UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN DEL TRANSPORTE DE VACUNAS MEDIANTE EL USO DE TECNOLOGÍA IOT Y MACHINE LEARNING

Curso académico 2020-2021

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

---

Documentos:

1.Memoria

2.Planos

3.Pliego de condiciones

4.Presupuesto

5.Anexos:

5.1: Documentación Código

5.2: Hojas de características

Tutores:

D. Ángel Francisco Perles Ivars

D. Jaime Laborda Macario

Autor:

D. José Herrero Ruiz

Agradezco a mis tutores, en especial a Ángel por toda la ayuda y apoyo que me ha ofrecido durante todo el desarrollo. A mi familia, por motivarme y apoyarme incondicionalmente. En particular, a mi madre, por luchar contra un cáncer y darme las fuerzas necesarias para seguir adelante. Gracias por ser mi mejor ejemplo de valentía.

Resumen

Con la llegada de la pandemia del Covid-19, la vacunación se ha convertido en uno de los pilares fundamentales para inmunizar a toda la población. Es por eso que, debido a esta necesidad, surge el problema logístico de hacer llegar a todo el mundo los viales de vacunación desde los laboratorios con la menor pérdida de eficacia posible.

Para lograr este objetivo es imprescindible considerar tanto la conservación de la cadena del frío durante todo el trayecto como el traslado cuidadoso de la carga de manera que ni se dañen las dosis ni se pierda parte de la efectividad.

Hoy en día, existen varias soluciones para el control de la cadena del frío, pero en ninguna se contempla el control de los movimientos soportados por la carga. Es por esta razón que, con la intención de monitorizar estos dos factores, se ha desarrollado este proyecto basado en la creación de un dispositivo apto para mantener la capacidad inmunizante de las vacunas mediante la creación de una inteligencia artificial entrenada para evaluar, clasificar e informar de cada uno de los movimientos más bruscos que se produzcan durante un recorrido.

Todo el sistema se ha implementado en una placa compuesta por un microprocesador que ejecuta la red neuronal en tiempo real, además de adquirir e informar sobre la temperatura de la carga y sobre los cambios térmicos que haya sufrido.

A pesar de las diversas tecnologías de comunicación inalámbricas que existen actualmente se ha empleado el Bluetooth de bajo consumo para la transmisión de datos ya que permite enviar toda la información al usuario con tan solo el uso de su teléfono móvil eliminando así la necesidad de manipular el dispositivo.

Este diseño exhibe la aplicabilidad de la inteligencia artificial para resolver problemas dónde la salud de las personas es el principal objetivo. Mostrando cómo la tecnología es capaz de mejorar nuestras vidas e incluso salvarlas.

Resum

Amb l'arribada de la pandèmia del Covid-19, la vacunació s'ha convertit en un dels pilars fonamentals per a immunitzar a tota la població. És per això que, a causa d'aquesta necessitat, sorgeix el problema logístic de fer arribar a tothom els vials de vacunació des dels laboratoris amb la menor pèrdua d'eficàcia possible.

Per a aconseguir aquest objectiu és imprescindible considerar tant la conservació de la cadena del fred durant tot el trajecte com el trasllat acurat de la càrrega de manera que ni es danyen les dosis ni es perda part de l'efectivitat.

Hui dia, existeixen diverses solucions per al control de la cadena del fred, però en cap es contempla el control dels moviments suportats per la càrrega. És per aquesta raó que, amb la intenció de monitorar aquests dos factors, s'ha desenvolupat aquest projecte basat en la creació d'un dispositiu apte per a mantindre la capacitat immunitzant de les vacunes mitjançant la creació d'una intel·ligència artificial entrenada per avaluar, classificar i informar de cadascun dels moviments més bruscos que es produïsquen durant un recorregut.

Tot el sistema s'ha implementat en una placa composta per un microprocessador que executa la xarxa neuronal en temps real, a més d'adquirir i informar sobre la temperatura de la càrrega i sobre els canvis tèrmics que haja patit.

Malgrat les diverses tecnologies de comunicació sense fils que existeixen actualment s'ha emprat el Bluetooth de baix consum per a la transmissió de dades ja que permet enviar tota la informació a l'usuari amb tan sols l'ús del seu telèfon mòbil eliminant així la necessitat de manipular el dispositiu.

Aquest disseny exhibeix l'aplicabilitat de la intel·ligència artificial per a resoldre problemes on la salut de les persones és el principal objectiu. Mostrant com la tecnologia és capaç de millorar les nostres vides i fins i tot salvar-les.

Summary

With the arrival of the Covid-19 pandemic, vaccination has become one of the fundamental pillars to immunize the entire population. That is why, due to this need, the logistical problem arises of getting the vaccination vials from the laboratories to everyone with the least possible loss of efficacy.

To achieve this goal, it is essential to consider both the preservation of the cold chain throughout the journey and the careful transfer of the load so that the doses are not damaged or part of the effectiveness is lost.

Nowadays, there are several solutions for the control of the cold chain, but none of them contemplate the control of the movements supported by the load. It is for this reason that, with the intention of monitoring these two factors, this project has been developed based on the creation of a device capable of maintaining the immunizing capacity of vaccines through the creation of an artificial intelligence trained to evaluate, classify and report of each of the most abrupt movements that might occur during transportation.

The entire system has been implemented on a board composed of a microprocessor that executes the neural network in real time, in addition to acquiring and reporting on the temperature of the load and the thermal changes it has undergone.

Despite the various wireless communication technologies that currently exist, low-consumption Bluetooth has been used for data transmission since it allows all the information to be sent to the user with only by using of their mobile phone, thus eliminating the need to manipulate the device.

This design exhibits the applicability of artificial intelligence to solve problems where people's health is the main objective. Showing how technology is capable of improving our lives and even saving them.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN DEL TRANSPORTE DE VACUNAS MEDIANTE EL USO DE TECNOLOGÍA IOT Y MACHINE LEARNING

---

1. Memoria

Curso académico 2020-2021

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

---

Autor:

D. José Herrero Ruiz

Tutores:

D. Ángel Francisco Perles Ivars

D. Jaime Laborda Macario

Índice de la memoria

Índice de figuras

Índice de tablas

Glosario de siglas y abreviaturas

IOT: Internet de las cosas

IA: Inteligencia Artificial

TF: Tensor Flow

FIFO: Primero en entrar, primero en salir

UART: Transmisor-Receptor Asíncrono Universal

MEMS: Tecnología electromecánica de dispositivos microscópicos

EI: Edge Impulse

SOC: Sistema en un chip

BLE: Bluetooth de baja energía

SPI: Interfaz de periféricos serie

I2C: Circuito inter-integrado

IMU: Sistema de unidad de medición inercial

GPIO: Entrada/Salida de Propósito General

DSP: Procesador de señales digitales

PWM: Modulación por ancho de pulsos

IDE: Entorno de desarrollo integrado

SRAM: Memoria estática de acceso aleatorio

TINYML: Técnica de aprendizaje automático

EPPROM: Memoria de solo lectura programable borrable

CLI: Interfaz de línea de comandos

CMD: Símbolo del sistema de Windows.

1. Objeto del proyecto

El propósito de este trabajo consiste en el diseño y desarrollo de un sistema de monitorización capaz de supervisar y verificar el correcto transporte de vacunas.

El objetivo principal, por lo tanto, consistirá en la detección de movimientos que puedan llegar a ser perjudiciales a la hora de trasladar una carga de un lugar a otro, debido a las inclemencias que surjan durante los trayectos de las mismas, ya que las distancias desde los laboratorios hasta los puntos de entrega pueden ser muy extensas, incluso en la mayoría de las ocasiones conllevando el cambio de diferentes medios de transporte.

En consecuencia, habrá que considerar en todo momento estas condiciones para crear un sistema lo suficientemente fiable y seguro, además de portable, que sea capaz de detectar correctamente todos los cambios que puedan ocurrir durante el trayecto.

Asimismo, el sistema diseñado tendrá que estar capacitado para controlar la temperatura a la cual se conserva la carga, puesto que la condición térmica es una de las más importantes a la hora de transportar cualquier tipo de material biológico, debido a la necesidad de conservación en su mismo estado, ya que, por el contrario, podría ocasionar una pérdida de potencia y eficacia vacunal.

Respecto a la programación implementada en el diseño, se va a hacer uso de la tecnología IoT que permite la interconexión de los diferentes dispositivos a la red, y de la inteligencia artificial, para que sea capaz de reconocer de la mejor forma posible los movimientos que se realicen a lo largo de una trayectoria.

Por último, considerando todo lo descrito previamente se va a poder diseñar una aplicación que sea capaz de monitorizar tanto la temperatura a la que deben viajar los viales como los movimientos más significativos, por ejemplo, si la carga se vuelca, si su inclinación no es la correcta o si recibe algún golpe que conlleve un movimiento brusco.

2. Antecedentes

La idea principal del proyecto deriva de los últimos acontecimientos vividos a lo largo de este último año. Con la llegada del covid-19 y la necesidad de desarrollo de una vacuna, surge la obligación de transporte de los viales desde los laboratorios hasta casi todos los países del mundo.

Debido a las condiciones de conservación tan restrictivas para la correcta conservación de la vacuna, como, por ejemplo, el mantenimiento por debajo del umbral de los -70 grados centígrados, resulta crucial el correcto traslado de las mismas.

Al mismo tiempo, florece la problemática de las empresas de transporte que deben garantizar que los portes han de adecuarse a las condiciones descritas previamente, obligándoles así a poseer algún sistema que respalde que la carga se ha transportado adecuadamente eximiéndoles de toda responsabilidad en el caso de que los viales lleguen dañados.

En consecuencia, a todos estos motivos explicados, resulta crucial este sistema de monitorización para que el transporte se efectúe de la manera más precisa y exacta posible, asegurando así el abastecimiento de vacunas a todos los lugares del mundo, evitando además cualquier contratiempo que pueda conllevar tanto el desperdicio de las vacunas como la pérdida de dinero.

3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes

Para el correcto desarrollo del trabajo será necesario establecer cuáles son las necesidades y circunstancias que envuelven al proyecto. A fin de tener una clara organización, fijaremos los factores a tener en cuenta y dividiremos el diseño en diferentes apartados.

3.1 Requisitos para el transporte de material sanitario

Según el Comité Asesor de Vacunas de la Asociación Española de Pediatría (2020), las condiciones fundamentales a la hora de abordar un sistema para el transporte de material sanitario son:

- Respetar y hacer cumplir las normas recomendadas por el laboratorio fabricante.
- Se debe realizar en contenedores especiales que garanticen la temperatura estable de conservación durante todo el trayecto.
- Deben quedar registrados en un albarán los siguientes datos: fecha de salida, lugar de destino, tipo de vacunas y presentación, cantidad de dosis, fecha de caducidad y lotes. (Capítulo 6 - Transporte y conservación de las vacunas).

Estos puntos clave indican que el desplazamiento de cualquier material biológico debe realizarse con la máxima precaución posible para que no se pierda la capacidad inmunizante.

3.2 Medida de variables

Uno de los factores más importantes a determinar es definir que movimientos van a ser reconocidos por el sistema. La idea principal es detectar los desplazamientos más bruscos que no correspondan con el movimiento del transporte por el cual está viajando la carga, es decir, golpes, traslaciones a altas velocidades, orientaciones incorrectas, y caída libre.

Además del movimiento, también será necesario medir la temperatura para tener un control sobre la cadena del frío de las vacunas.

En el caso de los sensores que se van a utilizar para medir estas variables, han de ser portables y con un consumo mínimo de potencia debido a que el sistema no va a poder ser conectado a la red eléctrica durante los desplazamientos.

3.3 Integración de Inteligencia Artificial

Para la medida de las variables, anteriormente mencionadas, se va a hacer uso de la inteligencia artificial, que tendrá la responsabilidad de interpretar los datos recogidos por los sensores y deberá ser capaz de reconocer cuáles son los movimientos que deben ser registrados, para posteriormente poder ser mostrados al final de un trayecto.

La inteligencia artificial, encargada de esta función, debe ser liviana, ya que va a tener que trabajar en procesadores con muy poca capacidad de procesamiento, debido a su pequeño tamaño.

3.4 Lectura e interpretación de datos

Una vez la información haya sido procesada por la inteligencia artificial, los datos recogidos durante los recorridos, tendrán que ser guardados en una memoria no volátil para que puedan ser leídos en cualquier momento.

Por consiguiente, la solución que se adopte, se encargará de enviar la información a otros dispositivos inteligentes como un smartphone, con el posterior tratamiento para que puedan ser leídos por cualquier persona. O, por el contrario, dichos datos tendrán que ser almacenados en una memoria interna del dispositivo, para que puedan ser extraídos cuándo se precise la lectura de los mismos.

4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

Teniendo en cuenta cuáles son los factores que debemos considerar para llevar a cabo el proyecto, se va a dividir el planteamiento de las soluciones alternativas por diferentes ámbitos de aplicación para tener una visión más clara a la hora de seleccionar la mejor solución posible.

Los apartados que van a ser objeto de estudio en esta sección son los siguientes:

-Sensores de temperatura: Se va a estudiar cual es el mejor sensor para este tipo de aplicaciones, teniendo en cuenta el tipo de carga que se está transportando.

-Plataformas de Inteligencia Artificial: Para la creación de las redes neuronales habrá que fijar el método de programación que resulta más efectivo a la hora de crear la inteligencia artificial.

-Placas de desarrollo para IA: Se van a examinar las diferentes opciones de placas capaces de soportar la ejecución de la inteligencia artificial con el menor tiempo de respuesta.

-Tratamiento de datos: Se establecerá cual es el mejor método para la recolección de los datos durante el trayecto.

-Portabilidad y almacenamiento de energía: Establecimiento del mejor método para la conservación de la energía.

4.1 Sensores de temperatura

Una de las principales necesidades del proyecto resulta en la capacidad de medir la temperatura, para poder conservar la cadena del frío de los viales de vacunación.

Como bien queda explicado en el apartado 3.1 de los requisitos para el transporte de material sanitario, uno de los puntos claves es que "Se debe realizar en contenedores especiales que garanticen la temperatura estable de conservación durante todo el trayecto".

Con el fin de lograr este objetivo, se ha propuesto el uso de los siguientes sensores, para poder monitorizar la temperatura durante el trayecto:

4.1.1 Sensor de temperatura LM35

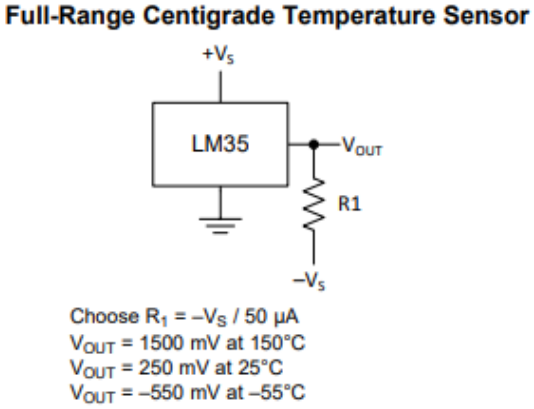
El sensor de temperatura LM35, es uno de los más conocidos por sus grandes prestaciones y su bajo coste. Posee una calibración de 1ºC de variación, lo que resultaría suficiente para el proyecto ya que las temperaturas a tratar van a ser siempre número enteros.

Además, por lo general, es capaz de abarcar un rango de temperaturas comprendido entre -55ºC y 150ºC.

Las características más destacables, las cuales podemos encontrar en el datasheet del fabricante (Texas Instrument, 1999) son:

- Calibrado directamente en grados Celsius (centígrados)
- Factor de escala lineal + 10 mV / ° C
- Precisión garantizada de 0,5 ° C (a 25 ° C)
- Clasificado para rango completo de −55 ° C a 150 ° C
- Adecuado para aplicaciones remotas
- Opera de 4 V a 30 V
- Drenaje de corriente inferior a 60 μA
- Bajo autocalentamiento, 0.08 ° C con aire en calma
- Salida de baja impedancia, 0,1 Ω para carga de 1 mA.

Figura 1. *Esquemático del sensor de temperatura LM35*



Nota. Esquema de conexión para configurar el sensor LM35 en modo de rango completo. Fuente: Texas Instrument, 1999.

El sensor LM35, es un buen candidato gracias a las capacidades que ofrece, ya que su rango de temperatura es de los más elevados en este tipo de sensores, y cuenta con un precio muy bajo.

La principal desventaja de este sensor es que, lógicamente, no viene integrado en la placa y debería ser instalado de forma externa, por lo tanto, resultaría en un aumento de tamaño del dispositivo en cuestión.

4.1.2 Sensor de temperatura HTS221

El sensor de temperatura de la marca ST, está diseñado en una placa ultra compacta y tiene la capacidad de medir también la humedad relativa del ambiente en el que se encuentra.

Este sensor se puede encontrar en una gran cantidad de placas compuestas por microprocesadores capaces de llevar a cabo aplicaciones del tipo IoT, como por ejemplo en la placa Arduino Nano 33 BLE Sense.

Su rango de temperatura es menor que el del sensor LM35, ya que oscila entre -40ºC y +120ºC.

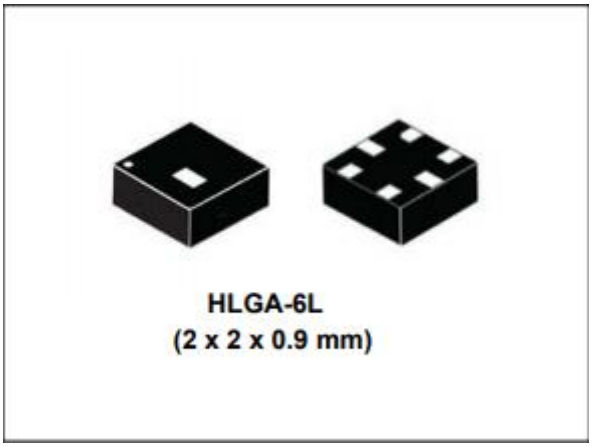
Sus principales características las podemos encontrar en el datasheet del fabricante (STMicroelectronics, 2016):

- Rango de humedad relativa de 0 a 100%
- Voltaje de suministro: 1,7 a 3,6 V
- Bajo consumo de energía: 2 μA a 1 Hz ODR
- ODR seleccionable de 1 Hz a 12,5 Hz
- Alta sensibilidad rH: 0.004% rH / LSB
- Precisión de humedad: ± 3,5% rH, 20 a + 80% rH
- Precisión de temperatura: ± 0,5 ° C, 15 a +40 ° C
- ADC integrado de 16 bits
- Datos de salida de temperatura y humedad de 16 bits
- Interfaces SPI e I²C

Las características ofrecen una viabilidad muy optimizada para el dispositivo que queremos diseñar ya que es capaz de combinar, la integración en la placa de desarrollo de Arduino Nano ofreciendo a la vez un rango de temperatura más que aceptable.

Con toda la suma de características que ofrece tanto de tamaño, de integración, la posibilidad de medir la humedad relativa del ambiente, su rango de temperatura y sobre todo el bajo consumo de energía hace que sea el candidato ideal para el diseño, ya que además viene integrado en la placa de Arduino y no será necesario ampliar el tamaño del dispositivo con otros sensores adicionales para la medición de temperatura.

Figura 2. *Representación visual del sensor de temperatura HTS221*



Nota. Esquema del sensor de temperatura ultra compacto de la marca STMicroelectronics, con las medidas de su tamaño de ocupación. Fuente: STMicroelectronics, 2016.

4.2 Plataformas de desarrollo de inteligencia artificial

Antes de comenzar, es necesario conocer que es la IA, y que tipos de aprendizaje existen según su finalidad.

> La inteligencia artificial es la habilidad de una máquina de presentar las mismas capacidades que los seres humanos, como el razonamiento, el aprendizaje, la creatividad y la capacidad de planear. La IA permite que los sistemas tecnológicos perciban su entorno, se relacionen con él, resuelvan problemas y actúen con un fin específico. (Duch Guillot, 2020).

Existen 3 tipos de aprendizaje:

- Aprendizaje Supervisado: Es un aprendizaje predictivo dónde la entrada y la salida se conocen. Los datos de entrada están clasificados y etiquetados.
- Aprendizaje No Supervisado: Parecido al supervisado, pero solamente se basan en el modelo predictivo, la salida no es de vital importancia y los datos de entrada no están clasificados.
- Aprendizaje Por Refuerzo: Consiste en un sistema de recompensas cuando la Inteligencia acierta en su salida, de esta manera se le enseña cuáles son las acciones correctas.

Para abordar el proyecto desde la mejor perspectiva posible es necesario que se parta desde el uso de TinyML para el procesamiento de los datos. Esta tecnología consiste en implementar una inteligencia artificial basada en el aprendizaje supervisado en un microprocesador de muy bajo consumo y con un código de programación de tan solo unos kilobytes.

Por ello, tanto el entorno como el lenguaje a utilizar para programar la IA va a ser clave para el correcto funcionamiento, ya que deberá permitir escribir programas de bajos recursos adaptados a las placas de procesamiento capaces de ejecutar tareas de redes neuronales.

A día de hoy, se puede encontrar una gran variedad de entornos de programación que nos permiten llevar a cabo estas tareas, pero para el caso de los microcontroladores existen dos grandes plataformas que debido a sus características hacen que sean ideales para su uso:

4.2.1 TensorFlow Lite

Es una plataforma de código abierto adaptada a los microprocesadores que nace de su predecesor TF, el cual, permite la programación de redes neuronales en muchos ámbitos, como, JavaScript, implementación en dispositivos móviles, y una herramienta extendida apta para el tratamiento de grandes cantidades de datos en modelos de alta producción.

TensorFlow Lite, está diseñado para microprocesadores ARM Cortex-M, además de ser compatible con otras plataformas como ESP32. Está codificado en C++ 11 y requiere una plataforma de 32 bits.

Las principales ventajas del uso de esta plataforma, es la versatilidad que ofrece para la creación de múltiples modelos de inteligencia artificial. Además de la sencillez que proporciona, gracias a los modelos de ejemplo que se pueden encontrar en los diferentes repositorios de Github.

Los pasos de creación de un modelo de IA vienen explicados por la propia plataforma en su página web (TensorFlow, 2021) y son:

- Entrena un modelo:
    - Genera un modelo de TensorFlow pequeño que pueda adaptarse al dispositivo de destino y contenga operaciones compatibles.
    - Convierte a un modelo de TensorFlow Lite con el conversor de TensorFlow Lite.
    - Convierte a un vector de bytes en C con herramientas estándar para almacenarlo en una memoria de programa de solo lectura en el dispositivo.
- Realiza inferencias en el dispositivo mediante la biblioteca de C++ y procesa los resultados.

A pesar de todas sus ventajas, TF tiene unas limitaciones, y es que la API de C++ requiere una administración manual de la memoria, además de que existe un subconjunto limitado de operaciones de TF Lite para microcontroladores.

4.2.2 Edge Impulse

Es una plataforma centrada en el desarrollo de inteligencia artificial para periféricos. Ofrece una gran cantidad de opciones desde un entorno muy sencillo de programación, que permite a los desarrolladores crear modelos neuronales avanzados.

Las principales ventajas de Edge Impulse es la sencillez de programación en IA, ya que gracias a su interfaz resulta muy sencillo modificar, recuperar y tratar los datos que van a ser utilizados por el modelo de la red neuronal.

Además, tiene una fácil implementación y una gran compatibilidad con muchas de las plataformas más utilizadas a día de hoy, resultando muy sencillo el proceso de portabilidad a las más importantes como, Arduino, Nvidia TensorRT, WebAssembly…

Las desventajas de esta plataforma residen en que no es completamente gratuita ya que funciona según unos planes de uso, los cuáles encontramos en su página web, y consisten en:

Tabla 1. *Planes de desarrollo de Edge Impulse*

Planes	Desarrollador	Empresa
Modelo	✔	✗
Colaboración en Equipo	✗	✔
Proyectos	5	Ilimitado
Tiempo de Cómputo Incluido	25 min	10000 min
Apoyo	Foro	Soporte comercial de servicio
Derechos Libres	✔	✔
Adquisición de datos de dispositivos en vivo	✔	✔
Diseñador de Impulsos TinyML	✔	✔
Bloques comunitarios	✔	✔
Bloques personalizados	✔	✔ Alojados
Clasificación en vivo	✔	✔
Modelo de Validación	✔	✔
Despliegue	✔	✔
Bloques privados empresariales	✗	✔
Depósito de datos en la nube de clientes	✗	✔
Transformación de datos	✗	✔

Nota. Fuente: Edge Impulse Inc. , 2020.

Como se puede apreciar en la tabla, el plan gratuito para desarrolladores es más que suficiente para crear un proyecto de IA. Y, a pesar de que el tiempo de cómputo de entrenamiento esté limitado a un tiempo de 25 minutos, será más que suficiente como para que la red neuronal aprenda cuáles son los movimientos que deberá reconocer.

Una vez expuestas las ventajas y desventajas de las dos principales plataformas, para el desarrollo de este trabajo se va a seleccionar el entorno de desarrollo Edge Impulse, debido a: las grandes ventajas que da en términos de visualización gráfica a la hora del entrenamiento, la posibilidad de implementación en Arduino, el control de diferentes versiones del proyecto, la clasificación en vivo de los movimientos, la posibilidad de volver a entrenar un modelo previamente creado y la sencillez de implementación en las placas con procesador ARM Cortex-M4.

Todo esto hace que, a pesar de que la comunidad de TF sea mucho más amplia, resulte más sencillo y fácil de implementar el uso de Edge Impulse, haciendo que sea la plataforma idónea para programar este tipo de proyectos en los que la versatilidad a la hora del aprendizaje automático resulta esencial para poder controlar todas las variables que comprometen a la red neuronal en tiempo real.

4.3 Placas de desarrollo para la IA

Una vez conocidos los entornos de desarrollo y el lenguaje en el cual se va a trabajar debemos establecer cuál es la mejor placa que satisface nuestros objetivos atendiendo a los requerimientos previamente explicados de: portabilidad y sensores necesarios.

Algunas de las placas que encontramos capaces de llevar a cabo esas funciones son:

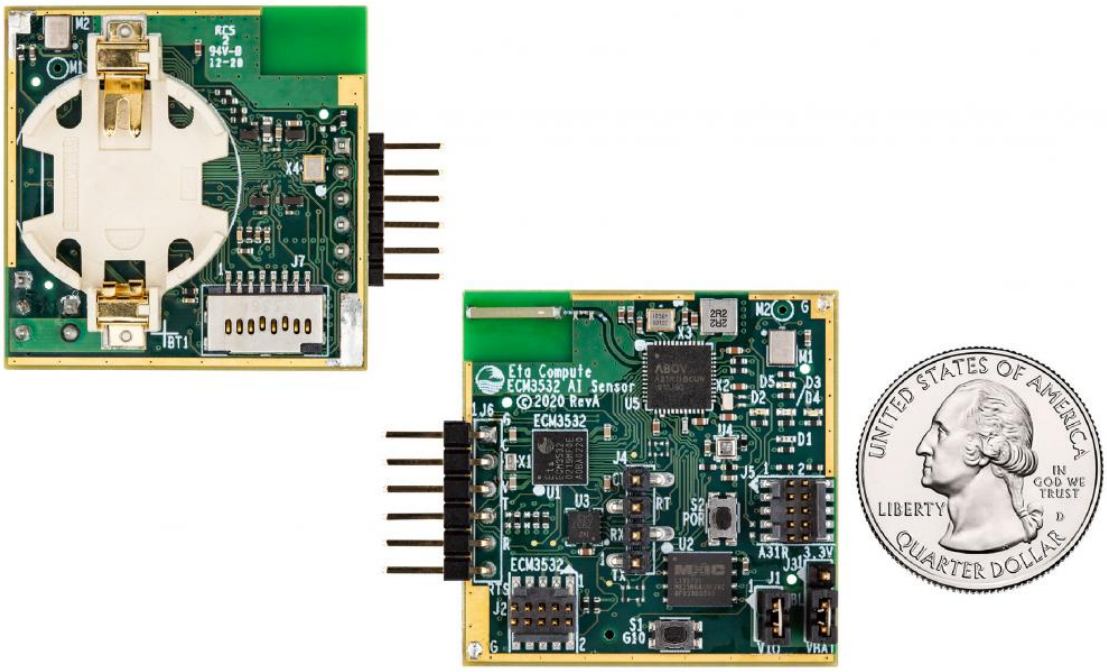
4.3.1 Eta Compute ECM3532 AI Sensor

Es una pequeña placa de desarrollo que presenta el ECM3532 TENSAI SoC con un microcontrolador Cortex-M3 y un CoolFlux DSP separado para acelerar las operaciones de aprendizaje automático. Las características de las placas se encuentran en la página del fabricante (Eta Compute, 2021) y son las siguientes:

- Placa de 1.4 x 1.4 pulgadas con sensores y procesador de sensor neural ECM3532.
    - 2 micrófonos MEMS: TDK-Invensense ICS-41350.
    - 1 x sensor de presión / temperatura: BOSCH BMP388.
    - 1 x MEMS de 6 ejes Acelerómetro/Giroscopio: TDK-Invensense ICM-20602.
- Soporte de batería para batería CR2032.
- Bluetooth de baja energía: BLE v4.2: ABOV A31R118 y antena.

- UART de 6 pines y puerto de alimentación.
- Flash serie de 64 Mbit para registro de datos.
- 5 Leds y un pulsador.

Figura 3. *Placa Eta Compute ECM3532 AI Sensor*



Nota. Comparación del tamaño de la placa Eta con un centavo estadounidense. Fuente: Eta Compute, 2021a.

Esta placa ofrece todo lo necesario para el proyecto e incluso trae su propio espacio para insertar una pila del tipo CR2032, solucionando un apartado importante en cuanto al diseño. Además, el tamaño de la placa es de tan solo 3,56 cm de lado, lo que hace que sea ideal. Pero el gran inconveniente de esta placa es el precio, ya que la podemos encontrar a partir de los 99€. Lo que consideramos que es un precio desorbitado en comparación a otras soluciones.

4.3.2 Himax WE-I Plus

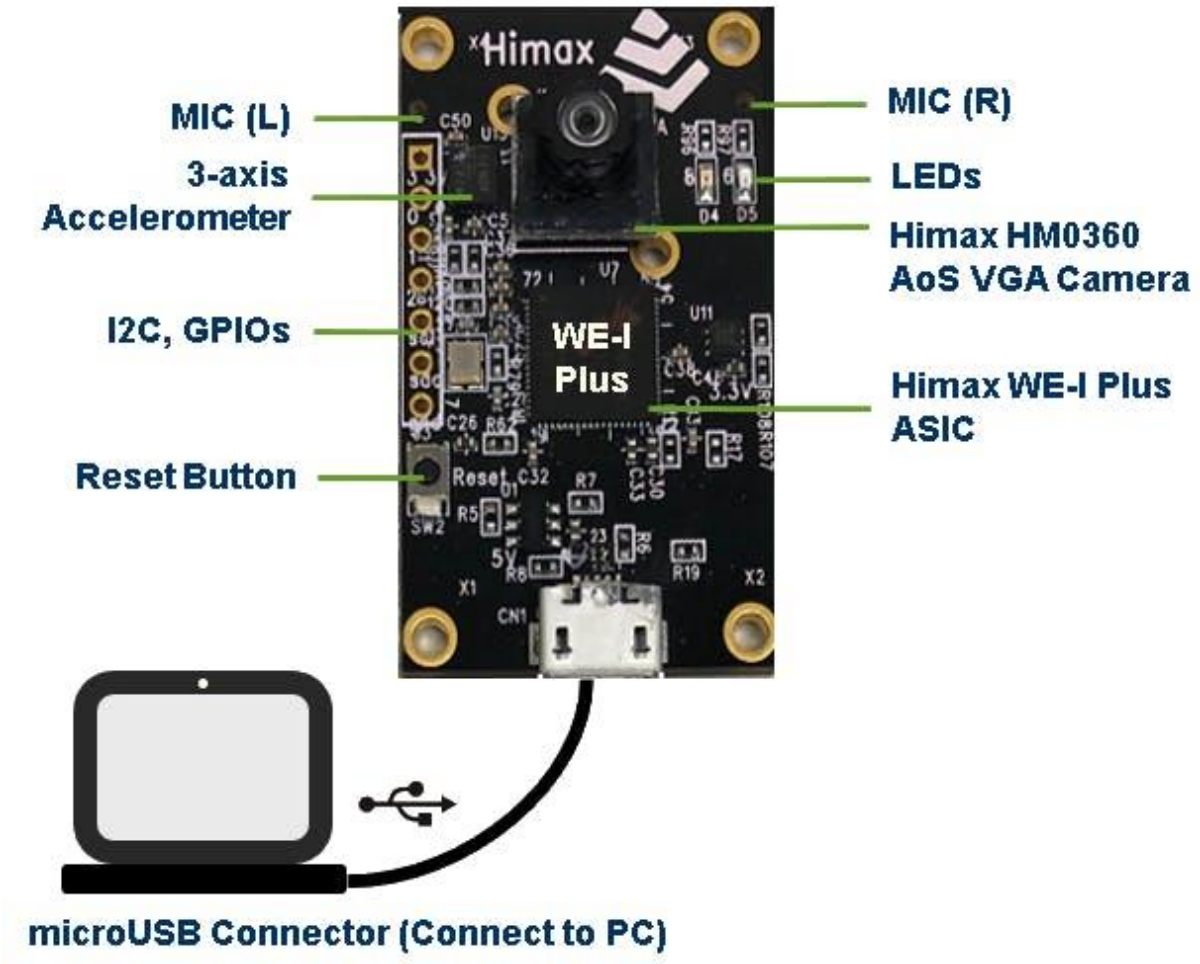
La placa compacta de la empresa Himax, es una buena candidata debido a las prestaciones que ofrece, ya que contiene la mayoría de los sensores a utilizar para el proyecto. Está optimizada para la implementación del modelo TinyML ya que ha sido desarrollada junto a Google para que sea compatible con TensorFlow Lite para microcontroladores.

Según el Datasheet de la empresa (Himax, 2020a), las características de la placa son:

- WE-I Plus ASIC (HX6537-A)
    - DSP EM9D ARC de 32 bits.
    - Frecuencia de reloj de 400 MHz.
    - 21 MB de SRAM.
    - Flash de 2 MB.
- En la placa
    - CCM VGA de potencia ultrabaja Himax HM3360 MS TM.
    - FTDI USB a SPI / I2C / UART.
    - Fuente de alimentación LDO (3.3/2.8/1.8/1.2V).
    - Acelerómetro de 3 ejes.
    - l x botón de reinicio.
    - 2 x micrófonos (UR).
    - 2 x Leds.
    - conector microUSB.
- Cabecera de expansión
    - 1 x Master 12C.
    - 3 x GPIOs.
    - Encendido / Tierra.

Como se puede apreciar, la placa de Himax ofrece un acelerómetro de 3 ejes, 2 micrófonos y 2 leds, debido a la necesidad de medir la temperatura de la carga que se va a transportar, es necesario un sensor de temperatura que en el caso de esta placa habría que incorporársela mediante una de las 3 entradas genéricas del chip. Además, la falta de giroscopio es otro de los grandes problemas ya que a pesar de que le instalemos uno externo se dificultará considerablemente el entrenamiento de la inteligencia artificial debido a que el acelerómetro y el giroscopio no estarán situados en el mismo punto, por lo tanto, los ejes de referencia serán distintos en cada uno de los sensores, provocando que los datos recogidos no sean válidos debido a la desincronización que puede existir en las medidas.

Figura 4. *Placa WE-I Plus EVB*



Nota. Vista de la ubicación de los sensores de la placa WE-I Plus EVB incluidos dentro de la pcb. Fuente: Himax, 2020b.

Debido a la falta de sensores de la placa, explicados previamente y al precio de la placa, que se puede encontrar a partir de 55,28€ en SparkFun. Se ha considerado que, aunque sí que podría ser una solución aceptable, no sería la más óptima posible.

4.3.3 B-L475E-IOT01A

El kit que nos ofrece STM32 para las aplicaciones IoT puede ser una alternativa muy rentable con una fiabilidad muy alta para el proyecto ya que incorpora todos los sensores necesarios para el diseño, desde el giroscopio hasta el sensor de temperatura. Además, con la certificación de AWS de Amazon promete una conectividad total con la nube gracias al módulo de radiofrecuencia que incorpora.

La placa se basa en un núcleo Cortex-M4 STM32L4, con soporte para Arduino Uno V3 y PMOD lo cual proporciona una interfaz ideal para conectar el microcontrolador con una gran variedad de dispositivos lógicos programables.

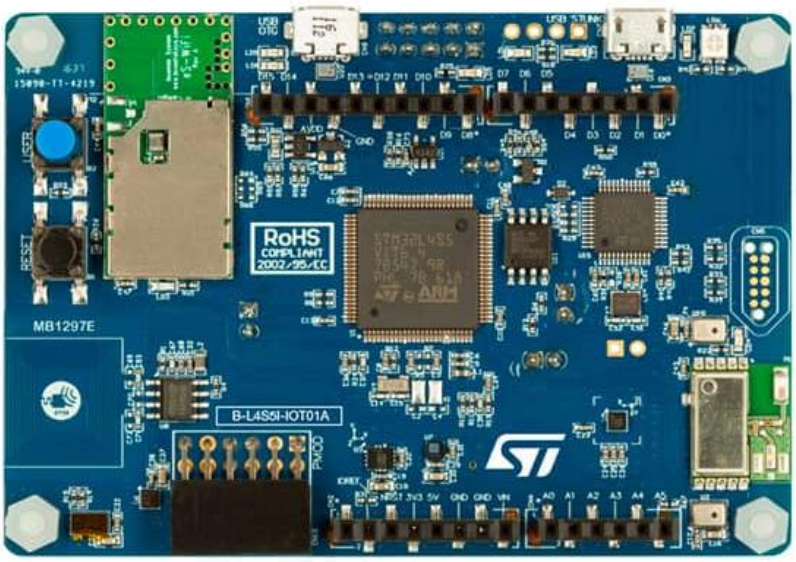
Las características que ofrece la placa según el fabricante (STM, 2021) son:

- STM32L4 Series MCUs based on Arm Cortex-M4 con 1 Mbyte de memoria de flash y 128 Kbytes de memoria SRAM, en el paquete LQFP100.
- Memoria Flash Quad-SPI (Macronix) de 64 Mbit.
- Bluetooth V4.1 módulo (SPBTLE-RF).
- Módulo de RF programable de baja potencia (868 MHz o 915 MHz) (SPSGRF-868 o SPSGRF-915).
- Módulo Wi-Fi compatible con 802.11 b / g / n de Inventek Systems (ISM43362-M3G-L44).
- Etiqueta NFC dinámica basada en M24SR con antena NFC.
- 2 micrófonos omnidireccionales digitales (MP34DT01).
- Sensor digital capacitivo para humedad relativa y temperatura (HTS221).
- Magnetómetro de 3 ejes de alto rendimiento (LIS3MDL).
- Acelerómetro 3D y giroscopio 3D (LSM6DSL).
- Barómetro de salida digital absoluta de 260-1260 hPa (LPS22HB).
- Sensor de tiempo de vuelo y detección de gestos (VL53L0X).
- 2 pulsadores (usuario y reset).
- USB OTG FS con conector Micro-AB.
- Conectores de expansión:
    - Arduino Uno V3.
    - PMOD.
- Opciones de fuente de alimentación flexibles:
    - ST LINK USB V_{BUS} o fuentes externas.
- Depurador / programador ST-LINK / V2-1 con USB integrado: almacenamiento masivo, puerto COM virtual y puerto de depuración.

- Biblioteca HAL completa de software gratuito que incluye una variedad de ejemplos, como parte del paquete STM32Cube MCU.
- Apoyo de una amplia variedad de entornos de desarrollo integrado (IDE), incluyendo IAR, Keil, GCC-based IDEs, Arm Mbed Enabled.
- Arm Mbed online.

Son muchas las opciones de las cuales se dispone si se selecciona la placa de STM32, pero se tendrá que tener en cuenta que el apartado anterior de la elección del entorno de programación debería ser cambiado por un entorno más amigable para esta clase de placas como puede ser Keil, o STMCube ya que va a permitir un mayor flexibilidad y sencillez a la hora de la programación.

Figura 5. *Placa B-L475E-IOT01A*



Nota. Vista gráfica del montaje de la placa B-L475E-IOT01A con la disposición de los sensores en la pcb. Fuente: (STMicroeletronics, 2017).

A pesar de todas las ventajas que incorpora la placa de STM, se ha considerado que, debido a su precio, que se puede encontrar a partir de 46€ en Digi-Key, no es la opción más viable, ya que, aunque incorpore mucha tecnología, esta no va a ser utilizada para el diseño por lo que supondría un malgaste importante de componentes como por ejemplo el NFC, o la antena de radiofrecuencia.

4.3.4 Arduino Nano 33 BLE Sense

Esta placa es la evolución del Arduino Nano tradicional, pero con la implementación de un procesador mucho más potente, el nRF52840 de Nordic Semiconductors con una CPU ARM Cortex-M4 de 32 bits que funciona a 64 MHz, permitiendo una capacidad de procesamiento excelente para todas las inferencias que ha de llevar a cabo la IA.

Las características que nos ofrece la placa Nano 33 vienen descritas en su página web:

Tabla 2. *Características de la placa Arduino Nano 33 BLE Sense*

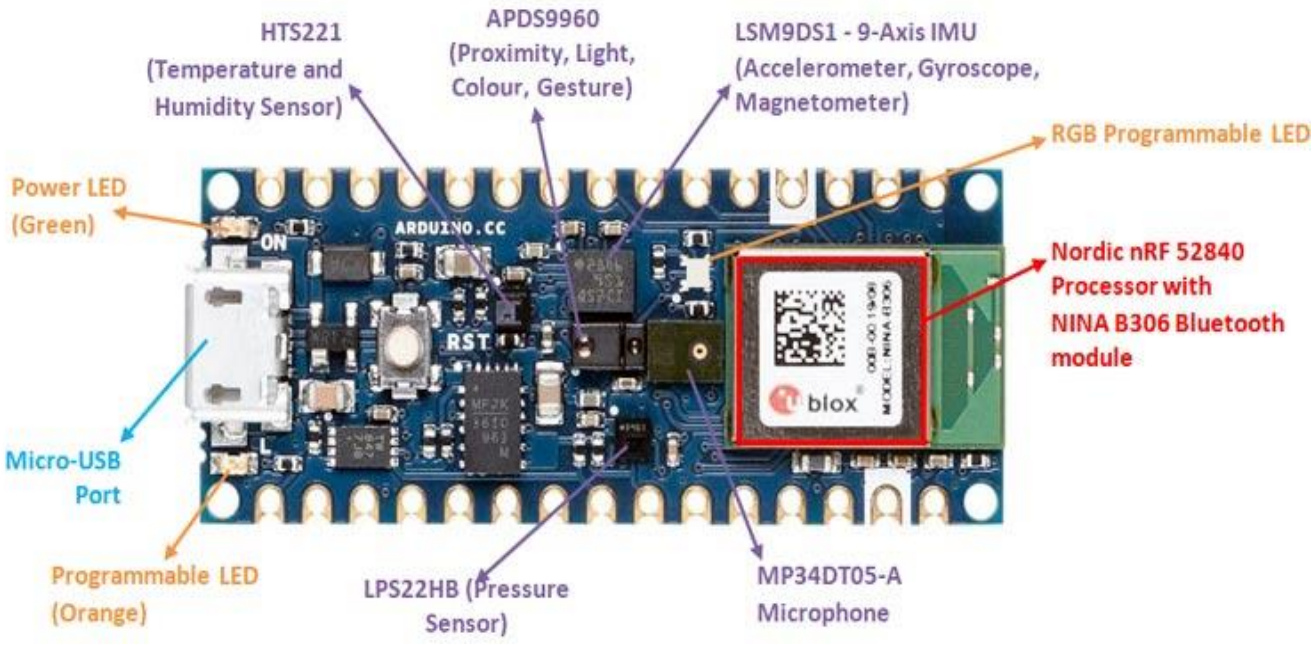
Microcontrolador	nRF52840
Tensión de funcionamiento	3,3 V
Voltaje de entrada (límite)	21V
Corriente CC por pin de E / S	15 mA
Velocidad de reloj	64 MHz
Memoria flash de la CPU	1 MB
SRAM	256 KB
EEPROM	Ninguno
Pines de entrada / salida digital	14
Pines PWM	Todos los pines digitales
UART	1
SPI	1
I2C	1
Pines de entrada analógica	8
Pines de salida analógica	Solo a través de PWM
Interrupciones externas	Todos los pines digitales
LED_BUILTIN	12
USB	Nativo en el procesador
IMU	LSM9DS1
Micrófono	MP34DT05
Gesto, luz, proximidad	APDS9960
Presión barométrica	LPS22HB
Temperatura, humedad	HTS221
Longitud	45 mm
Ancho	18 mm
Peso	5 gr

Nota. Fuente: Arduino,2021.

Son muchas las ventajas que nos proporciona la placa de Arduino gracias a su reducido tamaño y a un precio muy competitivo ya que la podemos encontrar a partir de 35€.

La solución que nos ofrece Arduino con su placa Nano 33 BLE Sense es la más completa ya que su placa nos ofrece los sensores necesarios para todo el procesamiento de datos, proporcionándonos además una compatibilidad completa con Edge Impulse, con una sencillez de integración de la interfaz de Arduino inigualable. Además, las bibliotecas a utilizar son facilitadas por el propio programa con tan solo una simple búsqueda.

Figura 6. *Placa Arduino Nano 33 BLE Sense*



Nota. Visualización de los componentes que incorpora la placa de Arduino, incluido todos los sensores, la alimentación por micro USB y el procesador nRF52840. Fuente: Ecuarobot, 2020.

Teniendo en cuenta todas las variables que son necesarias para seleccionar la mejor placa, desde los sensores que llevan implementados, hasta el entorno de desarrollo pasando por el tamaño, la capacidad de portabilidad, y el precio. Se ha concluido que la mejor elección era la placa Arduino Nano 33 BLE Sense, gracias al gran rendimiento que ofrece, y a la implementación de todos los sensores necesarios al menor precio

Además de todo esto, cabe recordar tanto su sencillez a la hora de implementar redes neuronales, como su bajo consumo a la hora de ejecutarlas, y todo esto sin olvidar a la gran comunidad que hay detrás de Arduino para solventar cualquier contratiempo ante una situación

problemática a la hora de llevar a cabo la programación. Todo en su conjunto hace que la placa Nano 33 BLE Sense sea ideal para este proyecto.

4.4 Tratamiento de datos

La recopilación de datos es uno de los aspectos fundamentales del proyecto debido a la necesidad que surge de tener que interpretar cuáles han sido los movimientos más significativos que ha sufrido la carga o si han existido cambios de temperatura bruscos que puedan afectar a la eficacia de los viales.

En consecuencia, se proponen las siguientes soluciones para el problema:

4.4.1 Almacenamiento interno

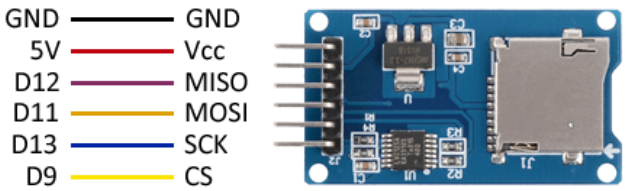
La propuesta de esta clase de medios supone la conexión de tarjetas de memoria para la recopilación de los datos.

4.4.1.1 Tarjeta MicroSD

La instalación de un módulo MicroSD Card Slot, puede resultar una buena solución, ya que de esta manera tendríamos una memoria no volátil, que permitiría recopilar todos los datos recogidos por la IA, para después ser tratados de una manera más visual, permitiéndonos la creación de gráficas de temperatura y de movimiento mediante herramientas como por ejemplo Microsoft Excel.

Además, actualmente con el desarrollo de la tecnología podemos encontrar tarjetas SD de hasta 256GB lo que otorgaría al dispositivo un almacenamiento para casi toda su vida útil.

Figura 7. *Módulo de tarjeta MicroSD*



Nota. Esquemático de conexiones de una tarjeta MicroSD. Fuente: Llamas, 2016.

En la Figura 8, se puede visualizar como se llevarían a cabo las conexiones del módulo lector a la placa de Arduino que se va a utilizar.

Aunque parece una solución muy adecuada gracias a la sencillez del sistema para recopilar datos. Se ha considerado que podría llegar a resultar molesto, tener que extraer la tarjeta SD cada vez que se quieran mostrar los resultados, además cuando la tarjeta no estuviese insertada, los datos no podrían ser leídos, perdiéndose información.

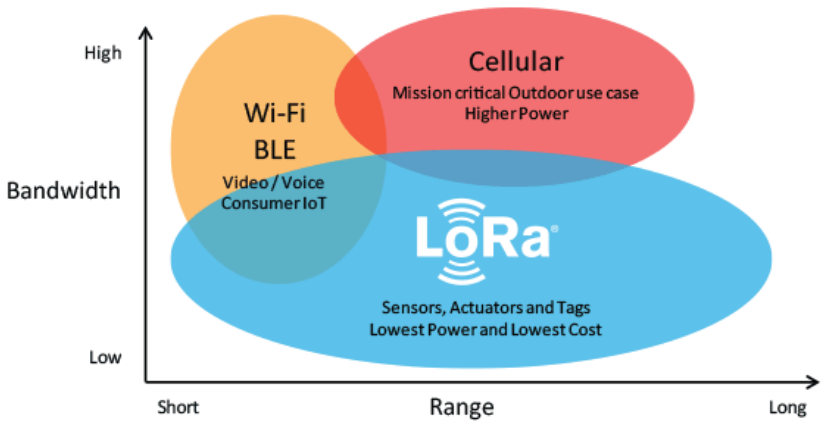
4.4.2 Transmisión inalámbrica

4.4.2.1 LoRa

Esta tecnología consiste en la implementación de un módulo de radiofrecuencia que funciona mediante nodos, es decir, la comunicación se realiza a través de enlaces LoRa, generalmente llamados Gateways, encargados de transportar la información a internet y finalmente a un usuario final.

Las principales ventajas de este medio de comunicación es su larga distancia y su bajo consumo. Se pueden llegar a alcanzar comunicaciones de hasta 15-20 km, superando así a la tecnología WiFi y Bluetooth. La principal desventaja es su poca capacidad de transmisión de tan solo 5.5kbps, lo que hace imposible la transmisión de audio y vídeo. Pero este aspecto no afectará ya que exclusivamente se van a transmitir datos de sensores con unos tamaños muy reducidos.

Figura 8. *Gráfico de anchos de banda de las tecnologías inalámbricas*

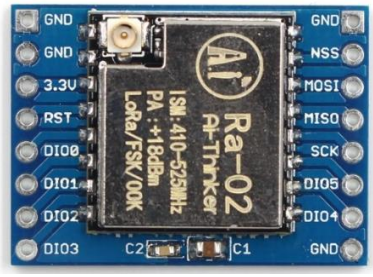


Nota. Visualización del ancho de banda de LoRa para el entendimiento de su funcionamiento y de su larga alcance debido la frecuencia de trabajo. Fuente: CircuitDigest, 2019.

En el caso del proyecto que vamos a diseñar, esta tecnología resulta ideal y casi brillante pero debido a que la placa seleccionada, Arduino Nano 33 BLE, no tiene implementada un

módulo LoRa, sería necesario conectarlo de forma externa. Además, la distancia a la que se transmiten los datos no es de una importancia tan significativa ya que cuando se viaja con una carga biológica las distancias a recorrer van a ser muy superiores a las 15-20 km, por lo tanto, cualquier tecnología capaz de comunicarse con el transportista situado generalmente a pocos metros será válida.

Figura 9. *Módulo Transceptor LoRa SX1278 Ra-02 433 Mhz con Base*



Nota. Visualización de un módulo para conexión LoRa mediante tecnología de radiofrecuencia. Fuente: Electronilab, s. f.

No obstante, si se quisiese implementar esta alternativa, podría ser muy interesante ya que, lógicamente mejoraría significativamente la calidad de la señal, aumentando así las especificaciones del conjunto.

4.4.2.2 Bluetooth Low Energy

La tecnología Bluetooth, mundialmente conocida y utilizada, se basa en la interconectividad de los dispositivos mediante un enlace de radiofrecuencia a 2,4GHz. Está especializada tanto en la transferencia de datos, como de audio y vídeo a distancias cortas.

Sin embargo, se va a hacer uso de Bluetooth Low Energy, una tecnología que resulta casi idéntica a la anterior, pero con unas diferencias que le convierten en el candidato perfecto.

Para continuar, es necesaria la explicación del funcionamiento de BLE y cómo se va a integrar en este proyecto.

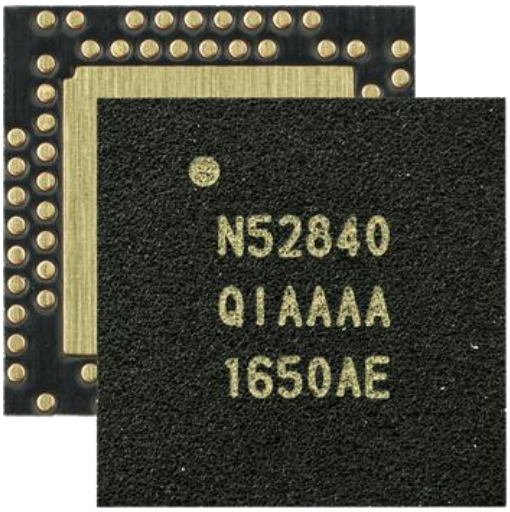
Bluetooth Low Energy consiste en: (Pastorino, 2020)

Al igual que Bluetooth, BLE opera en la banda ISM de 2.4 GHz. Sin embargo, a diferencia del Bluetooth clásico, BLE permanece en modo de suspensión constantemente, excepto cuando se inicia una conexión. Los tiempos de conexión reales

son solo de unos pocos milisegundos, a diferencia de Bluetooth, que tomaría más de 100 milisegundos. En las redes Bluetooth de baja energía los dispositivos pueden ser centrales o periféricos. Los dispositivos centrales (teléfonos inteligentes, tabletas, computadoras, etc.) tienen mayor capacidad de procesamiento y son responsables de controlar los dispositivos periféricos. Los dispositivos centrales generalmente ejecutan software creado específicamente para interactuar con dispositivos periféricos. Estos últimos sirven como sensores que recopilan datos y los envían a dispositivos centrales para su procesamiento. La clave del bajo consumo es que no procesan datos, solo lo recogen.

Gracias a su método de funcionamiento, consigue mantenerse en modo de suspensión constantemente, excepto en el momento de inicio de una conexión. Lo que le permite ahorrar mucha energía y así poder funcionar con una pequeña batería.

Figura 10. *Procesador nRF52840*



Nota. Procesador incluido en la placa de Arduino Nano 33, con tecnología Bluetooth Low Energy integrada. Fuente: Nordic Semiconductor, 2021.

El punto más a favor de esta técnica es que viene implementada en la placa que se va a utilizar, lo que hace que sea la solución ideal para este proyecto, ya que es capaz de llevar a cabo todas las funciones de transferencia de datos, sin la molestia de tocar el dispositivo y con tan solo la necesidad de poseer un smartphone.

4.5 Portabilidad y almacenamiento de energía

El aspecto de la portabilidad resulta crucial debido a que el dispositivo tiene que estar preparado para funcionar sin ninguna fuente de energía conectada a la red eléctrica.

Por ende, se derivan las siguientes soluciones:

4.5.1 Batería de 9V

Las pilas de 9V son una opción fiable debido al poco espacio que ocupan y a la facilidad de montaje ya que teniendo en cuenta que la entrada +Vin del Arduino Nano 33, puede oscilar desde los +4.5V hasta los +21V, solamente se tendría que llevar a cabo la instalación de los cables +Vin al positivo y GND a la referencia del Arduino para que este se encendiese.

Lo que resulta la forma más sencilla y rápida de poder mantener encendido el dispositivo durante el trayecto. Pero sería incómodo y costoso tener que cambiarlas cada vez que se agotase su carga. Además, el uso continuado de baterías no recargables implica un mayor riesgo para el medio ambiente.

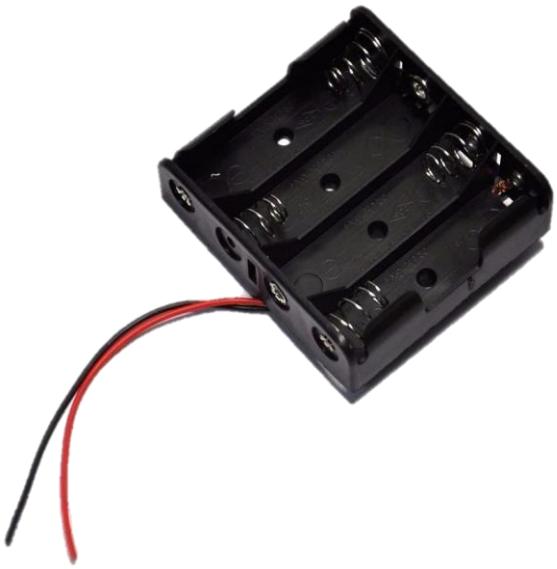
Figura 11. *Pila 9V*

Nota. Fuente: Vinibattery, s. f.

4.5.2 Pilas de 1.5V

Otra opción es la adquisición de pilas AA. Estas poseen un voltaje de 1.5 V por lo que sería necesario comprar un portapilas donde poder ubicar a las 4 en serie para obtener de esta manera el voltaje necesario, es decir, 4 Pilas x 1.5 V = 6 V.

Figura 12. *Portapilas para 4 pilas AA*



Nota. Fuente: Iberobotics,2021.

El principal inconveniente es que, como la corriente en serie se mantiene y estas pilas suelen oscilar desde los 1000mAh hasta los 2500 mAh según el tipo de componente del cual estén fabricadas, podría acarrear un gasto excesivo tener que cambiar las pilas cada poco tiempo. Además, en asuntos como el espacio de ocupación del dispositivo, el tamaño y el peso de 4 pilas AA en serie haría que fuese demasiado desorbitado.

Por eso, aunque si es cierto que las pilas AA, son las más económicas del mercado se ha considerado que no es la opción más viable, ya que la necesidad del cambio de pilas constante, hace que sea molesto, caro e incluso perjudicial para el medio ambiente.

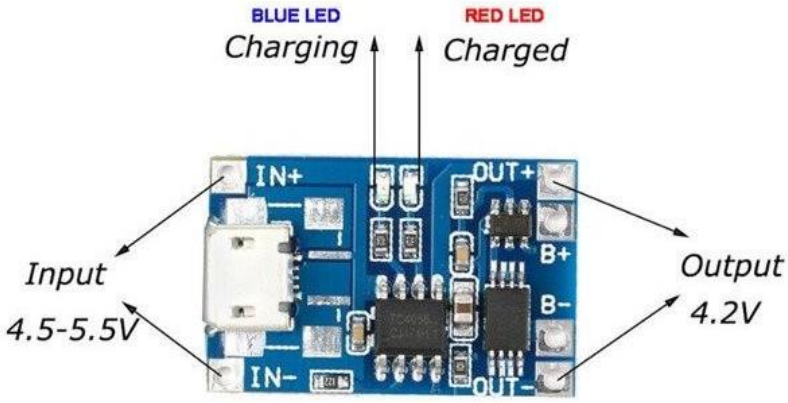
4.5.3 Batería recargable

Este tipo de pilas es más conocido como pilas 18650 y tienen una capacidad de almacenamiento muy elevada, en el caso de este proyecto se va emplear una batería de 9900mAh, lo que supone una carga mucho más que suficiente.

Pero sí que va a resultar necesaria la instalación de dos módulos adicionales tanto para la carga y descarga de la batería como para el acondicionamiento del voltaje de la señal de entrada.

Para la carga y descarga se va a hacer uso de la placa TP4056, un módulo que permite la carga de baterías del tipo 18650 mediante una conexión Micro USB, con un cargador que suministre un voltaje de entrada de 5V.

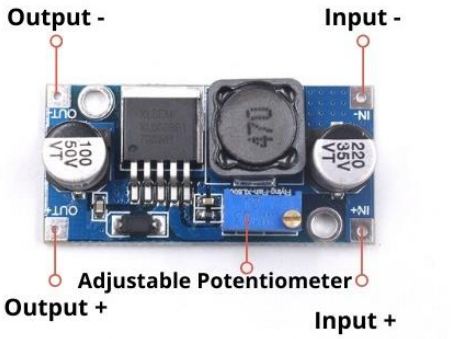
Figura 13. *Placa TP4056*



Nota. Esquema de conexionado de la placa TP4056 para baterías de tipo 18650. Fuente: Solectroshop, s. f.-a.

Debido a que la placa de Arduino Nano 33 BLE, debe de ser alimentada por el pin +Vin, y el mínimo voltaje permitido es de +4,5 V, será necesario incorporar, por ejemplo, el módulo XL6009 Convertidor DC a DC Boost Step-Up 3 V-32 V, que consiste en un regulador de voltaje capaz de aumentar la salida de la batería desde los 3,7 V hasta el voltaje deseado, que en este caso se podría hasta los +21 V, siendo este el máximo que permite la placa de Arduino.

Figura 14. *Placa XL6009 Convertidor DC a DC Booster Step-Up*



Nota. Esquema de conexionado de la placa XL6009 habilitada para la conversión de voltajes de corriente continua apta para la batería 18650 de 3,7 V utilizada en el proyecto. Fuente: Roboelements, s. f.

Con todo lo descrito previamente podemos concluir que este método de carga y descarga de la batería es el más adecuado debido a su poco tamaño y peso, ya que la batería 18650 mide y pesa lo mismo que una pila AA. Pero con una capacidad de carga muy superior, además de la posibilidad de poder cargarse mediante un micro USB.

Por todo esto, se ha llegado a la conclusión de que esta es la mejor opción para el diseño propuesto.

5. Descripción detallada de la solución adoptada

La finalidad de este apartado consiste en poder ver como se ha desarrollado la solución de principio a fin. Teniendo la posibilidad de entender todo el proceso de aprendizaje de la Inteligencia Artificial.

Adquiriendo de esta manera todas las capacidades para poder simular o recrear este tipo de proyectos, pudiendo solventar las errores más comunes y no tan comunes que surgen a la hora de programar redes neuronales y adaptarlas a microcontroladores.

Además, en cuanto a los aspectos técnicos se profundizará en todos los dispositivos electrónicos que se van a utilizar en el proyecto. Posibilitando así al lector, entender el porqué de la elección de todos los componentes a utilizar, y de dónde proviene la necesidad de uso de los mismos.

Toda la suma de las consideraciones anteriores va a estar muy presente durante este apartado, ya que resulta crucial no olvidarlas en todo momento, eliminando la posibilidad de que se olvide la finalidad del proyecto, que consiste en el transporte de material biológico y debe ser conservado y tratado con unas precauciones superiores a otros tipos de carga.

Esta solución incluye el proceso de creación tanto de software como de hardware del dispositivo de monitorización, fabricado a partir de placas independientes, intentado obtener el precio más ajustado posible y ofreciendo una excelente calidad, descartando cualquier opción comercial preexistente tanto para el control de la temperatura como para el reconocimiento de movimientos.

Siguiendo en la línea del trabajo, para facilitar la compresión tanto del proceso de fabricación como del diseño, este punto se dividirá en los siguientes subapartados:

- Empleo de sensores para el control de la temperatura
- Desarrollo e integración de la IA
- Establecimiento del protocolo de envío de datos al usuario
- Carga y portabilidad del dispositivo
- Diseño del software

5.1 Empleo de sensores para el control de temperatura

La necesidad de conservación de la cadena del frío, es uno de los retos más importantes de este proyecto, el hecho de medir temperaturas tan extremas de la manera más económica posible, hace que surja la obligación de emplear sensores térmicos capaces de conservar la carga sin perjudicarla.

Por ello, como se introdujo en el apartado 4.1 de soluciones alternativas, se llegó a la conclusión de que el uso del sensor ST HTS221, era la mejor opción debido las prestaciones que ofrece y gracias al ahorro de espacio que suponía no instalar un sensor de temperatura externo a la placa que se va a utilizar.

El rango de trabajo del sensor es de -40ºC hasta +120ºC, lo que igual si puede derivar en algún problema para transportar cierto material biológico que requiera de temperaturas extremas, pero por norma general, este rango es bastante adecuado y será apto para la mayoría aplicaciones.

Respecto a la programación del dispositivo, resulta muy sencilla, ya que las bibliotecas pueden ser descargas desde el propio programa de Arduino, incluyendo además ejemplos de uso. Si se quiere integrar en algún proyecto de Arduino, será necesario incluirla de la siguiente manera:

```
#include <Arduino_HTS221.h>
```

Para la lectura de las variables, se hará uso del siguiente código:

- `float temperature = HTS.readTemperature();`

De esta manera se guardará el valor de temperatura en la variable “temperature”.

Como es apreciable, su uso es muy sencillo, lo que facilita la programación para este tipo de aplicaciones en el que la monitorización de la temperatura resulta crucial. Además, el consumo de energía que necesita este sensor es de tan solo 2 μA a 1Hz, lo que hace que sea perfecto para el diseño de este dispositivo dónde el bajo consumo de batería es esencial.

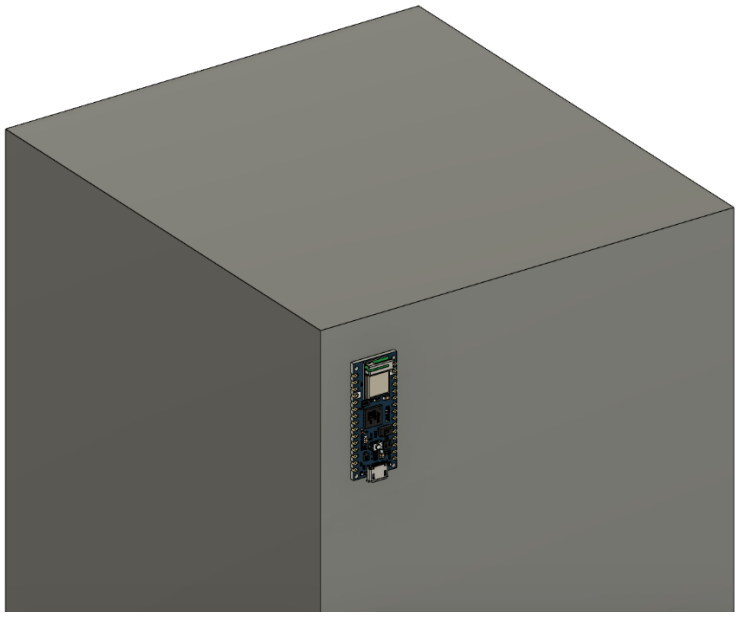
5.2 Desarrollo e integración de la inteligencia artificial

En este apartado se desarrollará todo lo derivado de la IA, cómo acceder al proceso de entrenamiento, cómo llevarlo a cabo, cómo establecer las neuronas a utilizar, cómo migrar el proyecto a la placa, etc.

5.2.1 Movimientos a detectar

Antes de establecer cualquier movimiento tendremos que tener en cuenta la orientación que va a mantener el dispositivo fijado a cualquier superficie, ya que así es como se va a entrenar a la red neuronal, si no dará fallos durante el reconocimiento.

Figura 15. *Orientación de la placa Arduino Nano 33*



Nota. Orientación que deberá tener la placa cuando se fije a cualquier superficie. Fuente: Elaboración propia

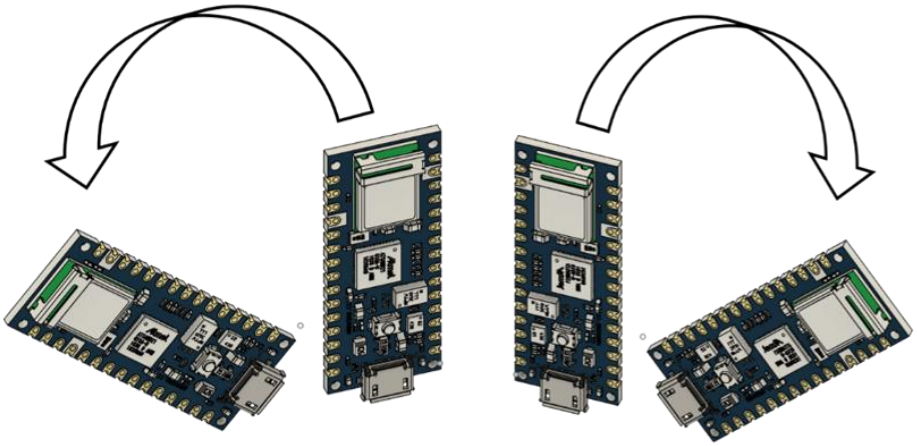
En referencia a los movimientos que deben ser reconocidos debido al impacto que le pueden generar a la carga se han definido los siguientes tipos:

5.2.1.1 Movimientos de vuelco

El vuelco, a la hora de transportar cargas especialmente sensibles, como por ejemplo los viales de vacunación, resulta de vital importancia, por ende, se van a procesar los siguientes datos de vuelco:

- Vuelco Lateral

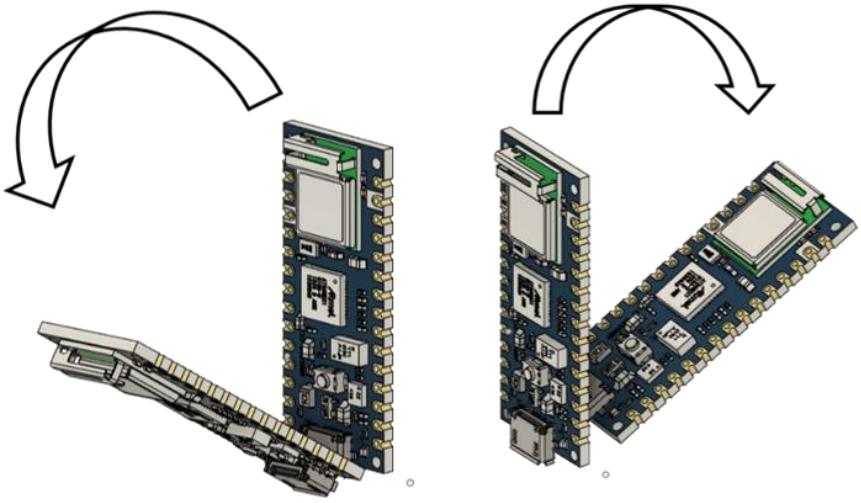
Figura 16. *Movimiento de vuelvo hacia la izquierda o hacia la derecha*



Nota. Fuente: Elaboración propia.

- Vuelco Frontal

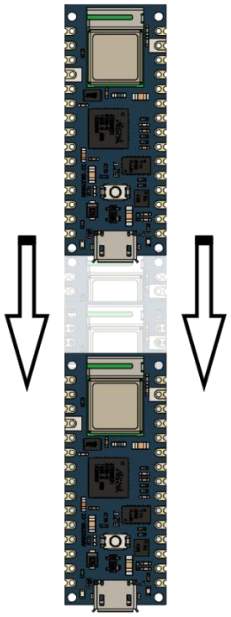
Figura 17. *Movimiento de vuelco hacia atrás o hacia delante*



Nota. Fuente: Elaboración propia.

5.2.1.2 Movimientos de caída libre

Figura 18. *Movimiento de caída libre*



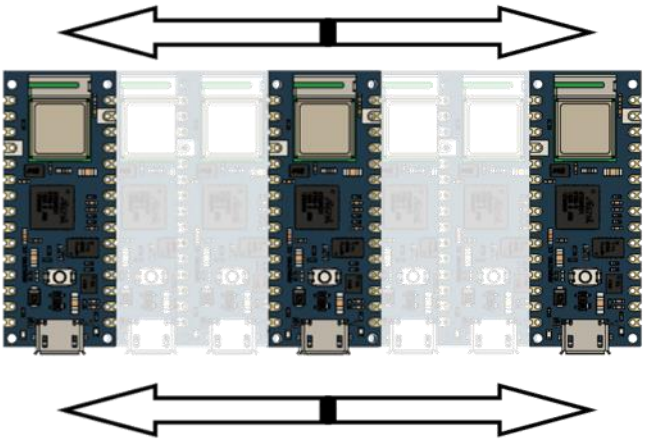
Nota. Fuente: Elaboración propia.

5.2.1.3 Movimientos de golpeo

En cuanto a los golpes que puede sufrir se van a definir los siguientes golpeos:

- Golpeo Lateral

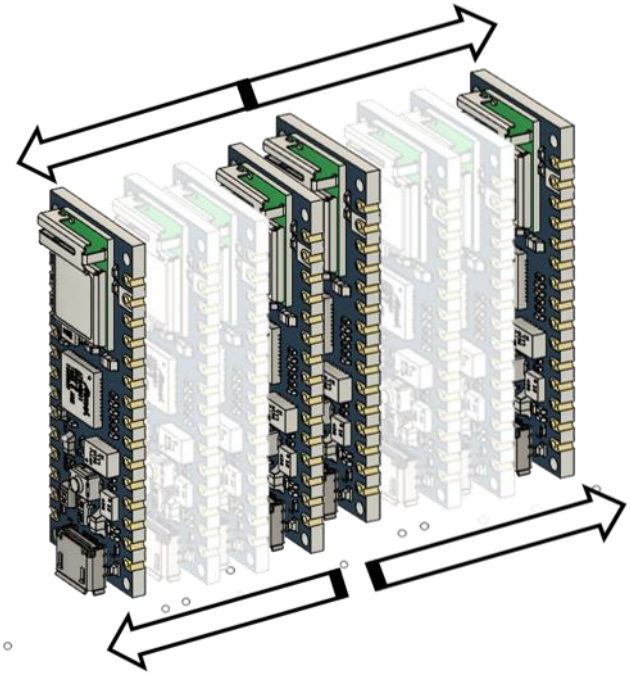
Figura 19. *Golpe de la carga hacia la izquierda o hacia la derecha*



Nota. Fuente: Elaboración propia.

- Golpeo Frontal

Figura 20. *Golpe de la carga hacia atrás o hacia delante*



Nota. Fuente: Elaboración propia.

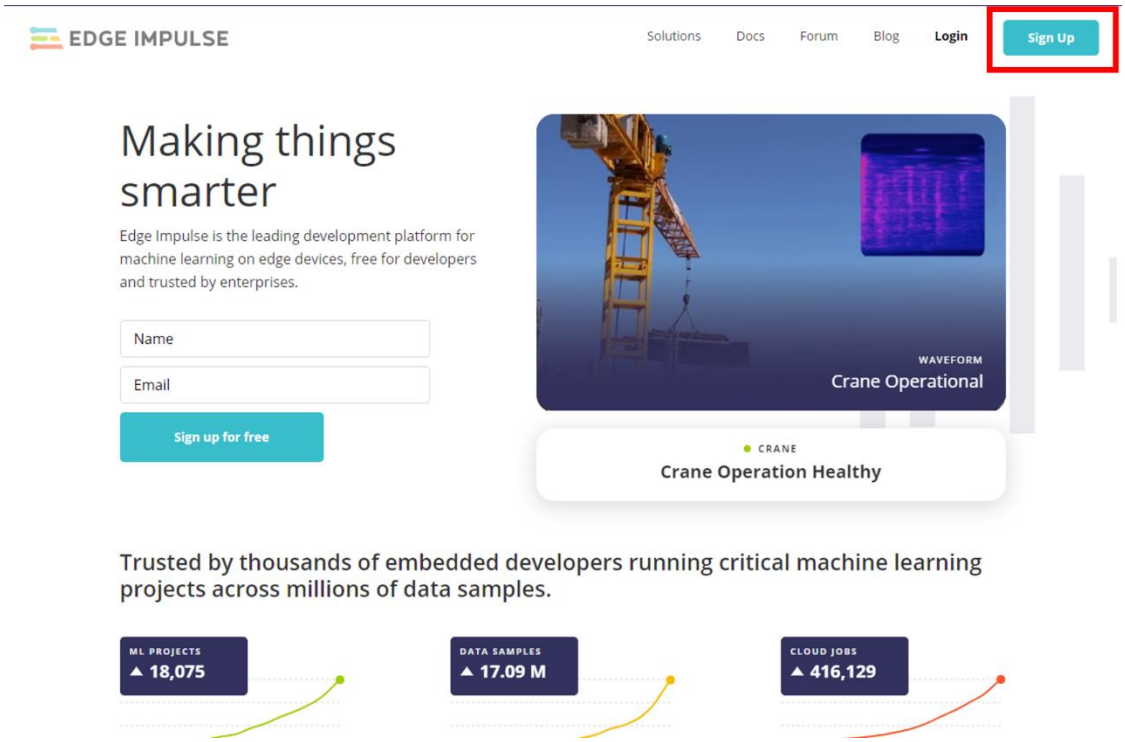
5.2.2 Acceso a la plataforma de entrenamiento

Una vez conocemos todos los movimientos necesarios para que el dispositivo funcione perfectamente, deberemos comenzar el proceso de entrenamiento de la IA.

Para ello, será necesario acceder a la página web de Edge Impulse, plataforma elegida para la creación de la red neuronal, gracias a la facilidad de uso, como bien se explicó en el punto 4.1.2 Edge Impulse.

Una vez accedemos, será necesario crear una cuenta, en el botón SIGN UP.

Figura 21. *Página web de Edge Impulse*



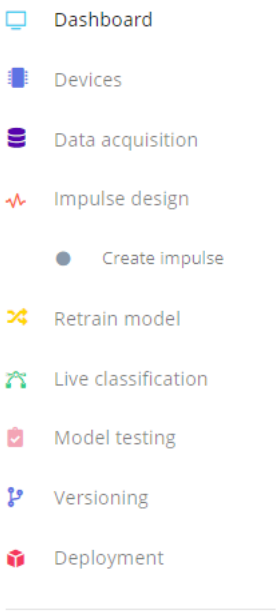
Nota. Fuente: Edge Impulse, 2021.

Una vez nos hayamos registrado en la plataforma, tendremos que crear un nuevo proyecto, y asignarle un nombre.

El proceso de creación de una red neuronal puede resultar complejo por eso se requiere prestar atención a todos los pasos, para comprender como funciona y de esta manera poder verificar las ilimitadas ventajas que presenta el uso de la Inteligencia Artificial en nuestro proyecto.

Edge, nos presenta una interfaz bastante amigable, que consiste en un menú izquierdo, el cual nos servirá de guía para facilitar la explicación.

Figura 22. *Menú Edge Impulse*



Nota. Fuente: Elaboración Propia.

El primer apartado consiste en el "Dashboard", es decir un tablero donde se presentará a modo de resumen el cómo se va avanzando durante el proceso de entrenamiento. Resulta muy necesario su uso ya que muestra una línea temporal que nos irá informando de cuánto falta para terminar el proceso.

Sin embargo, antes de continuar con el entrenamiento tendremos que llevar a cabo una serie de instalaciones que le permitirán a Edge detectar nuestra placa, para que podamos simular los movimientos y puedan ser recogidos durante la preparación de la IA.

Los pasos de este proyecto se van a llevar a cabo para el sistema operativo Windows 10, para el uso de cualquier otro sistema operativo, los pasos son realmente similares, y el software se puede encontrar fácilmente en la página oficial de Edge Impulse.

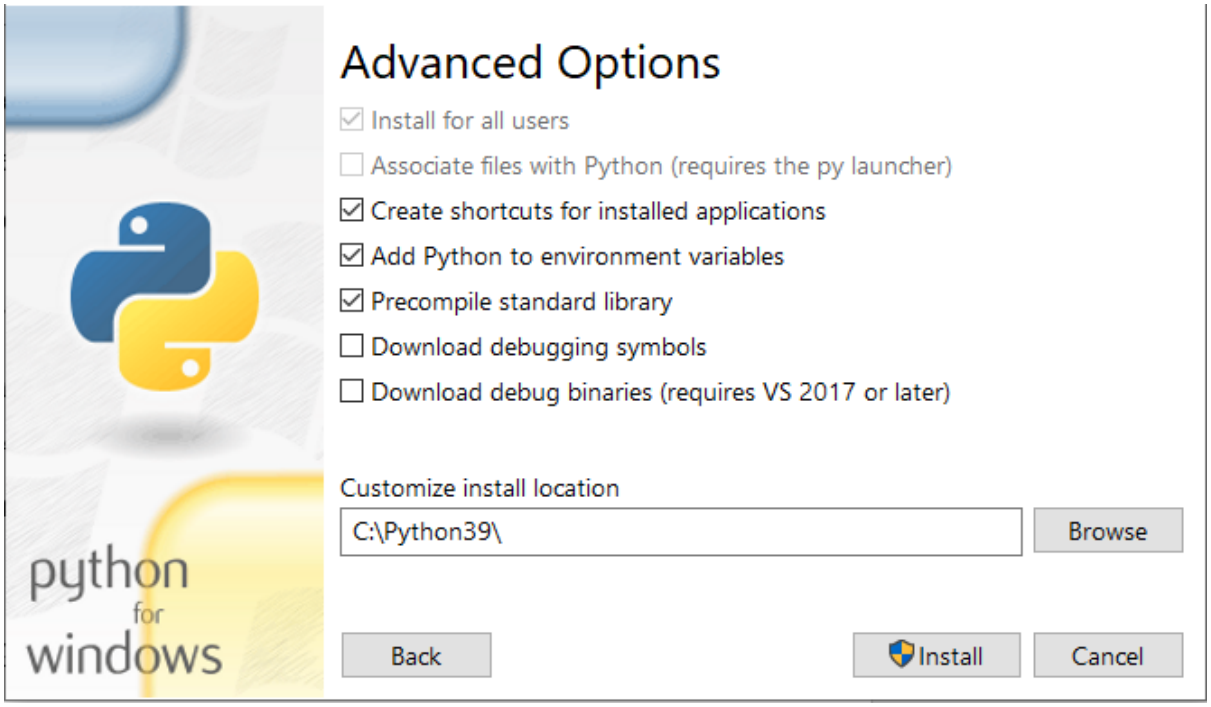
En primer lugar, será necesaria la instalación del software, Edge Impulse CLI.

CLI, consiste en una interfaz de línea de comandos que nos permitirá controlar dispositivos locales, como nuestra placa de Arduino, o incluso actuar como un proxy para sincronizar datos para dispositivos que no tienen conexión a internet, además de poder cargar y convertir archivos locales.

Para Instalar este software tendremos que descargar las siguientes aplicaciones:

- Python a partir de la versión 3: en la instalación hay que marcar la opción de "Add Python to environment variables" para que Windows incluya la ruta del programa en sus variables del entorno, permitiendo que los demás programas hagan uso de él.

Figura 23. *Instalación de Python*

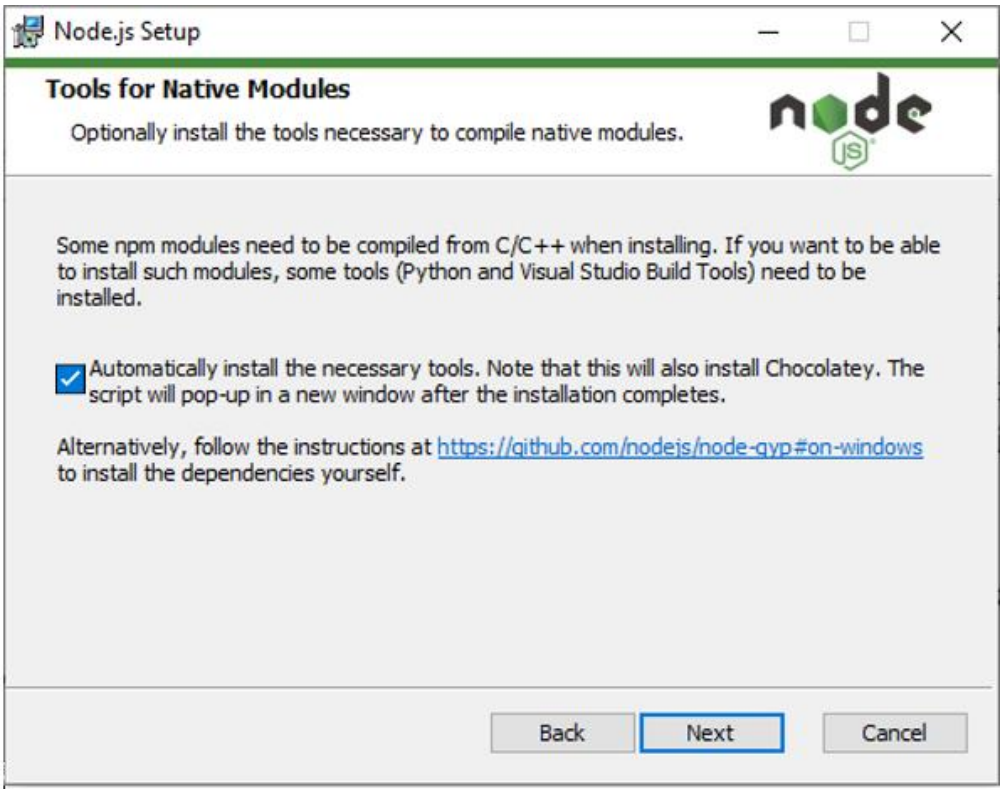


Nota. No olvidar añadir Python a las variables del entorno de Windows 10. Fuente: Elaboración propia.

- Node.js v14 o superior: La instalación de Node.js es imprescindible ya que va a permitir que podamos instalar la aplicación de Edge Impulse CLI, la cual está programada en JavaScript, razón por la que es necesaria la instalación de Node, ya que es el entorno de ejecución el que nos va a permitir poder tanto instalarla como ejecutarla.

  Debido a que el módulo de Edge Impulse de tipo npm, requiere el uso de Visual Studio, y de Python para poder programar la red neuronal, es necesario marcar la opción de "Automatically install the necessary tools". La cual instalará el paquete de Visual Studio Build Tools 2017 con todo lo necesario para poder instalar correctamente el módulo CLI.

Figura 24. *Instalación de Node Js*



Nota. No olvidar marcar la opción de los módulos npm. Fuente: Elaboración propia.

Una vez hayamos instalado las dos aplicaciones, será necesario que antes de instalar el CLI, instalemos el repositorio de bibliotecas de Arduino, llamado Arduino-CLI, que va a permitir que nuestra placa sea leída por el software de Edge, este repositorio es una solución todo en uno que contiene todas las librerías necesarias para el reconocimiento de cualquier placa de Arduino.

Para instalarlo, simplemente tendremos que descargar desde su página web CLI de Arduino, y seleccionar nuestro sistema operativo, que en este caso trabajamos con un Windows 10 de 64 Bits.

Una vez descargado tendremos que copiar la ruta donde se encuentre el archivo:

Ejemplo: C:\Users\Example \Desktop\arduino-cli_0.13.0_Windows_64bit

Esta ruta la deberemos copiar y acceder a las variables del entorno de Windows y copiarla en la variable "Path", de la siguiente manera:

Configuración → Acerca de → Configuración Avanzada del sistema →Opciones Avanzadas del sistema→Variables del entorno→Path→Editar→Nuevo.

En "Nuevo", deberemos pegar nuestra ruta y darle a "Aceptar. De esta forma Windows sabrá dónde acudir cuando al programa demande una llamada a Arduino-CLI.

Para finalizar nos quedará instalar el CLI, de Edge Impulse.

Tendremos que abrir el símbolo del sistema de Windows 10, escribiendo CMD, en el buscador, y a continuación deberemos ejecutar la siguiente línea:

```
npm install -g edge-impulse-cli
```

Una vez hayamos terminado de instalar el CLI, tendremos que actualizar el firmware la placa de Arduino Nano 33, que insertará el programa de Edge Impulse y configurará la placa para que el CLI la pueda detectar correctamente.

Esperaremos hasta que la placa parpadee y la resetearemos dándole una vez al botón "Reset" de la placa, para iniciar el nuevo firmware.

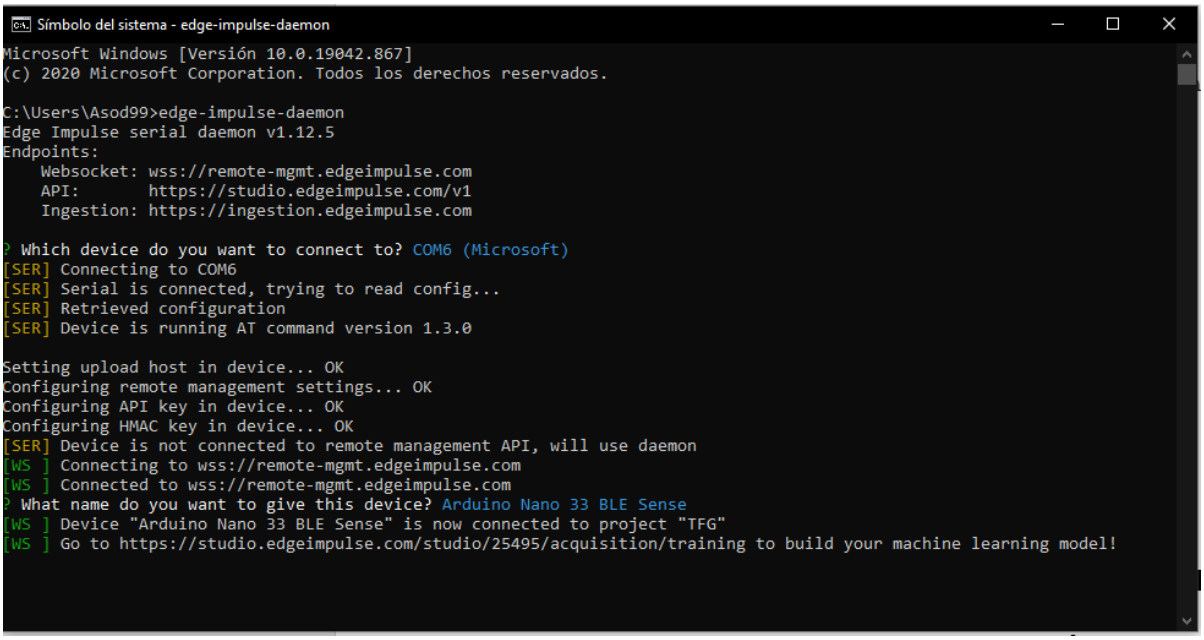
Una vez hayamos terminado, solamente nos quedará acceder al CMD, y ejecutar la siguiente línea de comandos:

```
edge-impulse-daemon
```

Si todo ha funcionado correctamente, nuestra placa ya estará preparada para que podamos entrenar a la IA, desde la página web de Edge Impulse.

Aviso: durante el entrenamiento no cerrar el CMD, porque es la conexión activa que mantiene enlazados a nuestro navegador con la placa.

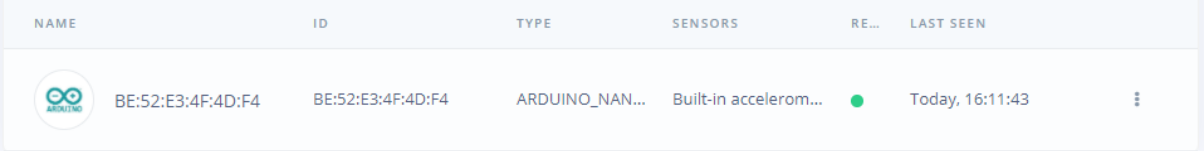
Figura 25. *CLI de Edge Impulse con conexión establecida*



Nota. Conexión Activa al puerto COM6 de la placa Arduino Nano 33 con la plataforma Edge Impulse. Fuente: Elaboración propia.

Una vez establecida la conexión con la placa, en la página de nuestro proyecto de IA, en el apartado “Devices”, nos debería aparecer nuestra placa con el nombre que le hemos asignado y la id asociada a la placa.

Figura 26. *Menú “Devices” de Edge Impulse*



Nota. Placa conectada a la plataforma Edge y preparada para el entrenamiento. Fuente: Elaboración propia.

Una vez tenemos todo conectado y la plataforma reconoce la placa es momento de adquirir datos y comenzar la etapa de entrenamiento.

5.2.3 Proceso de entrenamiento

Antes de comenzar el entrenamiento resulta indispensable resumir como funciona una inteligencia artificial para poder saber realmente que es lo que se está haciendo, ya que si no va a ser imposible su entendimiento.

Una inteligencia artificial consiste en una red neuronal, formada por "neuronas" artificiales, que intentan simular el comportamiento de las neuronas biológicas. Su funcionamiento matemático se basa en la suma de las entradas que tiene la neurona multiplicado por el peso de cada entrada, este peso es asignado según la importancia de la información contenida en la entrada.

En la práctica, lo que se hace es entrenar a la red neuronal, dándole una cantidad considerable de datos y clasificándolos para que más adelante pueda reconocer patrones parecidos y pueda asignarlos a la clasificación que le has enseñado previamente.

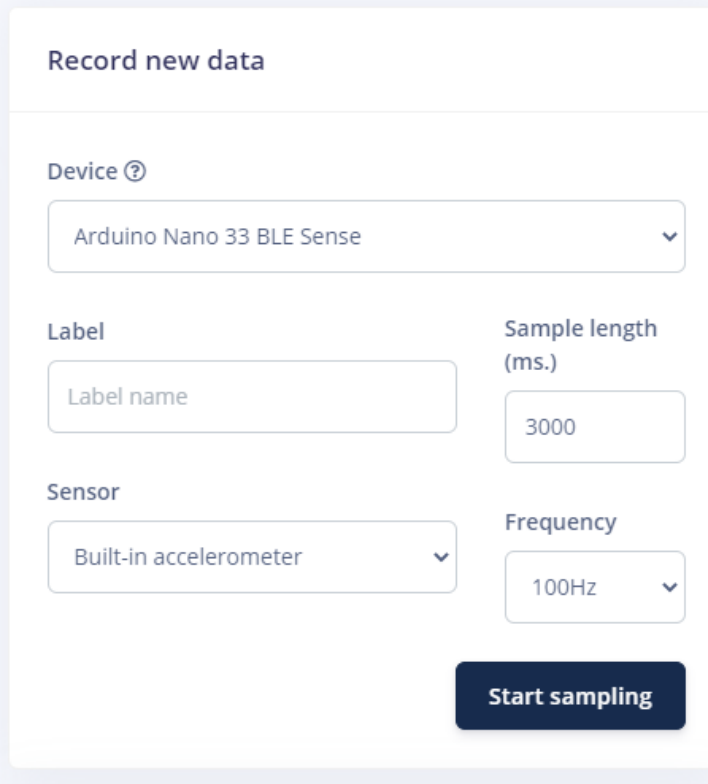
Una vez conocido este funcionamiento, se comenzará con el entrenamiento de la red neuronal, para ello, debemos dirigirnos al apartado de "Data Acquisition".

Se visualizarán 2 menús, uno llamado "Training data", dónde realizaremos los entrenamientos y desde dónde recopilaremos los datos. Y el otro llamado "Test data", dónde podremos registrar movimientos, para que cuando tengamos la IA, los clasifique, verificando así su funcionamiento.

Para comenzar, tendremos que fijarnos en el apartado "Record New Data", aquí podemos visualizar varios campos:

- Device: Placa que se va a usar.
- Label: Etiqueta del conjunto de datos a recoger.
- Sensor: Sensor que utilizaremos, en nuestro caso, el acelerómetro.
- Sample Length: Es el tiempo durante el que vamos a recoger los datos.
- Frequency: Es la frecuencia con la se va a leer cada dato nuevo.

Figura 27. *Record New Data de Edge Impulse.*



Nota. Menú para comenzar el entrenamiento de la IA de Edge Impulse. Fuente: Elaboración propia.

En nuestro caso, vamos a utilizar el Device: Arduino Nano 33 BLE Sense, el Label, lo iremos modificando conforme cambiemos el movimiento que estamos entrenando, y respecto al sensor, vamos utilizar el acelerómetro.

El valor, "Sample Lenght" se fijará en 3000 ms, de esta forma los datos se obtendrán de 3 en 3 segundos. Tiempo más que suficiente para simular cualquier movimiento. Y la frecuencia la dejaremos en 100Hz, debido a la elevada velocidad de los desplazamientos.

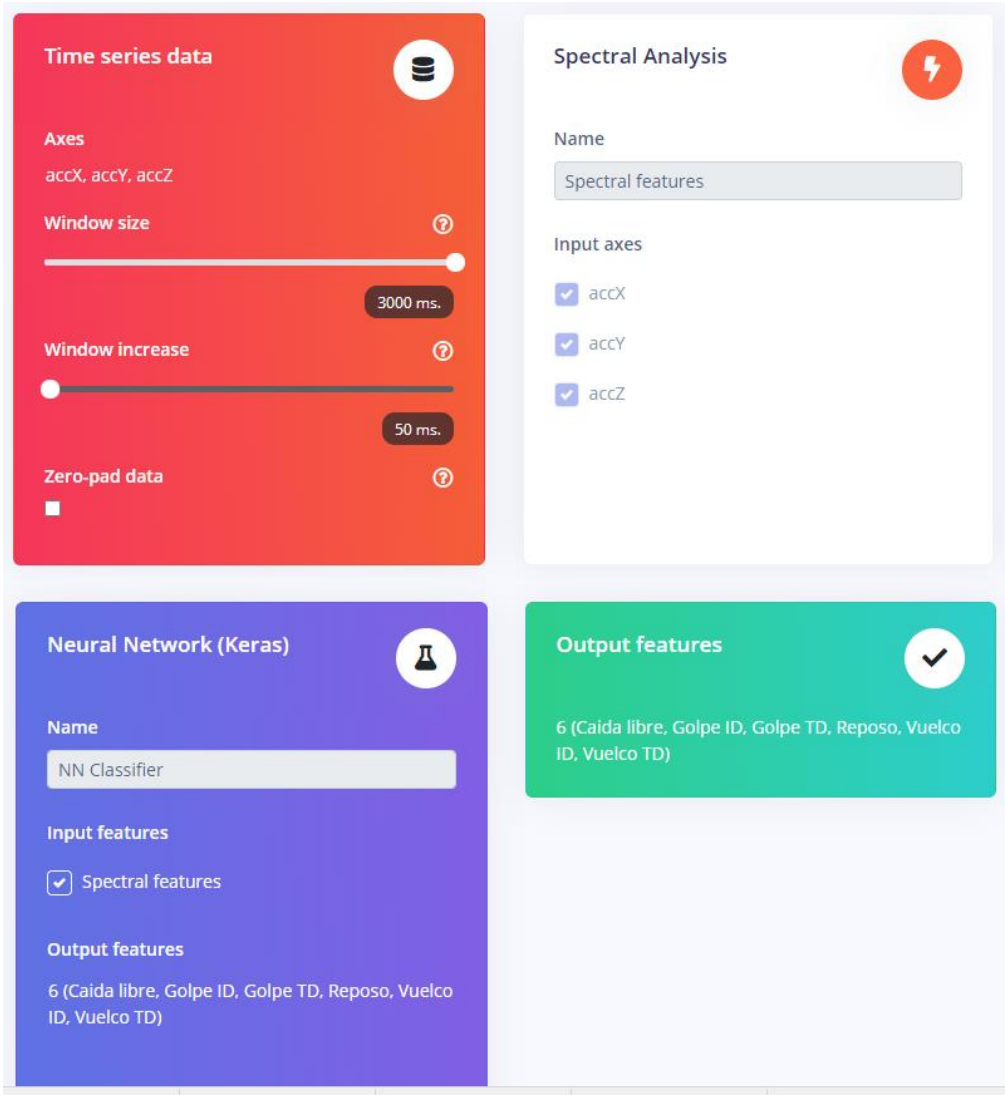
Una vez le demos a "Start Sampling" tendremos que recrear cada movimiento uno por uno hasta completar todos los movimientos, en total 5, pero también hay que añadir la posición de reposo, para que el movimiento de estabilidad quede también reconocido, y así la IA, pueda ser entrenada con más datos, aumentando considerablemente su precisión durante el proceso.

En este caso, el cómputo total de datos entrenados ha sido de 5 min 32 s, tiempo suficiente para registrar los movimientos necesarios.

Una vez tengamos los datos recogidos, tendremos que pasar al apartado de “Create Impulse”.

En este apartado lo que haremos, será crear “el impulso”, el cual consiste en tomar todos los datos sin procesar y mediante bloques de procesamiento predefinidos se extraerán las características más importantes para poder ser clasificados más adelante.

Figura 28. *Apartado Create Impulse de Edge*



Nota. Configuración del impulso para la clasificación de los datos recogidos. Fuente. Elaboración propia.

A partir aquí todos los pasos resultan cruciales, ya que, si no se selecciona la misma configuración la red neuronal, no conseguirá tener la precisión suficiente y no será capaz de reconocer ningún movimiento correctamente.

En el apartado de “Time series data” tendremos que prefijar el tamaño que va a tener nuestra ventana de entrenamiento. En este caso, como hemos generado un dato por cada movimiento, tendremos 1 movimiento por cada 3 segundos. Así que el tamaño de la ventana será de 3000ms.

En cuanto al punto de “Windows Increase”, podremos asignar cualquier valor, ya que no lo va a tener en cuenta debido a que la ventana posterior que creamos de 3 segundos ya ocupa todo el volumen del dato por lo tanto no va a poder dividir en ese dato en más ventanas ya que la posterior ocupa la totalidad.

Este apartado sería necesario en el caso de que hubiésemos realizado, por ejemplo 3 movimientos de vuelco derecho durante los 3 segundos que duraba el entrenamiento de cada dato. Entonces, sería necesario dividir el tamaño de la ventana entre 3, es decir 1000 ms, y el “Window increase” fijarlo en 1000 ms, para que hubiese creado, 3 sub ventanas dentro de ese dato.

Siguiendo, en el mismo menú nos encontramos con el clasificador de datos. Será necesario que elijamos el bloque de “Spectral Analysis”, que consiste en un clasificador de datos preprogramado por Edge Impulse y es capaz de analizar movimientos repetitivos, extraídos de los acelerómetros. Las características que ofrece son: la potencia y la frecuencia de una señal a lo largo del tiempo.

Una vez hayamos seleccionado el bloque tendremos que clicar los 3 ejes que nos aparecen: accX, accY, accZ.

Lo siguiente que visualizamos corresponde con el bloque de aprendizaje de la inteligencia artificial, estos bloques son los que van a definir cómo va a aprender y cómo va a clasificar los datos.

Se va a hacer uso del bloque, “Neural Network (Keras)”, el cual aprende patrones a partir de datos y puede aplicarlos a otros nuevos. Lo que resulta excelente para la categorización de movimientos de este estilo.

No olvidar, una vez seleccionado el bloque de aprendizaje, clicar en las características “Spectral Features”, para que posteriormente la IA, se entrene respecto a ese bloque de particularidades.

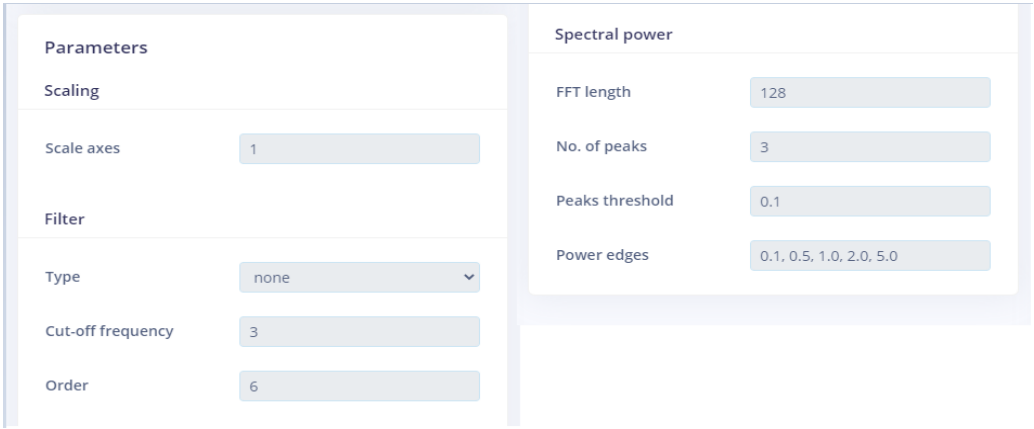
A continuación, avanzaremos en el menú, y accederemos al apartado de “Spectral Features”, este punto es el encargado de configurar el bloque de características para que prepare los datos que le van a ser entregados a la IA.

En el primer punto tenemos los parámetros, y nos aparecerá un gráfico de datos brutos dónde podremos visualizar cómo son los datos que hemos introducido. Y a la parte derecha, en el recuadro de “DSP result”, se nos muestra como es el resultado de los datos a la salida.

Respecto a la configuración, será necesario realizar pocos cambios, simplemente en la casilla de “Filter Type” podremos “None”, ya que no vamos aplicarle ningún filtro, ni paso alto ni paso bajo para limpiar los movimientos, debido a que si se lo aplicamos los movimientos perderán parte de su información primordial y serán muy parecidos, incrementado la dificultad de clasificación.

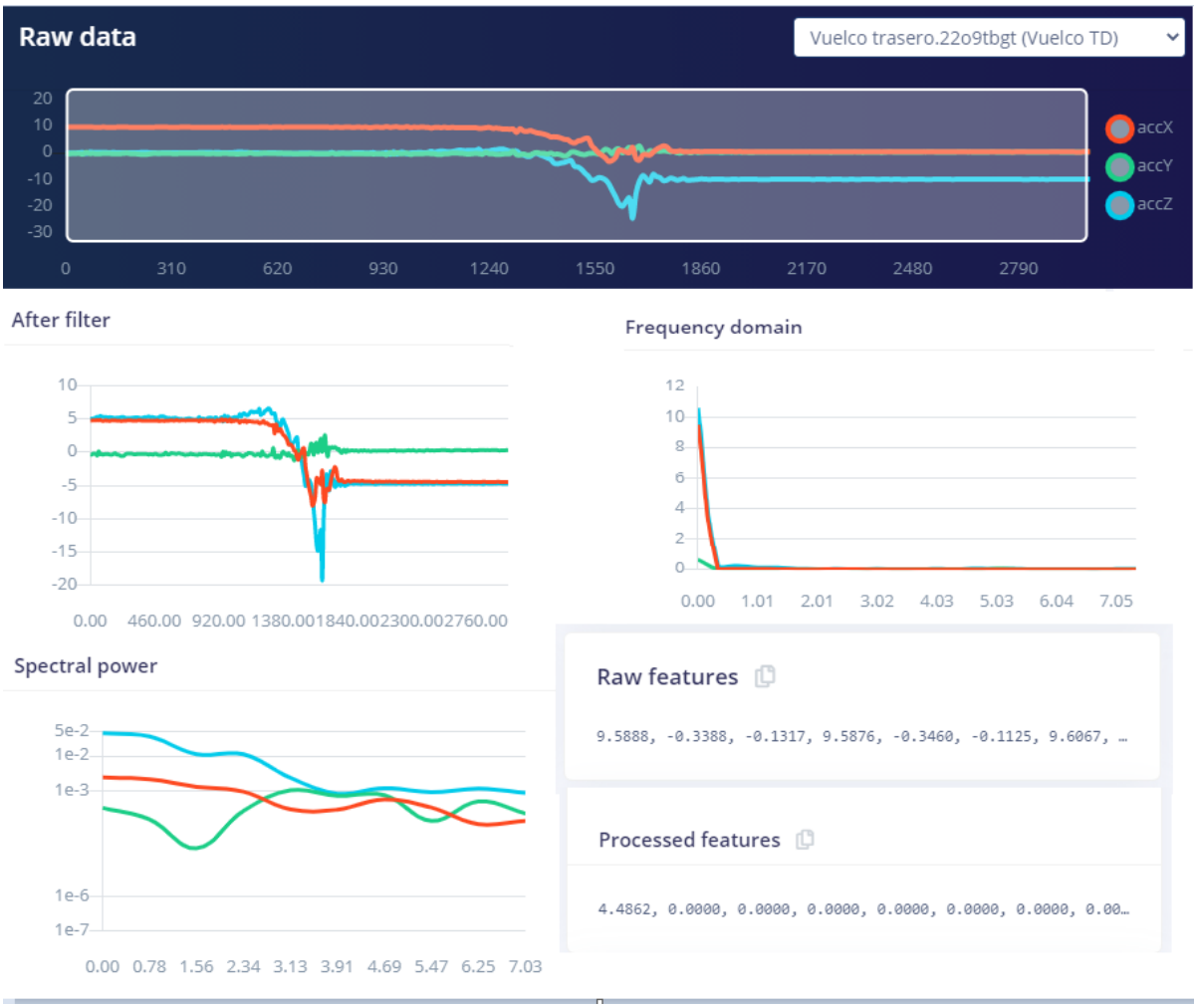
Todo lo demás no será necesario tocarlo ya que el aspecto de “Spectral Power” viene correctamente configurado para obtener la potencia correcta de los datos a la salida.

Figura 29. *Configuración de los parámetros del bloque de características*



Nota. Parámetros necesarios para la configuración de los datos de Edge. Fuente: Elaboración propia.

Figura 30. *Ejemplo de vuelco trasero con el procesado de las características*



Nota. Visualización del procesado de los datos de un vuelco trasero. Fuente: Elaboración propia.

La figura 30 resulta muy ilustrativa ya que resume en conjunto la extracción de las características de un movimiento concreto, como es, el dato de un vuelco trasero. Como, se puede apreciar debido a la falta del filtro la gráfica de “After filter” es igual que la de “Raw Data”, simplemente con zoom debido al estrechamiento de los ejes.

Unos de los aspectos más característicos, es la modificación de los valores numéricos en las características brutas, “Raw features” dónde se aprecian unos datos, los cuales, una vez procesados, “Processed features”, resultan extremadamente diferentes, gracias al tratamiento realizado.

Una vez tenemos los parámetros configurados, es momento de generar las características, en el apartado “Generate features”, cuando cliquemos en el botón verde, Edge impulse comenzará la generación de características de los datos introducidos previamente.

Cuando termine el proceso nos mostrará una gráfica muy útil, que permitirá observar cómo ha ido la clasificación de cada uno de los datos introducidos.

Figura 31. *Clasificación de las características del entrenamiento de la IA*

Nota. Gráfica de las características procesadas por el bloque espectral de Edge. Fuente: Elaboración propia.

La gráfica de la figura 31 muestra los movimientos clasificados por colores, y como se puede apreciar se observa una tendencia asociativa de los datos correspondientes a los mismos movimientos. Lo cual da a entender que el entrenamiento y el procesado de los datos se ha llevado a cabo de manera satisfactoria.

Para continuar, avanzaremos por el menú hasta la opción de "NN Classifier", en este apartado ya estaremos preparados para el entrenamiento de la IA.

Edge Impulse, ofrece 3 modos para el entrenamiento:

- Visual (simple) mode
- Keras (expert) mode
- Edit as iPython notebook

En este caso se elegirá la opción "Visual mode" gracias a la sencillez que ofrece.

Para un correcto entrenamiento se fijarán los siguientes parámetros:

- Number of training cycles: 200

200 entrenamientos serán más que suficientes para el entrenamiento de la IA, ya que los datos no son excesivos.

- Learning rate: 0.01

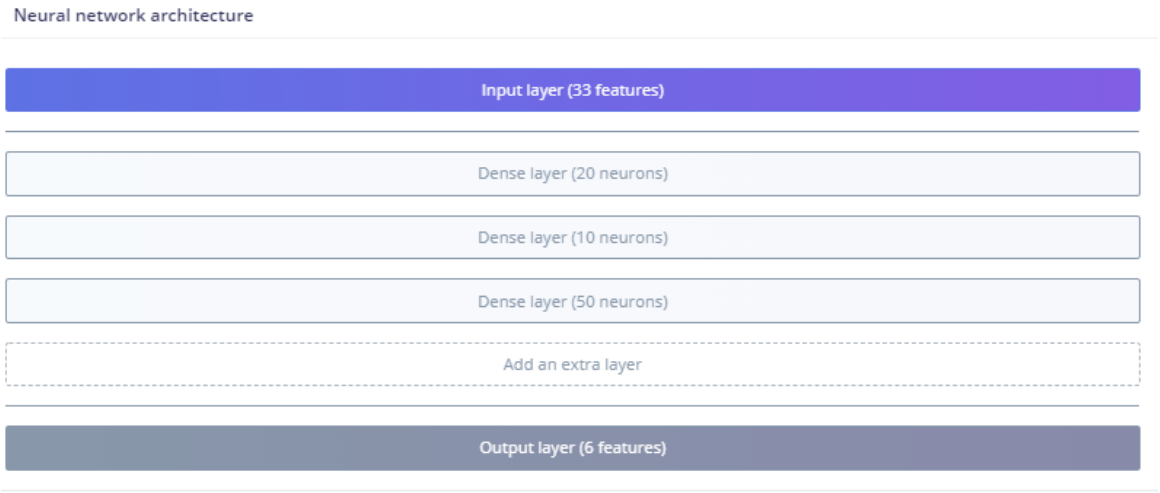
Este valor representa un índice de aprendizaje, es decir, cuánto va a aprender por cada entrenamiento. Si este valor es muy bajo, tardará mucho en aprender. Si, por el contrario, es muy alto aprenderá demasiado rápido. Las dos opciones derivan en una mala precisión de la IA.

- Minimum confidence rating: 0.7

Corresponde con el umbral mínimo como para que un valor devuelto por la IA sea considerado aceptable, todo lo que esté por debajo se considerará como incierto.

En cuanto a la arquitectura de la red neuronal se van a añadir 3 capas densas, las cuales son las más indicadas para este tipo de datos. Repartidas con diferentes números de neuronas. La primera tendrá 20 neuronas, la segunda 10 y la última 50 neuronas. Con este reparto será más que suficiente para los datos a clasificar ya que no son necesarias tantas neuronas para una cantidad tan reducida.

Figura 32. *Arquitectura de datos de la red neuronal.*



Nota. Distribución neuronal de las capas de la red y cuantificación de cada una de ellas. Fuente: Elaboración propia.

Una vez tengamos la red neuronal creada y entrenada, clicando en el botón de "Start Training", se generarán los datos de precisión de la IA indicando la cantidad de datos que ha fallado en una matriz de confusión que indicará la precisión obtenida por cada uno de los movimientos, y un gráfico de características con los datos que ha acertado o fallado.

Lógicamente, cuanto mayor sea el porcentaje de precisión, mejor será la IA que se ha creado.

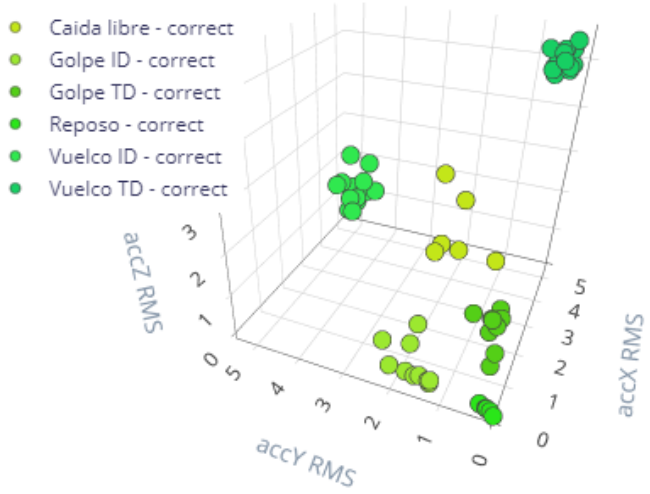
Figura 33. *Matriz de confusión de la IA creada*

	CAIDA LIBRE	GOLPE ID	GOLPE TD	REPOSO	VUELCO ID	VUELCO TD
CAIDA LIBRE	100%	0%	0%	0%	0%	
GOLPE ID	0%	100%	0%	0%	0%	
GOLPE TD	0%	0%	100%	0%	0%	
REPOSO	0%	0%	0%	100%	0%	
VUELCO ID	0%	0%	0%	0%	100%	
VUELCO TD	0%	0%	0%	0%	0%	100%
F1 SCORE		1.00	1.00	1.00	1.00	1.00

Nota. Porcentaje de acierto durante el proceso de entrenamiento para cada uno de los movimientos. Fuente: Elaboración propia.

En este caso la IA tiene una precisión del 100% lo que indica que los movimientos serán reconocidos perfectamente.

Figura 34. *Gráfico de características de la IA entrenada*



Nota. Los datos de entrenamiento utilizados por Edge Impulse han obtenido un 100% de acierto durante el entrenamiento de la IA. Fuente: Elaboración propia.

Además, Edge Impulse, informa del tiempo de inferencia que tarda la IA en reconocer los movimientos, de la cantidad de RAM utilizada en el proceso y del almacenamiento de ROM empleado.

Figura 35. *Datos de rendimiento de la IA en el microprocesador*

Nota. Tiempo de inferencia empleado en la detección de un movimiento, y cantidad de RAM y ROM empleadas por la placa Arduino Nano 33 BLE Sense. Fuente: Elaboración propia.

Los datos que nos ofrece Edge Impulse resultan de vital importancia para conocer previamente como va a ser el funcionamiento de la IA dentro del microprocesador. En el caso del dispositivo a diseñar, toda la información generada indica que el modelo tiene un buen rendimiento y que el funcionamiento debe ser excelente.

Una vez creada la IA, Edge, a continuación, ofrece una opción llamada, “Retrain Model”, que permite reentrenar a la IA, en el caso de que hayamos introducido nuevos datos de entrenamiento.

Además, también encontramos la opción de "Live Classification", donde podremos testear la IA, generando datos en tiempo real con la placa. Este proceso consiste en conectar la placa e indicar los mismos parámetros que hemos introducido para los datos del entrenamiento. El procedimiento consiste en recoger un dato de la placa, y clasificarlo al instante. De esta manera se puede comprobar rápidamente el funcionamiento de la IA.

5.2.4 Migración de la red neuronal

Para completar el proceso de creación de la IA, será necesario que creemos una versión del proyecto en el apartado de "Versioning", además, Edge Impulse, nos da la opción de hacer público el proyecto para que así pueda ser reutilizado por otras personas. Este proyecto se puede encontrar en el siguiente enlace: https://studio.edgeimpulse.com/public/25495/latest.

Una vez creada la versión, solo nos quedará desplegar la IA. Este proceso resulta muy sencillo gracias a que Edge compilará el proyecto completamente preparado y optimizado para la plataforma que deseemos.

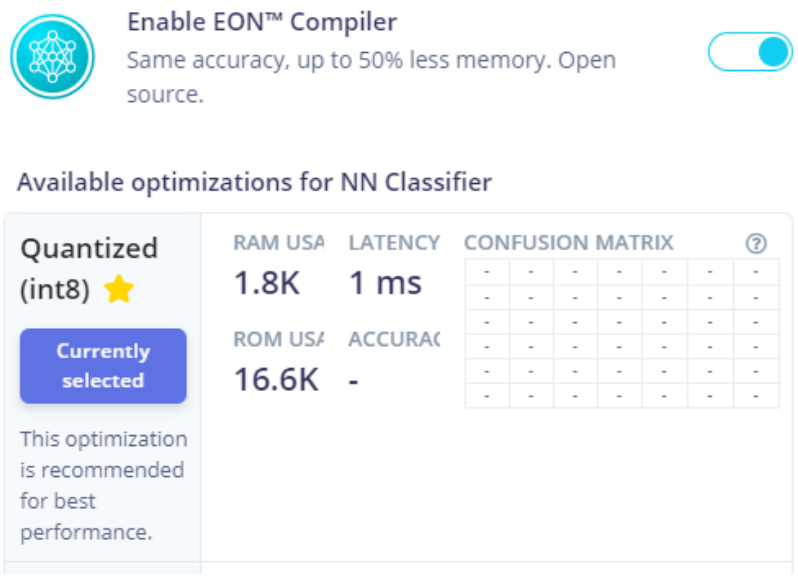
Aunque nos da la opción de compilarlo para la placa Arduino Nano 33 BLE Sense directamente, no vamos a seleccionar esta opción, ya que si no generará un software que no podrá ser editado más adelante.

Así que, se tendrá que elegir la opción de "Arduino Library", para que el código pueda ser editable y adaptable a las necesidades del proyecto.

Además, bajo de la selección de la plataforma, Edge, ofrece una serie de optimizaciones, gracias a la tecnología empleada por EON Compiler, una plataforma de código que compila las redes neuronales directamente en el código fuente de C++, logrando una reducción considerable de la latencia, la RAM y la ROM empleada por el dispositivo.

Se seleccionará la optimización recomendada por Edge debido a la mejora en las prestaciones que aporta.

Figura 36. *Selección optimizada por la tecnología EON Compiler.*



Nota. La optimización "Quantized" ofrece la mitad de latencia con un uso menor de la RAM y la ROM, gracias a la tecnología EON Compiler integrada por Edge Impulse. Fuente: Elaboración propia.

Con todo esto seleccionado, se generará un archivo .zip, que contendrá la Inteligencia Artificial creada completamente funcional y podrá ser implementado directamente como una biblioteca de Arduino. Permitiendo que le se puedan añadir las funcionalidades extra demandadas por el transporte de vacunas.

5.3 Establecimiento del protocolo de envío de datos al usuario

La comunicación con la persona encargada de supervisar y recopilar la información originada por el dispositivo es otro de los aspectos importantes para este proyecto.

En el caso de este diseño, se llegó a la conclusión de que el uso de la tecnología Bluetooth Low Energy era la más adecuada, ya que el rango de comunicación no tenía la necesidad de ser extremadamente largo. El aspecto de la sencillez y la reducción de dispositivos externos resultaba mucho más importante a la hora de abordar este apartado. Por ende, se optó por esta solución.

Este proyecto se va a configurar como un dispositivo periférico. Debido a que va a ser el encargado de enviar tanto la temperatura como los datos registrados a un dispositivo móvil.

El protocolo que se va a emplear se llama ATT y el perfil que lo implementa es conocido como GATT, encargado de enviar pequeños fragmentos de datos llamados "atributos" o "características". Principalmente, se basa en la creación de un servicio que lleva adjuntos estos atributos, los cuales deben ser configurados para enviar los datos que se quiera.

En este caso se va a configurar un servicio llamado "Data":

```
BLEService Data ("6240f192-c6e3-11eb-b8bc-0242ac130003");
```

Los datos que aparecen entre comillas, corresponden a la UUID, es decir, el identificador único universal, un número de 16 bytes, encargado de identificar información de forma exclusiva, en este caso se ha generado desde la página web de la Unión Internacional de Telecomunicaciones para tener un UUID exclusivo. Si se genera de acuerdo con uno de los mecanismos definidos en la Rec. UIT-T X.667 | ISO / IEC 9834-8, se garantiza que un UUID sea diferente de todos los demás UUID generados antes.

Aunque existe una lista de perfiles y servicios basados en GATT, la cual se puede encontrar en la página oficial de Bluetooth, se va a utilizar un UUID personalizado, modificando exclusivamente el último byte para cada atributo.

Los atributos que se van a implementar son los siguientes:

Nombre	UUID
➢ FreefallCharacteristic	00000000-0000-0000-0000-000000000001
➢ StrokeLCharacteristic	00000000-0000-0000-0000-000000000002
➢ StrokeFCharacteristic	00000000-0000-0000-0000-000000000003
➢ OverturnLCharacteristic	00000000-0000-0000-0000-000000000004
➢ OverturnFCharacteristic	00000000-0000-0000-0000-000000000005
➢ TemperatureCharacteristic	00000000-0000-0000-0000-00000000000a
➢ CriticalTemperatureCharacteristic	00000000-0000-0000-0000-00000000000b
➢ WriteTemperatureCharacteristic	00000000-0000-0000-0000-00000000000c

Los primeros 5 atributos corresponden con los movimientos que van a ser reconocidos por la IA. El atributo "A", hace referencia al envío de la temperatura y el "B", las veces que la temperatura fijada se ha superado en un intervalo de ±3ºC.

El atributo “C”, es especial, ya que no va a ser solo de lectura si no que se va a poder escribir cuál es la temperatura que se quiere fijar durante un trayecto.

Por lo tanto, todas las características tendrán como sufijos, “BLERead” y “BLENotify”, menos el último, “C”, que además tendrá “BLEWrite”.

El argumento BLENotify, será necesario ya que permitirá que el usuario reciba notificaciones cuando algún atributo cambie de valor.

Respecto a la inicialización y configuración se implementará de la siguiente manera:

```
BLE.begin();
```

```
BLE.setDeviceName( "Nombre del dispositivo" );
```

```
BLE.setLocalName( " Nombre del dispositivo " );
```

```
BLE.setAdvertisedService( Nombre del servicio );
```

Añadiremos las características al servicio:

```
Data.addCharacteristic(Característica a Asignar);
```

Y añadiremos el servicio al dispositivo BLE, además de poner a emitir al dispositivo:

```
BLE.addService(Nombre del servicio);
```

```
BLE.advertise();
```

En el caso de querer actualizar alguna variable se escribirá:

```
Nombredelacaracterística.writeValue(Valor a actualizar);
```

Y si queremos comprobar si la variable del establecimiento de la temperatura de la carga ha sido modificada implementaremos la siguiente línea:

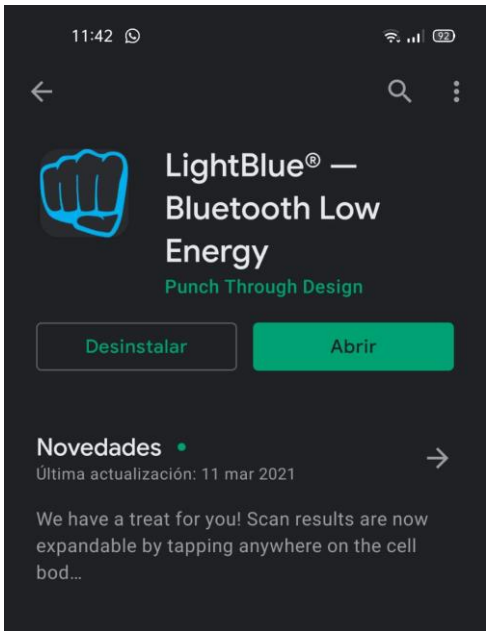
```
WriteTemperatureCharacteristic.written();
```

En el Anexo 0, se podrá ver en profundidad como se ha implementado la programación completa del Bluetooth.

5.3.1 Aplicación para la recepción de los datos

Para obtener los datos de la placa Arduino, será necesario que utilicemos una aplicación especializada en BLE. En este caso, se ha usado LightBlue, la cual, se encuentra disponible para descargar en las tiendas oficiales, tanto en Google Play, como en la App Store.

Figura 37. *Instalación de la aplicación LightBlue en Google Play*

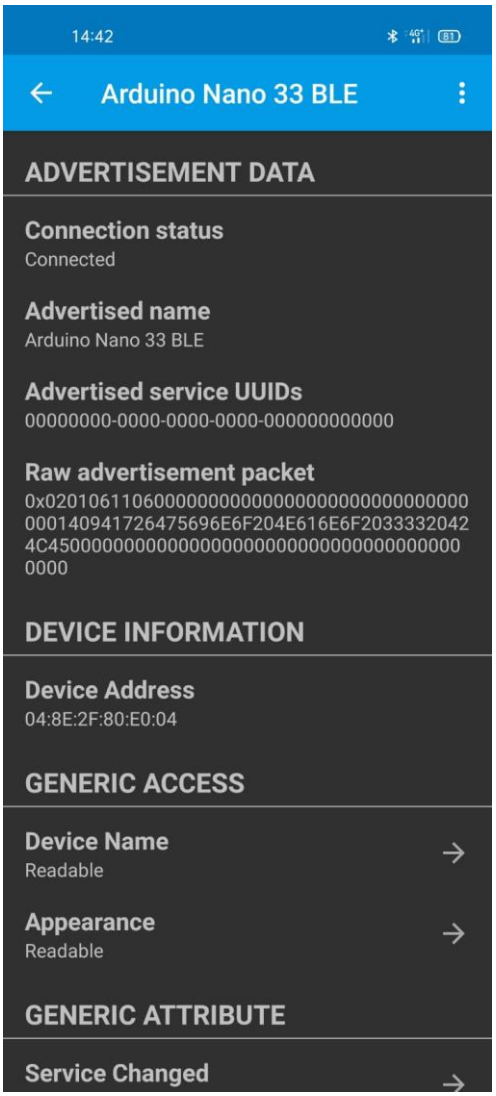


Nota. Fuente: Elaboración propia.

Una vez instalada la app, encontraremos una lista de los dispositivos cercanos que encuentre el smartphone. Tendremos que seleccionar el que aparezca con el nombre que le hemos asignado en el programa, en este caso, Arduino Nano 33 BLE.

Cuando hayamos accedido al dispositivo encontraremos una lista, dónde se mostrará la información del dispositivo, tal como, el status de la conexión, el UUID del servicio que está retransmitiendo, la MAC de la placa Arduino, y un apartado de acceso genérico que muestra el nombre del dispositivo y la apariencia.

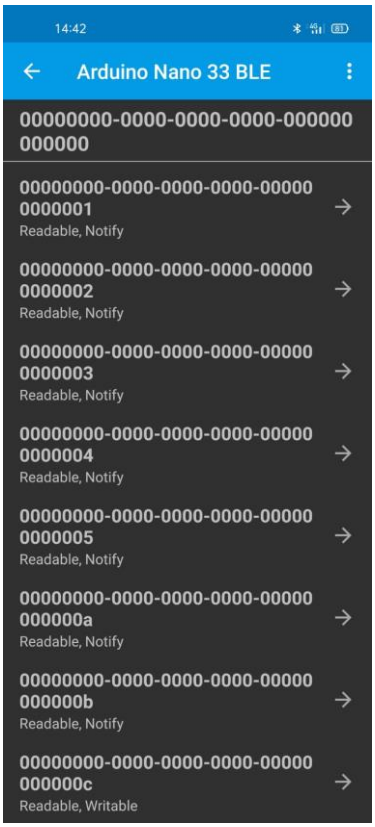
Figura 38. *LightBlue menú del dispositivo Arduino Nano 33 BLE*



Nota. Información genérica del dispositivo emisor BLE y de los servicios que ofrece. Fuente: Elaboración propia.

Si continuamos bajando podemos visualizar todos los atributos que hemos configurado para el servicio llamado “Data”.

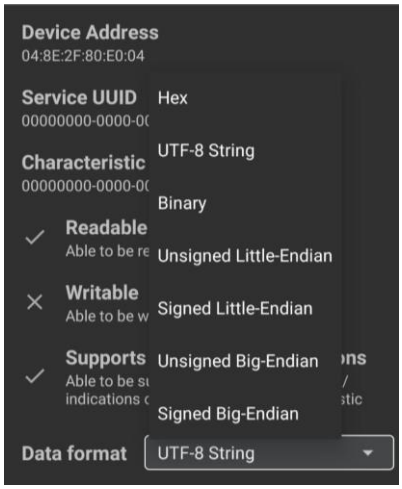
Figura 39. *LightBlue menú de atributos*



Nota. Atributos del servicio "Data" de la placa Arduino Nano 33 BLE. Fuente: Elaboración Propia.

En este menú podemos acceder a cada una de las características que deseemos para mostrar la información proporcionada por el dispositivo.

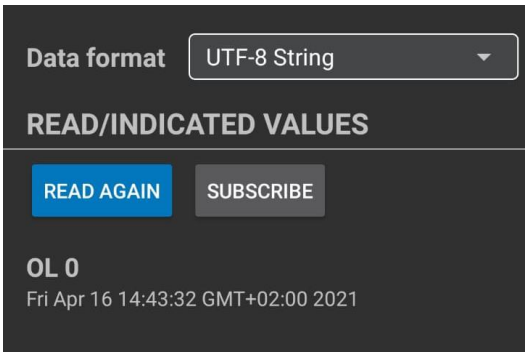
Figura 40. *LightBlue formato de datos del atributo caída libre*



Nota. Opciones de configuración de lectura para el atributo de caída libre. Fuente: Elaboración propia.

En la anterior figura podemos ver las diferentes opciones de visualización de los datos que nos ofrece LightBlue. En nuestro caso para que sean completamente legibles se deberá elegira la opción de UTF-8 String, ya que los datos en el programa los vamos a convertir en un string que va a estar compuesto por las dos primeras letras del movimiento, para que se pueda reconocer que número pertenece a cada movimiento, y seguidamente el número de veces que ha ocurrido.

Figura 41. *Lectura de datos del movimiento vuelvo lateral*



Nota. Fuente: Elaboración propia.

Para adquirir los datos deberemos pulsar el botón “Read again”, o suscribirnos, que hará llegar una notificación al teléfono cuando algún atributo se modifique.

Como podemos ver en la ilustración, el movimiento OL se ha realizado un total de 0 veces, y la lectura se tomó el 16 abril de 2021 a las 14:43:32.

La lista de movimientos se ha resumido con las siguientes inciales:

Tabla 3. *Relación de los movimientos con su abreviatura*

Movimientos	Variables en la programación	Movimiento en LightBlue
Golpe Lateral	StrokeL	SL
Vuelco Lateral	OverturnL	OL
Golpe Frontal	StrokeF	SF
Vuelco Frontal	OverturnF	OF
Caída Libre	Freefall	FF

Nota. Fuente: Elaboración propia.

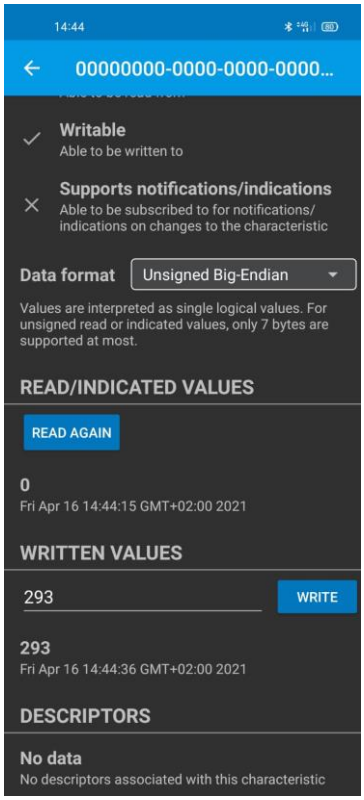
Además de las abreviaturas de la tabla 1 existen 2 más:

La variable CT, que indica cuántas veces se ha superado la temperatura fijada, funcionalidad la cual se explicará a continuación, y la variable T, que corresponde con la lectura en tiempo real de la temperatura del dispositivo.

Por último, se explicará la función de fijar la temperatura. Esta actividad consiste en dar la posibilidad de crear una herramienta capaz de avisar al usuario si la cadena del frío se ha roto. Consiste en la fijación de la temperatura mediante la variable "WriteTemperature", esta variable se comparará con la temperatura actual. Y en caso de que la temperatura oscile en más de 3 grados ya sean positivo o negativos, se incrementará la variable "CriticalTemperature"

De esta manera, cuando se quiera realizar un trayecto, el responsable de monitorizar la carga tendrá la posibilidad de establecer cuál es la temperatura a la que es necesaria que viaje la carga. En el caso de que no se establezca no sucederá nada, simplemente que la variable CT será siempre 0.

Figura 42. *Establecimiento de la temperatura mediante LightBlue*



Nota. Fijación para el control de la cadena del frío de la carga transportada. Fuente: Elaboración propia.

Para escribir la variable será necesaria seleccionar la opción “Unsigned Little-Endian”, debido a que la temperatura deberá introducirse en grados Kelvin, es decir, siempre desde una escala positiva y sin ningún carácter que no sea numérico. Por ende, la temperatura no tendrá signo, siendo “Unsigned”.

5.4 Carga y portabilidad del dispositivo

La carga y portabilidad son dos aspectos fundamentales ya que la falta de algún estos, generaría la inutilidad del dispositivo. Por ende, para su solución se han estudiado todas las opciones posibles, seleccionando la mejor posible.

La carga del dispositivo se realizará mediante el uso de una batería recargable del tipo 18650 ofreciendo 3,7V y 9900mAh. Esta batería no será suficiente para el Arduino ya que la mínima entrada de Vin es de 4,5V, obligando al uso de una placa que eleve ese voltaje.

Primero, se instalará un módulo de carga capaz de recargar la batería mediante un MicroUSB, para facilitar el uso del dispositivo, y hacer que pueda ser cargado con uno de los cables más genéricos actualmente.

Este módulo será el TP4056, y cargará la batería con un voltaje de 4.2V, como se puede comprobar en el datasheet del fabricante, ver. Anexo 02. El voltaje de carga es el adecuado para las baterías de este tipo, pero no llega a los 4.5V necesarios para el Arduino Nano 33 BLE, obligando a la instalación del módulo “Step-Up”. Este núcleo además posee dos LEDS, uno que indica si la batería se está cargando y otro que permite ver cuando está completamente cargada la batería, aumentando considerablemente su vida ya que elimina el exceso de carga que se le puede realizar procedente del desconocimiento de la cantidad que se ha cargado.

Para aumentar el voltaje, utilizaremos la placa XL6009, una placa capaz de aumentar un voltaje de entrada de 3V/32V a 5/35V con una corriente de hasta 3A, valores comprobables en el datasheet del fabricante, ver Anexo 0.

Respecto a la duración del dispositivo, hay que tener en cuenta cuál es la corriente utilizada por la placa. De forma práctica, se ha medido el consumo generado, siendo de 18,5mA.

Teniendo en cuenta este consumo y conociendo la capacidad de la batería podemos establecer una aproximación de la batería:

$$\frac{9900mAh}{18{,}5mA} = 535\ h$$

Lo que suma un total de 22 días de pleno funcionamiento, tiempo más que suficiente para que el dispositivo pueda ser cargado de nuevo.

Con todo esto, se completa un diseño para el dispositivo, capaces de llevar a cabo la función de monitorización y permitiendo una portabilidad completa gracias a las baterías y los módulos incorporados.

5.5 Diseño del software

Con todos los apartados explicados anteriormente, se creará un archivo que reúna el conjunto de programas anteriores, quedando así un archivo final, que sea el encargado de ejecutar a la red neuronal mediante la función:

```
run_inference_background()
```

Función la cual se ejecutará cuando el buffer de los datos recogidos por el acelerómetro quede completamente lleno, momento en el cuál, se invocará a la función del clasificador obtenida gracias a las librerías descargadas por Edge impulse:

```
run_classifier(&signal, &result, debug_nn);
```

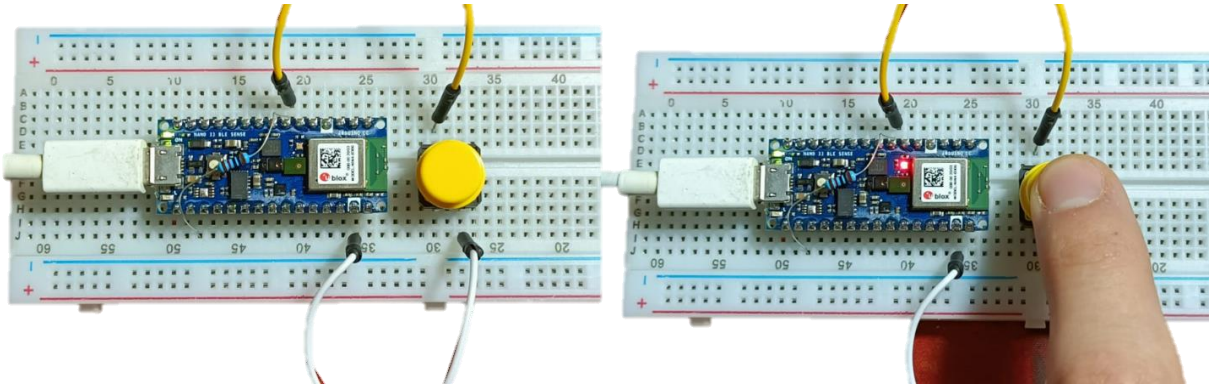
Este clasificador, organizará los movimientos por porcentajes de posibilidad, es decir, comparará los datos recibidos con los posibles movimientos y cuando el porcentaje de coincidencia sea superior al 70%, lo determinará definitivamente.

Una vez clasificado el movimiento, se guardará en una variable sumadora, encargada de llevar el recuento de cada uno de los movimientos, estas variables serán las que el usuario final podrá comprobar para cerciorarse de la cantidad de vuelcos y giros que han sucedido durante un trayecto.

En caso de que el trayecto haya finalizado y se requiera hacer un reseteo de todas las variables para comenzar una nueva monitorización, se ha asignado un pulsador al pin 4, en

modo Pull-UP, con una resistencia de 10 KΩ, conectada al pin de +3,3V que permitirá el reinicio completo del sistema cuando la lectura del pin sea “Low”.

Figura 43. *Esquema de configuración Pull-Up*



Nota. Conexión de la resistencia de 10 KΩ al pin de voltaje +3.3V, junto al pulsador conectado entre el pin 4 y Vref para establecer el reinicio de la configuración del programa. Fuente: Elaboración propia.

La variable llamada “RESET”, se comparará cada ciclo del programa con la lectura del pulsador permitiendo saber cuándo se ha presionado, y se encenderá el led rojo de la placa para dar aviso al usuario de que el reinicio se ha llevado a cabo correctamente:

Figura 44. *Configuración del programa para reiniciar variables*

```
Reset = digitalRead(4);
 if (Reset == LOW){
   digitalWrite(22, LOW);
   Freefall=0;
   StrokeL=0;
   StrokeF=0;
   OverturnL=0;
   OverturnF=0;
   Temperature=0;
   Tmax=0;
   Tmin=0;
   Tassign=0;
   CTemperature=0;
   Timer=0;
   StartT=true;
   previous=0;
   }
   if (Reset == HIGH){
   digitalWrite(22, HIGH);
   }
```

Nota. El pin 22 hace referencia al led rojo de la placa Arduino Nano 33 BLE Sense y el pin 4 es la conexión dónde se situará el pulsador. Fuente: Elaboración propia.

El programa completo se incluye en el Anexo 0, dónde se puede encontrar toda la programación del archivo “main”, además, también se recomienda el entendimiento de los archivos creados por Edge Impulse sobre la red Neuronal, los cuales se encuentran en el Anexo 0.

De esta manera, se complementará el entendimiento sobre el código y las múltiples funcionalidades del programa, adquiriendo una perspectiva mucho más amplia sobre la solución adoptada y capacitando a los lectores con la habilidad de crear proyectos similares a este, utilizando redes neuronales y microcontroladores.

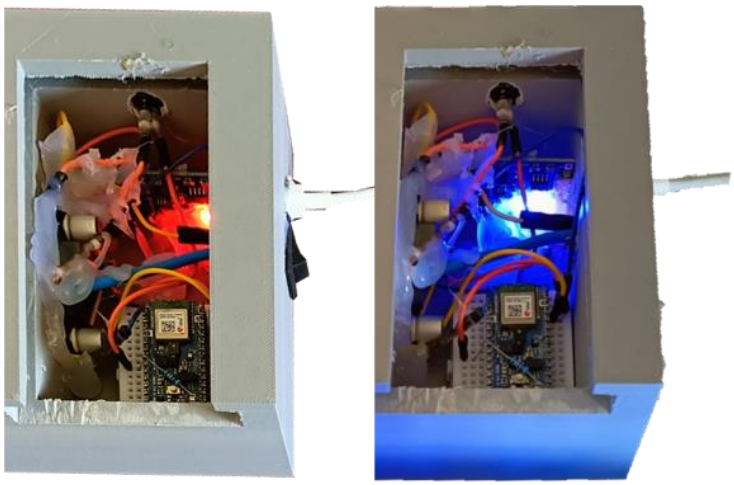
5.6 Prueba de funcionamiento

Para comprobar el funcionamiento del dispositivo se ha creado una versión beta dónde poder realizar diversos ensayos.

En la primera prueba se ha verificado la carga y portabilidad del dispositivo, conectándolo a un puerto MicroUSB confirmando que el módulo TP4056 cumple con su función y carga completamente la batería.

Cuando se conecta por primera vez se enciende el led rojo indicando que se está cargando, y cuando se completa la carga se apaga y se enciende el azul.

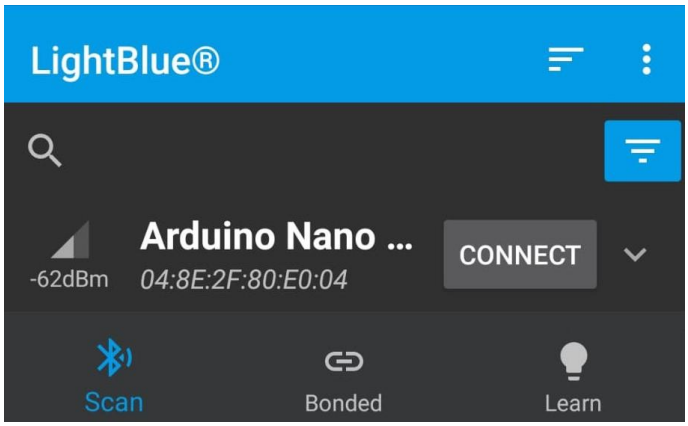
Figura 45. *Leds del módulo TP4056*



Nota. Vista del módulo de carga TP4056 con el led rojo indicando que la batería se está cargando y con el led azul indicando que la carga está completada. Fuente: Elaboración propia.

La segunda prueba ha consistido en comprobar que el sistema de conexión Bluetooth funcionaba, conectándolo con un smartphone y verificando que los datos eran recibidos y enviados correctamente.

Figura 46. *Detección del dispositivo mediante Bluetooth*



Nota. Conexión desde la aplicación LightBlue para comenzar la transmisión de datos. Fuente: Elaboración propia.

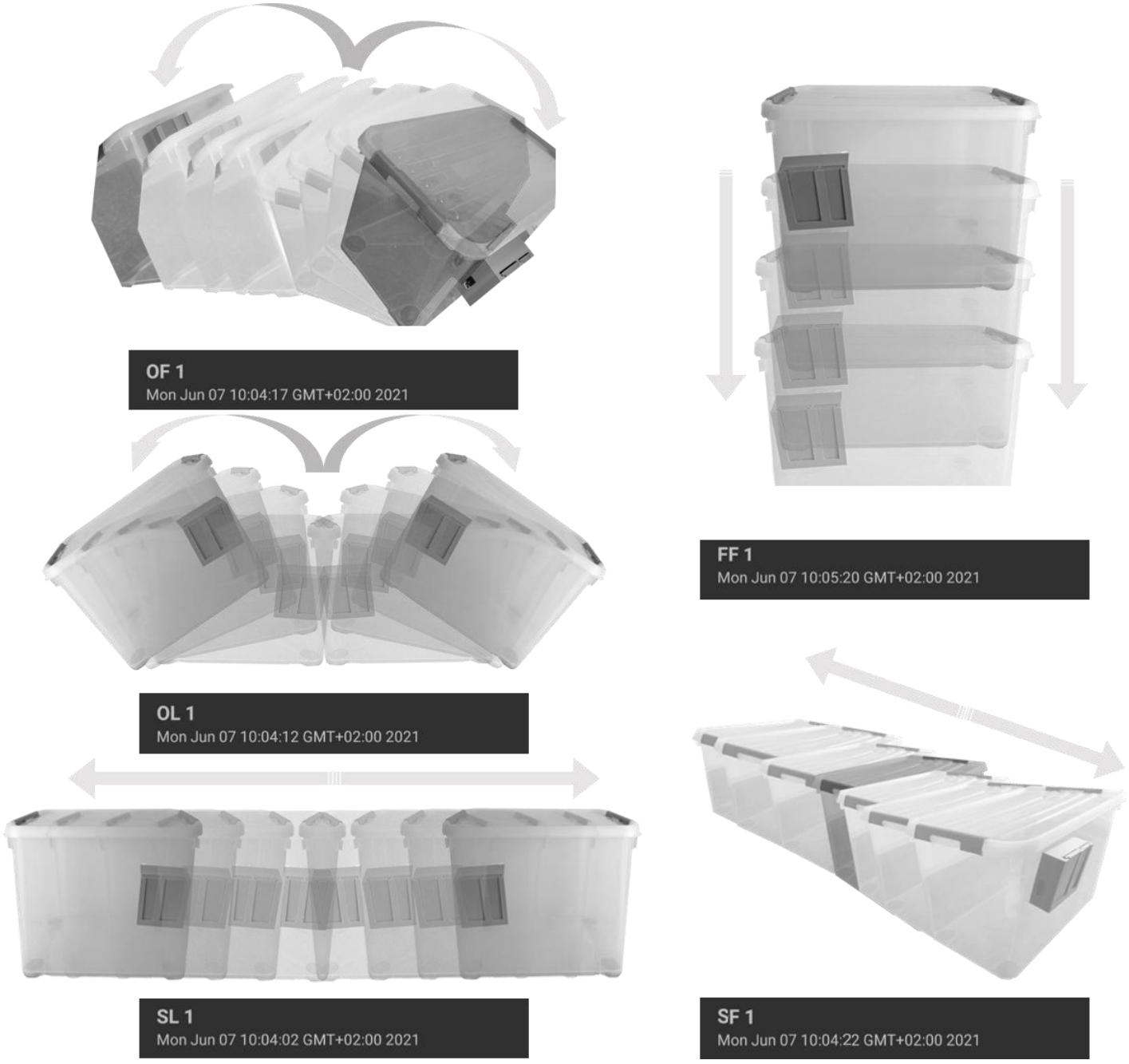
Una vez los sistemas de conexión y carga estaban comprobados se ha testeado el funcionamiento de la inteligencia artificial, fijando el dispositivo a una caja de pruebas lo más parecida a una caja de viales de vacunación. Aunque los viales viajasen en pallets el sistema funcionaría igualmente.

Para comprobar el sistema al completo se han simulado todos los movimientos de manera manual, volcando y golpeando la caja, comprobando que para cada uno de los desplazamientos el sistema enviaba mediante bluetooth todos los datos en tiempo real.

Como en el entrenamiento de la IA, los movimientos se entrenaron de forma pronunciada, las vibraciones que ha sufrido la carga no son detectadas por el sistema, eliminando posibles errores en las lecturas.

El único problema que ha surgido ha sido en la detección del movimiento de caída libre, ya que cuando el dispositivo, cae de forma brusca, no cae verticalmente recto, sino que, debido a las descompensaciones de peso de la caja, suele caer más rápido de un lado que del otro provocando que el sistema pueda detectarlo como un golpe lateral. Este problema podría solucionarse con un entrenamiento más exhaustivo de cada uno de los movimientos, pero hay que tener en cuenta que como es un acelerómetro los datos que se recogen para movimientos iguales pueden ser muy dispersos, provocando una mayor dificultad para solventar este error.

Figura 47. *Reconocimiento de movimientos*



Nota. Reconocimiento de movimientos durante prueba real del dispositivo. Fuente: Elaboración propia.

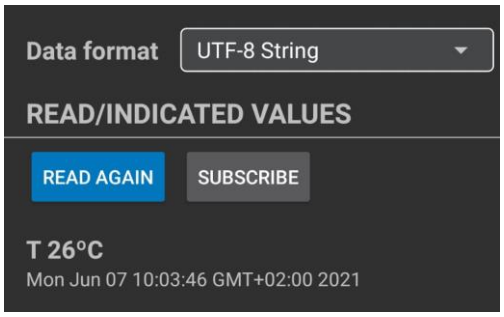
La tercera prueba que se le ha realizado ha sido la del botón de reset, verificando que cuando lo pulsas todas las variables registradas en el sistema son borradas y el led rojo se enciende para confirmarlo de forma visual.

Figura 48. *Led rojo de Reset del dispositivo*

Nota. Encendido del led rojo cuando se presiona el botón de reset. Fuente: Elaboración propia.

Por último, se ha comprobado el funcionamiento del sistema térmico, verificando que las lecturas de la temperatura eran correctas. Las temperaturas se han medido durante un par de horas, registrándolas y comparándolas con otro termómetro dando exactamente los mismos valores.

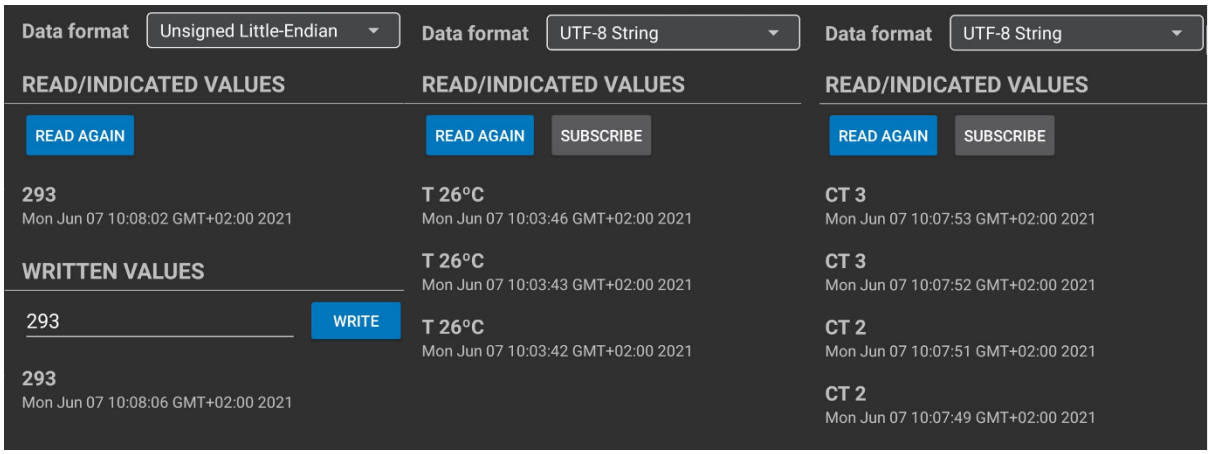
Figura 49. *Lectura de temperatura mediante Bluetooth*



Nota. Lectura de la temperatura ambiente del dispositivo. Fuente: Elaboración propia.

Además, también se ha verificado el funcionamiento del sistema de aviso térmico, prefijando una temperatura manual desde el smartphone de 20ºC, que es la temperatura que hacía en ese momento, y al dejar el sistema durante un par de horas, la temperatura llegó a aumentar a 26ºC, hasta 3 veces, y el sistema lo registró de manera correcta.

Figura 50. *Comprobación del sistema de aviso térmico*



Nota. Fijación de temperatura a 20ºC, cuando el sistema detecta que la temperatura difiere del valor prefijado registra el error en la variable CT. Fuente: Elaboración propia.

La duración máxima de la batería no ha podido ser testeada ya que nunca ha llegado a terminarse a pesar de estar más de 5 días en pleno funcionamiento, pero debería ser fiel a los cálculos teóricos teniendo una vida aproximada de 20 días.

6. Conclusiones

Este proyecto ha sido todo un reto personal, ya que abarcaba un campo muy grande tanto a nivel electrónico como de programación. El proceso de investigación y documentación sobre la creación de redes neuronales y su entrenamiento me ha permitido explorar las innumerables aplicaciones de la inteligencia artificial.

El diseño de este dispositivo me ha aportado una visión muy amplia sobre el proceso de creación de proyectos de esta envergadura, dónde tienes que enfrentarte a una serie de imprevistos problemas, que logran que saques tu máximo potencial para poder solventarlos.

Respecto al software he podido ampliar mis conocimientos sobre el entorno de programación de Arduino, gracias a tener que integrar la inteligencia artificial con los sensores de la placa y con la tecnología Bluetooth. Además, destacar el aprendizaje sobre la plataforma de Edge Impulse, que me ha permitido tanto programar como entrenar de la mejor forma posible a la red neuronal.

Gracias a este trabajo, he podido aplicar y ampliar mis conocimientos sobre los estudios del grado de ingeniería electrónica, debido al diseño e instalación de los diferentes módulos electrónicos que conforman el dispositivo y permiten que se cumplan los objetivos marcados al principio.

De esta manera, puedo afirmar que el dispositivo cumple con lo propuesto, y es capaz de llevar a cabo el reconocimiento de movimientos durante el transporte de los viales de vacunación, monitorizando además la temperatura para que se conserve la cadena del frío, y enviando todos los datos al usuario de manera inalámbrica mediante bluetooth. Por lo tanto, se puede concluir con que el diseño se ha ejecutado de manera efectiva y cumple con todos los requisitos tanto de funcionamiento como de portabilidad.

En definitiva, este diseño es una prueba más de como la tecnología puede ser empleada para solucionar problemas que nos afectan en nuestro día a día, siendo capaz de mejorar considerablemente nuestra calidad de vida. Del mismo modo, expone la gran versatilidad que ofrecen las redes neuronales, una tecnología en pleno auge con un futuro muy prometedor y unas posibilidades aún no exploradas.

7. Bibliografía

Arduino. (2021). *Arduino Nano 33 BLE Sense with headers*. Arduino Official Store. https://store.arduino.cc/arduino-nano-33-ble-sense-with-headers

CircuitDigest. (2019, 20 febrero). *Understanding LoRa Technology* [Fotografía]. https://circuitdigest.com/sites/default/files/inlineimages/u1/Understanding-LoRa-Technology.png

Comité Asesor de Vacunas. (2020). Transporte y conservación de las vacunas. Asociación Española de Pediatría. Disponible en: https://vacunasaep.org/documentos/manual/cap-6#3

Duch Guillot, J. (2020). ¿Qué es la inteligencia artificial y cómo se usa? *Noticias Parlamento Europeo*. Publicado. https://www.europarl.europa.eu/pdfs/news/expert/2020/9/story/20200827STO85804/20200827STO85804_es.pdf

Ecuarobot. (2020, 3 abril). *Arduino Nano 33 BLE Sense* [Fotografía]. http://ecuarobot.com/wp-content/uploads/2020/05/Arduino-Nano-33-BLE-Sense-Hardware-Overview.jpg

Edge Impulse. (2021). *Página web de Edge Impulse* [Fotografía]. https://www.edgeimpulse.com/

Edge Impulse Inc. (2020). *Industry Leading Solutions*. Edge Impulse. https://www.edgeimpulse.com/solutions

Electronilab. (s. f.). *Módulo Transceptor LoRa SX1278 Ra-02 433 Mhz con Base* [Fotografía]. https://cdn.electronilab.co/wp-content/uploads/2018/05/M%C3%B3dulo-Transceptor-LoRa-SX1278-Ra-02-433-Mhz-con-Base-3.jpg

Eta Compute. (2021a). *Placa Eta Compute ECM3532 AI Sensor* [Fotografía]. Eta Compute. https://etacompute.com/wp-content/uploads/2020/06/Page-images-03-1024x631.png

Eta Compute. (2021, 4 marzo). *TENSAI® AI Sensor Board*. https://etacompute.com/tensai-boards/

Himax. (2020, noviembre). *Technical Document*. Datasheet WE-I Plus EVB. https://cdn.sparkfun.com/assets/0/9/9/2/7/WE-I_Plus_EVB_Technical_Document_v02.pdf

Himax. (2020b, noviembre). *Vista Gráfica Placa Himax WE-I Plus EVB* [Fotografía]. https://hackster.imgix.net/uploads/attachments/1225493/image_jcXoV9nK5W.png?auto=compress%2Cformat&w=740&h=555&fit=max

TensorFlow. (2021). *TensorFlow Lite para microcontroladores*. Disponible en: https://www.tensorflow.org/lite/microcontrollers?hl=es

Iberobotics. (2021). *Portapilas 4 pilas AA LR6* [Fotografía]. https://www.iberobotics.com/wp-content/uploads/2018/05/porta_pila_AA_LR6_4_pilas_terminal_cable-e1526987562614.jpg

Llamas, L. (2016, 16 octubre). *Módulo tarjeta SD* [Fotografía]. Luis Llamas. https://www.luisllamas.es/wp-content/uploads/2016/10/arduino-micro-sd-esquema.png

Nordic Semiconductor. (2021). *nRF52840* [Fotografía]. https://www.nordicsemi.com/-/media/Images/Products/SoC/SoCs-dobble-top/nRF52-Series/nRF52840-QIAA.png?h=350&la=en&mw=350&w=350&hash=B883E5A775839D798942847F944B4C0BF941E364

Pastorino, C. (2020, 17 marzo). *Cómo funciona Bluetooth Low Energy: el protocolo estrella de IoT*. WeLiveSecurity. https://www.welivesecurity.com/la-es/2020/03/17/como-funciona-bluetooth-low-energy/

Roboelements. (s. f.). *XL9006 Module Pinout* [Fotografía]. https://www.roboelements.com/wp-content/uploads/2020/04/XL6009-Module-Pinout.jpg

Solectroshop. (s. f.-a). *Módulo TP4056 Cargador Baterias Litio con Protección micro USB* [Fotografía]. https://solectroshop.com/1575-large_default/modulo-tp4056-cargador-baterias-litio-con-proteccion.jpg

STMicroelectronics. (2016, agosto). *HTS221*. https://www.st.com/resource/en/datasheet/hts221.pdf

STMicroeletronics. (2017, 4 abril). *STMicroelectronics B-L475E-IOT01A Discovery Kit for IoT Node* [Fotografía]. https://eu.mouser.com/images/marketingid/2017/img/160178092_STMicro_B-L475E-IOT01ADiscoveryKit.png?v=051720.1042

STMicroelectronics. (2021). *B-L475E-IOT01A. STMicroelectronics.* https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html#overview

Texas Instrument. (1999, agosto). *LM35 Precision Centigrade Temperature Sensors*. https://www.ti.com/lit/ds/symlink/lm35.pdf

Vinibattery. (s. f.). *Pila 9V* [Fotografía]. https://vinibattery.com/wp-content/uploads/2018/06/DUR01925_02.jpg

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN DEL TRANSPORTE DE VACUNAS MEDIANTE EL USO DE TECNOLOGÍA IOT Y MACHINE LEARNING

2. Planos

Curso académico 2020-2021

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

Autor:

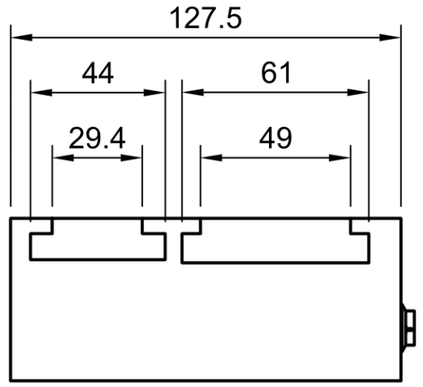
José Herrero Ruiz

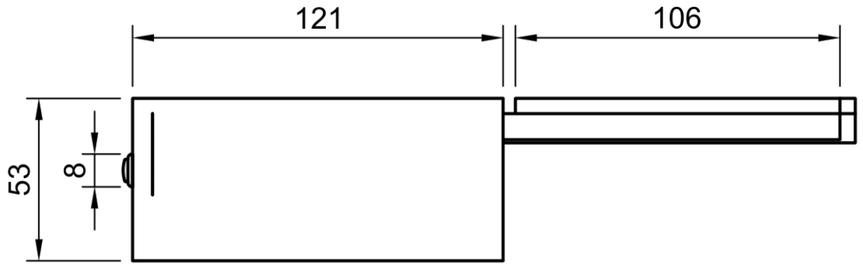
Tutores:

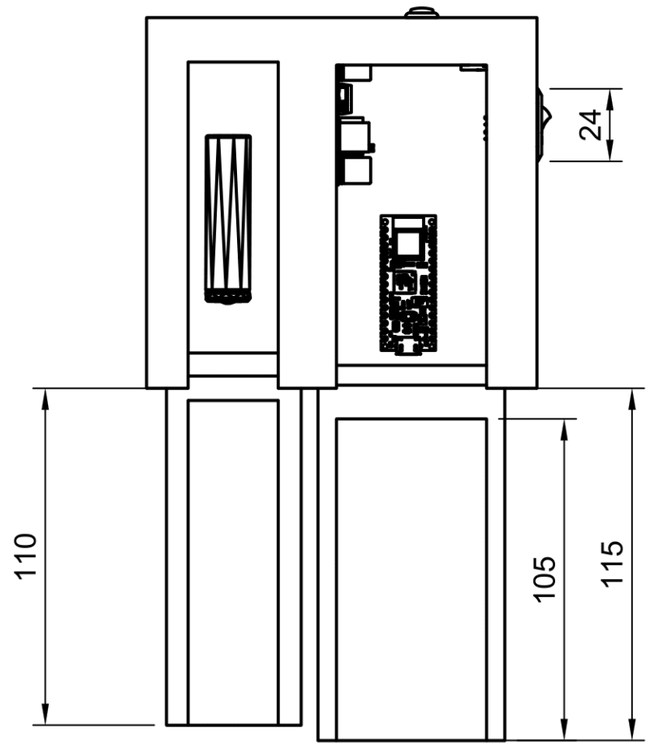
Ángel Francisco Perles Ivars

Jaime Laborda Macario

Índice de planos

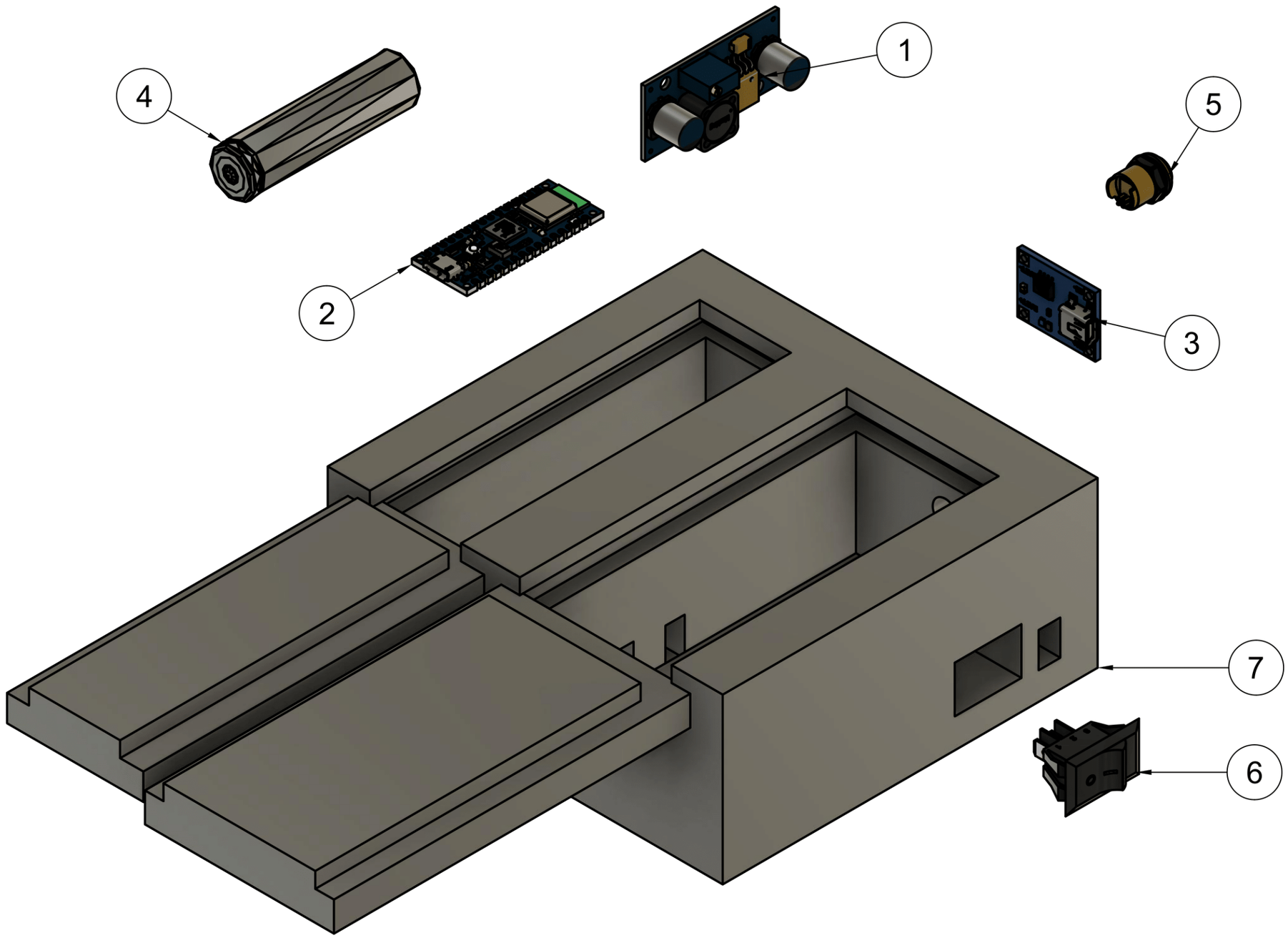




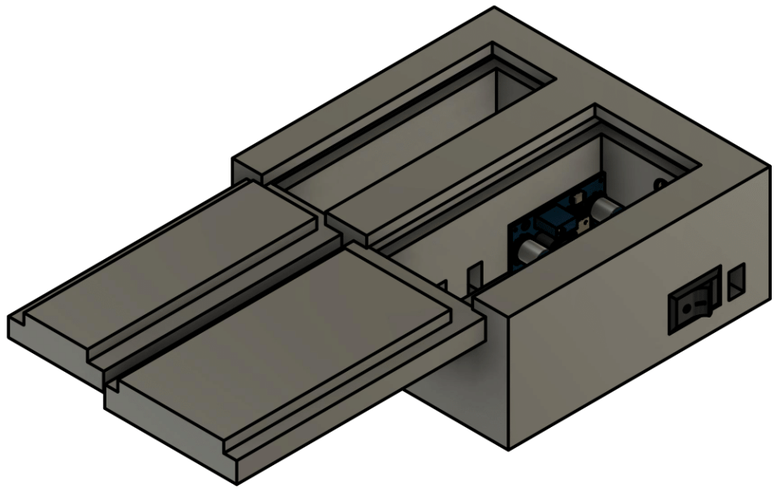


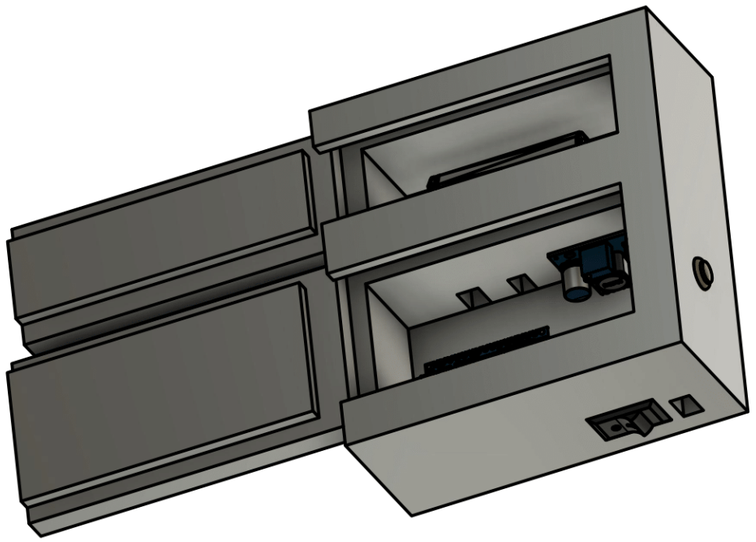
Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática UNIVERSITAT POLITÈCNICA DE VALÈNCIA — Escuela Técnica Superior de Ingeniería del Diseño	Proyecto: Sistema de monitorización del transporte de vacuna mediante Iot y machine learning
Autor: José Herrero Ruiz	Fecha: 15/04/2021 — Escala: 1:2 Título: Plano Dimensional — Nº Plano: 1

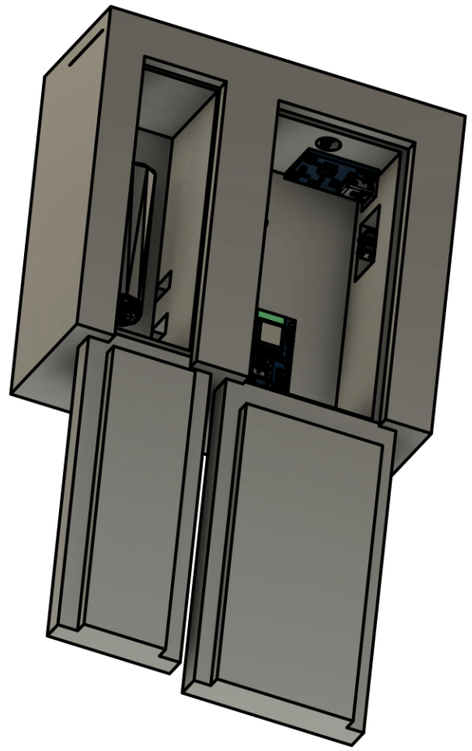
Parts List		
Item	Qty	Part Number
1	1	XL6009
2	1	Arduino Nano 33 BLE Sense
3	1	TP4056
4	1	Battery 18650
5	1	Button
6	1	Switch
7	1	Case

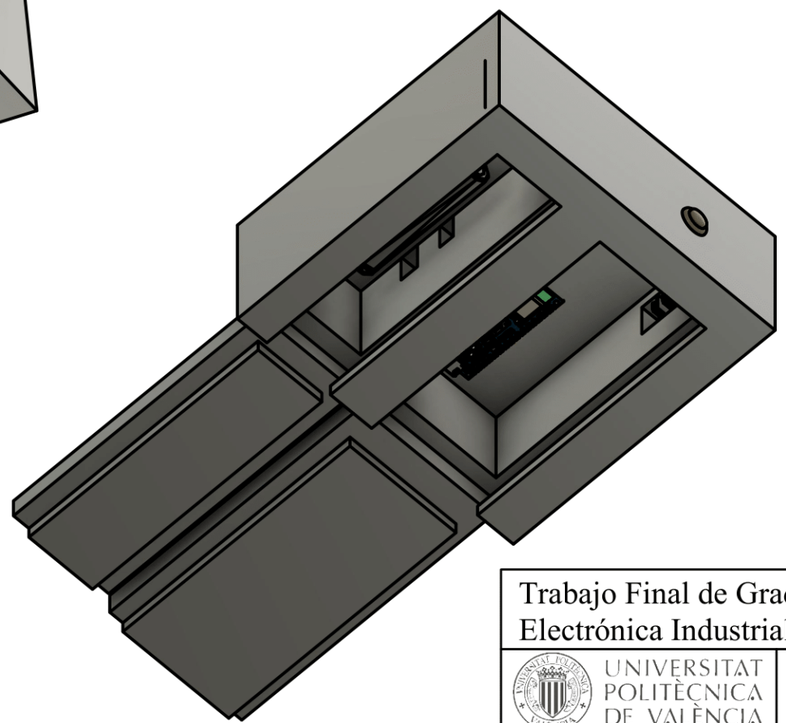


Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática UNIVERSITAT POLITÈCNICA DE VALÈNCIA — Escuela Técnica Superior de Ingeniería del Diseño	Proyecto: Sistema de monitorización del transporte de vacuna mediante Iot y machine learning
Autor: José Herrero Ruiz	Fecha: 15/04/2021 — Escala: 1:1 Título: Plano de Ensamblaje — Nº Plano: 2









Trabajo Final de Grado en Ingeniería Electrónica Industrial y Automática UNIVERSITAT POLITÈCNICA DE VALÈNCIA — Escuela Técnica Superior de Ingeniería del Diseño	Proyecto: Sistema de monitorización del transporte de vacuna mediante Iot y machine learning	
	Fecha: 15/04/2021	Escala: 1:1
Autor: José Herrero Ruiz	Título: Plano de Vistas Generales	Nº Plano: 3

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN DEL TRANSPORTE DE VACUNAS MEDIANTE EL USO DE TECNOLOGÍA IOT Y MACHINE LEARNING

3. Pliego De Condiciones

Curso académico 2020-2021

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

Autor:

José Herrero Ruiz

Tutores:

Ángel Francisco Perles Ivars

Jaime Laborda Macario

Índice Pliego de Condiciones

1. Objeto

El propósito de este documento es el establecimiento de los requisitos técnicos de fabricación y diseño del dispositivo de monitorización encargado de conservar en condiciones óptimas los viales de vacunación durante el transporte, mediante la recogida de datos procesados por una red neuronal.

La información de la cual se puede disponer en este documento tiene como objetivo el entendimiento completo del proceso de fabricación físico, fijando además las condiciones legales que deben cumplirse para que el dispositivo pueda ser introducido en el mercado sin generar ningún riesgo para la salud de las personas que lo manipulen, y asegurando la máxima garantía para todos los elementos que lo conforman.

Del mismo modo, se establecerán cuáles son las condiciones de instalación y mantenimiento que deben efectuarse durante el manejo del sistema, asentando un único método de conservación que certifique la estabilidad y protección del proceso.

2. Condiciones y normas de carácter general

2.1 Instalación

La instalación del dispositivo podrá ser realizada por la empresa compradora, las instrucciones del producto clarificarán todo el proceso de preparación y arranque.

Se recomienda que el dispositivo sea instalado por personal cualificado que entienda perfectamente el funcionamiento ya que podrían generarse errores durante el inicio o calibrado difícilmente solucionables si no se tienen conocimientos técnicos sobre la electrónica interna.

Además, será necesaria la asignación de un responsable que sea capaz de recoger todos los datos, entenderlos y saber cómo actuar ante situaciones en las que la carga a transportar esté sufriendo algún riesgo que produzca su inutilidad.

2.2 Seguridad

Respecto a la seguridad del dispositivo, deberá tratarse con especial cuidado, sin golpear la carcasa, ya que podría ocasionar problemas en las lecturas, originando datos erróneos e incluso la posibilidad de invalidar completamente un trayecto.

El dispositivo cumple las siguientes normativas de seguridad, las cuales deberán ser revisadas para cada actualización que se ejecute sobre el sistema:

- Real Decreto 186/2016, de 6 de mayo, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos.
- Orden IET/787/2013, de 25 de abril, por la que se aprueba el cuadro nacional de atribución de frecuencias
- Directiva 2011/65/UE del Parlamento Europeo y del Consejo, de 8 de junio de 2011, sobre restricciones a la utilización de determinadas sustancias peligrosas en aparatos eléctricos y electrónicos.
- Real Decreto 208/2005, en cual se establecen las medidas de prevención y la gestión de residuos sobre aparatos eléctricos y electrónicos.
- Real Decreto 255/2003, de 28 de febrero, por el que se aprueba el Reglamento sobre clasificación, envasado y etiquetado de preparados peligroso.

2.3 Utilización

Principalmente, para llevar a cabo un buen uso del dispositivo, se tendrá que establecer el lugar de colocación, cerciorándose de la existencia de espacio suficiente como para que no hayan colisiones con otros elementos de la carga, que puedan separar al dispositivo de su origen.

Una vez se haya localizado el sitio dónde va a fijarse, será necesario prestar atención a la orientación del sistema, este punto es uno de las más importantes, ya que, si se fija con una orientación incorrecta, todos los datos originados por la red neuronal, no tendrán ningún tipo de validez. Para lograr este punto, se ha colocado una flecha en la parte superior de la carcasa que indica al instalador hacia dónde ha de orientar el dispositivo.

Terminada la instalación del dispositivo, será necesario el empleo de un terminal móvil que tenga instalada la aplicación "LightBlue", y conectarse a la placa de Arduino, comprobando en el menú de atributos que la conexión está activa y que los datos tanto de movimiento como de temperatura llegan perfectamente.

El hecho de ser un prototipo indica que el dispositivo no está eximido de que aparezcan posibles fallos desconocidos durante su uso.

2.4 Mantenimiento

El mantenimiento del sistema va a resultar necesario e imprescindible, sobre todo en la parte inferior de la carcasa dónde se sitúan las sujeciones del dispositivo, debido al desgaste que sufre el material pegajoso de esa zona, que puede perder su eficacia, resultando inevitable su reemplazamiento.

Además, es altamente recomendable la comprobación de voltajes de la batería, ya que es uno de los elementos que más degradación sufre, y si fuera necesario cambiarla por otra del mismo modelo, 18650.

Las actualizaciones y mejoras del sistema corren a cargo del autor del proyecto.

3. Condiciones particulares y fabricación

3.1 Condiciones de los materiales

3.1.1 Materiales

A continuación, se presentan los materiales empleados durante la fabricación del dispositivo:

- Arduino Nano 33 BLE
- Módulo Step-up XL6009
- Módulo de carga TP4056
- Interruptor basculante
- Pulsador
- Batería 18650 3.7 V
- Resistencia de 10 KΩ
- Filamento ABS
- Impresora 3D
- 8 cables de cobre Unipolar 0,5 mm²
- Soldador
- Hilo con aleación de estaño, plomo y cobre (60/38/2)
- Termofusible

3.1.2 Fabricación de la carcasa del dispositivo

La fabricación de la caja dónde se situarán los componentes electrónicos se llevará a cabo mediante el uso de una Impresora 3D. El material que se va a emplear para la impresión es el filamento de ABS, un material muy tenaz, duro, rígido, y con resistencia química a la abrasión.

La impresora no necesita unas altas prestaciones, ya que la carcasa no posee formas geométricas que conlleven una gran dificultad de impresión. Simplemente se requerirá del archivo. STL que contiene el modelado 3D de la caja, tal y como se puede apreciar en el plano de explosión. Véase. Plano nº2.

3.2 Condiciones de fabricación

Las condiciones de fabricación se pueden realizar sin necesidad de maquinaria profesional, con unos costes de producción relativamente bajos.

El proceso se va a dividir principalmente en 3 grupos:

3.2.1 Conexiones eléctricas

La fase de interconexión deberá realizarse con unas condiciones determinadas, para garantizar la integridad y estabilidad del dispositivo.

Las condiciones a cumplir son:

- Empleo de guantes electroestáticos para evitar cualquier descarga eléctrica procedente de la persona que va a llevar a cabo la instalación.
- Zona de trabajo limpio para evitar colisiones con los componentes electrónicos y fallos en las soldaduras provocados por el polvo.
- Control de la temperatura y humedad en le zona de trabajo respetando los límites de los componentes.
- Desconectar las fuentes de alimentación de las placas de trabajo
- Comprobación de la continuidad de las soldaduras que se lleven a cabo, cerciorando el perfecto funcionamiento.

Durante todo el proceso de fabricación, la soldadura resulta necesaria para poder unir cualquier placa electrónica, por ello, es de obligado cumplimiento no soldar la placa durante un tiempo muy elevado para evitar el sobrecalentamiento en los demás componentes, ya que debido a la fragilidad de algunos módulos, como por ejemplo el del Bluetooth, se requerirá de un cuidado especial y no se deberá tocar con la punta del soldador, simplemente, habrá que soldar en el lugar de aplicación durante un tiempo máximo de 5 segundos.

3.2.2 Programación

El apartado del software tiene una gran relevancia ya que es el encargado de recoger e interpretar todos los datos durante el trayecto.

Sobre el volcado de la programación en el dispositivo, se realizará en el software de Arduino, siempre por personal cualificado que entienda el lenguaje de programación y esté capacitado para solventar los fallos que surjan durante el proceso. Además, previamente se

deberá hacer un test de manera independiente a cada uno de los sensores de la placa, para poder descartar cualquier fallo de hardware que provenga del fabricante.

El software es válido para su reutilización en cualquier otro sistema que integre exactamente el mismo hardware.

3.2.3 Montaje

Una vez terminada la programación y las conexiones de la placa principal a los módulos de carga, será necesario montar todo el conjunto dentro de la carcasa.

Este montaje se deberá llevar a cabo con la mayor seguridad y cautela, ya que podría perjudicar a algún componente, incluso la posibilidad de romper alguna de las soldaduras realizadas. En caso de que se produzca alguna rotura, se extraerá el dispositivo siempre respetando las condiciones de fabricación previamente explicadas y se procederá a solventar la rotura para posteriormente poder reanudar el proceso de montaje.

Una vez montado se deberá comprobar la continuidad de todas las conexiones, cerciorándose que el dispositivo va a funcionar perfectamente, cumpliendo las expectativas de garantía y seguridad de la normativa.

Además, queda terminantemente prohibido el uso de cualquier producto químico que pueda conllevar algún riesgo o peligrosidad para el medio ambiente.

3.3 Condiciones de entrega

El dispositivo se entregará al cliente en perfecto funcionamiento, con las instrucciones de montaje lo más clarificadas posible para facilitar el proceso de instalación.

Respecto a los plazos, el sistema se entregará en 2 semanas naturales desde la firma del contrato con el cliente. Una vez entregado, el producto poseerá 2 años de garantía, además se incluirá un servicio de actualizaciones del cual podrán disponer durante toda la vida útil del dispositivo. En caso de que el producto sea manipulado o la etiqueta de garantía esté dañada, se exime de toda responsabilidad a la empresa vendedora, quedando a cargo del comprador cualquier reparación.

Si en el momento de la entrega, el cliente, por alguna razón no justificada, resulta estar inaccesible e incomunicado, será completamente responsabilidad suya, cualquier retraso generado por esa causa, teniendo que responder además a los gastos derivados del transporte.

3.4 Prueba de servicio

Antes de realizar la entrega del dispositivo al cliente se deberán llevar a cabo las pruebas de servicio que corroboren el correcto funcionamiento de la unidad que está prevista ser vendida. Estas pruebas de servicio, tendrán que ser efectuadas por técnicos cualificados, siendo las siguientes:

- Examinar la continuidad de todas las soldaduras una vez ensamblado el sistema.
- Comprobar el estado de la batería, siendo su tensión nominal de 3.7V.
- Conectar un dispositivo bluetooth y asegurar la correcta transmisión de datos.
- Simular todos los movimientos, confirmando que el recuento sea correcto.
- Verificar la lectura de la temperatura comparándola con un termómetro externo.
- Confirmar que el puerto micro USB carga la batería correctamente.
- Cerciorarse de la integridad de la carcasa cerrando completamente el dispositivo.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN DEL TRANSPORTE DE VACUNAS MEDIANTE EL USO DE TECNOLOGÍA IOT Y MACHINE LEARNING

4. Presupuesto

Curso académico 2020-2021

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

Autor:

José Herrero Ruiz

Tutores:

Ángel Francisco Perles Ivars

Jaime Laborda Macario

Índice del presupuesto

Índice de tablas

1. Sobre el presupuesto

El presente documento tiene como finalidad mostrar cual es el coste total del desarrollo, diseño y fabricación del dispositivo de monitorización.

Los precios mostrados provienen de distribuidores autorizados, pero debido a la variabilidad del mercado en los productos electrónicos pueden cambiar sin previo aviso. Además, los modelos de las placas podrían ser actualizados o reemplazados por otros en el caso de que se dejasen de producir.

Para el desarrollo de este producto, se ha teniendo en cuenta la producción de un sistema de bajo coste que pudiese implementar una inteligencia artificial, es por eso que el presupuesto está dividido en apartados para así poder tener una perspectiva del coste real de un solo dispositivo, permitiendo individualizar los costes asociados al desarrollo y a la investigación, los cuales, son reutilizables para todos los productos futuros que se manufacturen.

No obstante, este presupuesto no es taxativo, ya que cualquier modificación por alguna de las partes deberá ser negociada, estando sujeto a cambios según conveniencia de los interesados.

2. Materiales

Considerar que el precio de las siguientes tablas hace referencia al precio de fabricación de un solo dispositivo.

2.1 Componentes Electrónicos

Tabla 1. *Presupuesto componentes electrónicos*

Unidades	Descripción	Cantidad	Fabricante	Ref. Fabricante	Precio	Total
EA	Arduino Nano 33 BLE Sense	1	Arduino	ARD-0141	34,99 €	34,99 €
EA	XL6009 Convertidor DC a DC Booster Step-Up	1	XL Semi	XL6009	2,23 €	2,23 €
EA	Módulo TP4056 Cargador Baterías Litio	1	Nanjing Top	TP4056	1,69 €	1,69 €
EA	Resistencia 10K Ohm 5% 0,25W 1/4W	1	VISHAY	SFR25000010	0,05 €	0,05€
EA	Batería Recargable 3,7V 9900 mAh Li-ion	1	GTF	TR18650	3,95 €	3,95 €
EA	Pulsador Normalmente Abierto 7mm	1	C & K	8531MZQE2	4,15 €	4,15€
EA	Interruptor Basculante	1	MULTICOMP	MP005716	1,47 €	1,47€
EA	Cable Cobre 0,5mm^2	8	LAPP	H05Z1-K	0,12 €	0,96 €
Total						49,49€

Nota. Fuente. Elaboración Propia.

2.2 Materiales de fabricación para la carcasa

La impresora 3D que se va a emplear permite el uso de filamento ABS, el cual emplearemos para la fabricación de la carcasa teniendo en cuenta que la pieza consumirá un total de 634,33 g.

Tabla 2. *Presupuesto materiales de fabricación para la carcasa*

Unidades	Descripción	Cantidad	Fabricante	Ref. Fabricante	Precio	Total
g	Filamento ABS, 1kg, 1,75mm	634,33	RS PRO	832-0453	32,57 €	32,57 €
h	Coste de las horas de impresión	20	-	-	0,5 €	10 €
EA	Coste de calibrado y preparación	1	-	-	2,00 €	2,00 €
Total						44,57 €

Nota. Fuente. Elaboración Propia.

2.3 Materiales para ensamblaje

Tabla 3. *Presupuesto de los materiales para ensamblaje*

Unidades	Descripción	Cantidad	Fabricante	Ref. Fabricante	Precio	Total
EA	Hilo Estaño 0.5mm 100g 60/38/2	1	Diotronic	HIF00639	4,57 €	4,57 €
EA	Barras de cola termofusible Ø 7,5 mm x 10 cm	2	Apli	13740	0,16 €	0,32 €
Total						4,89 €

Nota. Fuente. Elaboración Propia.

3. Mano de Obra

El coste de la mano de obra no incluye las horas de desarrollo de software ni de diseño del dispositivo. Es por eso, que la tabla muestra el coste de mano de obra solo para la construcción de un dispositivo de monitorización.

Los costes de la mano de obra están clasificados según la función que se desempeña en cada puesto. Los precios de la hora han sido obtenidos de los datos oficiales del IVE.

Tabla 4. *Presupuesto mano de obra*

Unidad	Descripción	Horas Parciales	Horas Totales	Precio	Total
h	MOOE.12a Peón		2	17,98 €	35,96 €
	Conexionado de placas	1			
	Ensamblado del dispositivo	1			
h	MOOE.8a Oficial 1º		0,5	19,23 €	9,62 €
	Carga del firmware en placa	0,5			
h	MOOE.2a Ingeniero Técnico		2	22,57 €	45,14 €
	Supervisión de la instalación	1			
	Puesta en marcha	1			
Total					90,72 €

Nota. Fuente. Elaboración Propia.

4. Diseño y desarrollo del dispositivo

A continuación, se muestra el coste total del diseño y desarrollo del dispositivo. Esta tabla es meramente informativa con la finalidad de mostrar cuál es el coste inicial y poder incluirlo como parte del desembolso inicial necesario para comenzar con la actividad productiva.

Tabla 5. *Presupuesto mano de obra del diseño y desarrollo*

Unidad	Descripción	Horas Parciales	Horas Totales	Precio	Total
h	MOOE.2a Ingeniero Técnico		30	22,57 €	677,10 €
	Preparación del entorno de programación	1			
	Entrenamiento de la Inteligencia Artificial	6			
	Migración e integración de la IA en Arduino	2			
	Obtención de librerías de los sensores a emplear	1			
	Programación del código fuente completo	18			
	Redacción de planos y presupuestos	2			
Total					677,10 €

Nota. Fuente. Elaboración Propia.

5. Resumen

5.1 Resumen sin gastos de desarrollo

Tabla 8. *Presupuesto sin gastos de desarrollo*

Descripción	Coste Parcial	Coste Total
Materiales		98,95 €
Componentes Electrónicos	49,49 €	
Materiales de fabricación para la carcasa	44,57 €	
Materiales para ensamblaje	4,89 €	
Mano de Obra		90,72 €
Total Sin Impuestos		189,67 €
IVA	21%	39,83 €
Gastos Adicionales	10%	18,97 €
Coste Total Del Dispositivo		248,47 €

Nota. Fuente. Elaboración Propia.

El coste total del dispositivo es de DOS CIENTOS CUARENTA Y OCHO CON CUARENTA Y SIETE CÉNTIMOS, IVA incluido.

5.2 Resumen con gastos de desarrollo

Tabla 9. *Presupuesto completo*

Descripción	Coste Parcial	Coste Total
Materiales		98,95 €
Componentes Electrónicos	49,49 €	
Materiales de fabricación para la carcasa	44,57 €	
Materiales para ensamblaje	4,89 €	
Mano de Obra		90,72 €
Diseño y Desarrollo		677,10 €
Total Sin Impuestos		866,77 €
IVA	21%	182,02 €
Gastos Adicionales	10%	86,68 €
Coste Total Del Dispositivo		1135,47 €

Nota. Fuente. Elaboración Propia.

El coste total del dispositivo incluido el diseño y desarrollo es de MIL CIENTO TREINTA Y CINCO CON CUARENTA Y SIETE CÉNTIMOS, IVA incluido.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

SISTEMA DE MONITORIZACIÓN DEL TRANSPORTE DE VACUNAS MEDIANTE EL USO DE TECNOLOGÍA IOT Y MACHINE LEARNING

5. Anexos

Curso académico 2020-2021

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

Autor:

José Herrero Ruiz

Tutores:

Ángel Francisco Perles Ivars

Jaime Laborda Macario

Índice de Anexos

5.1 Documentación Código

5.1.1 Programa principal con código propio

```
/* Includes ------------------------------------------------------------------*/
#include <tfg_inference.h>
#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>
#include <Arduino_HTS221.h>
/* Constant defines -------------------------------------------------------*/
#define CONVERT_G_TO_MS2    9.80665f
/* Private variables ------------------------------------------------------*/
static bool debug_nn = false; // Set this to true to see e.g. features
generated from the raw signal
static uint32_t run_inference_every_ms = 200;
static rtos::Thread inference_thread(osPriorityLow);
static float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };
static float inference_buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE];
static int Freefall=0,StrokeL=0, StrokeF=0,OverturnL=0,
OverturnF=0,Temperature=0,Tmax=0,Tmin=0,Tassign=0,CTemperature=0,Timer=0,Re
set=0;
static bool StartT=true;
static const char *previous;
char *FFdata="FF %d",*SLdata="SL %d",*SFdata="SF %d",*OLdata="OL
%d",*OFdata="OF %d",*Tdata="T %d°C",*CTdata="CT %d",FFChar[10]="
",SLChar[10]=" ",SFChar[10]=" ",OLChar[10]="",OFChar[10]=" ", TChar[10]="
",CTChar[10]=" ";
/* Forward declaration */
void run_inference_background();

BLEService Data("6240f192-c6e3-11eb-b8bc-0242ac130003");
BLECharacteristic FreefallCharacteristic("00000000-0000-0000-0000-
000000000001", BLERead | BLENotify,"New FFdata" );
BLECharacteristic StrokeLCharacteristic("00000000-0000-0000-0000-
000000000002", BLERead | BLENotify,"New SLdata" );
BLECharacteristic StrokeFCharacteristic("00000000-0000-0000-0000-
000000000003", BLERead | BLENotify,"New SFdata" );
BLECharacteristic OverturnLCharacteristic("00000000-0000-0000-0000-
000000000004", BLERead | BLENotify,"New OLdata" );
BLECharacteristic OverturnFCharacteristic("00000000-0000-0000-0000-
000000000005", BLERead | BLENotify,"New OFdata" );
BLECharacteristic TemperatureCharacteristic("00000000-0000-0000-0000-
00000000000a",  BLERead | BLENotify,"New Tdata");
BLECharacteristic CriticalTemperatureCharacteristic("00000000-0000-0000-
0000-00000000000b", BLERead | BLENotify,"New CTdata" );
BLEIntCharacteristic WriteTemperatureCharacteristic("00000000-0000-0000-
0000-00000000000c",  BLERead | BLEWrite);
```

```
void setup()
{

    IMU.begin();
    BLE.begin();
    HTS.begin();
    inference_thread.start(mbed::callback(&run_inference_background));
    BLE.setDeviceName( "Arduino Nano 33 BLE" );
    BLE.setLocalName( "Arduino Nano 33 BLE" );
    BLE.setAdvertisedService( Data );

    Data.addCharacteristic(FreefallCharacteristic);
    Data.addCharacteristic(StrokeLCharacteristic);
    Data.addCharacteristic(StrokeFCharacteristic);
    Data.addCharacteristic(OverturnLCharacteristic);
    Data.addCharacteristic(OverturnFCharacteristic);
    Data.addCharacteristic(TemperatureCharacteristic);
    Data.addCharacteristic(CriticalTemperatureCharacteristic);
    Data.addCharacteristic(WriteTemperatureCharacteristic);

    BLE.addService(Data);

    BLE.advertise();

    pinMode(22, OUTPUT);
    pinMode(4,INPUT);
}


/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/
void loop()
{
        /* Determine the next tick (and then sleep later) */
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);

        /* roll the buffer -3 points so we can overwrite the last one */
        numpy::roll(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, -3);

        /* read to the end of the buffer */
        IMU.readAcceleration(
            buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3],
            buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2],
            buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1]
        );

        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 3] *= CONVERT_G_TO_MS2;
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 2] *= CONVERT_G_TO_MS2;
```

```
        buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE - 1] *= CONVERT_G_TO_MS2;
```

```
/* and wait for next tick */
        uint64_t time_to_wait = next_tick - micros();
        delay((int)floor((float)time_to_wait / 1000.0f));
        delayMicroseconds(time_to_wait % 1000);

        if(Timer==500){
        Temperature = HTS.readTemperature()+273;
        Timer=0;
          if(StartT==true){
            Tmax=Temperature+2;
            Tmin=Temperature-2;
            StartT=false;
            }
           if(Temperature>Tmax or Temperature<Tmin)
                CTemperature++;
        }
        else Timer++;
        BLEDevice central = BLE.central();
        if (central.connected())
        {
          sprintf(FFChar, FFdata, Freefall);
          sprintf(SLChar, SLdata, StrokeL);
          sprintf(SFChar, SFdata, StrokeF);
          sprintf(OLChar, OLdata, OverturnL);
          sprintf(OFChar, OFdata, OverturnF);
          sprintf(TChar, Tdata, Temperature-273);
          sprintf(CTChar, CTdata, CTemperature);
         FreefallCharacteristic.writeValue(FFChar);
         StrokeLCharacteristic.writeValue(SLChar);
         StrokeFCharacteristic.writeValue(SFChar);
         OverturnLCharacteristic.writeValue(OLChar);
         OverturnFCharacteristic.writeValue(OFChar);
         TemperatureCharacteristic.writeValue(TChar);
         CriticalTemperatureCharacteristic.writeValue(CTChar);

         if(WriteTemperatureCharacteristic.written()){

          Tassign= WriteTemperatureCharacteristic.value();
          if( Tassign <233 or Tassign>393)
            {
               Tassign=0;
            TemperatureCharacteristic.writeValue((byte)0);
            }
            else{
              Tmax=Tassign+2;
              Tmin=Tassign-2;
              }
            }
          }
```

```
          Reset = digitalRead(4);
           if (Reset == LOW){
             digitalWrite(22, LOW);
             Freefall=0;
             StrokeL=0;
             StrokeF=0;
             OverturnL=0;
             OverturnF=0;
             Temperature=0;
             Tmax=0;
             Tmin=0;
             Tassign=0;
             CTemperature=0;
             Timer=0;
             StartT=true;
             previous=0;
             }
             if (Reset == HIGH){
             digitalWrite(22, HIGH);
             }

      }
```

5.1.2 Función run_inference_background de Edge con adaptación propia

```
/**
 * @brief      Run inferencing in the background.
 */
void run_inference_background()
{
    /* wait until we have a full buffer */
    delay((EI_CLASSIFIER_INTERVAL_MS * EI_CLASSIFIER_RAW_SAMPLE_COUNT) +
100);

    /* This is a structure that smoothens the output result.*/
    /* With the default settings 70% of readings should be the same before
classifying. */
    ei_classifier_smooth_t smooth;
    ei_classifier_smooth_init(&smooth, 10 /* no. of readings */, 7 /* min.
readings the same */, 0.8 /* min. confidence */, 0.3 /* max anomaly */);

    while (1) {
        /* copy the buffer */
        memcpy(inference_buffer, buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE
* sizeof(float));

        /* Turn the raw buffer in a signal which we can the classify */
        signal_t signal;
        int err = numpy::signal_from_buffer(inference_buffer,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
```

```
        /* Run the classifier */
        ei_impulse_result_t result = { 0 };
        run_classifier(&signal, &result, debug_nn);

        /* ei_classifier_smooth_update yields the predicted label */
        const char *prediction = ei_classifier_smooth_update(&smooth,
&result);
        if (prediction != previous ){
        if (prediction == "Caida libre"){
        Freefall++;}
       if (prediction == "Golpe ID"){
        StrokeL++;}
        if (prediction == "Golpe TD"){
        StrokeF++; }
        if (prediction == "Vuelco ID"){
        OverturnL++; }
        if (prediction == "Vuelco TD"){
        OverturnF++;}
        previous=prediction;
        }
        delay(run_inference_every_ms);
    }

    ei_classifier_smooth_free(&smooth);
}
```

5.1.3 Librerías obtenidas del repositorio de Arduino

5.1.3.1 Arduino_LSM9DS1

```
/*
  This file is part of the Arduino_LSM9DS1 library.
  Copyright (c) 2019 Arduino SA. All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301
USA
*/

#include "LSM9DS1.h"

#define LSM9DS1_ADDRESS            0x6b
```

```
#define LSM9DS1_WHO_AM_I            0x0f
#define LSM9DS1_CTRL_REG1_G         0x10
#define LSM9DS1_STATUS_REG          0x17
#define LSM9DS1_OUT_X_G             0x18
#define LSM9DS1_CTRL_REG6_XL        0x20
#define LSM9DS1_CTRL_REG8           0x22
#define LSM9DS1_OUT_X_XL            0x28

// magnetometer
#define LSM9DS1_ADDRESS_M           0x1e

#define LSM9DS1_CTRL_REG1_M         0x20
#define LSM9DS1_CTRL_REG2_M         0x21
#define LSM9DS1_CTRL_REG3_M         0x22
#define LSM9DS1_STATUS_REG_M        0x27
#define LSM9DS1_OUT_X_L_M           0x28

LSM9DS1Class::LSM9DS1Class(TwoWire& wire) :
  continuousMode(false), _wire(&wire)
{
}

LSM9DS1Class::~LSM9DS1Class()
{
}

int LSM9DS1Class::begin()
{
  _wire->begin();

  // reset
  writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG8, 0x05);
  writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG2_M, 0x0c);

  delay(10);

  if (readRegister(LSM9DS1_ADDRESS, LSM9DS1_WHO_AM_I) != 0x68) {
    end();

    return 0;
  }

  if (readRegister(LSM9DS1_ADDRESS_M, LSM9DS1_WHO_AM_I) != 0x3d) {
    end();

    return 0;
  }

  writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG1_G, 0x78); // 119 Hz,
2000 dps, 16 Hz BW
  writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG6_XL, 0x70); // 119 Hz, 4G

  writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG1_M, 0xb4); //
Temperature compensation enable, medium performance, 20 Hz
  writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG2_M, 0x00); // 4 Gauss
```

```
  writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG3_M, 0x00); //
Continuous conversion mode

  return 1;
}

void LSM9DS1Class::setContinuousMode() {
  // Enable FIFO (see docs
https://www.st.com/resource/en/datasheet/DM00103319.pdf)
  writeRegister(LSM9DS1_ADDRESS, 0x23, 0x02);
  // Set continuous mode
  writeRegister(LSM9DS1_ADDRESS, 0x2E, 0xC0);

  continuousMode = true;
}

void LSM9DS1Class::setOneShotMode() {
  // Disable FIFO (see docs
https://www.st.com/resource/en/datasheet/DM00103319.pdf)
  writeRegister(LSM9DS1_ADDRESS, 0x23, 0x00);
  // Disable continuous mode
  writeRegister(LSM9DS1_ADDRESS, 0x2E, 0x00);

  continuousMode = false;
}

void LSM9DS1Class::end()
{
  writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG3_M, 0x03);
  writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG1_G, 0x00);
  writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG6_XL, 0x00);

  _wire->end();
}

int LSM9DS1Class::readAcceleration(float& x, float& y, float& z)
{
  int16_t data[3];

  if (!readRegisters(LSM9DS1_ADDRESS, LSM9DS1_OUT_X_XL, (uint8_t*)data,
sizeof(data))) {
    x = NAN;
    y = NAN;
    z = NAN;

    return 0;
  }

  x = data[0] * 4.0 / 32768.0;
  y = data[1] * 4.0 / 32768.0;
  z = data[2] * 4.0 / 32768.0;

  return 1;
}

int LSM9DS1Class::accelerationAvailable()
```

```
{
  if (continuousMode) {
    // Read FIFO_SRC. If any of the rightmost 8 bits have a value, there is
data.
    if (readRegister(LSM9DS1_ADDRESS, 0x2F) & 63) {
      return 1;
    }
  } else {
    if (readRegister(LSM9DS1_ADDRESS, LSM9DS1_STATUS_REG) & 0x01) {
      return 1;
    }
  }

  return 0;
}

float LSM9DS1Class::accelerationSampleRate()
{
  return 119.0F;
}

int LSM9DS1Class::readGyroscope(float& x, float& y, float& z)
{
  int16_t data[3];

  if (!readRegisters(LSM9DS1_ADDRESS, LSM9DS1_OUT_X_G, (uint8_t*)data,
sizeof(data))) {
    x = NAN;
    y = NAN;
    z = NAN;

    return 0;
  }

  x = data[0] * 2000.0 / 32768.0;
  y = data[1] * 2000.0 / 32768.0;
  z = data[2] * 2000.0 / 32768.0;

  return 1;
}

int LSM9DS1Class::gyroscopeAvailable()
{
  if (readRegister(LSM9DS1_ADDRESS, LSM9DS1_STATUS_REG) & 0x02) {
    return 1;
  }

  return 0;
}

float LSM9DS1Class::gyroscopeSampleRate()
{
  return 119.0F;
}

int LSM9DS1Class::readMagneticField(float& x, float& y, float& z)
```

```
{
  int16_t data[3];

  if (!readRegisters(LSM9DS1_ADDRESS_M, LSM9DS1_OUT_X_L_M, (uint8_t*)data,
sizeof(data))) {
    x = NAN;
    y = NAN;
    z = NAN;

    return 0;
  }

  x = data[0] * 4.0 * 100.0 / 32768.0;
  y = data[1] * 4.0 * 100.0 / 32768.0;
  z = data[2] * 4.0 * 100.0 / 32768.0;

  return 1;
}

int LSM9DS1Class::magneticFieldAvailable()
{
  if (readRegister(LSM9DS1_ADDRESS_M, LSM9DS1_STATUS_REG_M) & 0x08) {
    return 1;
  }

  return 0;
}

float LSM9DS1Class::magneticFieldSampleRate()
{
  return 20.0;
}

int LSM9DS1Class::readRegister(uint8_t slaveAddress, uint8_t address)
{
  _wire->beginTransmission(slaveAddress);
  _wire->write(address);
  if (_wire->endTransmission() != 0) {
    return -1;
  }

  if (_wire->requestFrom(slaveAddress, 1) != 1) {
    return -1;
  }

  return _wire->read();
}

int LSM9DS1Class::readRegisters(uint8_t slaveAddress, uint8_t address,
uint8_t* data, size_t length)
{
  _wire->beginTransmission(slaveAddress);
  _wire->write(0x80 | address);
  if (_wire->endTransmission(false) != 0) {
    return -1;
  }
```

```
  if (_wire->requestFrom(slaveAddress, length) != length) {
    return 0;
  }

  for (size_t i = 0; i < length; i++) {
    *data++ = _wire->read();
  }

  return 1;
}

int LSM9DS1Class::writeRegister(uint8_t slaveAddress, uint8_t address,
uint8_t value)
{
  _wire->beginTransmission(slaveAddress);
  _wire->write(address);
  _wire->write(value);
  if (_wire->endTransmission() != 0) {
    return 0;
  }

  return 1;
}

#ifdef ARDUINO_ARDUINO_NANO33BLE
LSM9DS1Class IMU(Wire1);
#else
LSM9DS1Class IMU(Wire);
#endif
```

5.1.3.2 Arduino BLECharacteristic

```
/*
  This file is part of the ArduinoBLE library.
  Copyright (c) 2018 Arduino SA. All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301
USA
*/

#include "BLEProperty.h"
```

```
#include "local/BLELocalCharacteristic.h"
#include "remote/BLERemoteCharacteristic.h"

#include "BLECharacteristic.h"

BLECharacteristic::BLECharacteristic() :
  BLECharacteristic((BLELocalCharacteristic*)NULL)
{
}

BLECharacteristic::BLECharacteristic(BLELocalCharacteristic* local) :
  _local(local),
  _remote(NULL)
{
  if (_local) {
    _local->retain();
  }
}

BLECharacteristic::BLECharacteristic(BLERemoteCharacteristic* remote) :
  _local(NULL),
  _remote(remote)
{
  if (_remote) {
    _remote->retain();
  }
}

BLECharacteristic::BLECharacteristic(const char* uuid, uint8_t properties,
int valueSize, bool fixedLength) :
  BLECharacteristic(new BLELocalCharacteristic(uuid, properties, valueSize,
fixedLength))
{
}

BLECharacteristic::BLECharacteristic(const char* uuid, uint8_t properties,
const char* value) :
  BLECharacteristic(new BLELocalCharacteristic(uuid, properties, value))
{
}

BLECharacteristic::BLECharacteristic(const BLECharacteristic& other)
{
  _local = other._local;
  if (_local) {
    _local->retain();
  }

  _remote = other._remote;
  if (_remote) {
    _remote->retain();
  }
}

BLECharacteristic::~BLECharacteristic()
{
```

```
  if (_local && _local->release() <= 0) {
    delete _local;
  }

  if (_remote && _remote->release() <= 0) {
    delete _remote;
  }
}

const char* BLECharacteristic::uuid() const
{
  if (_local) {
    return _local->uuid();
  }

  if (_remote) {
    return _remote->uuid();
  }

  return "";
}

uint8_t BLECharacteristic::properties() const
{
  if (_local) {
    return _local->properties();
  }

  if (_remote) {
    return _remote->properties();
  }

  return 0;
}

int BLECharacteristic::valueSize() const
{
  if (_local) {
    return _local->valueSize();
  }

  if (_remote) {
    return _remote->valueLength();
  }

  return 0;
}

const uint8_t* BLECharacteristic::value() const
{
  if (_local) {
    return _local->value();
  }

  if (_remote) {
    return _remote->value();
```

```
  }

  return NULL;
}

int BLECharacteristic::valueLength() const
{
  if (_local) {
    return _local->valueLength();
  }

  if (_remote) {
    return _remote->valueLength();
  }

  return 0;
}

uint8_t BLECharacteristic::operator[] (int offset) const
{
  if (_local) {
    return (*_local)[offset];
  }

  if (_remote) {
    return (*_remote)[offset];
  }

  return 0;
}

int BLECharacteristic::readValue(uint8_t value[], int length)
{
  int bytesRead = 0;

  if (_local) {
    bytesRead = min(length, _local->valueLength());

    memcpy(value, _local->value(), bytesRead);
  }

  if (_remote) {
    // trigger a read if the updated value (notification/indication)
    // has already been read and the characteristic is readable
    if (_remote->updatedValueRead() && canRead()) {
      if (!read()) {
        // read failed
        return 0;
      }
    }

    bytesRead = min(length, _remote->valueLength());

    memcpy(value, _remote->value(), bytesRead);
  }
```

```
  return bytesRead;
}

int BLECharacteristic::readValue(void* value, int length)
{
  return readValue((uint8_t*)value, length);
}

int BLECharacteristic::readValue(uint8_t& value)
{
  value = 0;

  return readValue((uint8_t*)&value, sizeof(value));
}

int BLECharacteristic::readValue(int8_t& value)
{
  value = 0;

  return readValue((uint8_t*)&value, sizeof(value));
}

int BLECharacteristic::readValue(uint16_t& value)
{
  value = 0;

  return readValue((uint8_t*)&value, sizeof(value));
}

int BLECharacteristic::readValue(int16_t& value)
{
  value = 0;

  return readValue((uint8_t*)&value, sizeof(value));
}

int BLECharacteristic::readValue(uint32_t& value)
{
  value = 0;

  return readValue((uint8_t*)&value, sizeof(value));
}

int BLECharacteristic::readValue(int32_t& value)
{
  value = 0;

  return readValue((uint8_t*)&value, sizeof(value));
}

int BLECharacteristic::writeValue(const uint8_t value[], int length, bool
withResponse)
{
  if (_local) {
    return _local->writeValue(value, length);
  }
```

```
  if (_remote) {
    return _remote->writeValue(value, length, withResponse);
  }

  return 0;
}

int BLECharacteristic::writeValue(const void* value, int length, bool
withResponse)
{
  return writeValue((const uint8_t*)value, length, withResponse);
}

int BLECharacteristic::writeValue(const char* value, bool withResponse)
{
  if (_local) {
    return _local->writeValue(value);
  }

  if (_remote) {
    return _remote->writeValue(value, withResponse);
  }

  return 0;
}

int BLECharacteristic::writeValue(uint8_t value, bool withResponse)
{
  return writeValue((uint8_t*)&value, sizeof(value), withResponse);
}

int BLECharacteristic::writeValue(int8_t value, bool withResponse)
{
  return writeValue((uint8_t*)&value, sizeof(value), withResponse);
}

int BLECharacteristic::writeValue(uint16_t value, bool withResponse)
{
  return writeValue((uint8_t*)&value, sizeof(value), withResponse);
}

int BLECharacteristic::writeValue(int16_t value, bool withResponse)
{
  return writeValue((uint8_t*)&value, sizeof(value), withResponse);
}

int BLECharacteristic::writeValue(uint32_t value, bool withResponse)
{
  return writeValue((uint8_t*)&value, sizeof(value), withResponse);
}

int BLECharacteristic::writeValue(int32_t value, bool withResponse)
{
  return writeValue((uint8_t*)&value, sizeof(value), withResponse);
}
```

```
int BLECharacteristic::broadcast()
{
  if (_local) {
    return _local->broadcast();
  }

  return 0;
}

bool BLECharacteristic::written()
{
  if (_local) {
    return _local->written();
  }

  return false;
}

bool BLECharacteristic::subscribed()
{
  if (_local) {
    return _local->subscribed();
  }

  return false;
}

bool BLECharacteristic::valueUpdated()
{
  if (_remote) {
    return _remote->valueUpdated();
  }

  return false;
}

void BLECharacteristic::addDescriptor(BLEDescriptor& descriptor)
{
  if (_local) {
    return _local->addDescriptor(descriptor);
  }
}

BLECharacteristic::operator bool() const
{
  return (_local != NULL) || (_remote != NULL);
}

BLELocalCharacteristic* BLECharacteristic::local()
{
  return _local;
}

void BLECharacteristic::setEventHandler(int event,
BLECharacteristicEventHandler eventHandler)
```

```
{
  if (_local) {
    _local->setEventHandler((BLECharacteristicEvent)event, eventHandler);
  }

  if (_remote) {
    _remote->setEventHandler((BLECharacteristicEvent)event, eventHandler);
  }
}

int BLECharacteristic::descriptorCount() const
{
  if (_remote) {
    return _remote->descriptorCount();
  }

  return 0;
}

bool BLECharacteristic::hasDescriptor(const char* uuid) const
{
  return hasDescriptor(uuid, 0);
}

bool BLECharacteristic::hasDescriptor(const char* uuid, int index) const
{
  if (_remote) {
    int count = 0;
    int numDescriptors = _remote->descriptorCount();

    for (int i = 0; i < numDescriptors; i++) {
      BLERemoteDescriptor* d = _remote->descriptor(i);

      if (strcasecmp(uuid, d->uuid()) == 0) {
        if (count == index) {
          return true;
        }

        count++;
      }
    }
  }

  return false;
}

BLEDescriptor BLECharacteristic::descriptor(int index) const
{
  if (_remote) {
    return BLEDescriptor(_remote->descriptor(index));
  }

  return BLEDescriptor();
}

BLEDescriptor BLECharacteristic::descriptor(const char * uuid) const
```

```
{
  return descriptor(uuid, 0);
}

BLEDescriptor BLECharacteristic::descriptor(const char * uuid, int index)
const
{
  if (_remote) {
    int count = 0;
    int numDescriptors = _remote->descriptorCount();

    for (int i = 0; i < numDescriptors; i++) {
      BLERemoteDescriptor* d = _remote->descriptor(i);

      if (strcasecmp(uuid, d->uuid()) == 0) {
        if (count == index) {
          return BLEDescriptor(d);
        }

        count++;
      }
    }
  }

  return BLEDescriptor();
}

bool BLECharacteristic::canRead()
{
  if (_remote) {
    return (properties() & BLERead) != 0;
  }

  return false;
}

bool BLECharacteristic::read()
{
  if (_remote) {
    return _remote->read();
  }

  return false;
}

bool BLECharacteristic::canWrite()
{
  if (_remote) {
    return (properties() & (BLEWrite | BLEWriteWithoutResponse)) != 0;
  }

  return false;
}

bool BLECharacteristic::canSubscribe()
{
```

```
  if (_remote) {
    return (properties() & (BLENotify | BLEIndicate)) != 0;
  }

  return false;
}

bool BLECharacteristic::subscribe()
{
  if (_remote) {
    return _remote->writeCccd((properties() & BLEIndicate) ? 0x0002 :
0x0001);
  }

  return false;
}

bool BLECharacteristic::canUnsubscribe()
{
  return canSubscribe();
}

bool BLECharacteristic::unsubscribe()
{
  if (_remote) {
    return _remote->writeCccd(0x0000);
  }

  return false;
}
```

5.1.3.3 Arduino HTS221

```
/*
  This file is part of the Arduino_HTS221 library.
  Copyright (c) 2019 Arduino SA. All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301
USA
*/

#include <Wire.h>
```

```
#include "HTS.h"

#define HTS221_ADDRESS   0x5F

#define HTS221_WHO_AM_I_REG          0x0f
#define HTS221_CTRL1_REG             0x20
#define HTS221_CTRL2_REG             0x21
#define HTS221_STATUS_REG            0x27
#define HTS221_HUMIDITY_OUT_L_REG    0x28
#define HTS221_TEMP_OUT_L_REG        0x2a
#define HTS221_H0_rH_x2_REG          0x30
#define HTS221_H1_rH_x2_REG          0x31
#define HTS221_T0_degC_x8_REG        0x32
#define HTS221_T1_degC_x8_REG        0x33
#define HTS221_T1_T0_MSB_REG         0x35
#define HTS221_H0_T0_OUT_REG         0x36
#define HTS221_H1_T0_OUT_REG         0x3a
#define HTS221_T0_OUT_REG            0x3c
#define HTS221_T1_OUT_REG            0x3e

HTS221Class::HTS221Class(TwoWire& wire) :
  _wire(&wire)
{
}

int HTS221Class::begin()
{
  _wire->begin();

  if (i2cRead(HTS221_WHO_AM_I_REG) != 0xbc) {
    end();

    return 0;
  }

  readHTS221Calibration();

  // enable HTS221
  i2cWrite(HTS221_CTRL1_REG, 0x80);

  return 1;
}

void HTS221Class::end()
{
  // disable HTS221
  i2cWrite(HTS221_CTRL1_REG, 0x00);

  _wire->end();
}

float HTS221Class::readTemperature(int units)
{
  // trigger one shot
  i2cWrite(HTS221_CTRL2_REG, 0x01);
```

```
  // wait for completion
  while ((i2cRead(HTS221_STATUS_REG) & 0x01) == 0) {
    yield();
  }

  // read value and convert
  int16_t tout = i2cRead16(HTS221_TEMP_OUT_L_REG);
  float reading = (tout * _hts221TemperatureSlope +
_hts221TemperatureZero);
  if (units == FAHRENHEIT) { // Fahrenheit = (Celsius * 9 / 5) + 32
    return (reading * 9.0 / 5.0) + 32.0;
  } else {
    return reading;
  }
}

float HTS221Class::readHumidity()
{
  // trigger one shot
  i2cWrite(HTS221_CTRL2_REG, 0x01);

  // wait for completion
  while ((i2cRead(HTS221_STATUS_REG) & 0x02) == 0) {
    yield();
  }

  // read value and convert
  int16_t hout = i2cRead16(HTS221_HUMIDITY_OUT_L_REG);

  return (hout * _hts221HumiditySlope + _hts221HumidityZero);
}

int HTS221Class::i2cRead(uint8_t reg)
{
  _wire->beginTransmission(HTS221_ADDRESS);
  _wire->write(reg);
  if (_wire->endTransmission(false) != 0) {
    return -1;
  }

  if (_wire->requestFrom(HTS221_ADDRESS, 1) != 1) {
    return -1;
  }

  return _wire->read();
}

int HTS221Class::i2cWrite(uint8_t reg, uint8_t val)
{
  _wire->beginTransmission(HTS221_ADDRESS);
  _wire->write(reg);
  _wire->write(val);
  if (_wire->endTransmission() != 0) {
    return 0;
  }
```

```
  return 1;
}

void HTS221Class::readHTS221Calibration()
{
  uint8_t h0rH = i2cRead(HTS221_H0_rH_x2_REG);
  uint8_t h1rH = i2cRead(HTS221_H1_rH_x2_REG);

  uint16_t t0degC = i2cRead(HTS221_T0_degC_x8_REG) |
((i2cRead(HTS221_T1_T0_MSB_REG) & 0x03) << 8);
  uint16_t t1degC = i2cRead(HTS221_T1_degC_x8_REG) |
((i2cRead(HTS221_T1_T0_MSB_REG) & 0x0c) << 6);

  int16_t h0t0Out = i2cRead16(HTS221_H0_T0_OUT_REG);
  int16_t h1t0Out = i2cRead16(HTS221_H1_T0_OUT_REG);

  int16_t t0Out = i2cRead16(HTS221_T0_OUT_REG);
  int16_t t1Out = i2cRead16(HTS221_T1_OUT_REG);

  // calculate slopes and 0 offset from calibration values,
  // for future calculations: value = a * X + b

  _hts221HumiditySlope = (h1rH - h0rH) / (2.0 * (h1t0Out - h0t0Out));
  _hts221HumidityZero = (h0rH / 2.0) - _hts221HumiditySlope * h0t0Out;

  _hts221TemperatureSlope = (t1degC - t0degC) / (8.0 * (t1Out - t0Out));
  _hts221TemperatureZero = (t0degC / 8.0) - _hts221TemperatureSlope *
t0Out;
}

#ifdef ARDUINO_ARDUINO_NANO33BLE
HTS221Class HTS(Wire1);
#else
HTS221Class HTS(Wire);
#endif
```

5.1.4 Librerías IA generadas por Edge

5.1.4.1 Ei_classifier_config

```
/* Edge Impulse inferencing library
 * Copyright (c) 2021 EdgeImpulse Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
copy
 * of this software and associated documentation files (the "Software"), to
deal
 * in the Software without restriction, including without limitation the
rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE
 * SOFTWARE.
 */

#ifndef _EI_CLASSIFIER_CONFIG_H_
#define _EI_CLASSIFIER_CONFIG_H_

// clang-format off
#ifndef EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN
#if defined(__MBED__)
    #include "mbed.h"
    #if (MBED_VERSION < MBED_ENCODE_VERSION(5, 7, 0))
        #define EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN      0
    #else
        #define EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN      1
    #endif // Mbed OS 5.7 version check
#elif defined(__TARGET_CPU_CORTEX_M0) ||
defined(__TARGET_CPU_CORTEX_M0PLUS) || defined(__TARGET_CPU_CORTEX_M3) ||
defined(__TARGET_CPU_CORTEX_M4) || defined(__TARGET_CPU_CORTEX_M7) ||
defined(ARDUINO_NRF52_ADAFRUIT)
    #define EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN      1
#else
    #define EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN      0
#endif
```

```
#endif // EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN

// CMSIS-NN falls back to reference kernels when __ARM_FEATURE_DSP and
__ARM_FEATURE_MVE are not defined
// we should never use those... So disable CMSIS-NN in that case and throw
a warning
#if EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN == 1
    #if !defined(__ARM_FEATURE_DSP) && !defined(__ARM_FEATURE_MVE)
        #pragma message( \
            "CMSIS-NN enabled, but neither __ARM_FEATURE_DSP nor
__ARM_FEATURE_MVE defined. Falling back.")
        #undef EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN
        #define EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN 0
    #endif
#endif // EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN == 1

#if EI_CLASSIFIER_TFLITE_ENABLE_CMSIS_NN == 1
#define CMSIS_NN                    1
#endif

#ifndef EI_CLASSIFIER_TFLITE_ENABLE_ARC
#ifdef CPU_ARC
#define EI_CLASSIFIER_TFLITE_ENABLE_ARC             1
#else
#define EI_CLASSIFIER_TFLITE_ENABLE_ARC             0
#endif // CPU_ARC
#endif // EI_CLASSIFIER_TFLITE_ENABLE_ARC

// clang-format on
#endif // _EI_CLASSIFIER_CONFIG_H_
```

5.1.4.2 Ei_run_classifier

```
/* Edge Impulse inferencing library
 * Copyright (c) 2021 EdgeImpulse Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
copy
 * of this software and associated documentation files (the "Software"), to
deal
 * in the Software without restriction, including without limitation the
rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

```
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE
 * SOFTWARE.
 */

#ifndef _EDGE_IMPULSE_RUN_CLASSIFIER_H_
#define _EDGE_IMPULSE_RUN_CLASSIFIER_H_

#include "model-parameters/model_metadata.h"
#if EI_CLASSIFIER_HAS_ANOMALY == 1
#include "model-parameters/anomaly_clusters.h"
#endif
#include "ei_run_dsp.h"
#include "ei_classifier_types.h"
#include "ei_classifier_smooth.h"
#if defined(EI_CLASSIFIER_HAS_SAMPLER) && EI_CLASSIFIER_HAS_SAMPLER == 1
#include "ei_sampler.h"
#endif
#include "edge-impulse-sdk/porting/ei_classifier_porting.h"
#include "model-parameters/dsp_blocks.h"

#if (EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE) &&
(EI_CLASSIFIER_COMPILED != 1)
#include <cmath>
#include "edge-impulse-sdk/tensorflow/lite/micro/all_ops_resolver.h"
#include "edge-impulse-sdk/tensorflow/lite/micro/micro_error_reporter.h"
#include "edge-impulse-sdk/tensorflow/lite/micro/micro_interpreter.h"
#include "edge-impulse-sdk/tensorflow/lite/schema/schema_generated.h"
#include "edge-impulse-sdk/tensorflow/lite/version.h"
#include "edge-impulse-sdk/classifier/ei_aligned_malloc.h"

#include "tflite-model/tflite-trained.h"
#if defined(EI_CLASSIFIER_HAS_TFLITE_OPS_RESOLVER) &&
EI_CLASSIFIER_HAS_TFLITE_OPS_RESOLVER == 1
#include "tflite-model/tflite-resolver.h"
#endif // EI_CLASSIFIER_HAS_TFLITE_OPS_RESOLVER

#if defined(EI_CLASSIFIER_ENABLE_DETECTION_POSTPROCESS_OP)
namespace tflite {
namespace ops {
namespace micro {
extern TfLiteRegistration *Register_TFLite_Detection_PostProcess(void);
}  // namespace micro
}  // namespace ops

extern float post_process_boxes[10 * 4 * sizeof(float)];
extern float post_process_classes[10];
extern float post_process_scores[10];

}  // namespace tflite
#endif
```

```
static tflite::MicroErrorReporter micro_error_reporter;
static tflite::ErrorReporter* error_reporter = &micro_error_reporter;
#elif EI_CLASSIFIER_COMPILED == 1
#include "edge-impulse-sdk/tensorflow/lite/c/common.h"
#include "edge-impulse-
sdk/tensorflow/lite/kernels/internal/tensor_ctypes.h"
#include "tflite-model/trained_model_compiled.h"
#include "edge-impulse-sdk/classifier/ei_aligned_malloc.h"

namespace tflite {
extern float post_process_boxes[10 * 4 * sizeof(float)];
extern float post_process_classes[10];
extern float post_process_scores[10];
} // namespace tflite

#elif EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE_FULL

#include "tensorflow/lite/c/common.h"
#include "tensorflow/lite/interpreter.h"
#include "tensorflow/lite/kernels/register.h"
#include "tensorflow/lite/model.h"
#include "tensorflow/lite/optional_debug_tools.h"
#include "tflite-model/tflite-trained.h"

#elif EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_NONE
// noop
#else
#error "Unknown inferencing engine"
#endif

#if ECM3532
void*    __dso_handle = (void*) &__dso_handle;
#endif

#ifdef __cplusplus
namespace {
#endif // __cplusplus

/* Function prototypes ---------------------------------------------------
- */
extern "C" EI_IMPULSE_ERROR run_inference(ei::matrix_t *fmatrix,
ei_impulse_result_t *result, bool debug);
extern "C" EI_IMPULSE_ERROR run_classifier_image_quantized(signal_t
*signal, ei_impulse_result_t *result, bool debug);
static EI_IMPULSE_ERROR can_run_classifier_image_quantized();
static void calc_cepstral_mean_and_var_normalization_mfcc(ei_matrix
*matrix, void *config_ptr);
static void calc_cepstral_mean_and_var_normalization_mfe(ei_matrix *matrix,
void *config_ptr);
static void calc_cepstral_mean_and_var_normalization_spectrogram(ei_matrix
*matrix, void *config_ptr);

/* Private variables -----------------------------------------------------
- */
#if EI_CLASSIFIER_LABEL_COUNT > 0
```

```
ei_impulse_maf classifier_maf[EI_CLASSIFIER_LABEL_COUNT] = {{0}};
#else
ei_impulse_maf classifier_maf[0];
#endif
static size_t slice_offset = 0;
static bool feature_buffer_full = false;

/* Private functions ------------------------------------------------------
- */

/**
 * @brief      Run a moving average filter over the classification result.
 *             The size of the filter determines the response of the
filter.
 *             It is now set to the number of slices per window.
 * @param      maf             Pointer to maf object
 * @param[in]  classification  Classification output on current slice
 *
 * @return     Averaged classification value
 */
extern "C" float run_moving_average_filter(ei_impulse_maf *maf, float
classification)
{
    maf->running_sum -= maf->maf_buffer[maf->buf_idx];
    maf->running_sum += classification;
    maf->maf_buffer[maf->buf_idx] = classification;

    if (++maf->buf_idx >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW >> 1)) {
        maf->buf_idx = 0;
    }

#if (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW > 1)
    return maf->running_sum / (float)(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW
>> 1);
#else
    return maf->running_sum;
#endif
}

/**
 * @brief      Reset all values in filter to 0
 *
 * @param      maf   Pointer to maf object
 */
static void clear_moving_average_filter(ei_impulse_maf *maf)
{
    maf->running_sum = 0;

    for (int i = 0; i < (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW >> 1); i++)
{
        maf->maf_buffer[i] = 0.f;
    }
}

/**
 * @brief      Init static vars
```

```
 */
extern "C" void run_classifier_init(void)
{
    slice_offset = 0;
    feature_buffer_full = false;

    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        clear_moving_average_filter(&classifier_maf[ix]);
    }
}

/**
 * @brief      Fill the complete matrix with sample slices. From there, run
inference
 *             on the matrix.
 *
 * @param      signal  Sample data
 * @param      result  Classification output
 * @param[in]  debug   Debug output enable boot
 * @param      enable_maf Enables the moving average filter
 *
 * @return     The ei impulse error.
 */
extern "C" EI_IMPULSE_ERROR run_classifier_continuous(signal_t *signal,
ei_impulse_result_t *result,
                                                      bool debug = false,
bool enable_maf = true)
{
    static ei::matrix_t static_features_matrix(1,
EI_CLASSIFIER_NN_INPUT_FRAME_SIZE);
    if (!static_features_matrix.buffer) {
        return EI_IMPULSE_ALLOC_FAILED;
    }

    EI_IMPULSE_ERROR ei_impulse_error = EI_IMPULSE_OK;

    uint64_t dsp_start_ms = ei_read_timer_ms();

    size_t out_features_index = 0;
    size_t feature_size;
    bool is_mfcc = false;
    bool is_mfe = false;
    bool is_spectrogram = false;

    for (size_t ix = 0; ix < ei_dsp_blocks_size; ix++) {
        ei_model_dsp_t block = ei_dsp_blocks[ix];

        if (out_features_index + block.n_output_features >
EI_CLASSIFIER_NN_INPUT_FRAME_SIZE) {
            ei_printf("ERR: Would write outside feature buffer\n");
            return EI_IMPULSE_DSP_ERROR;
        }

        ei::matrix_t fm(1, block.n_output_features,
                        static_features_matrix.buffer + out_features_index
+ slice_offset);
```

```
        /* Switch to the slice version of the mfcc feature extract function
*/
        if (block.extract_fn == extract_mfcc_features) {
            block.extract_fn = &extract_mfcc_per_slice_features;
            is_mfcc = true;
        }
        else if (block.extract_fn == extract_spectrogram_features) {
            block.extract_fn = &extract_spectrogram_per_slice_features;
            is_spectrogram = true;
        }
        else if (block.extract_fn == extract_mfe_features) {
            block.extract_fn = &extract_mfe_per_slice_features;
            is_mfe = true;
        }
        else {
            ei_printf("ERR: Unknown extract function, only MFCC, MFE and
spectrogram supported\n");
            return EI_IMPULSE_DSP_ERROR;
        }

        int ret = block.extract_fn(signal, &fm, block.config,
EI_CLASSIFIER_FREQUENCY);
        if (ret != EIDSP_OK) {
            ei_printf("ERR: Failed to run DSP process (%d)\n", ret);
            return EI_IMPULSE_DSP_ERROR;
        }

        if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
            return EI_IMPULSE_CANCELED;
        }

        out_features_index += block.n_output_features;

        feature_size = (fm.rows * fm.cols);
    }

    /* For as long as the feature buffer isn't completely full, keep moving
the slice offset */
    if (feature_buffer_full == false) {
        slice_offset += feature_size;

        if (slice_offset > (EI_CLASSIFIER_NN_INPUT_FRAME_SIZE -
feature_size)) {
            feature_buffer_full = true;
            slice_offset -= feature_size;
        }
    }

    result->timing.dsp = ei_read_timer_ms() - dsp_start_ms;

    if (debug) {
        ei_printf("\r\nFeatures (%d ms.): ", result->timing.dsp);
        for (size_t ix = 0; ix < static_features_matrix.cols; ix++) {
            ei_printf_float(static_features_matrix.buffer[ix]);
            ei_printf(" ");
```

```
        }
        ei_printf("\n");
    }

#if EI_CLASSIFIER_INFERENCING_ENGINE != EI_CLASSIFIER_NONE
    if (debug) {
        ei_printf("Running neural network...\n");
    }
#endif

    if (feature_buffer_full == true) {
        dsp_start_ms = ei_read_timer_ms();
        ei::matrix_t classify_matrix(1, EI_CLASSIFIER_NN_INPUT_FRAME_SIZE);

        /* Create a copy of the matrix for normalization */
        for (size_t m_ix = 0; m_ix < EI_CLASSIFIER_NN_INPUT_FRAME_SIZE;
m_ix++) {
            classify_matrix.buffer[m_ix] =
static_features_matrix.buffer[m_ix];
        }

        if (is_mfcc) {
            calc_cepstral_mean_and_var_normalization_mfcc(&classify_matrix,
ei_dsp_blocks[0].config);
        }
        else if (is_spectrogram) {

calc_cepstral_mean_and_var_normalization_spectrogram(&classify_matrix,
ei_dsp_blocks[0].config);
        }
        else if (is_mfe) {
            calc_cepstral_mean_and_var_normalization_mfe(&classify_matrix,
ei_dsp_blocks[0].config);
        }
        result->timing.dsp += ei_read_timer_ms() - dsp_start_ms;

        ei_impulse_error = run_inference(&classify_matrix, result, debug);

        if (enable_maf) {
            for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    #if EI_CLASSIFIER_OBJECT_DETECTION != 1
                result->classification[ix].value =
                    run_moving_average_filter(&classifier_maf[ix], result-
>classification[ix].value);
    #endif
            }
        }

        /* Shift the feature buffer for new data */
        for (size_t i = 0; i < (EI_CLASSIFIER_NN_INPUT_FRAME_SIZE -
feature_size); i++) {
            static_features_matrix.buffer[i] =
static_features_matrix.buffer[i + feature_size];
        }
    }
    return ei_impulse_error;
```

```
}

#if EI_CLASSIFIER_OBJECT_DETECTION

/**
 * Fill the result structure from an unquantized output tensor
 * (we don't support quantized here a.t.m.)
 */
static void fill_result_struct_f32(ei_impulse_result_t *result, float
*data, float *scores, float *labels, bool debug) {
    for (size_t ix = 0; ix < EI_CLASSIFIER_OBJECT_DETECTION_COUNT; ix++) {

        float score = scores[ix];
        float label = labels[ix];

#if EI_CLASSIFIER_TFLITE_INPUT_DATATYPE == EI_CLASSIFIER_DATATYPE_INT8
        // so for i8 inputs this seems to be 0.5..1.0 instead of 0.0..1.0
        // let's fix it by hand
        score = (score - 0.5f) * 2;
#endif

        if (score >= EI_CLASSIFIER_OBJECT_DETECTION_THRESHOLD) {
            float ystart = data[(ix * 4) + 0];
            float xstart = data[(ix * 4) + 1];
            float yend = data[(ix * 4) + 2];
            float xend = data[(ix * 4) + 3];

            if (xstart < 0) xstart = 0;
            if (xstart > 1) xstart = 1;
            if (ystart < 0) ystart = 0;
            if (ystart > 1) ystart = 1;
            if (yend < 0) yend = 0;
            if (yend > 1) yend = 1;
            if (xend < 0) xend = 0;
            if (xend > 1) xend = 1;
            if (xend < xstart) xend = xstart;
            if (yend < ystart) yend = ystart;

            if (debug) {
                ei_printf("%s (%f): %f [ %f, %f, %f, %f ]\n",
                    ei_classifier_inferencing_categories[(uint32_t)label],
label, score, xstart, ystart, xend, yend);
            }

            result->bounding_boxes[ix].label =
ei_classifier_inferencing_categories[(uint32_t)label];
            result->bounding_boxes[ix].x = static_cast<uint32_t>(xstart *
static_cast<float>(EI_CLASSIFIER_INPUT_WIDTH));
            result->bounding_boxes[ix].y = static_cast<uint32_t>(ystart *
static_cast<float>(EI_CLASSIFIER_INPUT_HEIGHT));
            result->bounding_boxes[ix].width = static_cast<uint32_t>((xend
- xstart) * static_cast<float>(EI_CLASSIFIER_INPUT_WIDTH));
            result->bounding_boxes[ix].height = static_cast<uint32_t>((yend
- ystart) * static_cast<float>(EI_CLASSIFIER_INPUT_HEIGHT));
            result->bounding_boxes[ix].value = score;
        }
```

```
        else {
            result->bounding_boxes[ix].value = 0.0f;
        }
    }
}

#else

/**
 * Fill the result structure from a quantized output tensor
 */
static void fill_result_struct_i8(ei_impulse_result_t *result, int8_t
*data, float zero_point, float scale, bool debug) {
    for (uint32_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        float value = static_cast<float>(data[ix] - zero_point) * scale;

        if (debug) {
            ei_printf("%s:\t", ei_classifier_inferencing_categories[ix]);
            ei_printf_float(value);
            ei_printf("\n");
        }
        result->classification[ix].label =
ei_classifier_inferencing_categories[ix];
        result->classification[ix].value = value;
    }
}

/**
 * Fill the result structure from an unquantized output tensor
 */
static void fill_result_struct_f32(ei_impulse_result_t *result, float
*data, bool debug) {
    for (uint32_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        float value = data[ix];

        if (debug) {
            ei_printf("%s:\t", ei_classifier_inferencing_categories[ix]);
            ei_printf_float(value);
            ei_printf("\n");
        }
        result->classification[ix].label =
ei_classifier_inferencing_categories[ix];
        result->classification[ix].value = value;
    }
}

#endif // EI_CLASSIFIER_OBJECT_DETECTION

#if (EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE)

/**
 * Setup the TFLite runtime
 *
 * @param      ctx_start_ms       Pointer to the start time
 * @param      input              Pointer to input tensor
 * @param      output             Pointer to output tensor
```

```
 * @param      micro_interpreter  Pointer to interpreter (for non-compiled
models)
 * @param      micro_tensor_arena Pointer to the arena that will be
allocated
 *
 * @return  EI_IMPULSE_OK if successful
 */
static EI_IMPULSE_ERROR inference_tflite_setup(uint64_t *ctx_start_ms,
TfLiteTensor** input, TfLiteTensor** output,
#if EI_CLASSIFIER_OBJECT_DETECTION
    TfLiteTensor** output_labels,
    TfLiteTensor** output_scores,
#endif
#if (EI_CLASSIFIER_COMPILED != 1)
    tflite::MicroInterpreter** micro_interpreter,
#endif
    uint8_t** micro_tensor_arena) {
#if (EI_CLASSIFIER_COMPILED == 1)
    TfLiteStatus init_status = trained_model_init(ei_aligned_malloc);
    if (init_status != kTfLiteOk) {
        ei_printf("Failed to allocate TFLite arena (error code %d)\n",
init_status);
        return EI_IMPULSE_TFLITE_ARENA_ALLOC_FAILED;
    }
#else
    // Create an area of memory to use for input, output, and intermediate
arrays.
    uint8_t *tensor_arena = (uint8_t*)ei_aligned_malloc(16,
EI_CLASSIFIER_TFLITE_ARENA_SIZE);
    if (tensor_arena == NULL) {
        ei_printf("Failed to allocate TFLite arena (%d bytes)\n",
EI_CLASSIFIER_TFLITE_ARENA_SIZE);
        return EI_IMPULSE_TFLITE_ARENA_ALLOC_FAILED;
    }
    *micro_tensor_arena = tensor_arena;
#endif

    *ctx_start_ms = ei_read_timer_ms();

    static bool tflite_first_run = true;

#if (EI_CLASSIFIER_COMPILED != 1)
        static const tflite::Model* model = nullptr;
#endif

#if (EI_CLASSIFIER_COMPILED != 1)
    // ======
    // Initialization code start
    // This part can be run once, but that would require the TFLite arena
    // to be allocated at all times, which is not ideal (e.g. when doing
MFCC)
    // ======
    if (tflite_first_run) {
        // Map the model into a usable data structure. This doesn't involve
any
        // copying or parsing, it's a very lightweight operation.
```

```
        model = tflite::GetModel(trained_tflite);
        if (model->version() != TFLITE_SCHEMA_VERSION) {
            error_reporter->Report(
                "Model provided is schema version %d not equal "
                "to supported version %d.",
                model->version(), TFLITE_SCHEMA_VERSION);
            ei_aligned_free(tensor_arena);
            return EI_IMPULSE_TFLITE_ERROR;
        }
    }
#endif

#if (EI_CLASSIFIER_COMPILED != 1)
#ifdef EI_TFLITE_RESOLVER
    EI_TFLITE_RESOLVER
#else
    tflite::AllOpsResolver resolver;
#endif
#if defined(EI_CLASSIFIER_ENABLE_DETECTION_POSTPROCESS_OP)
    resolver.AddCustom("TFLite_Detection_PostProcess",
tflite::ops::micro::Register_TFLite_Detection_PostProcess());
#endif
#endif // EI_CLASSIFIER_COMPILED != 1

#if (EI_CLASSIFIER_COMPILED == 1)
    *input = trained_model_input(0);
    *output = trained_model_output(0);
#if EI_CLASSIFIER_OBJECT_DETECTION
    *output_scores =
trained_model_output(EI_CLASSIFIER_TFLITE_OUTPUT_SCORE_TENSOR);
    *output_labels =
trained_model_output(EI_CLASSIFIER_TFLITE_OUTPUT_LABELS_TENSOR);
#endif // EI_CLASSIFIER_OBJECT_DETECTION
#else
    // Build an interpreter to run the model with.
    tflite::MicroInterpreter *interpreter = new tflite::MicroInterpreter(
        model, resolver, tensor_arena, EI_CLASSIFIER_TFLITE_ARENA_SIZE,
error_reporter);

    *micro_interpreter = interpreter;

    // Allocate memory from the tensor_arena for the model's tensors.
    TfLiteStatus allocate_status = interpreter->AllocateTensors();
    if (allocate_status != kTfLiteOk) {
        error_reporter->Report("AllocateTensors() failed");
        ei_aligned_free(tensor_arena);
        return EI_IMPULSE_TFLITE_ERROR;
    }

    // Obtain pointers to the model's input and output tensors.
    *input = interpreter->input(0);
    *output = interpreter->output(0);
#if EI_CLASSIFIER_OBJECT_DETECTION
    *output_scores = interpreter-
>output(EI_CLASSIFIER_TFLITE_OUTPUT_SCORE_TENSOR);
```

```
    *output_labels = interpreter-
>output(EI_CLASSIFIER_TFLITE_OUTPUT_LABELS_TENSOR);
#endif // EI_CLASSIFIER_OBJECT_DETECTION
#endif

    // Assert that our quantization parameters match the model
    if (tflite_first_run) {
        assert((*input)->type == EI_CLASSIFIER_TFLITE_INPUT_DATATYPE);
        assert((*output)->type == EI_CLASSIFIER_TFLITE_OUTPUT_DATATYPE);
#if EI_CLASSIFIER_OBJECT_DETECTION
        assert((*output_scores)->type ==
EI_CLASSIFIER_TFLITE_OUTPUT_DATATYPE);
        assert((*output_labels)->type ==
EI_CLASSIFIER_TFLITE_OUTPUT_DATATYPE);
#endif
#if defined(EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED) ||
defined(EI_CLASSIFIER_TFLITE_OUTPUT_QUANTIZED)
        if (EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED) {
            assert((*input)->params.scale ==
EI_CLASSIFIER_TFLITE_INPUT_SCALE);
            assert((*input)->params.zero_point ==
EI_CLASSIFIER_TFLITE_INPUT_ZEROPOINT);
        }
        if (EI_CLASSIFIER_TFLITE_OUTPUT_QUANTIZED) {
            assert((*output)->params.scale ==
EI_CLASSIFIER_TFLITE_OUTPUT_SCALE);
            assert((*output)->params.zero_point ==
EI_CLASSIFIER_TFLITE_OUTPUT_ZEROPOINT);
        }
#endif
        tflite_first_run = false;
    }
    return EI_IMPULSE_OK;
}

/**
 * Run TFLite model
 *
 * @param   ctx_start_ms    Start time of the setup function (see above)
 * @param   output          Output tensor
 * @param   interpreter     TFLite interpreter (non-compiled models)
 * @param   tensor_arena    Allocated arena (will be freed)
 * @param   result          Struct for results
 * @param   debug           Whether to print debug info
 *
 * @return  EI_IMPULSE_OK if successful
 */
static EI_IMPULSE_ERROR inference_tflite_run(uint64_t ctx_start_ms,
    TfLiteTensor* output,
#if EI_CLASSIFIER_OBJECT_DETECTION
    TfLiteTensor* labels_tensor,
    TfLiteTensor* scores_tensor,
#endif
#if (EI_CLASSIFIER_COMPILED != 1)
    tflite::MicroInterpreter* interpreter,
#endif
```

```
     uint8_t* tensor_arena,
    ei_impulse_result_t *result,
    bool debug) {
#if (EI_CLASSIFIER_COMPILED == 1)
    trained_model_invoke();
#else
    // Run inference, and report any error
    TfLiteStatus invoke_status = interpreter->Invoke();
    if (invoke_status != kTfLiteOk) {
        error_reporter->Report("Invoke failed (%d)\n", invoke_status);
        ei_aligned_free(tensor_arena);
        return EI_IMPULSE_TFLITE_ERROR;
    }
    delete interpreter;
#endif

    uint64_t ctx_end_ms = ei_read_timer_ms();

    result->timing.classification = ctx_end_ms - ctx_start_ms;

    // Read the predicted y value from the model's output tensor
    if (debug) {
        ei_printf("Predictions (time: %d ms.):\n", result-
>timing.classification);
    }
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
    fill_result_struct_f32(result, tflite::post_process_boxes,
tflite::post_process_scores, tflite::post_process_classes, debug);
    // fill_result_struct_f32(result, output->data.f, scores_tensor-
>data.f, labels_tensor->data.f, debug);
#else
    bool int8_output = output->type == TfLiteType::kTfLiteInt8;
    if (int8_output) {
        fill_result_struct_i8(result, output->data.int8, output-
>params.zero_point, output->params.scale, debug);
    }
    else {
        fill_result_struct_f32(result, output->data.f, debug);
    }logo
#endif

#if (EI_CLASSIFIER_COMPILED == 1)
    trained_model_reset(ei_aligned_free);
#else
    ei_aligned_free(tensor_arena);
#endif

    if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
        return EI_IMPULSE_CANCELED;
    }

    return EI_IMPULSE_OK;
}
#endif // (EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE)

/**
```

```
 * @brief      Do inferencing over the processed feature matrix
 *
 * @param      fmatrix  Processed matrix
 * @param      result   Output classifier results
 * @param[in]  debug    Debug output enable
 *
 * @return     The ei impulse error.
 */
extern "C" EI_IMPULSE_ERROR run_inference(
    ei::matrix_t *fmatrix,
    ei_impulse_result_t *result,
    bool debug = false)
{
#if (EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE)
    {
        uint64_t ctx_start_ms;
        TfLiteTensor* input;
        TfLiteTensor* output;
#if EI_CLASSIFIER_OBJECT_DETECTION
        TfLiteTensor* output_scores;
        TfLiteTensor* output_labels;
#endif
        uint8_t* tensor_arena;

#if (EI_CLASSIFIER_COMPILED == 1)
        EI_IMPULSE_ERROR init_res = inference_tflite_setup(&ctx_start_ms,
&input, &output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
            &output_labels,
            &output_scores,
    #endif
            &tensor_arena);
#else
        tflite::MicroInterpreter* interpreter;
        EI_IMPULSE_ERROR init_res = inference_tflite_setup(&ctx_start_ms,
&input, &output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
            &output_labels,
            &output_scores,
    #endif
            &interpreter, &tensor_arena);
#endif
        if (init_res != EI_IMPULSE_OK) {
            return init_res;
        }

        // Place our calculated x value in the model's input tensor
#if EI_CLASSIFIER_OBJECT_DETECTION
        bool uint8_input = input->type == TfLiteType::kTfLiteUInt8;
        for (size_t ix = 0; ix < fmatrix->rows * fmatrix->cols; ix++) {
            if (uint8_input) {
                float pixel = (float)fmatrix->buffer[ix];
                input->data.uint8[ix] = static_cast<uint8_t>((pixel /
EI_CLASSIFIER_TFLITE_INPUT_SCALE) + EI_CLASSIFIER_TFLITE_INPUT_ZEROPOINT);
            }
            else {
```

```
                input->data.f[ix] = fmatrix->buffer[ix];
            }
        }
#else
        bool int8_input = input->type == TfLiteType::kTfLiteInt8;
        for (size_t ix = 0; ix < fmatrix->rows * fmatrix->cols; ix++) {
            // Quantize the input if it is int8
            if (int8_input) {
                input->data.int8[ix] = static_cast<int8_t>(round(fmatrix-
>buffer[ix] / input->params.scale) + input->params.zero_point);
                // printf("float %ld : %d\r\n", ix, input->data.int8[ix]);
            } else {
                input->data.f[ix] = fmatrix->buffer[ix];
            }
        }
#endif

#if (EI_CLASSIFIER_COMPILED == 1)
        EI_IMPULSE_ERROR run_res = inference_tflite_run(ctx_start_ms,
output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
            output_labels,
            output_scores,
    #endif
            tensor_arena, result, debug);
#else
        EI_IMPULSE_ERROR run_res = inference_tflite_run(ctx_start_ms,
output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
            output_labels,
            output_scores,
    #endif
            interpreter, tensor_arena, result, debug);
#endif

        if (run_res != EI_IMPULSE_OK) {
            return run_res;
        }
    }

#elif EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE_FULL

    {
        static std::unique_ptr<tflite::FlatBufferModel> model = nullptr;
        static std::unique_ptr<tflite::Interpreter> interpreter = nullptr;
        if (!model) {
            model = tflite::FlatBufferModel::BuildFromBuffer((const
char*)trained_tflite, trained_tflite_len);
            if (!model) {
                ei_printf("Failed to build TFLite model from buffer\n");
                return EI_IMPULSE_TFLITE_ERROR;
            }

            tflite::ops::builtin::BuiltinOpResolver resolver;
            tflite::InterpreterBuilder builder(*model, resolver);
            builder(&interpreter);
```

```
            if (!interpreter) {
                ei_printf("Failed to construct interpreter\n");
                return EI_IMPULSE_TFLITE_ERROR;
            }

            if (interpreter->AllocateTensors() != kTfLiteOk) {
                ei_printf("AllocateTensors failed\n");
                return EI_IMPULSE_TFLITE_ERROR;
            }
        }

        // Obtain pointers to the model's input and output tensors.
    #if EI_CLASSIFIER_OBJECT_DETECTION == 1
        #if EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1
            int8_t* input = interpreter->typed_input_tensor<int8_t>(0);
        #else
            float* input = interpreter->typed_input_tensor<float>(0);
        #endif
    #elif EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1
        int8_t* input = interpreter->typed_input_tensor<int8_t>(0);
    #else
        float* input = interpreter->typed_input_tensor<float>(0);
    #endif

        if (!input) {
            return EI_IMPULSE_INPUT_TENSOR_WAS_NULL;
        }

        for (uint32_t ix = 0; ix < fmatrix->rows * fmatrix->cols; ix++) {
    #if EI_CLASSIFIER_OBJECT_DETECTION == 1
        #if EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1
            float pixel = (float)fmatrix->buffer[ix];
            input[ix] = static_cast<uint8_t>((pixel /
EI_CLASSIFIER_TFLITE_INPUT_SCALE) + EI_CLASSIFIER_TFLITE_INPUT_ZEROPOINT);
        #else
            input[ix] = fmatrix->buffer[ix];
        #endif
    #elif EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1
            input[ix] = static_cast<int8_t>(round(fmatrix->buffer[ix] /
EI_CLASSIFIER_TFLITE_INPUT_SCALE) + EI_CLASSIFIER_TFLITE_INPUT_ZEROPOINT);
    #else
            input[ix] = fmatrix->buffer[ix];
    #endif
        }

        uint64_t ctx_start_ms = ei_read_timer_ms();

        interpreter->Invoke();

        uint64_t ctx_end_ms = ei_read_timer_ms();

        result->timing.classification = ctx_end_ms - ctx_start_ms;
    #if EI_CLASSIFIER_TFLITE_OUTPUT_QUANTIZED == 1
        int8_t* out_data = interpreter->typed_output_tensor<int8_t>(0);
    #else
```

```
        float* out_data = interpreter->typed_output_tensor<float>(0);
    #endif

        if (!out_data) {
            return EI_IMPULSE_OUTPUT_TENSOR_WAS_NULL;
        }

        if (debug) {
            ei_printf("Predictions (time: %d ms.):\n", result-
>timing.classification);
        }

#if EI_CLASSIFIER_OBJECT_DETECTION == 1
        float *scores_tensor = interpreter-
>typed_output_tensor<float>(EI_CLASSIFIER_TFLITE_OUTPUT_SCORE_TENSOR);
        float *label_tensor = interpreter-
>typed_output_tensor<float>(EI_CLASSIFIER_TFLITE_OUTPUT_LABELS_TENSOR);
        if (!scores_tensor) {
            return EI_IMPULSE_SCORE_TENSOR_WAS_NULL;
        }
        if (!label_tensor) {
            return EI_IMPULSE_LABEL_TENSOR_WAS_NULL;
        }
        fill_result_struct_f32(result, out_data, scores_tensor,
label_tensor, debug);
#else

    #if EI_CLASSIFIER_TFLITE_OUTPUT_QUANTIZED == 1
        fill_result_struct_i8(result, out_data,
EI_CLASSIFIER_TFLITE_OUTPUT_ZEROPOINT, EI_CLASSIFIER_TFLITE_OUTPUT_SCALE,
debug);
    #else
        fill_result_struct_f32(result, out_data, debug);
    #endif

#endif

        // on Linux we're not worried about free'ing (for now)
    }

#elif (EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TENSAIFLOW)
    {
        uint64_t ctx_start_ms = ei_read_timer_ms();
        int8_t *input;
        int8_t output[EI_CLASSIFIER_LABEL_COUNT];

        input = (int8_t *)ei_malloc(fmatrix->rows * fmatrix->cols);

        if (!input) {
            return EI_IMPULSE_ALLOC_FAILED;
        }

        for (size_t ix = 0; ix < fmatrix->rows * fmatrix->cols; ix++) {
            input[ix] = static_cast<int8_t>(
                round(fmatrix->buffer[ix] /
EI_CLASSIFIER_TFLITE_INPUT_SCALE) +
```

```
                EI_CLASSIFIER_TFLITE_INPUT_ZEROPOINT);
        }

        /* Run tensaiflow inference */
        infer(input, output);

        for (uint32_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
            float value;
            // Dequantize the output if it is int8
            value = static_cast<float>(output[ix] -
EI_CLASSIFIER_TFLITE_OUTPUT_ZEROPOINT) *
                EI_CLASSIFIER_TFLITE_OUTPUT_SCALE;

            if (debug) {
                ei_printf("%s:\t",
ei_classifier_inferencing_categories[ix]);
                ei_printf_float(value);
                ei_printf("\n");
            }
            result->classification[ix].label =
ei_classifier_inferencing_categories[ix];
            result->classification[ix].value = value;
        }

        result->timing.classification = ei_read_timer_ms() - ctx_start_ms;

        ei_free(input);
    }

#endif

#if EI_CLASSIFIER_HAS_ANOMALY == 1

    // Anomaly detection
    {
        uint64_t anomaly_start_ms = ei_read_timer_ms();

        float input[EI_CLASSIFIER_ANOM_AXIS_SIZE];
        for (size_t ix = 0; ix < EI_CLASSIFIER_ANOM_AXIS_SIZE; ix++) {
            input[ix] = fmatrix->buffer[EI_CLASSIFIER_ANOM_AXIS[ix]];
        }
        standard_scaler(input, ei_classifier_anom_scale,
ei_classifier_anom_mean, EI_CLASSIFIER_ANOM_AXIS_SIZE);
        float anomaly = get_min_distance_to_cluster(
            input, EI_CLASSIFIER_ANOM_AXIS_SIZE,
ei_classifier_anom_clusters, EI_CLASSIFIER_ANOM_CLUSTER_COUNT);

        uint64_t anomaly_end_ms = ei_read_timer_ms();

        if (debug) {
            ei_printf("Anomaly score (time: %d ms.): ",
static_cast<int>(anomaly_end_ms - anomaly_start_ms));
            ei_printf_float(anomaly);
            ei_printf("\n");
        }
```

```
        result->timing.anomaly = anomaly_end_ms - anomaly_start_ms;

        result->anomaly = anomaly;
    }

#endif

    if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
        return EI_IMPULSE_CANCELED;
    }

    return EI_IMPULSE_OK;
}

extern "C" EI_IMPULSE_ERROR run_inference_i16(
    ei::matrix_i32_t *fmatrix,
    ei_impulse_result_t *result,
    bool debug = false)
{
#if EI_CLASSIFIER_OBJECT_DETECTION
    return EI_IMPULSE_NOT_SUPPORTED_WITH_I16;
#else

#if (EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE)
    {
        uint64_t ctx_start_ms;
        TfLiteTensor* input;
        TfLiteTensor* output;
        uint8_t* tensor_arena;

#if (EI_CLASSIFIER_COMPILED == 1)
        EI_IMPULSE_ERROR init_res = inference_tflite_setup(&ctx_start_ms,
&input, &output,
            &tensor_arena);
#else
        tflite::MicroInterpreter* interpreter;
        EI_IMPULSE_ERROR init_res = inference_tflite_setup(&ctx_start_ms,
&input, &output,
            &interpreter,
            &tensor_arena);
#endif
        if (init_res != EI_IMPULSE_OK) {
            return init_res;
        }

        EIDSP_i16 scale;
        numpy::float_to_int16(&input->params.scale, &scale, 1);

        // Place our calculated x value in the model's input tensor
        bool int8_input = input->type == TfLiteType::kTfLiteInt8;
        for (size_t ix = 0; ix < fmatrix->rows * fmatrix->cols; ix++) {
            // Quantize the input if it is int8
            if (int8_input) {
                int32_t calc = (int32_t)fmatrix->buffer[ix] << 8; // Shift
for scaler
                calc /= scale;
```

```
                calc += 0x80; // Round by adding 0.5
                calc >>= 8; // Shift to int8_t domain
                input->data.int8[ix] = static_cast<int8_t>(calc + input-
>params.zero_point);
            } else {
                numpy::int16_to_float((EIDSP_i16 *)&fmatrix->buffer[ix],
&input->data.f[ix], 1);
            }
        }

#if (EI_CLASSIFIER_COMPILED == 1)
        EI_IMPULSE_ERROR run_res = inference_tflite_run(ctx_start_ms,
output,
            tensor_arena, result, debug);
#else
        EI_IMPULSE_ERROR run_res = inference_tflite_run(ctx_start_ms,
output,
            interpreter, tensor_arena, result, debug);
#endif

        if (run_res != EI_IMPULSE_OK) {
            return run_res;
        }
    }

#elif EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE_FULL

    ei_printf("ERR: run_classifier_i16 is not supported with full
TensorFlow Lite\n");
    return EI_IMPULSE_TFLITE_ERROR;

#endif

#if EI_CLASSIFIER_HAS_ANOMALY == 1

    // Anomaly detection
    {
        uint64_t anomaly_start_ms = ei_read_timer_ms();

        float input[EI_CLASSIFIER_ANOM_AXIS_SIZE];
        for (size_t ix = 0; ix < EI_CLASSIFIER_ANOM_AXIS_SIZE; ix++) {
            // input[ix] = fmatrix->buffer[EI_CLASSIFIER_ANOM_AXIS[ix]];
            // numpy::int16_to_float(&fmatrix-
>buffer[EI_CLASSIFIER_ANOM_AXIS[ix]], &input[ix], 1);
            input[ix] = (float)fmatrix->buffer[EI_CLASSIFIER_ANOM_AXIS[ix]]
/ 32768.f;
        }
        standard_scaler(input, ei_classifier_anom_scale,
ei_classifier_anom_mean, EI_CLASSIFIER_ANOM_AXIS_SIZE);
        float anomaly = get_min_distance_to_cluster(
            input, EI_CLASSIFIER_ANOM_AXIS_SIZE,
ei_classifier_anom_clusters, EI_CLASSIFIER_ANOM_CLUSTER_COUNT);

        uint64_t anomaly_end_ms = ei_read_timer_ms();

        if (debug) {
```

```
            ei_printf("Anomaly score (time: %d ms.): ",
static_cast<int>(anomaly_end_ms - anomaly_start_ms));
            ei_printf_float(anomaly);
            ei_printf("\n");
        }

        result->timing.anomaly = anomaly_end_ms - anomaly_start_ms;

        result->anomaly = anomaly;
    }

#endif

    if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
        return EI_IMPULSE_CANCELED;
    }

    return EI_IMPULSE_OK;
#endif // OBJECT_DETECTION
}

/**
 * Run the classifier over a raw features array
 * @param raw_features Raw features array
 * @param raw_features_size Size of the features array
 * @param result Object to store the results in
 * @param debug Whether to show debug messages (default: false)
 */
extern "C" EI_IMPULSE_ERROR run_classifier(
    signal_t *signal,
    ei_impulse_result_t *result,
    bool debug = false)
{
#if EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1 &&
EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE
    // Shortcut for quantized image models
    if (can_run_classifier_image_quantized() == EI_IMPULSE_OK) {
        return run_classifier_image_quantized(signal, result, debug);
    }
#endif

    // if (debug) {
    // static float buf[1000];
    // printf("Raw data: ");
    // for (size_t ix = 0; ix < 16000; ix += 1000) {
    //     int r = signal->get_data(ix, 1000, buf);
    //     for (size_t jx = 0; jx < 1000; jx++) {
    //         printf("%.0f, ", buf[jx]);
    //     }
    // }
    // printf("\n");
    // }

    memset(result, 0, sizeof(ei_impulse_result_t));

    ei::matrix_t features_matrix(1, EI_CLASSIFIER_NN_INPUT_FRAME_SIZE);
```

```
    uint64_t dsp_start_ms = ei_read_timer_ms();

    size_t out_features_index = 0;

    for (size_t ix = 0; ix < ei_dsp_blocks_size; ix++) {
        ei_model_dsp_t block = ei_dsp_blocks[ix];

        if (out_features_index + block.n_output_features >
EI_CLASSIFIER_NN_INPUT_FRAME_SIZE) {
            ei_printf("ERR: Would write outside feature buffer\n");
            return EI_IMPULSE_DSP_ERROR;
        }

        ei::matrix_t fm(1, block.n_output_features, features_matrix.buffer
+ out_features_index);

        int ret = block.extract_fn(signal, &fm, block.config,
EI_CLASSIFIER_FREQUENCY);
        if (ret != EIDSP_OK) {
            ei_printf("ERR: Failed to run DSP process (%d)\n", ret);
            return EI_IMPULSE_DSP_ERROR;
        }

        if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
            return EI_IMPULSE_CANCELED;
        }

        out_features_index += block.n_output_features;
    }

    result->timing.dsp = ei_read_timer_ms() - dsp_start_ms;

    if (debug) {
        ei_printf("Features (%d ms.): ", result->timing.dsp);
        for (size_t ix = 0; ix < features_matrix.cols; ix++) {
            ei_printf_float(features_matrix.buffer[ix]);
            ei_printf(" ");
        }
        ei_printf("\n");
    }

#if EI_CLASSIFIER_INFERENCING_ENGINE != EI_CLASSIFIER_NONE
    if (debug) {
        ei_printf("Running neural network...\n");
    }
#endif

    return run_inference(&features_matrix, result, debug);
}

#if defined(EI_CLASSIFIER_USE_QUANTIZED_DSP_BLOCK) &&
EI_CLASSIFIER_USE_QUANTIZED_DSP_BLOCK == 1

extern "C" EI_IMPULSE_ERROR run_classifier_i16(
    signal_i16_t *signal,
```

```
    ei_impulse_result_t *result,
    bool debug = false)
{

    memset(result, 0, sizeof(ei_impulse_result_t));

    ei::matrix_i32_t features_matrix(1, EI_CLASSIFIER_NN_INPUT_FRAME_SIZE);

    uint64_t dsp_start_ms = ei_read_timer_ms();

    size_t out_features_index = 0;

    for (size_t ix = 0; ix < ei_dsp_blocks_size; ix++) {
        ei_model_dsp_i16_t block = ei_dsp_blocks_i16[ix];

        if (out_features_index + block.n_output_features >
EI_CLASSIFIER_NN_INPUT_FRAME_SIZE) {
            ei_printf("ERR: Would write outside feature buffer\n");
            return EI_IMPULSE_DSP_ERROR;
        }

        ei::matrix_i32_t fm(1, block.n_output_features,
features_matrix.buffer + out_features_index);

        int ret = block.extract_fn(signal, &fm, ei_dsp_blocks[ix].config,
EI_CLASSIFIER_FREQUENCY);

        if (ret != EIDSP_OK) {
            ei_printf("ERR: Failed to run DSP process (%d)\n", ret);
            return EI_IMPULSE_DSP_ERROR;
        }

        if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
            return EI_IMPULSE_CANCELED;
        }

        out_features_index += block.n_output_features;
    }

    result->timing.dsp = ei_read_timer_ms() - dsp_start_ms;

    if (debug) {
        ei_printf("Features (%d ms.): ", result->timing.dsp);
        for (size_t ix = 0; ix < features_matrix.cols; ix++) {
            ei_printf_float((float)features_matrix.buffer[ix] / 32768.f);
            ei_printf(" ");
        }
        ei_printf("\n");
    }

#if EI_CLASSIFIER_INFERENCING_ENGINE != EI_CLASSIFIER_NONE
    if (debug) {
        ei_printf("Running neural network...\n");
    }
#endif
```

```
    return run_inference_i16(&features_matrix, result, debug);
}
#endif //EI_CLASSIFIER_USE_QUANTIZED_DSP_BLOCK

/**
 * @brief      Calculates the cepstral mean and variable normalization.
 *
 * @param      matrix      Source and destination matrix
 * @param      config_ptr  ei_dsp_config_mfcc_t struct pointer
 */
static void calc_cepstral_mean_and_var_normalization_mfcc(ei_matrix
*matrix, void *config_ptr)
{
    ei_dsp_config_mfcc_t *config = (ei_dsp_config_mfcc_t *)config_ptr;

    /* Modify rows and colums ration for matrix normalization */
    matrix->rows = EI_CLASSIFIER_NN_INPUT_FRAME_SIZE / config-
>num_cepstral;
    matrix->cols = config->num_cepstral;

    // cepstral mean and variance normalization
    int ret = speechpy::processing::cmvnw(matrix, config->win_size, true,
false);
    if (ret != EIDSP_OK) {
        ei_printf("ERR: cmvnw failed (%d)\n", ret);
        return;
    }

    /* Reset rows and columns ratio */
    matrix->rows = 1;
    matrix->cols = EI_CLASSIFIER_NN_INPUT_FRAME_SIZE;
}

/**
 * @brief      Calculates the cepstral mean and variable normalization.
 *
 * @param      matrix      Source and destination matrix
 * @param      config_ptr  ei_dsp_config_mfe_t struct pointer
 */
static void calc_cepstral_mean_and_var_normalization_mfe(ei_matrix *matrix,
void *config_ptr)
{
    ei_dsp_config_mfe_t *config = (ei_dsp_config_mfe_t *)config_ptr;

    /* Modify rows and colums ration for matrix normalization */
    matrix->rows = EI_CLASSIFIER_NN_INPUT_FRAME_SIZE / config->num_filters;
    matrix->cols = config->num_filters;

    // cepstral mean and variance normalization
    int ret = speechpy::processing::cmvnw(matrix, config->win_size, false,
true);
    if (ret != EIDSP_OK) {
        ei_printf("ERR: cmvnw failed (%d)\n", ret);
        return;
    }
```

```
    /* Reset rows and columns ratio */
    matrix->rows = 1;
    matrix->cols = EI_CLASSIFIER_NN_INPUT_FRAME_SIZE;
}

/**
 * @brief      Calculates the cepstral mean and variable normalization.
 *
 * @param      matrix      Source and destination matrix
 * @param      config_ptr  ei_dsp_config_spectrogram_t struct pointer
 */
static void calc_cepstral_mean_and_var_normalization_spectrogram(ei_matrix
*matrix, void *config_ptr)
{
    ei_dsp_config_spectrogram_t *config = (ei_dsp_config_spectrogram_t
*)config_ptr;

    /* Modify rows and colums ration for matrix normalization */
    matrix->cols = config->fft_length / 2 + 1;
    matrix->rows = EI_CLASSIFIER_NN_INPUT_FRAME_SIZE / matrix->cols;

    int ret = numpy::normalize(matrix);
    if (ret != EIDSP_OK) {
        ei_printf("ERR: normalization failed (%d)\n", ret);
        return;
    }

    /* Reset rows and columns ratio */
    matrix->rows = 1;
    matrix->cols = EI_CLASSIFIER_NN_INPUT_FRAME_SIZE;
}

/**
 * Check if the current impulse could be used by
'run_classifier_image_quantized'
 */
__attribute__((unused)) static EI_IMPULSE_ERROR
can_run_classifier_image_quantized() {
#if (EI_CLASSIFIER_INFERENCING_ENGINE != EI_CLASSIFIER_TFLITE)
    return EI_IMPULSE_UNSUPPORTED_INFERENCING_ENGINE;
#endif

#if EI_CLASSIFIER_HAS_ANOMALY == 1
    return EI_IMPULSE_ONLY_SUPPORTED_FOR_IMAGES;
#endif

    // Check if we have a quantized NN
#if EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED != 1
    return EI_IMPULSE_ONLY_SUPPORTED_FOR_IMAGES;
#endif

    // And if we have one DSP block which operates on images...
    if (ei_dsp_blocks_size != 1 || ei_dsp_blocks[0].extract_fn !=
extract_image_features) {
        return EI_IMPULSE_ONLY_SUPPORTED_FOR_IMAGES;
    }
```

```
    return EI_IMPULSE_OK;
}

#if EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1 &&
EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE
/**
 * Special function to run the classifier on images, only works on TFLite
models (either interpreter or EON)
 * that allocates a lot less memory by quantizing in place. This only works
if 'can_run_classifier_image_quantized'
 * returns EI_IMPULSE_OK.
 */
extern "C" EI_IMPULSE_ERROR run_classifier_image_quantized(
    signal_t *signal,
    ei_impulse_result_t *result,
    bool debug = false)
{
    EI_IMPULSE_ERROR verify_res = can_run_classifier_image_quantized();
    if (verify_res != EI_IMPULSE_OK) {
        return verify_res;
    }

    memset(result, 0, sizeof(ei_impulse_result_t));

#if (EI_CLASSIFIER_INFERENCING_ENGINE != EI_CLASSIFIER_TFLITE)
    return EI_IMPULSE_UNSUPPORTED_INFERENCING_ENGINE;
#else
    uint64_t ctx_start_ms;
    TfLiteTensor* input;
    TfLiteTensor* output;
#if EI_CLASSIFIER_OBJECT_DETECTION
    TfLiteTensor* output_scores;
    TfLiteTensor* output_labels;
#endif
    uint8_t* tensor_arena;

#if (EI_CLASSIFIER_COMPILED == 1)
    EI_IMPULSE_ERROR init_res = inference_tflite_setup(&ctx_start_ms,
&input, &output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
        &output_labels,
        &output_scores,
    #endif
        &tensor_arena);
#else
    tflite::MicroInterpreter* interpreter;
    EI_IMPULSE_ERROR init_res = inference_tflite_setup(&ctx_start_ms,
&input, &output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
        &output_labels,
        &output_scores,
    #endif
        &interpreter,
        &tensor_arena);
#endif
```

```
    if (init_res != EI_IMPULSE_OK) {
        return init_res;
    }

    if (input->type != TfLiteType::kTfLiteInt8) {
        return EI_IMPULSE_ONLY_SUPPORTED_FOR_IMAGES;
    }

    uint64_t dsp_start_ms = ei_read_timer_ms();

    // features matrix maps around the input tensor to not allocate any
memory
    ei::matrix_i8_t features_matrix(1, EI_CLASSIFIER_NN_INPUT_FRAME_SIZE,
input->data.int8);

    // run DSP process and quantize automatically
    int ret = extract_image_features_quantized(signal, &features_matrix,
ei_dsp_blocks[0].config, EI_CLASSIFIER_FREQUENCY);
    if (ret != EIDSP_OK) {
        ei_printf("ERR: Failed to run DSP process (%d)\n", ret);
        return EI_IMPULSE_DSP_ERROR;
    }

    if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
        return EI_IMPULSE_CANCELED;
    }

    result->timing.dsp = ei_read_timer_ms() - dsp_start_ms;

    if (debug) {
        ei_printf("Features (%d ms.): ", result->timing.dsp);
        for (size_t ix = 0; ix < features_matrix.cols; ix++) {
            ei_printf_float((features_matrix.buffer[ix] -
EI_CLASSIFIER_TFLITE_INPUT_ZEROPOINT) * EI_CLASSIFIER_TFLITE_INPUT_SCALE);
            ei_printf(" ");
        }
        ei_printf("\n");
    }

    ctx_start_ms = ei_read_timer_ms();

#if (EI_CLASSIFIER_COMPILED == 1)
    EI_IMPULSE_ERROR run_res = inference_tflite_run(ctx_start_ms, output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
        output_labels,
        output_scores,
    #endif
        tensor_arena, result, debug);
#else
    EI_IMPULSE_ERROR run_res = inference_tflite_run(ctx_start_ms, output,
    #if EI_CLASSIFIER_OBJECT_DETECTION
        output_labels,
        output_scores,
    #endif
        interpreter, tensor_arena, result, debug);
#endif
```

```
    if (run_res != EI_IMPULSE_OK) {
        return run_res;
    }

    return EI_IMPULSE_OK;
#endif // EI_CLASSIFIER_INFERENCING_ENGINE != EI_CLASSIFIER_TFLITE
}
#endif // #if EI_CLASSIFIER_TFLITE_INPUT_QUANTIZED == 1 &&
EI_CLASSIFIER_INFERENCING_ENGINE == EI_CLASSIFIER_TFLITE

#if EIDSP_SIGNAL_C_FN_POINTER == 0

/**
 * Run the impulse, if you provide an instance of sampler it will also
persist the data for you
 * @param sampler Instance to an **initialized** sampler
 * @param result Object to store the results in
 * @param data_fn Function to retrieve data from sensors
 * @param debug Whether to log debug messages (default false)
 */
__attribute__((unused)) EI_IMPULSE_ERROR run_impulse(
#if defined(EI_CLASSIFIER_HAS_SAMPLER) && EI_CLASSIFIER_HAS_SAMPLER == 1
        EdgeSampler *sampler,
#endif
        ei_impulse_result_t *result,
#ifdef __MBED__
        mbed::Callback<void(float*, size_t)> data_fn,
#else
        std::function<void(float*, size_t)> data_fn,
#endif
        bool debug = false) {

    float *x = (float*)calloc(EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE,
sizeof(float));
    if (!x) {
        return EI_IMPULSE_OUT_OF_MEMORY;
    }

    uint64_t next_tick = 0;

    uint64_t sampling_us_start = ei_read_timer_us();

    // grab some data
    for (int i = 0; i < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; i +=
EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME) {
        uint64_t curr_us = ei_read_timer_us() - sampling_us_start;

        next_tick = curr_us + (EI_CLASSIFIER_INTERVAL_MS * 1000);

        data_fn(x + i, EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME);
#if defined(EI_CLASSIFIER_HAS_SAMPLER) && EI_CLASSIFIER_HAS_SAMPLER == 1
        if (sampler != NULL) {
            sampler->write_sensor_data(x + i,
EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME);
        }
```

```
 #endif

        if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
            free(x);
            return EI_IMPULSE_CANCELED;
        }

        while (next_tick > ei_read_timer_us() - sampling_us_start);
    }

    result->timing.sampling = (ei_read_timer_us() - sampling_us_start) /
1000;

    signal_t signal;
    int err = numpy::signal_from_buffer(x,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
    if (err != 0) {
        free(x);
        ei_printf("ERR: signal_from_buffer failed (%d)\n", err);
        return EI_IMPULSE_DSP_ERROR;
    }

    EI_IMPULSE_ERROR r = run_classifier(&signal, result, debug);
    free(x);
    return r;
}

#if defined(EI_CLASSIFIER_USE_QUANTIZED_DSP_BLOCK) &&
EI_CLASSIFIER_USE_QUANTIZED_DSP_BLOCK == 1

__attribute__((unused)) EI_IMPULSE_ERROR run_impulse_i16(
#if defined(EI_CLASSIFIER_HAS_SAMPLER) && EI_CLASSIFIER_HAS_SAMPLER == 1
        EdgeSampler *sampler,
#endif
        ei_impulse_result_t *result,
#ifdef __MBED__
        mbed::Callback<void(signed short*, size_t)> data_fn,
#else
        std::function<void(signed short*, size_t)> data_fn,
#endif
        bool debug = false) {

    int16_t x[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };

    uint64_t next_tick = 0;

    uint64_t sampling_us_start = ei_read_timer_us();

    // grab some data
    for (int i = 0; i < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; i +=
EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME) {
        uint64_t curr_us = ei_read_timer_us() - sampling_us_start;

        next_tick = curr_us + (EI_CLASSIFIER_INTERVAL_MS * 1000);

        data_fn(x + i, EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME);
```

```
 #if defined(EI_CLASSIFIER_HAS_SAMPLER) && EI_CLASSIFIER_HAS_SAMPLER == 1
        if (sampler != NULL) {
            sampler->write_sensor_data(x + i,
EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME);
        }
#endif

        if (ei_run_impulse_check_canceled() == EI_IMPULSE_CANCELED) {
            return EI_IMPULSE_CANCELED;
        }

        while (next_tick > ei_read_timer_us() - sampling_us_start);
    }

    result->timing.sampling = (ei_read_timer_us() - sampling_us_start) /
1000;

    signal_i16_t signal;
    int err = numpy::signal_from_buffer_i16(x,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
    if (err != 0) {
        ei_printf("ERR: signal_from_buffer failed (%d)\n", err);
        return EI_IMPULSE_DSP_ERROR;
    }

    return run_classifier_i16(&signal, result, debug);
}
#endif //EI_CLASSIFIER_USE_QUANTIZED_DSP_BLOCK

#if defined(EI_CLASSIFIER_HAS_SAMPLER) && EI_CLASSIFIER_HAS_SAMPLER == 1
/**
 * Run the impulse, does not persist data
 * @param result Object to store the results in
 * @param data_fn Function to retrieve data from sensors
 * @param debug Whether to log debug messages (default false)
 */
__attribute__((unused)) EI_IMPULSE_ERROR run_impulse(
        ei_impulse_result_t *result,
#ifdef __MBED__
        mbed::Callback<void(float*, size_t)> data_fn,
#else
        std::function<void(float*, size_t)> data_fn,
#endif
        bool debug = false) {
    return run_impulse(NULL, result, data_fn, debug);
}
#endif

#endif // #if EIDSP_SIGNAL_C_FN_POINTER == 0

#ifdef __cplusplus
}
#endif // __cplusplus

#endif // _EDGE_IMPULSE_RUN_CLASSIFIER_H_
```

5.1.4.3 Trained_model_compiled

```
 /* Generated by Edge Impulse
*
* Permission is hereby granted, free of charge, to any person obtaining a
copy
* of this software and associated documentation files (the "Software"), to
deal
* in the Software without restriction, including without limitation the
rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included
in
* all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE
* SOFTWARE.
*/
// Generated on: 09.04.2021 17:41:14

#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include "edge-impulse-sdk/tensorflow/lite/c/builtin_op_data.h"
#include "edge-impulse-sdk/tensorflow/lite/c/common.h"
#include "edge-impulse-sdk/tensorflow/lite/micro/kernels/micro_ops.h"

#if EI_CLASSIFIER_PRINT_STATE
#if defined(__cplusplus) && EI_C_LINKAGE == 1
extern "C" {
    extern void ei_printf(const char *format, ...);
}
#else
extern void ei_printf(const char *format, ...);
#endif
#endif

#if defined __GNUC__
#define ALIGN(X) __attribute__((aligned(X)))
#elif defined _MSC_VER
#define ALIGN(X) __declspec(align(X))
#elif defined __TASKING__
#define ALIGN(X) __align(X)
#endif
```

```
 namespace {

constexpr int kTensorArenaSize = 176;

#if defined(EI_CLASSIFIER_ALLOCATION_STATIC)
uint8_t tensor_arena[kTensorArenaSize] ALIGN(16);
#elif defined(EI_CLASSIFIER_ALLOCATION_STATIC_HIMAX)
#pragma Bss(".tensor_arena")
uint8_t tensor_arena[kTensorArenaSize] ALIGN(16);
#pragma Bss()
#elif defined(EI_CLASSIFIER_ALLOCATION_STATIC_HIMAX_GNU)
uint8_t tensor_arena[kTensorArenaSize] ALIGN(16)
__attribute__((section(".tensor_arena")));
#else
#define EI_CLASSIFIER_ALLOCATION_HEAP 1
uint8_t* tensor_arena = NULL;
#endif

static uint8_t* tensor_boundary;
static uint8_t* current_location;

template <int SZ, class T> struct TfArray {
  int sz; T elem[SZ];
};
enum used_operators_e {
  OP_FULLY_CONNECTED, OP_SOFTMAX,  OP_LAST
};
struct TensorInfo_t { // subset of TfLiteTensor used for initialization
from constant memory
  TfLiteAllocationType allocation_type;
  TfLiteType type;
  void* data;
  TfLiteIntArray* dims;
  size_t bytes;
  TfLiteQuantization quantization;
};
struct NodeInfo_t { // subset of TfLiteNode used for initialization from
constant memory
  struct TfLiteIntArray* inputs;
  struct TfLiteIntArray* outputs;
  void* builtin_data;
  used_operators_e used_op_index;
};

TfLiteContext ctx{};
TfLiteTensor tflTensors[14];
TfLiteRegistration registrations[OP_LAST];
TfLiteNode tflNodes[5];

const TfArray<2, int> tensor_dimension0 = { 2, { 1,33 } };
const TfArray<1, float> quant0_scale = { 1, { 0.093370683491230011, } };
const TfArray<1, int> quant0_zero = { 1, { -128 } };
const TfLiteAffineQuantization quant0 = { (TfLiteFloatArray*)&quant0_scale,
(TfLiteIntArray*)&quant0_zero, 0 };
```

```
 const ALIGN(8) int32_t tensor_data1[20] = { 526, -126, 452, 517, 3, -328,
-264, 182, 47, 141, 202, -214, 13, 299, 837, 45, 530, -273, 0, -73, };
const TfArray<1, int> tensor_dimension1 = { 1, { 20 } };
const TfArray<1, float> quant1_scale = { 1, { 0.00058937229914590716, } };
const TfArray<1, int> quant1_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant1 = { (TfLiteFloatArray*)&quant1_scale,
(TfLiteIntArray*)&quant1_zero, 0 };
const ALIGN(8) int32_t tensor_data2[10] = { 747, -139, 369, -297, -13, 436,
-364, -709, -216, -517, };
const TfArray<1, int> tensor_dimension2 = { 1, { 10 } };
const TfArray<1, float> quant2_scale = { 1, { 0.00043212089804001153, } };
const TfArray<1, int> quant2_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant2 = { (TfLiteFloatArray*)&quant2_scale,
(TfLiteIntArray*)&quant2_zero, 0 };
const ALIGN(8) int32_t tensor_data3[50] = { -646, -132, -725, -61, -176,
176, -108, -141, -601, -18, -67, 1809, 1297, -88, -195, -490, -115, -885, -
144, -142, 20, -155, 2, 25, 968, -183, -367, 1416, 27, -316, 1366, -185, -
1304, -309, -293, -29, -171, -228, 0, -3, 1345, -87, -87, 1069, 13, -428,
659, 72, -194, -15, };
const TfArray<1, int> tensor_dimension3 = { 1, { 50 } };
const TfArray<1, float> quant3_scale = { 1, { 0.0006739548989571631, } };
const TfArray<1, int> quant3_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant3 = { (TfLiteFloatArray*)&quant3_scale,
(TfLiteIntArray*)&quant3_zero, 0 };
const ALIGN(8) int32_t tensor_data4[6] = { -293, 391, 164, 585, -435, -366,
};
const TfArray<1, int> tensor_dimension4 = { 1, { 6 } };
const TfArray<1, float> quant4_scale = { 1, { 0.000858235580381006, } };
const TfArray<1, int> quant4_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant4 = { (TfLiteFloatArray*)&quant4_scale,
(TfLiteIntArray*)&quant4_zero, 0 };
const ALIGN(8) int8_t tensor_data5[20*33] = {
  -41, -41, -118, 14, -26, 13, -60, -38, -8, 31, 17, 58, -4, 23, 0, -5, 67,
-24, 36, 17, 21, -69, 39, -58, -57, -21, -50, 16, -10, -19, -9, -34, 6,
  -4, 1, -10, -31, 0, 39, 23, 43, -53, 24, 17, -35, -21, -55, -39, -31, -
29, -53, 36, 14, -13, -38, -53, -40, 7, -40, 3, 14, -38, 3, 31, 38, -58,
  -2, -41, -43, 42, 16, 18, -46, -13, 0, 20, -11, -47, -28, -5, 30, -51, -
38, -18, 44, -56, 46, -1, 39, 12, 9, 66, 18, 1, 7, -32, 3, 79, -18,
  -78, -21, -76, 8, -4, -6, -60, 8, -9, 18, -27, -55, -12, 0, -28, -53, -
27, -30, 50, 14, -66, -55, 73, -8, -5, 46, -8, -27, 28, 10, 4, 56, 6,
  54, 25, -43, 32, -1, 43, -66, -21, 64, 36, -1, -48, -2, -53, -62, -51, -
11, -62, 12, -20, 2, -45, 45, 1, -29, -24, 22, 35, 30, 0, 37, -6, -36,
  -15, -16, 37, 5, 19, 29, 38, 18, -40, -41, 8, -48, 30, 25, 66, 40, -16,
53, 36, 43, 28, 9, -45, 9, -13, 44, 36, -19, 48, 17, -9, 23, 9,
  -54, 33, -26, -32, -40, -2, -3, -12, -35, 17, -56, -55, -67, -50, 32, -2,
-25, 10, 14, -31, -2, -19, 18, -54, -6, -20, -46, -10, -6, -14, 5, -30, -
52,
  -32, 72, -84, 0, -68, 68, -17, -49, 57, -64, -16, -28, -33, -36, -15, -
26, 12, -52, 3, -12, -10, -39, -61, -7, -8, -42, 27, 15, 16, 37, -25, 72, -
34,
  53, 1, -51, -64, 15, 8, -34, -10, 6, -57, -69, 101, 8, 80, -24, -59, 20,
9, 38, 88, 106, 96, -14, -14, -32, 35, -59, -6, 28, -1, -31, 9, -37,
  -5, 26, 6, 39, -44, 9, 9, -20, 57, 22, 25, -6, -18, 66, 19, 5, 20, 50,
43, -10, 109, 38, -9, 103, 5, -19, -2, -18, 51, 30, -30, -5, -41,
  -43, -38, 18, -57, 42, -35, 10, -2, 86, 108, 70, 34, 35, -21, 6, 5, -9,
51, -16, -52, -111, -72, 62, -67, -42, -5, -61, 21, 10, -44, 47, -12, -69,
```

```
  81, 19, -18, -18, -5, 46, -1, 37, 51, 43, -31, 72, 10, -12, -29, 0, 6,
17, 41, 93, 55, 9, -12, 8, 33, -42, -28, 39, -46, 32, -37, 55, -21,
  62, 53, 78, -36, 27, -29, 0, -23, -23, 38, -4, -75, -71, 85, 35, -52, 20,
-12, -50, -40, 2, 40, 127, -7, -52, -10, -44, 32, 47, -22, 15, -39, -41,
  17, 2, 26, 19, -6, -21, 15, -10, 11, 32, 42, 76, 17, -10, -10, -3, -43, -
41, 14, 59, -35, -4, 27, -43, -62, 27, 16, -1, 32, -23, -64, 42, 4,
  -105, -52, 52, -25, 25, -18, 27, 33, -11, -33, -65, 12, -53, -31, 9, -4,
-29, -24, -22, -53, 1, 8, -35, 17, 87, -10, -51, -7, -22, 3, -10, -15, 23,
  -56, 47, -11, 9, -19, 38, -38, -26, -45, 19, -38, 44, -33, -7, 56, -31, -
7, 42, 33, 43, 13, -19, -15, -46, -10, -20, 27, 11, 1, 5, -73, -55, -41,
  -46, 57, 49, 26, 18, 16, 11, -25, 4, 29, 19, -60, -20, -62, -23, 35, 44,
36, -44, -40, -57, -74, -12, 39, 70, 41, 7, 17, 68, 49, 17, 12, 22,
  -35, -9, -63, -52, 40, -29, 1, 38, -40, -5, -6, 5, -3, -28, -19, -28, -8,
-19, -28, 29, -60, -27, 10, -46, 35, -17, -48, 39, 21, 27, 27, 2, -1,
  -51, 4, 70, 52, 31, 36, 16, -36, -14, -8, -30, -39, 47, 28, -7, 90, -9,
20, -44, -25, 21, -37, -33, 52, -44, -40, 50, -17, 12, -3, 28, 10, 28,
  3, 0, 59, 14, 35, -15, 23, -45, 4, 48, 12, 85, -8, -16, 40, 14, 7, -30,
15, 23, -40, 20, 30, -59, 61, 51, 13, 41, -40, -52, -85, -53, 69,
};
const TfArray<2, int> tensor_dimension5 = { 2, { 20,33 } };
const TfArray<1, float> quant5_scale = { 1, { 0.006312177050858736, } };
const TfArray<1, int> quant5_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant5 = { (TfLiteFloatArray*)&quant5_scale,
(TfLiteIntArray*)&quant5_zero, 0 };
const ALIGN(8) int8_t tensor_data6[10*20] = {
  -11, -10, 89, 32, 47, -61, -27, 0, -54, -47, 75, -36, 127, 48, -110, -12,
-61, 1, -31, 4,
  -39, 9, -30, 15, 15, -3, 42, -26, -22, -44, 1, -47, -14, -27, -46, -52, -
25, 31, 21, -60,
  -39, 30, 29, -87, 45, -3, 40, -36, 36, 75, -49, -31, 92, -51, 80, -5, -
11, 26, -54, -8,
  -35, -11, -9, -56, -18, 26, 32, 52, 46, -30, 16, 8, 11, -59, -57, 69, 21,
10, 6, -38,
  8, -3, -15, -44, -82, 58, -18, -18, -49, 102, -85, -58, -33, 4, 38, 19,
14, -57, 61, 11,
  48, 43, -57, -6, -52, -14, -7, -24, 96, -54, -12, 56, -45, 29, -85, 38, -
62, -10, -67, 38,
  29, -44, 14, 37, 41, -26, -2, 30, 9, 46, -43, 33, -75, -20, -25, -23, 21,
-21, 20, -42,
  16, 51, -29, -65, -15, -4, -16, -87, 113, 66, 34, 102, -60, 11, -68, -31,
-43, -4, -27, 35,
  -64, 35, -10, 11, 49, -13, 22, -66, -12, -44, -34, 44, 86, 52, -28, 15, -
54, 45, -16, -7,
  -15, 10, 68, 55, -38, 38, 27, -33, 37, -25, -58, 8, -51, -15, -58, 8, 67,
44, -19, 59,
};
const TfArray<2, int> tensor_dimension6 = { 2, { 10,20 } };
const TfArray<1, float> quant6_scale = { 1, { 0.0071480576880276203, } };
const TfArray<1, int> quant6_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant6 = { (TfLiteFloatArray*)&quant6_scale,
(TfLiteIntArray*)&quant6_zero, 0 };
const ALIGN(8) int8_t tensor_data7[50*10] = {
  23, -18, 15, -1, 0, 21, -8, 12, 31, -8,
  12, 11, 20, -10, 2, -15, 14, -18, 21, 12,
  -43, 15, -9, -1, -5, 27, 16, 35, -3, 5,
  -25, 15, 26, -5, 27, -17, -13, -13, -2, 0,
```

```
  4, -17, 52, 9, 28, 21, 8, 28, -5, -2,
  3, 4, -23, 0, 2, -19, 12, 16, 12, 20,
  -26, 14, 28, 1, 1, -2, -10, -10, -4, 19,
  69, -17, -12, -18, 2, -11, 13, 0, 10, 19,
  -18, 9, -10, -5, -1, -7, 12, 23, -8, 22,
  24, -7, -18, 5, 12, 8, 10, 10, 14, 14,
  -16, -3, 21, -6, -3, 0, 18, -23, -4, 2,
  -24, 10, 36, 2, 3, 6, -19, -13, -11, -7,
  -19, 17, 24, -4, 8, 41, -11, 20, -56, -14,
  32, 2, 32, 3, -12, 8, 10, -10, 15, -14,
  14, 15, -19, 0, 10, 7, -2, 10, -1, 4,
  3, -10, 12, 11, -2, 4, -6, 20, 18, -1,
  55, 10, 9, -14, -11, -3, 21, -5, 22, 21,
  10, -9, 12, -8, -7, -11, -1, 31, 13, 2,
  33, -11, -11, -24, 13, 8, -6, -15, -37, 4,
  11, 14, 15, -3, -14, -12, -21, -13, 7, 4,
  -16, -16, -30, -5, -15, 16, -7, 28, 9, -9,
  0, 4, 24, -11, 7, -11, 16, 8, 7, -4,
  12, 3, -30, 7, -12, 9, -4, -10, 6, -6,
  -5, -5, -29, 20, -7, 15, 16, 13, -19, 10,
  -10, -18, 4, -5, 7, -47, 19, 26, -21, -19,
  32, -2, 32, -5, -18, 3, 16, 0, 16, -13,
  7, 20, 44, -5, -4, 18, 15, -19, 2, 7,
  58, -4, -15, -19, -21, -3, -2, -4, -24, 16,
  48, 9, 5, 1, -19, -4, 5, -14, 3, 22,
  -2, -19, 8, -14, 18, 15, 24, 28, 12, 1,
  -21, -10, 15, -10, 11, 31, -15, 34, 1, -4,
  -127, -7, 11, 18, 2, -11, 5, -87, -10, -7,
  -22, 3, 3, -16, -21, 0, -15, 29, 13, 12,
  7, 12, 5, -21, 12, 13, 6, -21, -15, 6,
  0, -3, -18, -23, -19, 1, 4, -20, -7, -11,
  -18, 14, -4, 6, -17, -10, 9, 8, 7, 16,
  28, 6, 28, -1, -25, -7, -12, -24, 10, 4,
  20, 15, -25, -2, 17, -9, -23, -1, -8, 17,
  -14, -13, -16, -3, -8, -13, -10, -9, -15, -2,
  3, -2, 3, 21, 31, 47, -7, 29, 5, -7,
  22, -10, 1, 35, 7, -13, 0, 2, -33, 21,
  -9, -9, 34, -24, 4, -6, 13, -30, 15, -9,
  13, 18, -18, 13, -13, 13, -5, 25, 6, -9,
  47, -4, -17, 22, 15, -7, -14, 1, -5, -14,
  -12, 3, -16, 5, -6, 26, -14, 28, 8, -6,
  -13, 10, -18, 4, 13, 12, -20, -5, -22, 14,
  -6, 7, 4, 7, 9, 1, 17, -5, 21, -12,
  -15, 7, -34, 16, 7, 13, -6, -15, -22, 21,
  -4, -20, 0, 2, 8, 2, 19, -11, 17, 5,
  5, 18, 31, 1, -6, -7, 21, -19, 5, 9,
};
const TfArray<2, int> tensor_dimension7 = { 2, { 50,10 } };
const TfArray<1, float> quant7_scale = { 1, { 0.016756035387516022, } };
const TfArray<1, int> quant7_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant7 = { (TfLiteFloatArray*)&quant7_scale,
(TfLiteIntArray*)&quant7_zero, 0 };
const ALIGN(8) int8_t tensor_data8[6*50] = {
  47, -8, -14, 9, 16, -1, -11, 14, -9, 9, -15, -15, -45, 7, 0, 3, 19, 21,
25, -16, -8, -10, 16, -26, -20, 7, -17, -33, -20, 12, 6, -3, -32, 18, 9, 8,
-20, 19, -11, 25, -34, -6, -5, 0, 0, 18, 6, 14, 3, -24,
```

```
  -42, 8, -19, 15, 21, -12, 2, -24, -39, -15, 15, 11, 36, -3, -14, -16, -
16, -56, -18, -17, 19, -3, -8, 3, -13, -22, -5, -13, -11, 1, 41, 5, -127,
1, 0, -18, -21, -18, 2, 28, 0, 15, 9, 1, 25, -17, 14, -13, 12, 7,
  -21, -14, -16, -25, -23, 14, -7, 9, 11, 15, 12, -11, -54, 12, 2, -19, 15,
-19, 18, -8, 5, -18, 14, 16, -21, -11, -18, 23, 20, -22, -10, 6, -11, -11,
1, 14, 4, -1, 0, -2, 5, 3, 6, 30, -31, 13, 0, 23, -1, -9,
  -66, -27, -55, -21, -41, 2, -39, -30, -24, -37, -13, 67, 70, -30, -24, -
30, -36, -19, -9, -13, -43, -3, -26, -33, 58, -35, -30, 59, -22, -51, 40, -
20, -39, -10, -27, 5, -19, -25, 9, -29, 30, -17, -47, 81, -51, 2, 30, -7, -
24, -4,
  -26, -21, 11, -26, 23, 6, -21, -1, 21, -2, 0, -43, -11, -14, -4, 18, 0,
34, 6, -32, 11, 1, -11, 4, 10, -15, -7, -48, -18, 27, -5, 4, 65, -7, 2, 0,
-16, -18, -3, 12, -19, -6, 11, -33, 5, 1, 1, -11, 5, -9,
  1, 15, -16, -13, -2, -5, -12, 15, -7, -12, 23, 4, -45, 19, -10, 2, 22, -
2, -40, 1, -6, 12, -14, 5, -5, 11, 15, -29, 10, 3, -12, -13, 7, -2, -6, 18,
20, -31, 4, -35, -38, 17, -10, -7, -13, 1, 18, -30, 9, 20,
};
const TfArray<2, int> tensor_dimension8 = { 2, { 6,50 } };
const TfArray<1, float> quant8_scale = { 1, { 0.017898241057991982, } };
const TfArray<1, int> quant8_zero = { 1, { 0 } };
const TfLiteAffineQuantization quant8 = { (TfLiteFloatArray*)&quant8_scale,
(TfLiteIntArray*)&quant8_zero, 0 };
const TfArray<2, int> tensor_dimension9 = { 2, { 1,20 } };
const TfArray<1, float> quant9_scale = { 1, { 0.060452912002801895, } };
const TfArray<1, int> quant9_zero = { 1, { -128 } };
const TfLiteAffineQuantization quant9 = { (TfLiteFloatArray*)&quant9_scale,
(TfLiteIntArray*)&quant9_zero, 0 };
const TfArray<2, int> tensor_dimension10 = { 2, { 1,10 } };
const TfArray<1, float> quant10_scale = { 1, { 0.040221620351076126, } };
const TfArray<1, int> quant10_zero = { 1, { -128 } };
const TfLiteAffineQuantization quant10 = {
(TfLiteFloatArray*)&quant10_scale, (TfLiteIntArray*)&quant10_zero, 0 };
const TfArray<2, int> tensor_dimension11 = { 2, { 1,50 } };
const TfArray<1, float> quant11_scale = { 1, { 0.047950834035873413, } };
const TfArray<1, int> quant11_zero = { 1, { -128 } };
const TfLiteAffineQuantization quant11 = {
(TfLiteFloatArray*)&quant11_scale, (TfLiteIntArray*)&quant11_zero, 0 };
const TfArray<2, int> tensor_dimension12 = { 2, { 1,6 } };
const TfArray<1, float> quant12_scale = { 1, { 0.24192795157432556, } };
const TfArray<1, int> quant12_zero = { 1, { 56 } };
const TfLiteAffineQuantization quant12 = {
(TfLiteFloatArray*)&quant12_scale, (TfLiteIntArray*)&quant12_zero, 0 };
const TfArray<2, int> tensor_dimension13 = { 2, { 1,6 } };
const TfArray<1, float> quant13_scale = { 1, { 0.00390625, } };
const TfArray<1, int> quant13_zero = { 1, { -128 } };
const TfLiteAffineQuantization quant13 = {
(TfLiteFloatArray*)&quant13_scale, (TfLiteIntArray*)&quant13_zero, 0 };
const TfLiteFullyConnectedParams opdata0 = { kTfLiteActRelu,
kTfLiteFullyConnectedWeightsFormatDefault, false, false };
const TfArray<3, int> inputs0 = { 3, { 0,5,1 } };
const TfArray<1, int> outputs0 = { 1, { 9 } };
const TfLiteFullyConnectedParams opdata1 = { kTfLiteActRelu,
kTfLiteFullyConnectedWeightsFormatDefault, false, false };
const TfArray<3, int> inputs1 = { 3, { 9,6,2 } };
const TfArray<1, int> outputs1 = { 1, { 10 } };
```

```
 const TfLiteFullyConnectedParams opdata2 = { kTfLiteActRelu,
kTfLiteFullyConnectedWeightsFormatDefault, false, false };
const TfArray<3, int> inputs2 = { 3, { 10,7,3 } };
const TfArray<1, int> outputs2 = { 1, { 11 } };
const TfLiteFullyConnectedParams opdata3 = { kTfLiteActNone,
kTfLiteFullyConnectedWeightsFormatDefault, false, false };
const TfArray<3, int> inputs3 = { 3, { 11,8,4 } };
const TfArray<1, int> outputs3 = { 1, { 12 } };
const TfLiteSoftmaxParams opdata4 = { 1 };
const TfArray<1, int> inputs4 = { 1, { 12 } };
const TfArray<1, int> outputs4 = { 1, { 13 } };
const TensorInfo_t tensorData[] = {
  { kTfLiteArenaRw, kTfLiteInt8, tensor_arena + 0,
(TfLiteIntArray*)&tensor_dimension0, 33, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant0))}, },
  { kTfLiteMmapRo, kTfLiteInt32, (void*)tensor_data1,
(TfLiteIntArray*)&tensor_dimension1, 80, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant1))}, },
  { kTfLiteMmapRo, kTfLiteInt32, (void*)tensor_data2,
(TfLiteIntArray*)&tensor_dimension2, 40, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant2))}, },
  { kTfLiteMmapRo, kTfLiteInt32, (void*)tensor_data3,
(TfLiteIntArray*)&tensor_dimension3, 200, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant3))}, },
  { kTfLiteMmapRo, kTfLiteInt32, (void*)tensor_data4,
(TfLiteIntArray*)&tensor_dimension4, 24, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant4))}, },
  { kTfLiteMmapRo, kTfLiteInt8, (void*)tensor_data5,
(TfLiteIntArray*)&tensor_dimension5, 660, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant5))}, },
  { kTfLiteMmapRo, kTfLiteInt8, (void*)tensor_data6,
(TfLiteIntArray*)&tensor_dimension6, 200, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant6))}, },
  { kTfLiteMmapRo, kTfLiteInt8, (void*)tensor_data7,
(TfLiteIntArray*)&tensor_dimension7, 500, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant7))}, },
  { kTfLiteMmapRo, kTfLiteInt8, (void*)tensor_data8,
(TfLiteIntArray*)&tensor_dimension8, 300, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant8))}, },
  { kTfLiteArenaRw, kTfLiteInt8, tensor_arena + 48,
(TfLiteIntArray*)&tensor_dimension9, 20, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant9))}, },
  { kTfLiteArenaRw, kTfLiteInt8, tensor_arena + 80,
(TfLiteIntArray*)&tensor_dimension10, 10, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant10))}, },
  { kTfLiteArenaRw, kTfLiteInt8, tensor_arena + 0,
(TfLiteIntArray*)&tensor_dimension11, 50, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant11))}, },
  { kTfLiteArenaRw, kTfLiteInt8, tensor_arena + 64,
(TfLiteIntArray*)&tensor_dimension12, 6, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant12))}, },
  { kTfLiteArenaRw, kTfLiteInt8, tensor_arena + 0,
(TfLiteIntArray*)&tensor_dimension13, 6, {kTfLiteAffineQuantization,
const_cast<void*>(static_cast<const void*>(&quant13))}, },
};const NodeInfo_t nodeData[] = {
```

```
  { (TfLiteIntArray*)&inputs0, (TfLiteIntArray*)&outputs0,
const_cast<void*>(static_cast<const void*>(&opdata0)), OP_FULLY_CONNECTED,
},
  { (TfLiteIntArray*)&inputs1, (TfLiteIntArray*)&outputs1,
const_cast<void*>(static_cast<const void*>(&opdata1)), OP_FULLY_CONNECTED,
},
  { (TfLiteIntArray*)&inputs2, (TfLiteIntArray*)&outputs2,
const_cast<void*>(static_cast<const void*>(&opdata2)), OP_FULLY_CONNECTED,
},
  { (TfLiteIntArray*)&inputs3, (TfLiteIntArray*)&outputs3,
const_cast<void*>(static_cast<const void*>(&opdata3)), OP_FULLY_CONNECTED,
},
  { (TfLiteIntArray*)&inputs4, (TfLiteIntArray*)&outputs4,
const_cast<void*>(static_cast<const void*>(&opdata4)), OP_SOFTMAX, },
};
static std::vector<void*> overflow_buffers;
static TfLiteStatus AllocatePersistentBuffer(struct TfLiteContext* ctx,
                                             size_t bytes, void** ptr)
{
  if (current_location - bytes < tensor_boundary) {
    // OK, this will look super weird, but.... we have CMSIS-NN buffers
which
    // we cannot calculate beforehand easily.
    *ptr = malloc(bytes);
    if (*ptr == NULL) {
      printf("ERR: Failed to allocate persistent buffer of size %d\n",
(int)bytes);
      return kTfLiteError;
    }
    overflow_buffers.push_back(*ptr);
    return kTfLiteOk;
  }

  current_location -= bytes;

  *ptr = current_location;
  return kTfLiteOk;
}
typedef struct {
  size_t bytes;
  void *ptr;
} scratch_buffer_t;
static std::vector<scratch_buffer_t> scratch_buffers;

static TfLiteStatus RequestScratchBufferInArena(struct TfLiteContext* ctx,
size_t bytes,
                                                int* buffer_idx) {
  scratch_buffer_t b;
  b.bytes = bytes;

  TfLiteStatus s = AllocatePersistentBuffer(ctx, b.bytes, &b.ptr);
  if (s != kTfLiteOk) {
    return s;
  }

  scratch_buffers.push_back(b);
```

```
  *buffer_idx = scratch_buffers.size() - 1;

  return kTfLiteOk;
}

static void* GetScratchBuffer(struct TfLiteContext* ctx, int buffer_idx) {
  if (buffer_idx > static_cast<int>(scratch_buffers.size()) - 1) {
    return NULL;
  }
  return scratch_buffers[buffer_idx].ptr;
}
} // namespace

  TfLiteStatus trained_model_init( void*(*alloc_fnc)(size_t,size_t) ) {
#ifdef EI_CLASSIFIER_ALLOCATION_HEAP
  tensor_arena = (uint8_t*) alloc_fnc(16, kTensorArenaSize);
  if (!tensor_arena) {
    printf("ERR: failed to allocate tensor arena\n");
    return kTfLiteError;
  }
#endif
  tensor_boundary = tensor_arena;
  current_location = tensor_arena + kTensorArenaSize;
  ctx.AllocatePersistentBuffer = &AllocatePersistentBuffer;
  ctx.RequestScratchBufferInArena = &RequestScratchBufferInArena;
  ctx.GetScratchBuffer = &GetScratchBuffer;
  ctx.tensors = tflTensors;
  ctx.tensors_size = 14;
  for(size_t i = 0; i < 14; ++i) {
    tflTensors[i].type = tensorData[i].type;
    tflTensors[i].is_variable = 0;

#if defined(EI_CLASSIFIER_ALLOCATION_HEAP)
    tflTensors[i].allocation_type = tensorData[i].allocation_type;
#else
    tflTensors[i].allocation_type = (tensor_arena <= tensorData[i].data &&
tensorData[i].data < tensor_arena + kTensorArenaSize) ? kTfLiteArenaRw :
kTfLiteMmapRo;
#endif
    tflTensors[i].bytes = tensorData[i].bytes;
    tflTensors[i].dims = tensorData[i].dims;

#if defined(EI_CLASSIFIER_ALLOCATION_HEAP)
    if(tflTensors[i].allocation_type == kTfLiteArenaRw){
      uint8_t* start = (uint8_t*) ((uintptr_t)tensorData[i].data +
(uintptr_t) tensor_arena);

     tflTensors[i].data.data =  start;
    }
    else{
       tflTensors[i].data.data = tensorData[i].data;
    }
#else
    tflTensors[i].data.data = tensorData[i].data;
#endif // EI_CLASSIFIER_ALLOCATION_HEAP
```

```
    tflTensors[i].quantization = tensorData[i].quantization;
    if (tflTensors[i].quantization.type == kTfLiteAffineQuantization) {
      TfLiteAffineQuantization const* quant = ((TfLiteAffineQuantization
const*)(tensorData[i].quantization.params));
      tflTensors[i].params.scale = quant->scale->data[0];
      tflTensors[i].params.zero_point = quant->zero_point->data[0];
    }
    if (tflTensors[i].allocation_type == kTfLiteArenaRw) {
      auto data_end_ptr = (uint8_t*)tflTensors[i].data.data +
tensorData[i].bytes;
      if (data_end_ptr > tensor_boundary) {
        tensor_boundary = data_end_ptr;
      }
    }
  }
  if (tensor_boundary > current_location /* end of arena size */) {
    printf("ERR: tensor arena is too small, does not fit model - even
without scratch buffers\n");
    return kTfLiteError;
  }
  registrations[OP_FULLY_CONNECTED] =
*tflite::ops::micro::Register_FULLY_CONNECTED();
  registrations[OP_SOFTMAX] = *tflite::ops::micro::Register_SOFTMAX();

  for(size_t i = 0; i < 5; ++i) {
    tflNodes[i].inputs = nodeData[i].inputs;
    tflNodes[i].outputs = nodeData[i].outputs;
    tflNodes[i].builtin_data = nodeData[i].builtin_data;
    tflNodes[i].custom_initial_data = nullptr;
    tflNodes[i].custom_initial_data_size = 0;
    if (registrations[nodeData[i].used_op_index].init) {
      tflNodes[i].user_data =
registrations[nodeData[i].used_op_index].init(&ctx, (const
char*)tflNodes[i].builtin_data, 0);
    }
  }
  for(size_t i = 0; i < 5; ++i) {
    if (registrations[nodeData[i].used_op_index].prepare) {
      TfLiteStatus status =
registrations[nodeData[i].used_op_index].prepare(&ctx, &tflNodes[i]);
      if (status != kTfLiteOk) {
        return status;
      }
    }
  }
  return kTfLiteOk;
}

static const int inTensorIndices[] = {
  0,
};
TfLiteTensor* trained_model_input(int index) {
  return &ctx.tensors[inTensorIndices[index]];
}

static const int outTensorIndices[] = {
```

```
  13,
};
TfLiteTensor* trained_model_output(int index) {
  return &ctx.tensors[outTensorIndices[index]];
}

TfLiteStatus trained_model_invoke() {
  for(size_t i = 0; i < 5; ++i) {
    TfLiteStatus status =
registrations[nodeData[i].used_op_index].invoke(&ctx, &tflNodes[i]);

#if EI_CLASSIFIER_PRINT_STATE
    ei_printf("layer %lu\n", i);
    ei_printf("    inputs:\n");
    for (size_t ix = 0; ix < tflNodes[i].inputs->size; ix++) {
      auto d = tensorData[tflNodes[i].inputs->data[ix]];

      size_t data_ptr = (size_t)d.data;

      if (d.allocation_type == kTfLiteArenaRw) {
        data_ptr = (size_t)tensor_arena + data_ptr;
      }

      if (d.type == TfLiteType::kTfLiteInt8) {
        int8_t* data = (int8_t*)data_ptr;
        ei_printf("        %lu (%zu bytes, ptr=%p, alloc_type=%d, type=%d):
", ix, d.bytes, data, (int)d.allocation_type, (int)d.type);
        for (size_t jx = 0; jx < d.bytes; jx++) {
          ei_printf("%d ", data[jx]);
        }
      }
      else {
        float* data = (float*)data_ptr;
        ei_printf("        %lu (%zu bytes, ptr=%p, alloc_type=%d, type=%d):
", ix, d.bytes, data, (int)d.allocation_type, (int)d.type);
        for (size_t jx = 0; jx < d.bytes / 4; jx++) {
          ei_printf("%f ", data[jx]);
        }
      }
      ei_printf("\n");
    }
    ei_printf("\n");

    ei_printf("    outputs:\n");
    for (size_t ix = 0; ix < tflNodes[i].outputs->size; ix++) {
      auto d = tensorData[tflNodes[i].outputs->data[ix]];

      size_t data_ptr = (size_t)d.data;

      if (d.allocation_type == kTfLiteArenaRw) {
        data_ptr = (size_t)tensor_arena + data_ptr;
      }

      if (d.type == TfLiteType::kTfLiteInt8) {
        int8_t* data = (int8_t*)data_ptr;
```

```
        ei_printf("        %lu (%zu bytes, ptr=%p, alloc_type=%d, type=%d):
", ix, d.bytes, data, (int)d.allocation_type, (int)d.type);
        for (size_t jx = 0; jx < d.bytes; jx++) {
          ei_printf("%d ", data[jx]);
        }
      }
      else {
        float* data = (float*)data_ptr;
        ei_printf("        %lu (%zu bytes, ptr=%p, alloc_type=%d, type=%d):
", ix, d.bytes, data, (int)d.allocation_type, (int)d.type);
        for (size_t jx = 0; jx < d.bytes / 4; jx++) {
          ei_printf("%f ", data[jx]);
        }
      }
      ei_printf("\n");
    }
    ei_printf("\n");
#endif // EI_CLASSIFIER_PRINT_STATE

    if (status != kTfLiteOk) {
      return status;
    }
  }
  return kTfLiteOk;
}

TfLiteStatus trained_model_reset( void (*free_fnc)(void* ptr) ) {
#ifdef EI_CLASSIFIER_ALLOCATION_HEAP
  free_fnc(tensor_arena);
#endif
  scratch_buffers.clear();
  for (size_t ix = 0; ix < overflow_buffers.size(); ix++) {
    free(overflow_buffers[ix]);
  }
  overflow_buffers.clear();
  return kTfLiteOk;
}
```

5.2 Hojas de Características

5.2.1 XL6009

XLSEMI

XL6009

400KHz 60V 4A Switching Current Boost / Buck-Boost / Inverting DC/DC Converter

Features

- Wide 5V to 32V Input Voltage Range
- Positive or Negative Output Voltage Programming with a Single Feedback Pin
- Current Mode Control Provides Excellent Transient Response
- 1.25V reference adjustable version
- Fixed 400KHz Switching Frequency
- Maximum 4A Switching Current
- SW PIN Built in Over Voltage Protection
- Excellent line and load regulation
- EN PIN TTL shutdown capability
- Internal Optimize Power MOSFET
- High efficiency up to 94%
- Built in Frequency Compensation
- Built in Soft-Start Function
- Built in Thermal Shutdown Function
- Built in Current Limit Function
- Available in TO263-5L package

Applications

- EPC / Notebook Car Adapter
- Automotive and Industrial Boost / Buck-Boost / Inverting Converters
- Portable Electronic Equipment

General Description

The XL6009 regulator is a wide input range, current mode, DC/DC converter which is capable of generating either positive or negative output voltages. It can be configured as either a boost, flyback, SEPIC or inverting converter. The XL6009 built in N-channel power MOSFET and fixed frequency oscillator, current-mode architecture results in stable operation over a wide range of supply and output voltages.

The XL6009 regulator is special design for portable electronic equipment applications.

TO263-5L

Figure1. Package Type of XL6009

Pin Configurations

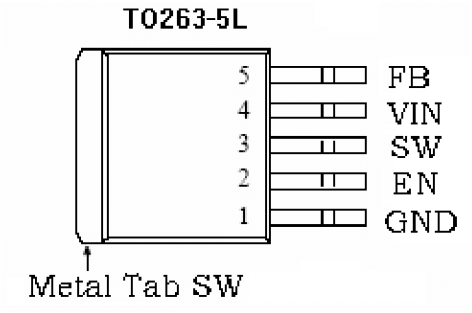


Figure2. Pin Configuration of XL6009 (Top View)

Table 1 Pin Description

Pin Number	Pin Name	Description
1	GND	Ground Pin.
2	EN	Enable Pin. Drive EN pin low to turn off the device, drive it high to turn it on. Floating is default high.
3	SW	Power Switch Output Pin (SW).
4	VIN	Supply Voltage Input Pin. XL6009 operates from a 5V to 32V DC voltage. Bypass Vin to GND with a suitably large capacitor to eliminate noise on the input.
5	FB	Feedback Pin (FB). Through an external resistor divider network, FB senses the output voltage and regulates it. The feedback threshold voltage is 1.25V.

Function Block

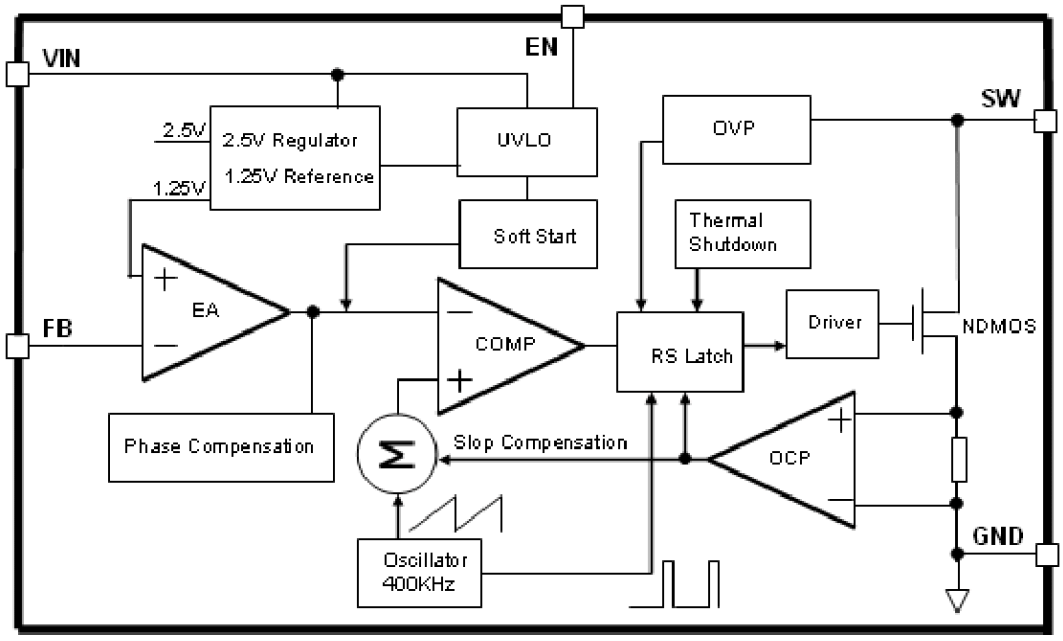


Figure3. Function Block Diagram of XL6009

Typical Application Circuit

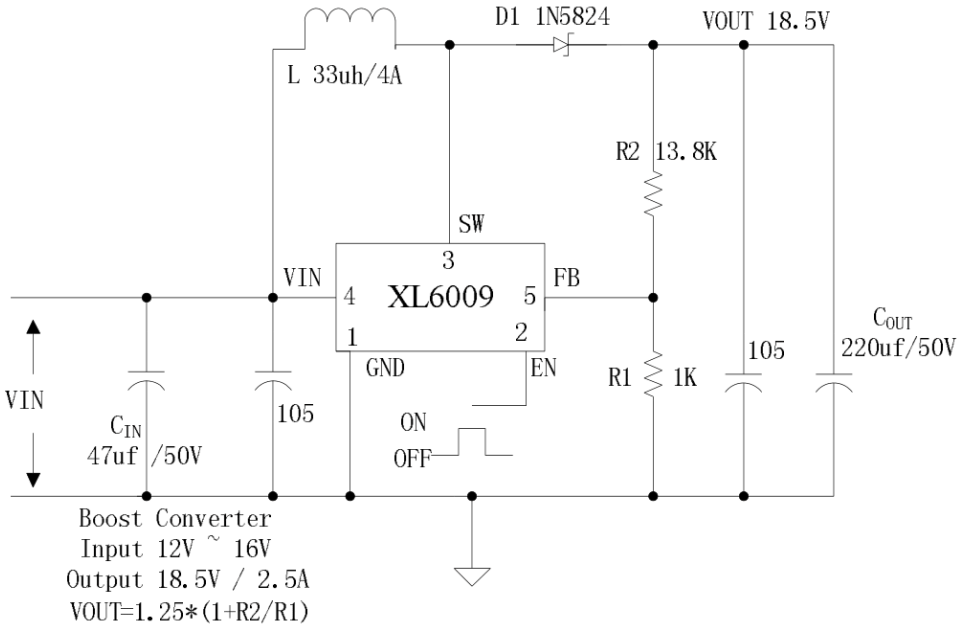


Figure4. XL6009 Typical Application Circuit (Boost Converter)

Ordering Information

Package	Temperature Range	Part Number	Marking ID	Packing Type
		Lead Free	Lead Free	
		XL6009E1	XL6009E1	Tube
		XL6009TRE1	XL6009E1	Tape & Reel

XLSEMI Pb-free products, as designated with "E1" suffix in the par number, are RoHS compliant.

Absolute Maximum Ratings（Note1）

Parameter	Symbol	Value	Unit
Input Voltage	Vin	-0.3 to 36	V
Feedback Pin Voltage	V_{FB}	-0.3 to Vin	V
EN Pin Voltage	V_{EN}	-0.3 to Vin	V
Output Switch Pin Voltage	V_{Output}	-0.3 to 60	V
Power Dissipation	P_D	Internally limited	mW
Thermal Resistance (TO263-5L) (Junction to Ambient, No Heatsink, Free Air)	R_{JA}	30	°C/W
Operating Junction Temperature	T_J	-40 to 125	°C
Storage Temperature	T_{STG}	-65 to 150	°C
Lead Temperature (Soldering, 10 sec)	T_{LEAD}	260	°C
ESD (HBM)		>2000	V

Note1: Stresses greater than those listed under Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

XL6009 Electrical Characteristics

T_a = 25℃;unless otherwise specified.

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Unit
System parameters test circuit figure4						
VFB	Feedback Voltage	Vin = 12V to 16V, Vout=18V Iload=0.1A to 2A	1.213	1.25	1.287	V
Efficiency	ŋ	Vin=12V ,Vout=18.5V Iout=2A	-	92	-	%

Electrical Characteristics (DC Parameters)

Vin = 12V, GND=0V, Vin & GND parallel connect a 220uf/50V capacitor; Iout=0.5A, T_a = 25℃; the others floating unless otherwise specified.

Parameters	Symbol	Test Condition	Min.	Typ.	Max.	Unit
Input operation voltage	Vin		5		32	V
Shutdown Supply Current	I_{STBY}	V_{EN}=0V		70	100	uA
Quiescent Supply Current	I_q	V_{EN} =2V, V_{FB} =Vin		2.5	5	mA
Oscillator Frequency	Fosc		320	400	480	Khz
Switch Current Limit	I_L	V_{FB} =0		4		A
Output Power NMOS	Rdson	Vin=12V, I_{SW}=4A		110	120	mohm
EN Pin Threshold	V_{EN}	High (Regulator ON) Low (Regulator OFF)		1.4 0.8		V
EN Pin Input Leakage Current	I_H	V_{EN} =2V (ON)		3	10	uA
	I_L	V_{EN} =0V (OFF)		3	10	uA
Max. Duty Cycle	D_{MAX}	V_{FB}=0V		90		%

Schottky Diode Selection Table

Current	Surface Mount	Through Hole	VR (The same as system maximum input voltage)				
			20V	30V	40V	50V	60V
1A		√	1N5817	1N5818	1N5819		
3A		√	1N5820	1N5821	1N5822		
		√	MBR320	MBR330	MBR340	MBR350	MBR360
	√		SK32	SK33	SK34	SK35	SK36
	√			30WQ03	30WQ04	30WQ05	
		√		31DQ03	31DQ04	31DQ05	
		√	SR302	SR303	SR304	SR305	SR306
5A		√	1N5823	1N5824	1N5825		
		√	SR502	SR503	SR504	SR505	SR506
		√	SB520	SB530	SB540	SB550	SB560
	√			50WQ03	50WQ04	50WQ05	

Typical System Application for EPC/Notebook Car Adapter – Boost (Output 18.5V/2.5A)

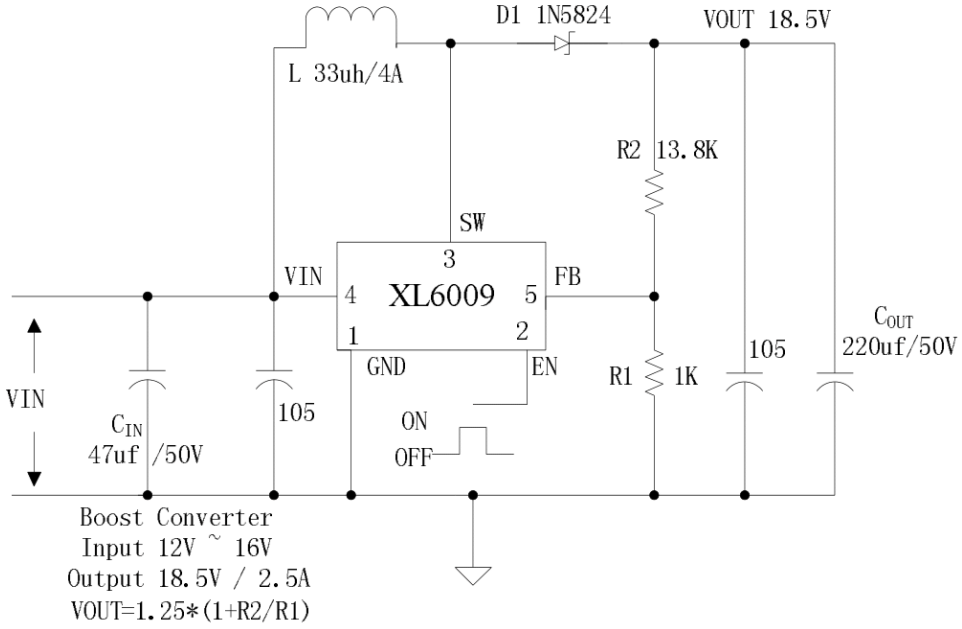


Figure5. XL6009 Typical System Application (Boost Converter)

Typical System Application for Portable Notebook Car Adapter

– SEPIC Buck-Boost Topology (Input 10V~30V, Output 12V/2A)

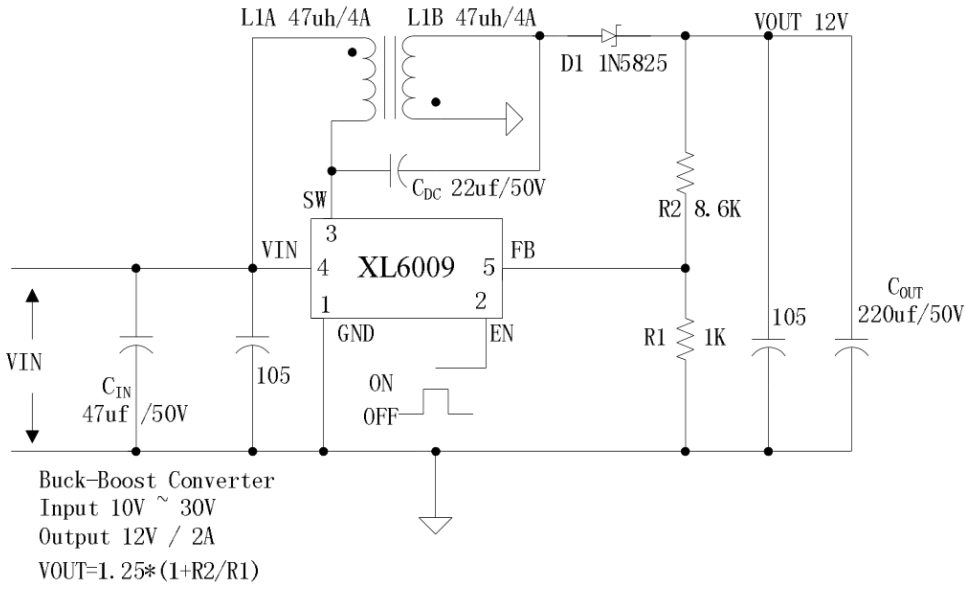


Figure6. XL6009 Typical System Application (SEPIC Buck-Boost Converter)

Typical System Application for Inverting Converter

– SEPIC Inverting Topology (Input 10V~30V, Output + -12V/1A)

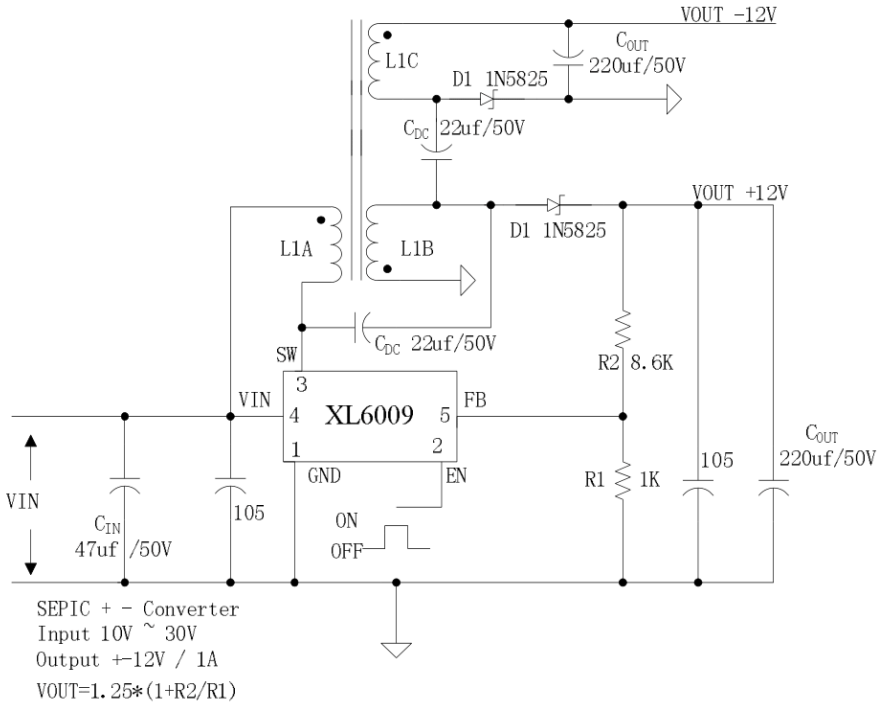
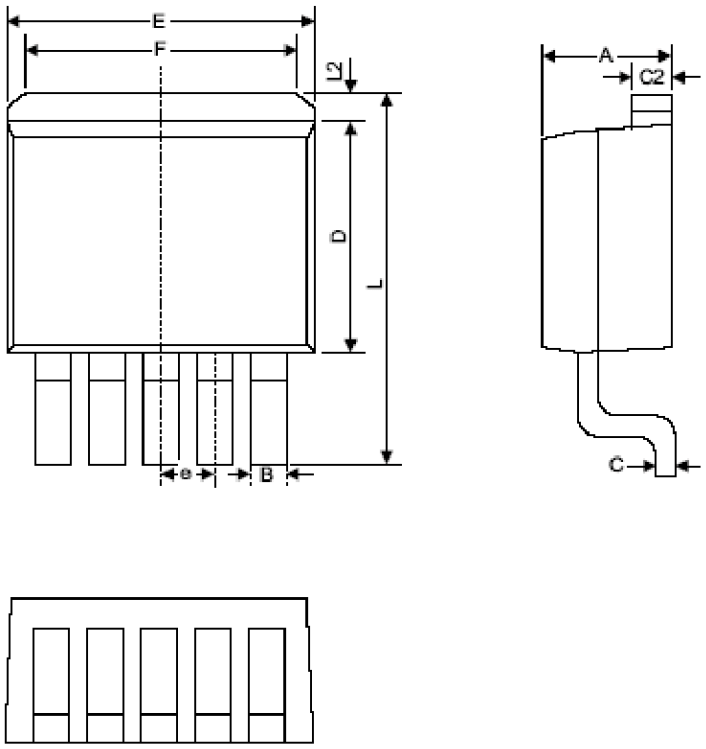


Figure7. XL6009 Typical System Application (SEPIC Inverting Converter)

Package Information

TO263-5L



Symbol	Dimensions In Millimeters		Dimensions In Inches	
	Min	Max	Min	Max
A	4.440	4.650	0.175	0.183
B	0.710	0.970	0.028	0.038
C	0.360	0.640	0.014	0.025
C2	1.255	1.285	0.049	0.051
D	8.390	8.890	0.330	0.350
E	9.960	10.360	0.392	0.408
e	1.550	1.850	0.061	0.073
F	6.360	7.360	0.250	0.290
L	13.950	14.750	0.549	0.581
L2	1.120	1.420	0.044	0.056

5.2.2 TP 4056

南京拓微集成电路有限公司
NanJing Top Power ASIC Corp.

TP4056 1A Standalone Linear Li-Ion Battery Charger with Thermal Regulation in SOP-8

DESCRIPTION

The TP4056 is a complete constant-current/constant-voltage linear charger for single cell lithium-ion batteries. Its SOP package and low external component count make the TP4056 ideally suited for portable applications. Furthermore, the TP4056 can work within USB and wall adapter.

No blocking diode is required due to the internal PMOSFET architecture and have prevent to negative Charge Current Circuit. Thermal feedback regulates the charge current to limit the die temperature during high power operation or high ambient temperature. The charge voltage is fixed at 4.2V, and the charge current can be programmed externally with a single resistor. The TP4056 automatically terminates the charge cycle when the charge current drops to 1/10th the programmed value after the final float voltage is reached.

TP4056 Other features include current monitor, under voltage lockout, automatic recharge and two status pin to indicate charge termination and the presence of an input voltage.

FEATURES

- Programmable Charge Current Up to 1000mA
- No MOSFET, Sense Resistor or Blocking Diode Required
- Complete Linear Charger in SOP-8 Package for Single Cell Lithium-Ion Batteries
- Constant-Current/Constant-Voltage
- Charges Single Cell Li-Ion Batteries Directly from USB Port
- Preset 4.2V Charge Voltage with 1.5% Accuracy
- Automatic Recharge
- two Charge Status Output Pins
- C/10 Charge Termination
- 2.9V Trickle Charge Threshold (TP4056)
- Soft-Start Limits Inrush Current
- Available Radiator in 8-Lead SOP Package, the Radiator need connect GND or impending

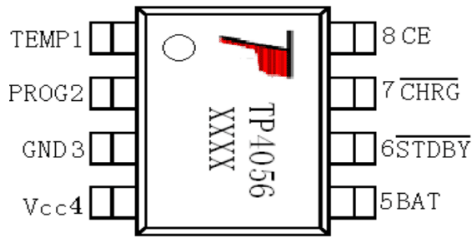
ABSOLUTE MAXIMUM RATINGS

- Input Supply Voltage(V_{CC})：-0.3V～8V
- TEMP：-0.3V～10V
- CE：-0.3V～10V
- BAT Short-Circuit Duration：Continuous
- BAT Pin Current：1200mA
- PROG Pin Current：1200uA
- Maximum Junction Temperature：145℃
- Operating Ambient Temperature Range：-40℃～85℃
- Lead Temp.(Soldering, 10sec)：260℃

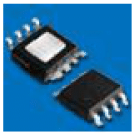
APPLICATIONS

- Cellular Telephones, PDAs, GPS
- Charging Docks and Cradles
- Digital Still Cameras, Portable Devices
- USB Bus-Powered Chargers,Chargers

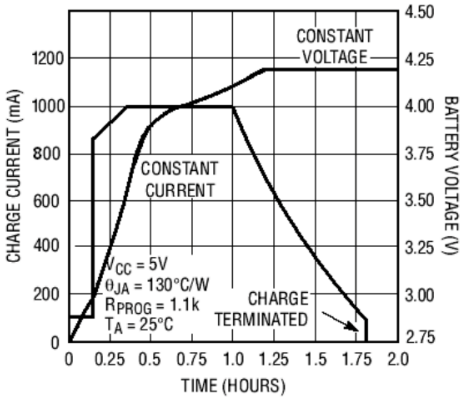
PACKAGE/ORDER INFORMATION



SOP－8

photo	ORDER PART NUMBER
	TP4056-42-SOP8-PP
	PART MARKING TP4056

Complete Charge Cycle (1000mAh Battery)



TEMP(Pin 1) :Temperature Sense Input Connecting TEMP pin to NTC thermistor's output in Lithium ion battery pack. If TEMP pin's voltage is below 45% or above 80% of supply voltage V_{IN} for more than 0.15S, this means that battery's temperature is too high or too low, charging is suspended. The temperature sense function can be disabled by grounding the TEMP pin.

PROG(Pin 2): Constant Charge Current Setting and Charge Current Monitor Pin charge current is set by connecting a resistor R_{ISET} from this pin to GND. When in precharge mode, the ISET pin's voltage is regulated to 0.2V. When in constant charge current mode, the ISET pin's voltage is regulated to 2V.In all modes during charging, the voltage on ISET pin can be used to measure the charge current as follows:

$$I_{BAT} = \frac{V_{PROG}}{R_{PROG}} \times 1200 \quad (V_{PROG}=1V)$$

GND(Pin3): Ground Terminal

Vcc(Pin 4): Positive Input Supply Voltage V_{IN} is the power supply to the internal circuit. When V_{IN} drops to within 30mv of the BAT pin voltage, TP4056 enters low power sleep mode, dropping BAT pin's current to less than 2uA.

BAT(Pin5): Battery Connection Pin. Connect the positive terminal of the battery to BAT pin. BAT pin draws less than 2uA current in chip disable mode or in sleep mode. BAT pin provides charge current to the battery and provides regulation voltage of 4.2V.

$\overline{\text{STDBY}}$(Pin6): Open Drain Charge Status Output When the battery Charge Termination, the $\overline{\text{STDBY}}$ pin is pulled low by an internal switch, otherwise $\overline{\text{STDBY}}$ pin is in high impedance state.

$\overline{\text{CHRG}}$ (Pin7): Open Drain Charge Status Output When the battery is being charged, the $\overline{\text{CHRG}}$ pin is pulled low by an internal switch, otherwise $\overline{\text{CHRG}}$ pin is in high impedance state.

CE(Pin8): Chip Enable Input. A high input will put the device in the normal operating mode. Pulling the CE pin to low level will put the YP4056 into disable mode. The CE pin can be driven by TTL or CMOS logic level.

ELECTRICAL CHARACTERISTICS

The ● denotes specifications which apply over the full operating temperature range, otherwise specifications are at **T_A=25℃， V_{CC} =5V**， unless otherwise noted.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
V_{CC}	Input Supply Voltage		●	4.0	5	8.0	V
I_{CC}	Input Supply Current	Charge Mode, R_{PROG} = 1.2k	●		150	500	μA
		StandbyMode(Charge Terminated)	●		55	100	μA
		Shutdown Mode (R_{PROG} Not Connected,V_{CC} < V_{BAT}, or V_{CC} < V_{UV})	●		55	100	μA
V_{FLOAL}	Regulated Output (Float) Voltage	0℃≤T_A≤85℃， I_{BAT}=40mA		4.137	4.2	4.263	V
I_{BAT}	BAT Pin Current	R_{PROG} = 2.4k, Current Mode	●	450	500	550	mA
	Text condition:VBAT=4.0V	R_{PROG} = 1.2k, Current Mode	●	950	1000	1050	mA
		Standby Mode, V_{BAT} = 4.2V	●	0	−2.5	−6	μA
I_{TRIKL}	Trickle Charge Current	V_{BAT}<V_{TRIKL}, R_{PROG}=1.2K	●	120	130	140	mA
V_{TRIKL}	Trickle Charge Threshold Voltage	R_{PROG}=1.2K, V_{BAT} Rising		2.8	2.9	3.0	V
V_{TRHYS}	Trickle Charge Hysteresis Voltage	R_{PROG}=1.2K		60	80	100	mV
T_{LIM}	Junction Temperature in Constant Temperature Mode				145		℃

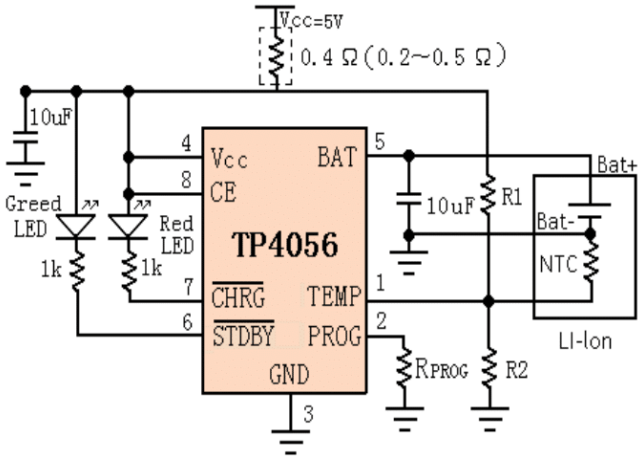
indicator light state

Charge state	Red LED $\overline{CHRG}$	Greed LED $\overline{STDBY}$
charging	bright	extinguish
Charge Termination	extinguish	bright
Vin too low; Temperature of battery too low or too high; no battery	extinguish	extinguish
BAT PIN Connect 10u Capacitance; No battery	Greed LED bright，Red LED Coruscate T=1-4 S	

Rprog Current Setting

R_{PROG} (k)	I_{BAT} (mA)
10	130
5	250
4	300
3	400
2	580
1.66	690
1.5	780
1.33	900
1.2	1000

TYPICAL APPLICATIONS



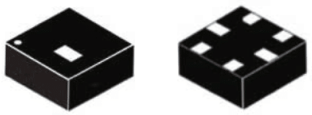
5.2.3 HTS 221

life.augmented

HTS221

Capacitive digital sensor for relative humidity and temperature

Datasheet - production data



HLGA-6L
(2 x 2 x 0.9 mm)

Features

- 0 to 100% relative humidity range
- Supply voltage: 1.7 to 3.6 V
- Low power consumption: 2 µA @ 1 Hz ODR
- Selectable ODR from 1 Hz to 12.5 Hz
- High rH sensitivity: 0.004% rH/LSB
- Humidity accuracy: ± 3.5% rH, 20 to +80% rH
- Temperature accuracy: ± 0.5 °C,15 to +40 °C
- Embedded 16-bit ADC
- 16-bit humidity and temperature output data
- SPI and I²C interfaces
- Factory calibrated
- Tiny 2 x 2 x 0.9 mm package
- ECOPACK® compliant

Applications

- Air conditioning, heating and ventilation
- Air humidifiers
- Refrigerators
- Wearable devices
- Smart home automation
- Industrial automation
- Respiratory equipment
- Asset and goods tracking

Description

The HTS221 is an ultra-compact sensor for relative humidity and temperature. It includes a sensing element and a mixed signal ASIC to provide the measurement information through digital serial interfaces.

The sensing element consists of a polymer dielectric planar capacitor structure capable of detecting relative humidity variations and is manufactured using a dedicated ST process.

The HTS221 is available in a small top-holed cap land grid array (HLGA) package guaranteed to operate over a temperature range from -40 °C to +120 °C.

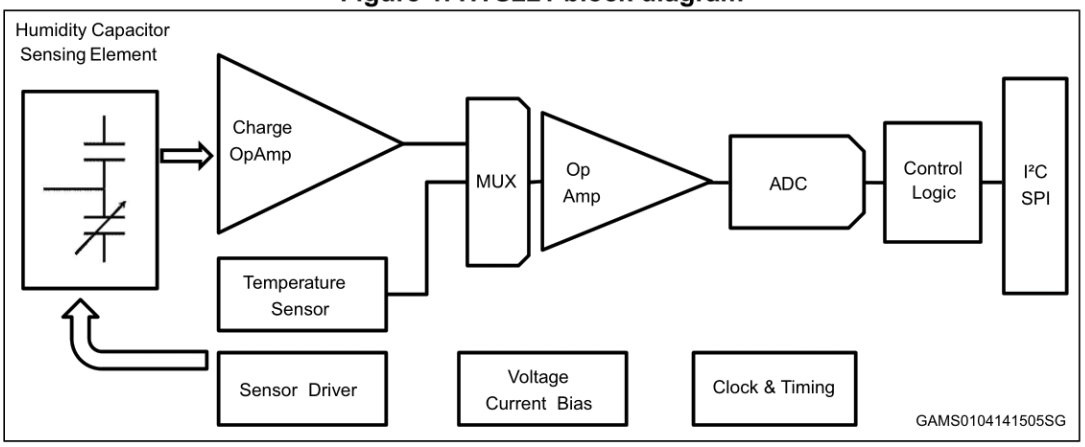
Table 1. Device summary

Order code	Temperature range [°C]	Package	Packing
HTS221TR	-40 to +120	HLGA-6L	Tape and reel

August 2016 DocID026333 Rev 4
This is information on a product in full production. www.st.com

1 HTS221 block diagram

Figure 1. HTS221 block diagram



1.1 Pin information

Figure 2. Pin configuration (bottom view)

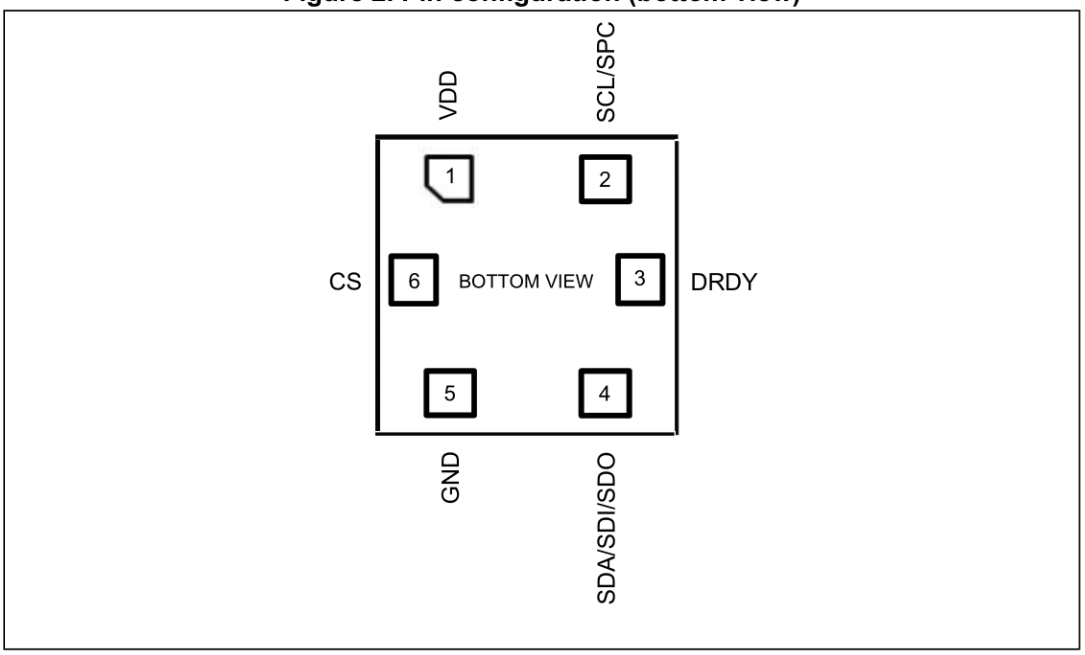


Table 2. Pin description

Pin n°	Name	Function
1	V_{DD}	Power supply
2	SCL/SPC	I²C serial clock (SCL) SPI serial port clock (SPC)
3	DRDY	Data Ready output signal
4	SDA/SDI/SDO	I²C serial data (SDA) 3-wire SPI serial data input /output (SDI/SDO)
5	GND	Ground
6	SPI enable	I²C/SPI mode selection (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)

2 Sensor parameters and electrical specifications

Conditions at V_{DD} = 2.5 V, T = 25 °C, unless otherwise noted.

Table 3. Humidity and temperature parameter specifications

Symbol	Parameter	Test condition	Min.	Typ.(1)	Max.	Unit
Hop	Operating humidity range		0	–	100	% rH
Hbit	Humidity output data			16	–	bit
Hs	Humidity sensitivity			0.004		%rH/LSB
				256		LSB/%rH
Hacc	Humidity accuracy(2)	20 to 80% rH		±3.5		% rH
		0 to 100% rH		±5		
Hnoise	Humidity noise(3)			0.03		RMS
Hhys	Humidity hysteresis			±1		% rH
Hstep	Humidity response time(4)	t @ 63%		10		s
Hdrift	Humidity long-term drift	20 to 80% rH		0.5		%rH/yr
Top	Operating temperature range		-40	–	120	°C
Tbit	Temperature output data		–	16	–	bit
Ts	Temperature sensitivity			0.016		°C/LSB
				64		LSB/°C
Tacc	Temperature accuracy	15 to 40 °C		±0.5		°C
		0 to 60 °C		±1		
Tnoise	Temperature noise(3)			0.007		RMS
Tstep	Temperature response time	t @ 63%		15		s
Tdrift	Temperature long-term drift	T = 0 to 80 °C			0.05	°C/yr
ODR	Humidity and temperature digital output data rate			1/7/12.5		Hz

1. Typical specifications are not guaranteed
2. Accuracy in non condensing environment including hysteresis
3. Default value; noise value can be modified by *AV_CONF (10h)*
4. Valid at 25 °C and 1 m/s airflow

Table 4. Electrical characteristics

Symbol	Parameter	Test condition	Min.	Typ.(1)	Max.	Unit
V_{DD}	Supply voltage		1.7	–	3.6	V
I_{DD}	Supply current (2)	1 Hz, 25 °C, 2.5 V		2		µA
$I_{DD}P_{DN}$	Supply current in power-down mode T = 25 °C	25 °C, 2.5 V	–	0.5	–	µA

1. Typical specifications are not guaranteed
2. Refer to *Table 16*.

2.1 Communication interface characteristics

2.1.1 SPI - serial peripheral interface

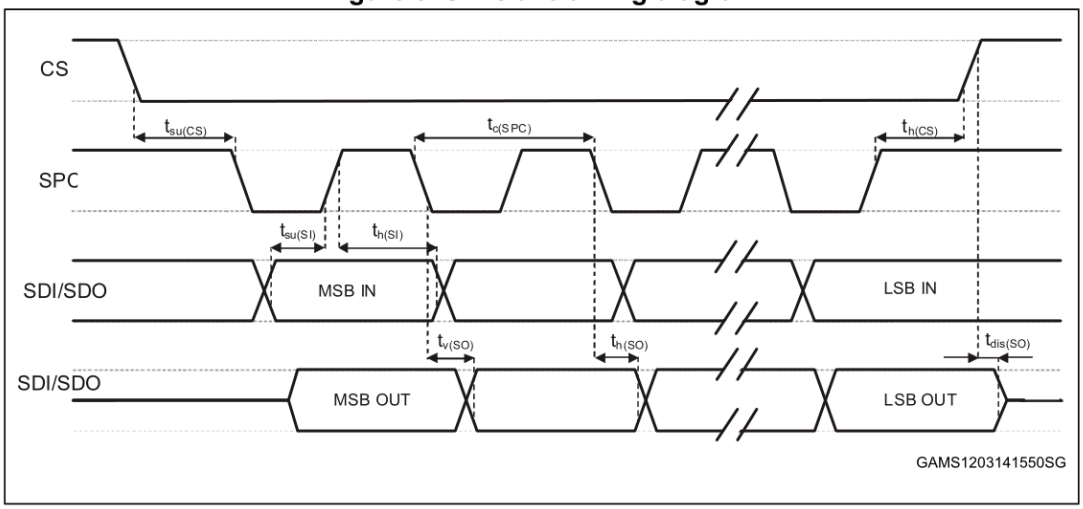
Subject to general operating conditions for V_{DD} and T_{OP}

Table 5. SPI slave timing values

Symbol	Parameter	Value (1)		Unit
		Min.	Max.	
$t_{c(SPC)}$	SPI clock cycle	100		ns
$f_{c(SPC)}$	SPI clock frequency		10	MHz
$t_{su(CS)}$	CS setup time	6		ns
$t_{h(CS)}$	CS hold time	8		
$t_{su(SI)}$	SDI input setup time	5		
$t_{h(SI)}$	SDI input hold time	15		
$t_{v(SO)}$	SDO valid output time		50	
$t_{h(SO)}$	SDO output hold time	9		
$t_{dis(SO)}$	SDO output disable time		50	

1. Values are guaranteed at 10 MHz clock frequency for SPI, based on characterization results, not tested in production.

Figure 3. SPI slave timing diagram



Note: Measurement points are done at $0.2 \cdot V_{DD}$ and $0.8 \cdot V_{DD}$, for both ports.

2.1.2 I²C - control interface

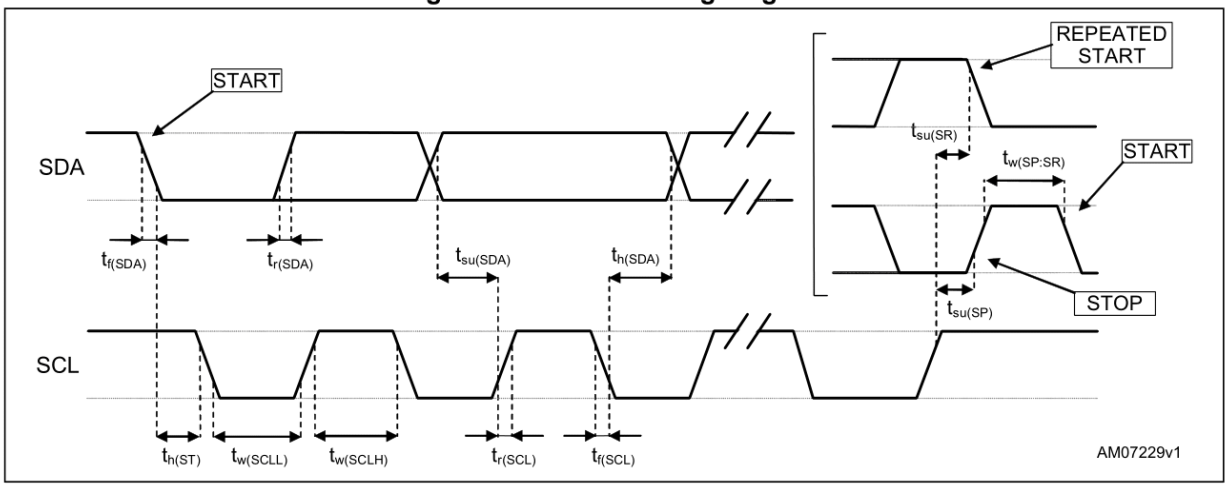
Subject to general operating conditions for V_{DD} and T_{OP}.

Table 6. I²C slave timing values

Symbol	Parameter (1)	I²C standard mode (1)		I²C fast mode (1)		Unit
		Min.	Max.	Min.	Max.	
$f_{(SCL)}$	SCL clock frequency	0	100	0	400	kHz
$t_{w(SCLL)}$	SCL clock low time	4.7		1.3		µs
$t_{w(SCLH)}$	SCL clock high time	4.0		0.6		
$t_{su(SDA)}$	SDA setup time	250		100		ns
$t_{h(SDA)}$	SDA data hold time	0.01	3.45	0	0.9	µs
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time		1000	$20 + 0.1C_b$ (2)	300	ns
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time		300	$20 + 0.1C_b$ (2)	300	
$t_{h(ST)}$	START condition hold time	4		0.6		µs
$t_{su(SR)}$	Repeated START condition setup time	4.7		0.6		
$t_{su(SP)}$	STOP condition setup time	4		0.6		
$t_{w(SP:SR)}$	Bus free time between STOP and START condition	4.7		1.3		

1. Data based on standard I²C protocol requirement, not tested in production.
2. C_b = total capacitance of one bus line, in pF.

Figure 4. I²C slave timing diagram



Note: Measurement points are done at 0.2·V_{DD} and 0.8·V_{DD}, for both ports.

2.2 Absolute maximum ratings

Stress above those listed as “Absolute maximum ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 7. Absolute maximum ratings

Symbol	Ratings	Maximum value	Unit
V_{DD}	Supply voltage	-0.3 to 4.8	V
V_{IN}	Input voltage on any control pin	-0.3 to V_{DD} +0.3	V
T_{STG}	Storage temperature range	-40 to +125	°C
ESD	Electrostatic discharge protection	2 (HBM)	kV

Note: Supply voltage on any pin should never exceed 4.8 V.

This device is sensitive to mechanical shock, improper handling can cause permanent damage to the part.

This device is sensitive to electrostatic discharge (ESD), improper handling can cause permanent damage to the part.

3 Functionality

The HTS221 is a digital humidity and temperature sensor, packaged in an HLGA holed package.The device includes the sensing element and an IC (integrated circuit) interface able to take information from the sensing element and provide a digital signal to the application, communicating through I²C/SPI interfaces with the host controller.

3.1 IC interface

The complete measurement chain consists of a low-noise capacitive amplifier, which converts the capacitive imbalance of the humidity sensor into an analog voltage signal, and an analog-to-digital converter is used to generate the digital information.

The converter is coupled with a dedicated hardware (HW) averaging filter to remove the high-frequency component and reduce the serial interface traffic.

The relative humidity and temperature data can be accessed through an I²C/SPI interface, making the device particularly suitable for direct interfacing with a microcontroller.

3.2 Factory calibration

The IC (integrated circuit) interface is factory calibrated and the coefficients required to convert the ADC 16-bit values into rH% or degrees Celsius can be read through the internal registers of the sensor. Further calibration by the user is not required.

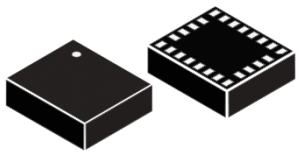
5.2.4 LSM9DS1

life.augmented

LSM9DS1

iNEMO inertial module:
3D accelerometer, 3D gyroscope, 3D magnetometer

Datasheet - production data



LGA-24L (3.5x3x1.0 mm)

Features

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
- ±2/±4/±8/±16 *g* linear acceleration full scale
- ±4/±8/±12/±16 gauss magnetic full scale
- ±245/±500/±2000 dps angular rate full scale
- 16-bit data output
- SPI / I^2C serial interfaces
- Analog supply voltage 1.9 V to 3.6 V
- “Always-on” eco power mode down to 1.9 mA
- Programmable interrupt generators
- Embedded temperature sensor
- Embedded FIFO
- Position and motion detection functions
- Click/double-click recognition
- Intelligent power saving for handheld devices
- ECOPACK®, RoHS and “Green” compliant

Applications

- Indoor navigation
- Smart user interfaces
- Advanced gesture recognition
- Gaming and virtual reality input devices
- Display/map orientation and browsing

Description

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.

The LSM9DS1 has a linear acceleration full scale of ±2*g*/±4*g*/±8/±16 *g,* a magnetic field full scale of ±4/±8/±12/±16 gauss and an angular rate of ±245/±500/±2000 dps.

The LSM9DS1 includes an I^2C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.

Magnetic, accelerometer and gyroscope sensing can be enabled or set in power-down mode separately for smart power management.

The LSM9DS1 is available in a plastic land grid array package (LGA) and it is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

Table 1. Device summary

Part number	Temperature range [°C]	Package	Packing
LSM9DS1	-40 to +85	LGA-24L	Tray
LSM9DS1TR	-40 to +85	LGA-24L	Tape and reel

This is information on a product in full production. www.st.com

1 Pin description

Figure 1. Pin connections

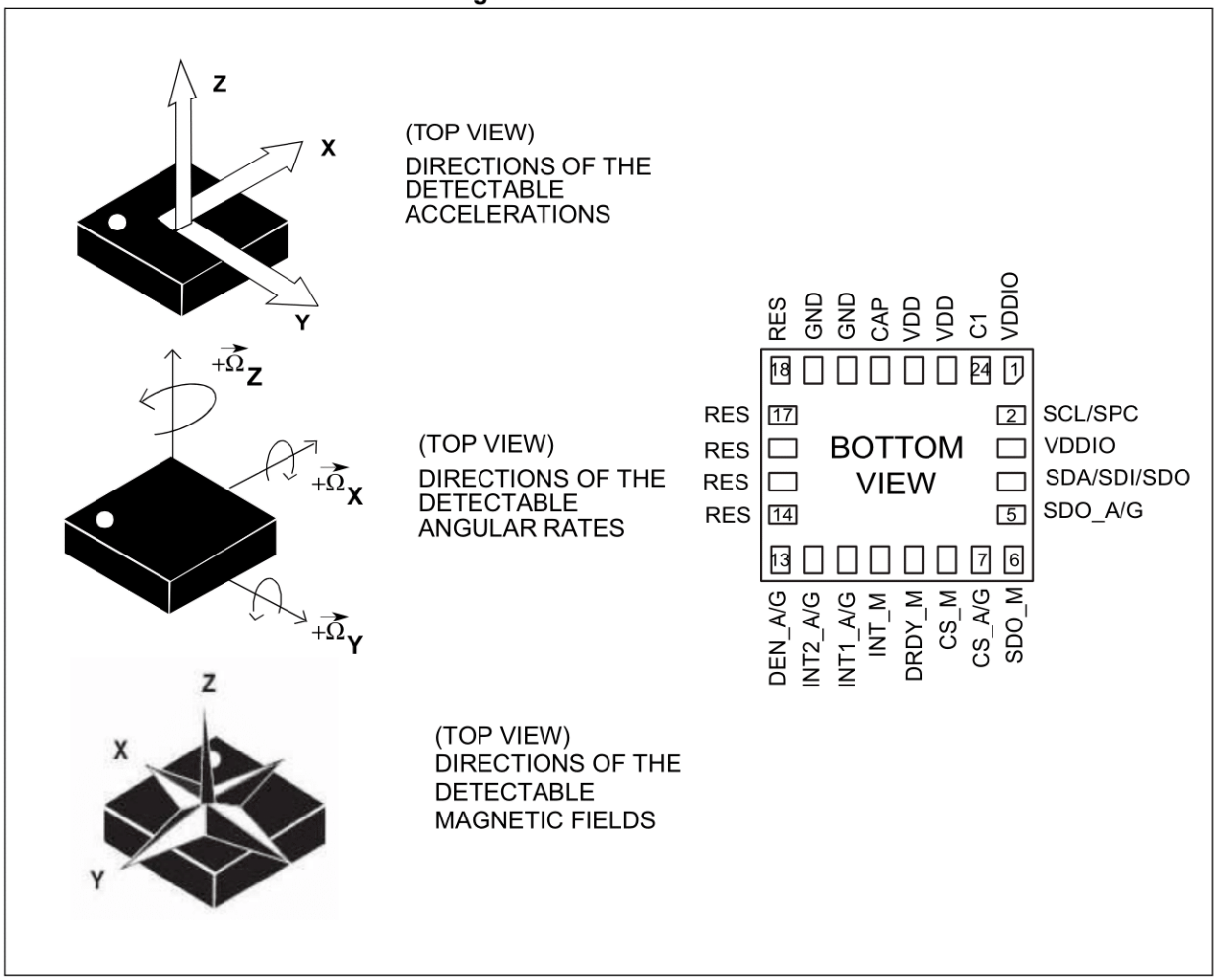


Table 2. Pin description

Pin #	Name	Function
1	VDDIO(1)	Power supply for I/O pins
2	SCL/SPC	I^2C serial clock (SCL) / SPI serial port clock (SPC)
3	VDDIO(2)	Power supply for I/O pins
4	SDA/SDI/SDO	I^2C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
5	SDO_A/G	SPI serial data output (SDO) for the accelerometer and gyroscope I^2C least significant bit of the device address (SA0) for the accelerometer and gyroscope
6	SDO_M	SPI serial data output (SDO) for the magnetometer I^2C least significant bit of the device address (SA0) for the magnetometer
7	CS_A/G	SPI enable I^2C/SPI mode selection for the accelerometer and gyroscope (1: SPI idle mode / I^2C communication enabled; 0: SPI communication mode / I^2C disabled)
8	CS_M	SPI enable I^2C/SPI mode selection for the magnetometer (1: SPI idle mode / I^2C communication enabled; 0: SPI communication mode / I^2C disabled)
9	DRDY_M	Magnetic sensor data ready
10	INT_M	Magnetic sensor interrupt
11	INT1_A/G	Accelerometer and gyroscope interrupt 1
12	INT2_A/G	Accelerometer and gyroscope interrupt 2
13	DEN_A/G	Accelerometer and gyroscope data enable
14	RES	Reserved. Connected to GND.
15	RES	Reserved. Connected to GND.
16	RES	Reserved. Connected to GND.
17	RES	Reserved. Connected to GND.
18	RES	Reserved. Connected to GND.
19	GND	0 V supply
20	GND	0 V supply
21	CAP	Connected to GND with ceramic capacitor(3)
22	VDD(4)	Power supply
23	VDD(5)	Power supply
24	C1	Capacitor connection (C1 = 100 nF)

1. Recommended 100 nF filter capacitor.
2. Recommended 100 nF filter capacitor.
3. 10 nF (±10%), 16 V. 1 nF minimum value has to be guaranteed under 11 V bias condition.
4. Recommended 100 nF plus 10 µF capacitors.
5. Recommended 100 nF plus 10 µF capacitors.

2 Module specifications

2.1 Sensor characteristics

@ Vdd = 2.2 V, T = 25 °C unless otherwise noted[a]

Table 3. Sensor characteristics

Symbol	Parameter	Test conditions	Min.	Typ.[1]	Max.	Unit
LA_FS	Linear acceleration measurement range			±2		*g*
				±4		
				±8		
				±16		
M_FS	Magnetic measurement range			±4		gauss
				±8		
				±12		
				±16		
G_FS	Angular rate measurement range			±245		dps
				±500		
				±2000		
LA_So	Linear acceleration sensitivity	Linear acceleration FS = ±2 *g*		0.061		mg/LSB
		Linear acceleration FS = ±4 *g*		0.122		
		Linear acceleration FS = ±8 *g*		0.244		
		Linear acceleration FS = ±16 *g*		0.732		
M_GN	Magnetic sensitivity	Magnetic FS = ±4 gauss		0.14		mgauss/ LSB
		Magnetic FS = ±8 gauss		0.29		
		Magnetic FS = ±12 gauss		0.43		
		Magnetic FS = ±16 gauss		0.58		
G_So	Angular rate sensitivity	Angular rate FS = ±245 dps		8.75		mdps/ LSB
		Angular rate FS = ±500 dps		17.50		
		Angular rate FS = ±2000 dps		70		
LA_TyOff	Linear acceleration typical zero-g level offset accuracy[2]	FS = ±8 *g*		±90		mg
M_TyOff	Zero-gauss level [3]	FS = ±4 gauss		±1		gauss
G_TyOff	Angular rate typical zero-rate level [4]	FS = ±2000 dps		±30		dps
M_DF	Magnetic disturbance field	Zero-gauss offset starts to degrade			50	gauss
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed
2. Typical zero-*g* level offset value after soldering
3. Typical zero-gauss level value after test and trimming
4. Typical zero rate level offset value after MSL3 preconditioning

a. The product is factory calibrated at 2.2 V. The operational power supply range is from 1.9 V to 3.6 V.

2.2 Electrical characteristics

@ Vdd = 2.2 V, T = 25 °C unless otherwise noted[b]

Table 4. Electrical characteristics

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
Vdd	Supply voltage		1.9		3.6	V
Vdd_IO	Module power supply for I/O		1.71		Vdd+0.1	
Idd_XM	Current consumption of the accelerometer and magnetic sensor in normal mode (2)			600		µA
Idd_G	Gyroscope current consumption in normal mode(3)			4.0		mA
Top	Operating temperature range		-40		+85	°C
Trise	Time for power supply rising(4)		0.01		100	ms
Twait	Time delay between Vdd_IO and Vdd(4)		0		10	ms

1. Typical specifications are not guaranteed
2. Magnetic sensor in high-resolution mode (ODR = 20 Hz), accelerometer sensor in normal mode, gyroscope in power-down mode
3. Accelerometer and magnetic sensor in power-down mode
4. Please refer to *Section 2.2.1: Recommended power-up sequence* for more details.

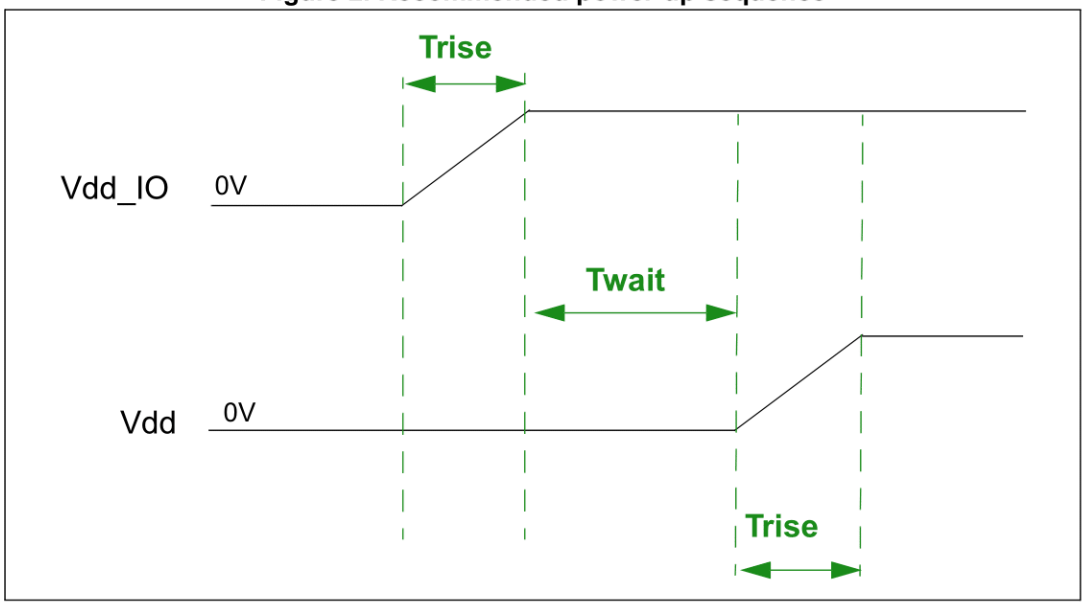
b. LSM9DS1 is factory calibrated at 2.2 V.

2.2.1 Recommended power-up sequence

For the power-up sequence please refer to the following figure, where:

- Trise is the time for the power supply to rise from 10% to 90% of its final value
- Twait is the delay between the end of the Vdd_IO ramp (90% of its final value) and the start of the Vdd ramp

Figure 2. Recommended power-up sequence



2.3 Temperature sensor characteristics

@ Vdd = 2.2 V, T = 25 °C unless otherwise noted [c]

Table 5. Temperature sensor characteristics

Symbol	Parameter	Test condition	Min.	Typ.(1)	Max.	Unit
TODR	Temperature refresh rate	Gyro OFF(2)		50		Hz
		Gyro ON		59.5		
TSen	Temperature sensitivity(3)			16		LSB/°C
Top	Operating temperature range		-40		+85	°C

1. Typical specifications are not guaranteed.
2. When the accelerometer ODR is set to 10 Hz and the gyroscope part is turned off, the TODR value is 10 Hz.
3. The output of the temperature sensor is 0 (typ.) at 25 °C

c. The product is factory calibrated at 2.2 V.

2.4 Communication interface characteristics

2.4.1 SPI - serial peripheral interface

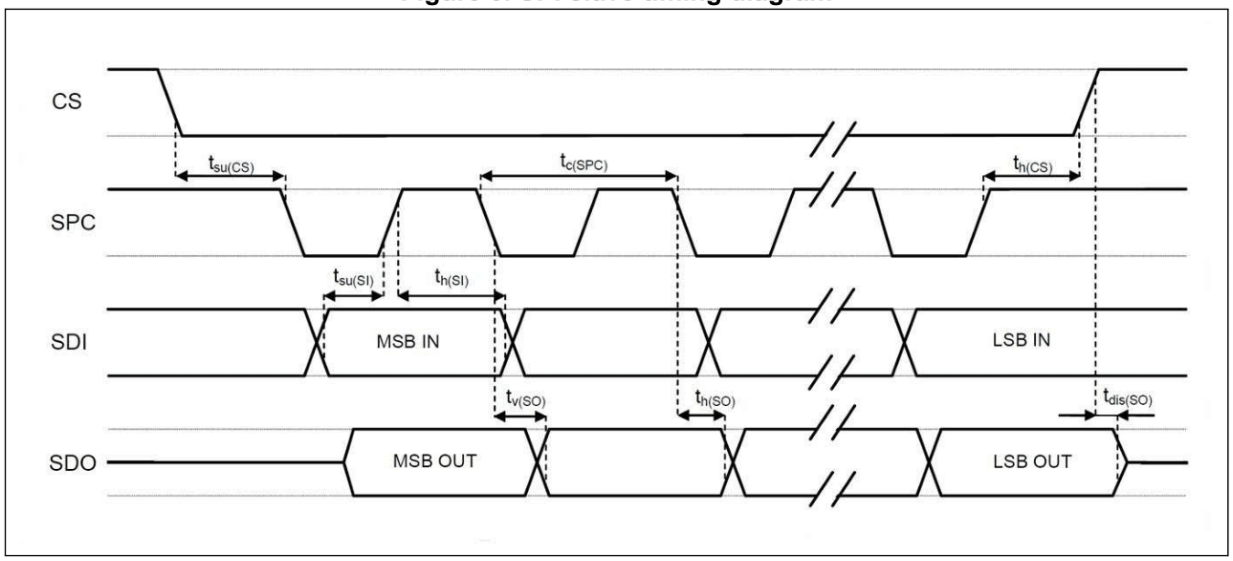
Subject to general operating conditions for Vdd and Top.

Table 6. SPI slave timing values

Symbol	Parameter	Value(1)		Unit
		Min	Max	
$t_{c(SPC)}$	SPI clock cycle	100		ns
$f_{c(SPC)}$	SPI clock frequency		10	MHz
$t_{su(CS)}$	CS setup time	5		ns
$t_{h(CS)}$	CS hold time	20		
$t_{su(SI)}$	SDI input setup time	5		
$t_{h(SI)}$	SDI input hold time	15		
$t_{v(SO)}$	SDO valid output time		50	
$t_{h(SO)}$	SDO output hold time	5		
$t_{dis(SO)}$	SDO output disable time		50	

1. Values are guaranteed at 10 MHz clock frequency for SPI with both 4 and 3 wires, based on characterization results, not tested in production

Figure 3. SPI slave timing diagram



Note: Measurement points are done at 0.2·Vdd_IO and 0.8·Vdd_IO, for both input and output ports.

2.4.2 I^2C - inter-IC control interface

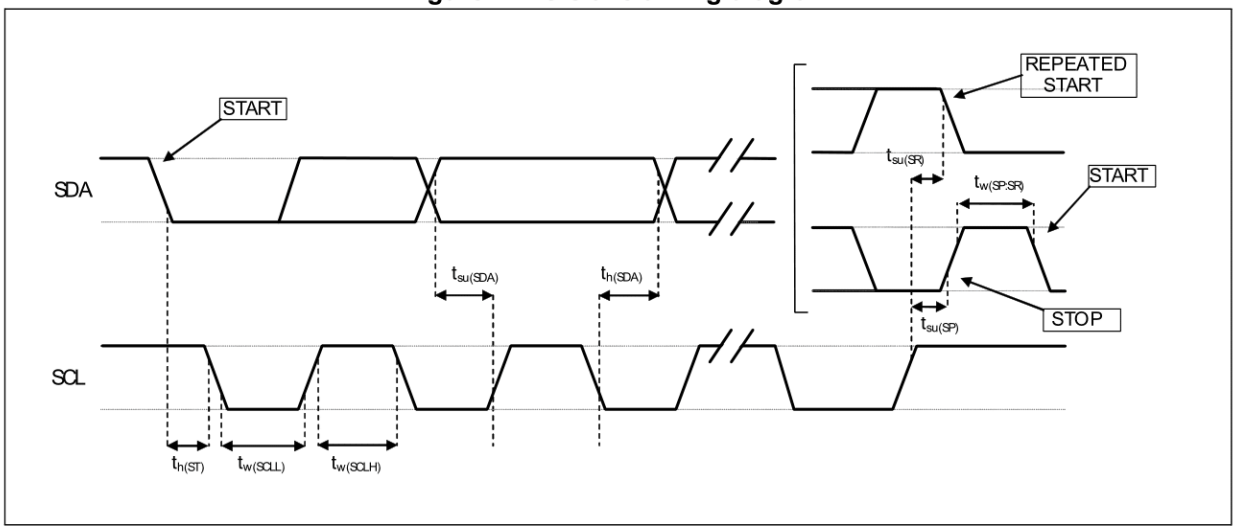
Subject to general operating conditions for Vdd and Top.

Table 7. I^2C slave timing values

Symbol	Parameter	I^2C Standard mode(1)		I^2C Fast mode (1)		Unit
		Min	Max	Min	Max	
$f_{(SCL)}$	SCL clock frequency	0	100	0	400	kHz
$t_{w(SCLL)}$	SCL clock low time	4.7		1.3		µs
$t_{w(SCLH)}$	SCL clock high time	4.0		0.6		
$t_{su(SDA)}$	SDA setup time	250		100		ns
$t_{h(SDA)}$	SDA data hold time	0	3.45	0	0.9	µs
$t_{h(ST)}$	START condition hold time	4		0.6		µs
$t_{su(SR)}$	Repeated START condition setup time	4.7		0.6		
$t_{su(SP)}$	STOP condition setup time	4		0.6		
$t_{w(SP:SR)}$	Bus free time between STOP and START condition	4.7		1.3		

1. Data based on standard I^2C protocol requirement, not tested in production.

Figure 4. I^2C slave timing diagram



Note: Measurement points are done at 0.2·Vdd_IO and 0.8·Vdd_IO, for both ports

2.5 Absolute maximum ratings

Stresses above those listed as “Absolute maximum ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device under these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 8. Absolute maximum ratings

Symbol	Ratings	Maximum value	Unit
Vdd	Supply voltage	-0.3 to 4.8	V
Vdd_IO	I/O pins supply voltage	-0.3 to 4.8	V
Vin	Input voltage on any control pin (including CS_A/G, CS_M, SCL/SPC, SDA/SDI/SDO, SDO_A/G, SDO_M)	0.3 to Vdd_IO +0.3	V
A_{UNP}	Acceleration (any axis)	3,000 for 0.5 ms	g
		10,000 for 0.1 ms	g
M_{EF}	Maximum exposed field	1000	gauss
ESD	Electrostatic discharge protection (HBM)	2	kV
T_{STG}	Storage temperature range	-40 to +125	°C

Note: Supply voltage on any pin should never exceed 4.8 V.

This device is sensitive to mechanical shock, improper handling can cause permanent damage to the part.

This device is sensitive to electrostatic discharge (ESD), improper handling can cause permanent damage to the part.

2.6 Terminology

2.6.1 Sensitivity

Linear acceleration sensitivity can be determined, for example, by applying 1 *g* acceleration to the device. Because the sensor can measure DC accelerations, this can be done easily by pointing the selected axis towards the ground, noting the output value, rotating the sensor 180 degrees (pointing towards the sky) and noting the output value again. By doing so, ±1 *g* acceleration is applied to the sensor. Subtracting the larger output value from the smaller one, and dividing the result by 2, leads to the actual sensitivity of the sensor. This value changes very little over temperature and over time. The sensitivity tolerance describes the range of sensitivities of a large number of sensors.

An angular rate gyroscope is device that produces a positive-going digital output for counterclockwise rotation around the axis considered. Sensitivity describes the gain of the sensor and can be determined by applying a defined angular velocity to it. This value changes very little over temperature and time.

Magnetic sensor sensitivity describes the gain of the sensor and can be determined, for example, by applying a magnetic field of 1 *gauss* to it.

2.6.2 Zero-*g*, zero-rate and zero-*gauss* level

Linear acceleration zero-*g* level offset (TyOff) describes the deviation of an actual output signal from the ideal output signal if no acceleration is present. A sensor in a steady state on a horizontal surface will measure 0 *g* on both the X-axis and Y-axis, whereas the Z-axis will measure 1 *g*. Ideally, the output is in the middle of the dynamic range of the sensor (content of OUT registers 00h, data expressed as two's complement number). A deviation from the ideal value in this case is called zero-*g* offset.

Offset is to some extent a result of stress to MEMS sensor and therefore the offset can slightly change after mounting the sensor onto a printed circuit board or exposing it to extensive mechanical stress. Offset changes little over temperature, see "Linear acceleration zero-*g* level change vs. temperature" in *Table 3*. The zero-*g* level tolerance (TyOff) describes the standard deviation of the range of zero-*g* levels of a group of sensors.

Zero-rate level describes the actual output signal if there is no angular rate present. The zero-rate level of precise MEMS sensors is, to some extent, a result of stress to the sensor and therefore the zero-rate level can slightly change after mounting the sensor onto a printed circuit board or after exposing it to extensive mechanical stress. This value changes very little over temperature and time.

Zero-*g*auss level offset (M_TyOff) describes the deviation of an actual output signal from the ideal output if no magnetic field is present.

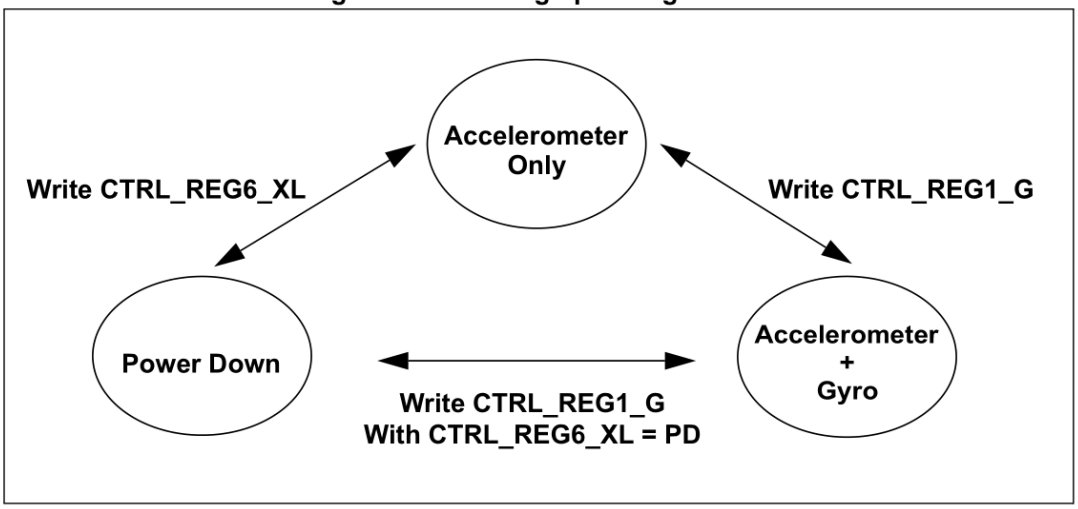
3 LSM9DS1 functionality

3.1 Operating modes

In the LSM9DS1 the accelerometer and gyroscope have two operating modes available: only accelerometer active and gyroscope in power down or both accelerometer and gyroscope sensors active at the same ODR. Switching from one mode to the other requires one write operation: writing to *CTRL_REG6_XL (20h)*, the accelerometer operates in normal mode and the gyroscope is powered down, writing to *CTRL_REG1_G (10h)* both accelerometer and gyroscope are activated at the same ODR.

Figure 5 depicts both modes of operation from power down.

Figure 5. Switching operating modes



The magnetic sensor has three operating modes available: power-down (default), continuous-conversion mode and single-conversion mode. Switching from power-down to the other modes requires one write operation to *CTRL_REG3_M (22h)*, setting values in the MD[1:0] bits. For the output of the magnetic data compensated by temperature, the TEMP_COMP bit in *CTRL_REG1_M (20h)* must be set to '1'.

3.2 Gyroscope power modes

In the LSM9DS1, the gyroscope can be configured in three different operating modes: power-down, low-power and normal mode.

Low-power mode is available for lower ODR (14.9, 59.5, 119 Hz) while for greater ODR (238, 476, 952 Hz) the device is automatically in normal mode. *Table* summarizes the ODR configuration (ODR_G[2:0] bits set in *CTRL_REG1_G (10h)*) and corresponding power modes.

To enable low-power mode, the LP_mode bit in *CTRL_REG3_G (12h)* has to be set to '1'.

Low-power mode allows reaching low power consumption while maintaining the device always on, refer to *Table 10*.

Table 9. Gyroscope operating modes

ODR_G [2:0]	ODR [Hz]	Power mode
000	Power down	Power-down
001	14.9	Low-power/Normal mode
010	59.5	Low-power/Normal mode
011	119	Low-power/Normal mode
100	238	Normal mode
101	476	Normal mode
110	952	Normal mode

Table 10. Operating mode current consumption

ODR [Hz]	Power mode	Current consumption(1) [mA]
14.9	Low-power	1.9
59.5	Low-power	2.4
119	Low-power	3.1
238	Normal mode	4.3
476	Normal mode	4.3
952	Normal mode	4.3

1. Typical values of gyroscope and accelerometer current consumption are based on characterization data.

Table 11. Accelerometer turn-on time

ODR [Hz]	BW = 400 Hz(1)	BW = 200 Hz(1)	BW = 100 Hz(1)	BW = 50 Hz(1)
14.9	0	0	0	0
59.5	0	0	0	0
119	1	1	1	2
238	1	1	2	4
476	1	2	4	7
952	2	4	7	14

1. The table contains the number of samples to be discarded after switching between power-down mode and normal mode.

Table 12. Gyroscope turn-on time

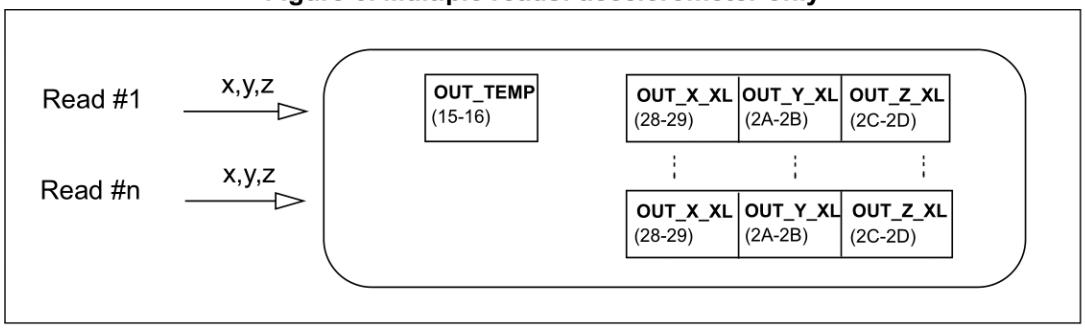
ODR [Hz]	LPF1 only(1)	LPF1 and LPF2(1)
14.9	2	LPF2 not available
59.5 or 119	3	13
238	4	14
476	5	15
952	8	18

1. The table contains the number of samples to be discarded after switching between low-power mode and normal mode.

3.3 Accelerometer and gyroscope multiple reads (burst)

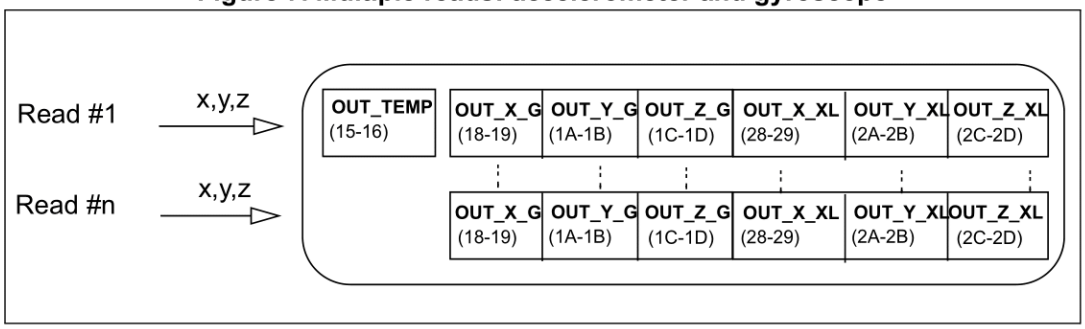
When only accelerometer is activated and the gyroscope is in power down, starting from *OUT_X_XL (28h - 29h)* multiple reads can be performed. Once *OUT_Z_XL (2Ch - 2Dh)* is read, the system automatically restarts from *OUT_X_XL (28h - 29h)* (see *Figure 6*).

Figure 6. Multiple reads: accelerometer only



When both accelerometer and gyroscope sensors are activated at the same ODR, starting from *OUT_X_G (18h - 19h)* multiple reads can be performed. Once *OUT_Z_XL (2Ch - 2Dh)* is read, the system automatically restarts from *OUT_X_G (18h - 19h)* (see *Figure 7*).

Figure 7. Multiple reads: accelerometer and gyroscope



3.4 Block diagram

Figure 8. Accelerometer and gyroscope digital block diagram

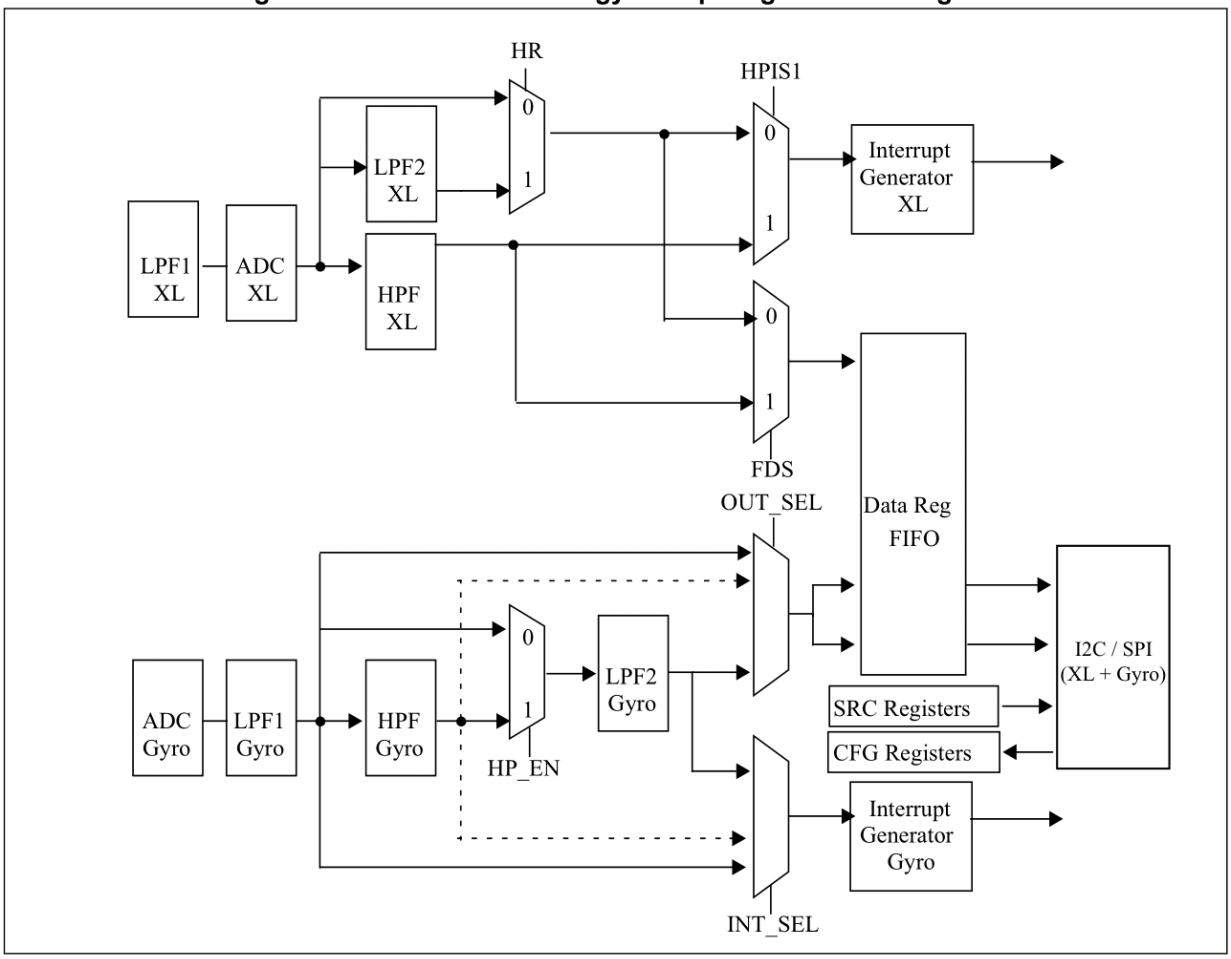


Figure 9. Magnetometer block diagram