

INFORME DEL TRABAJO PROFESIONAL



---

**Implementación de un guante electrónico con  
machine learning aplicado a la traducción de gestos  
de lengua de señas Argentina.**

---

13 de abril de 2022

Darius Ismael Maitia Petros - 95436

Jazmin Sofía Ferreiro Cedones - 97266

## **Reconocimientos**

El presente trabajo práctico profesional de final de carrera fue desarrollado bajo la supervisión del Ing. Pablo Deymonnaz como tutor y del Ing. Sebastián García Marra como co-tutor a lo largo de un periodo que se inició en Enero del 2021 hasta su finalización el 13 de Abril de 2022.

No queríamos dejar de agradecerles por habernos guiado y asesorado desinteresadamente a lo largo de todo este tiempo aportando un valioso conocimiento.

# Índice

|   |           |
|---|-----------|
| <b>I Propósito del trabajo práctico</b>                         | <b>8</b>  |
| <b>1. Resumen</b>   | <b>8</b>  |
| <b>2. Estado de situación</b>                                   | <b>8</b>  |
| 2.1. Relevancia . . . . .                                       | 9         |
| 2.2. Investigación . . . . .                                    | 11        |
| 2.2.1. Interpretación de gestos de la lengua de señas . . . . . | 11        |
| 2.2.2. Guante electrónico . . . . .                             | 13        |
| 2.2.3. Procesamiento de datos . . . . .                         | 15        |
| 2.2.4. Integración, despliegue y aprendizaje continuo . . . . . | 16        |
| 2.2.5. Rust embebido . . . . .                                  | 18        |
| <b>3. Alcance del proyecto</b>                                  | <b>19</b> |
| 3.1. Objetivo . . . . .   | 21        |
| 3.1.1. Subobjetivos . . . . .                                   | 21        |
| 3.2. Historias de usuario . . . . .                             | 22        |
| <b>II Implementación del trabajo</b>                            | <b>24</b> |
| <b>4. Implementación del hardware</b>                           | <b>24</b> |
| 4.1. Embebido . . . . .   | 24        |
| 4.2. Sensores . . . . .   | 25        |
| 4.3. Autonomía del dispositivo . . . . .                        | 27        |
| 4.4. Estructura del guante . . . . .                            | 32        |
| 4.5. Proceso de fabricación del guante . . . . .                | 33        |

|  |           |
|--|-----------|
| 4.5.1. Elementos necesarios . . . . .                      | 33        |
| 4.5.1.1. Herramientas: . . . . .                           | 33        |
| 4.5.1.2. Componentes electrónicos . . . . .                | 34        |
| 4.5.2. Preparación de los componentes . . . . .            | 35        |
| 4.5.2.1. Impresión de la carcasa . . . . .                 | 35        |
| 4.5.2.2. ESP32 . . . . .                                   | 35        |
| 4.5.2.3. MPU6050 . . . . .                                 | 35        |
| 4.5.2.4. MT3608 . . . . .                                  | 36        |
| 4.5.3. Preparación del cableado . . . . .                  | 36        |
| 4.5.4. Soldar los cables al ESP32 . . . . .                | 38        |
| 4.5.4.1. Pines AD0 . . . . .                               | 38        |
| 4.5.4.2. Pines SDA, SCL y Tensión . . . . .                | 38        |
| 4.5.4.3. Tierra (Ground) . . . . .                         | 39        |
| 4.5.4.4. Pin de tensión de 5V . . . . .                    | 39        |
| <b>5. Funcionalidades</b>                                  | <b>41</b> |
| 5.1. Calibración . . . . .                                 | 41        |
| 5.2. Recolección de datos . . . . .                        | 43        |
| 5.3. Interpretaciones . . . . .                            | 44        |
| <b>6. Implementación del software del sistema embebido</b> | <b>45</b> |
| 6.1. Communication/Ble . . . . .                           | 45        |
| 6.2. TasksManager . . . . .                                | 47        |
| 6.3. Glove . . . . .                                       | 50        |
| 6.4. Sensors . . . . .                                     | 53        |
| 6.5. Interpretation . . . . .                              | 56        |
| <b>7. Programación de la aplicación</b>                    | <b>60</b> |

|   |           |
|---|-----------|
| 7.1. Flutter . . . . .                                | 60        |
| 7.2. Interfaz de usuario . . . . .                    | 61        |
| 7.2.1. Navegación . . . . .                           | 61        |
| 7.2.2. Conexión y configuración . . . . .             | 62        |
| 7.2.3. Recolección de datos . . . . .                 | 65        |
| 7.2.4. Gestión de archivos . . . . .                  | 67        |
| 7.2.5. Visualizador de datos recolectados . . . . .   | 68        |
| 7.2.6. Visualizador de datos en tiempo real . . . . . | 70        |
| 7.2.7. Interpretaciones . . . . .                     | 71        |
| 7.2.8. Acerca de . . . . .                            | 72        |
| 7.3. Implementación del software de la app . . . . .  | 73        |
| 7.3.1. Comunicación bluetooth . . . . .               | 73        |
| 7.3.1.1. BluetoothSpecification . . . . .             | 74        |
| 7.3.1.2. BluetoothBackend . . . . .                   | 74        |
| 7.3.2. Representación de las mediciones . . . . .     | 78        |
| 7.3.3. Recolección de datos . . . . .                 | 78        |
| 7.3.3.1. Measurements_collector.dart . . . . .        | 78        |
| 7.3.4. Gestión de los archivos . . . . .              | 81        |
| 7.3.5. Mediciones recibidas . . . . .                 | 81        |
| 7.3.5.1. Formato de las mediciones . . . . .          | 82        |
| 7.3.5.2. Representación de las mediciones . . . . .   | 84        |
| 7.3.6. Navegación . . . . .                           | 86        |
| 7.3.7. Interpretaciones . . . . .                     | 86        |
| 7.4. Dependencias . . . . .                           | 87        |
| <b>8. Visualizador de datos en Jupyter</b>            | <b>88</b> |

|   |            |
|---|------------|
| <b>9. Machine Learning embebido</b>                         | <b>91</b>  |
| 9.1. Tensorflow Lite . . . . .                              | 92         |
| 9.2. Redes Neuronales . . . . .                             | 92         |
| 9.3. Evaluación del modelo . . . . .                        | 96         |
| 9.4. Alimentar el set de datos . . . . .                    | 98         |
| 9.5. Bloque de procesamiento inicial . . . . .              | 101        |
| 9.6. Entrenamiento . . . . .                                | 105        |
| 9.7. Depliegue continuo y aprendizaje continuo . . . . .    | 109        |
| 9.8. Resultados . . . . .                                   | 111        |
| <b>10. Gestión del proyecto</b>                             | <b>111</b> |
| <b>11. Desafíos</b>   | <b>115</b> |
| 11.1. Comunicación con los sensores . . . . .               | 115        |
| 11.1.1. Problemática . . . . .                              | 115        |
| 11.1.2. Solución . . . . .                                  | 115        |
| 11.2. Frecuencia . . . . .                                  | 118        |
| 11.3. Memoria . . . . .                                     | 120        |
| <b>12. Conclusiones</b>                                     | <b>122</b> |
| 12.1. Acerca del desarrollo del hardware . . . . .          | 122        |
| 12.2. Acerca de la programación de los embebidos . . . . .  | 123        |
| 12.3. Acerca del desarrollo de la aplicación . . . . .      | 125        |
| 12.4. Acerca del machine learning embebido . . . . .        | 126        |
| 12.5. Acerca del entrenamiento de la red neuronal . . . . . | 127        |
| 12.6. Acerca de la funcionalidad . . . . .                  | 128        |
| <b>A. Rust y el Esp32</b>                                   | <b>133</b> |
| A.1. Compilación cruzada . . . . .                          | 133        |

|   |            |
|---|------------|
| A.2. Pruebas de concepto . . . . .                  | 135        |
| A.3. Dependencias . . . . .                         | 136        |
| A.4. Hardware Abstraction Layer . . . . .           | 138        |
| A.5. Panic halt . . . . .                           | 139        |
| A.5.1. Despliegue . . . . .                         | 139        |
| A.6. Rust y lectura de sensores MPU6050 . . . . .   | 139        |
| A.7. Rust y wifi/bluetooth . . . . .                | 146        |
| A.8. Comandos AT . . . . .                          | 147        |
| A.8.1. Propuesta alternativa . . . . .              | 150        |
| <b>B. Gestación del proyecto</b>                    | <b>151</b> |
| B.1. Motivaciones . . . . .                         | 151        |
| B.2. Ideas de proyectos y consideraciones . . . . . | 151        |
| <b>C. Minutas de reunión</b>                        | <b>156</b> |
| <b>D. Repositorios</b>                              | <b>166</b> |

## Parte I

# Propósito del trabajo práctico

### 1. Resumen

Los gestos que se realizan con las manos son una parte natural de la comunicación humana. Están presentes en todos los idiomas. Sin embargo, para personas con discapacidad en el habla oral, los gestos no son solo una herramienta sino un medio de comunicación en lo que constituye la lengua de señas.

En el presente trabajo práctico profesional se intenta hacer uso de tecnología en auge como son la inteligencia artificial y los dispositivos IoT para ayudar a personas con alguna discapacidad en el habla. Se propone la implementación de un guante electrónico que mediante el uso de inteligencia artificial embebida en un microcontrolador permita la traducción de gestos a frases escritas o audibles con la ayuda de una aplicación de celular.

### 2. Estado de situación

En las últimas décadas hemos vivenciado un incremento de la capacidad del tráfico de datos a través de internet, así como también el abaratamiento de los costos de hardware y la proliferación de dispositivos de bajo costo con una capacidad de procesamiento cada vez mayor.

Estos dispositivos permiten tomar mediciones de sensores en tiempo real y automatizar la recolección de datos desde distintos puntos de un sistema. La interconexión de múltiples dispositivos como parte de una infraestructura que conecta objetos físicos con sistemas informáticos a través de la red constituyen lo que se conoce hoy en día como Internet de las cosas o IoT (por sus siglas en inglés *Internet of Things*). Este paradigma facilita la obtención

de datos que antes eran demasiado costosos o complejos de obtener, automatizando la recolección de mediciones en tiempo real. Esto permite el desarrollo de servicios y aplicaciones integradas con las que cada vez más organizaciones pueden explotar la información disponible y tomar mejores decisiones. Esto es lo que se conoce como *Big Data*, que conlleva el desafío de procesar un gran volumen de datos heterogéneos que son generados desde distintos entornos.

El procesamiento se puede realizar en plataformas en la nube, recibiendo un flujo ininterrumpido de datos, y también (o al menos en parte) en los mismos dispositivos embebidos encargados de la obtención de los datos, lo que recibe el nombre de *edge processing*. En efecto, es tal la capacidad de procesamiento de los microcontroladores disponibles comercialmente en la actualidad que es posible realizar tareas computacionalmente complejas con un alto nivel de eficacia dentro del dispositivo, tales como algoritmos de machine learning. Esto hace que los sistemas embebidos constituyan en el presente una valiosa herramienta para afrontar problemas existentes en una amplia gama de rubros.

Dentro de las oportunidades que brinda este escenario, en este trabajo se plantea la posibilidad de hacer uso de estas tecnologías poniéndolas al servicio de las personas que padecen algún tipo de discapacidad del habla mediante la implementación de un sistema distribuido constituido por un guante electrónico que permita traducir gestos de lengua de señas utilizando machine learning. Este dispositivo apuntará a reducir la brecha comunicacional existente entre las personas mudas o con dificultades para hablar y la mayoría de las personas que no padecen este tipo de problemas y no están familiarizadas con una comunicación de señas.

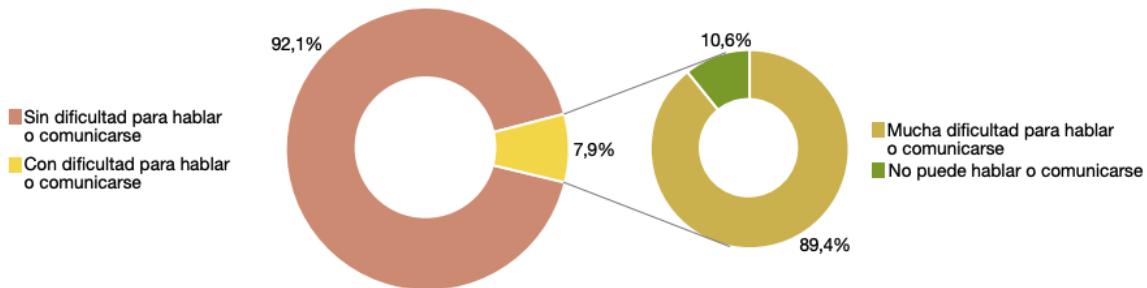
## 2.1. Relevancia

Impulsar la igualdad de oportunidades para todos y la inclusión social de las personas con discapacidad es una preocupación de las sociedades modernas y un tema clave en los

objetivos de desarrollo sostenible de la Organización de las Naciones Unidas planteados de cara al 2030 [25].

Según datos del año 2010, en Argentina la discapacidad auditiva corresponde al 18% de las discapacidades, de las cuales la dificultad auditiva representa el 86,6% y la sordera al 13,4% [13]. Si bien no existe un censo exacto de cuántos sordos hay en el país, asociaciones y personas interesadas calculan un número mayor a 70.000 sordos y más de 450.000 con algún tipo de discapacidad auditiva [11].

De acuerdo a un estudio realizado por el INDEC en el año 2018 (Estudio Nacional sobre el Perfil de las Personas con Discapacidad 2018) de la población de 6 años y más con alguna dificultad, que representa el 10,2% del total de la población de 6 años y más, el 7,9% tiene dificultad para hablar o comunicarse. Al evaluar la distribución por grado de dificultad de este grupo poblacional, se observa que el 89,4% tiene mucha dificultad para hablar o comunicarse y el 10,6% restante indica no poder hacerlo.



**Figura 1:** Población con dificultad del habla y la comunicación de 6 años y más, por grado de dificultad. [23]

La lengua de señas argentina (LSA) facilita sustancialmente la comunicación en la comunidad de sordos. Sin embargo aunque no se dispone de datos acerca del número de usuarios de la LSA podemos intuir que la cantidad de personas hablantes de esta lengua es reducida en nuestro país<sup>1</sup>. Esto hace que para una persona con impedimento del habla sea muy difícil

<sup>1</sup>Podemos basarnos en números de otro país suponiendo que la proporción entre personas hablantes de una lengua de señas y la cantidad de habitantes en ese país se corresponde con la de Argentina, por ejemplo 2006 en Estados Unidos había entre 250000 y 500000 personas hablantes de la lengua de señas estadounidense (ASL por sus siglas en inglés de “american sign language”) [3] para una población de 298,4 millones de habitantes,

poder establecer comunicaciones con otras personas. La alternativa de usar comunicación escrita es incómoda, impersonal e incluso inviable en una situación de emergencia, por esto es que desarrollar una herramienta que sirva para traducir lengua de señas es de interés para las personas que padecen este tipo de discapacidad.

## **2.2. Investigación**

A continuación se hace un breve resumen de la investigación realizada acerca de distintos aspectos que atañen al proyecto.

### **2.2.1. Interpretación de gestos de la lengua de señas**

Hoy en día existen varios proyectos informáticos que buscan hacer uso de la inteligencia artificial para traducir la lengua de señas utilizada por la comunidad sorda. Generalmente este tipo de proyectos hacen uso del reconocimiento de imágenes para traducir las señas[10]. Si bien estos proyectos tienen potencial, poseen algunas desventajas inherentes a cómo se obtienen los datos y cómo se entrena nodelos de inteligencia artificial con ellos. El reconocimiento de imágenes requiere que dispongamos de un gran conjunto de imágenes de entrenamiento para que el algoritmo sea capaz de reconocer gestos en una amplia variedad de entornos y en múltiples circunstancias, por ejemplo dicho set de entrenamiento ha de disponer de imágenes de las manos con distintos colores de piel para evitar que únicamente se reconozcan los gestos de personas con una determinada tez. También es necesario que las fotografías varíen en cuanto a su luminosidad, el entorno y la perspectiva de las mismas.

Por otro lado, la utilización de imágenes fijas constituye en sí mismo una restricción, dado que solamente se puede entrenar la red neuronal para identificar gestos estáticos de las manos, cuando el lenguaje de señas también se compone de movimientos dinámicos. Es por esto que muchos proyectos sobre este tema se limitan a traducir el abecedario del lenguaje con lo que a lo sumo un 0,0016% de la población de Estados Unidos habla la lengua de señas local.

de signos.

Por último, cabe mencionar que se requiere de una cámara enfocando a la persona para poder traducir sus gestos. Esto constituye un limitante en cuanto a la usabilidad de la solución, dado que solamente es útil en contextos muy limitados cuando en realidad la necesidad de una herramienta así se da en casi la totalidad de las situaciones cotidianas que vive una persona sorda.

Un enfoque superador es el de desarrollar un guante electrónico con el fin de utilizar inteligencia artificial combinada con la capacidad de los sensores y de los sistemas embebidos para extraer únicamente información sobre la posición de las manos y los dedos a medida que realizan los gestos cuando hablan en lengua de signos. De esta manera se evitan los problemas mencionados asociados al entrenamiento de redes neuronales para el reconocimiento de imágenes. La obtención de datos se vuelve mucho más sencilla y no se ve empañada por el entorno en el que se efectúe. Además, se resuelve el inconveniente que surge de la necesidad de tener una cámara enfocando al usuario ya que un dispositivo con sensores puede ser portable y se podría utilizar en situaciones cotidianas que involucran interacción con otras personas.

Si bien un guante electrónico tiene los beneficios mencionados, existen otras complicaciones que se deben tener en cuenta a la hora de interpretar una señal de LSA, a saber:

- la delimitación de las señas: cuando termina una señal y cuando inicia la señal siguiente
- la coarticulación: cuando una señal se ve afectada por la señal precedente o sucesora
- la temporalidad de la señal: qué rápido se efectúa la señal, sabiendo que esto puede variar dependiendo de la persona y de la situación.

Además, si el proyecto se vuelve más ambicioso y buscamos traducir no solo una señal sino una frase de lengua de señas, se añaden los problemas naturales asociados a la tra-

ducción de un idioma a otro. Las expresiones en lengua de señas argentina no siempre se corresponden palabra por palabra con una frase de la lengua castellana que utilizamos; las señas constituyen ideogramas que no solo representan palabras sino que también representan ideas que en castellano se deberían expresar como un conjunto de palabras. También se trata de una lengua que tiene una estructura gramatical diferente donde por ejemplo los verbos van al final de la oración y tampoco se utilizan artículos.

### 2.2.2. Guante electrónico

El concepto de guante electrónico no es nuevo, se remonta a los años 70 cuando las computadoras de escritorio se empezaban a volver una realidad y se investigaba acerca de cómo debía ser la interfaz hombre-máquina de estos dispositivos. Este tipo de interfaz se conoce como interfaz natural para el usuario (NUI<sup>2</sup>).

La interfaz hombre-máquina popularizada mundialmente fue la del mouse y el teclado con soporte visual a través de una pantalla (GUI<sup>3</sup>). Sin embargo, algunos investigadores pensaban que la utilización de un guante electrónico con sensores en los dedos podía ser un dispositivo viable para resolver esta cuestión. Su argumento se basaba en que la forma más natural de interactuar con nuestro entorno es a través de nuestras manos; las utilizamos para realizar tareas cotidianas y la interacción con la computadora no es una excepción. No obstante, cuando trabajamos con computadoras, la dexteridad que experimentamos con nuestras manos no se transmite a la máquina con las herramientas tradicionales de hardware que son el mouse y el teclado.

Los primeros en llevar esta idea a la práctica fueron un grupo de investigación del MIT en 1982 en un proyecto denominado “Put that there” [18] en el que desplazaban objetos en una pantalla usando gestos de la mano. Para ello utilizaban un nuevo sensor llamado Polhemus capaz de comunicar a la computadora la posición de la mano del usuario. Posteriormente se

---

<sup>2</sup>Del inglés *Natural user interface*: interfaz natural para el usuario

<sup>3</sup>Del inglés *Graphical user interface*: interfaz gráfica de usuario

llegaron a desarrollar productos que se han distribuido masivamente tales como el Power Glove en el año 1989, el cual consistía en un guante de realidad virtual con el que el usuario podía interactuar con la consola de Nintendo. Este producto recibió críticas por su falta de precisión y fue discontinuado porque la cantidad de juegos y aplicaciones que eran compatibles con la interfaz era muy limitada. A pesar de sus falencias, el Power Glove permaneció en el imaginario popular como un objeto de culto. La idea de emplear guantes para una NUI se vio por varios años como algo futurista y se ha llegado a reflejar en películas de ciencia ficción, como es el caso de Minority Report del año 2003 en la cual se puede ver a Tom Cruise manejar una pantalla holográfica con unos guantes especiales[5].

En la actualidad hay varias startups que están recuperando la idea del guante electrónico como interfaz persona-computadora. Por ejemplo Gest[7] consiste en un dispositivo con sensores en los dedos que interpreta gestos y permite realizar diversas acciones en varios programas de diseño gráfico entre otras aplicaciones. Otros ejemplos son Mocap Pro[9] y Plexus Vr [8], guantes diseñados para trabajar con realidad virtual.

Existen diversos tipos de guantes electrónicos con distintas características de hardware. El Power Glove por ejemplo hacía uso de sensores de flexión<sup>4</sup> capaces de medir el grado de flexión de los dedos[19]. Hay varios tipos de sensores de flexión: algunos utilizan fibra óptica y miden la variación de la luz que transita por estas fibras para determinar el grado de flexión, otros están basados en tinta conductora cuya resistencia varía en función del grado de doblez; el PowerGlove usaba estos últimos. Muchos aficionados buscan desarrollar sus propios guantes de realidad virtual y usan sensores basados en tinta conductora dado que ofrecen un alto grado de fiabilidad en sus mediciones. La desventaja de estos sensores radica en su precio, son mucho más costosos que otro tipo de sensores<sup>5</sup>, pero para aplicaciones de realidad virtual en las que se quiere reflejar con fidelidad el movimiento de las manos y dedos en una mano virtual resulta necesario. Los sensores de flexión ópticos son considerablemente

---

<sup>4</sup>habitualmente conocidos por su nombre en inglés *flex sensors*

<sup>5</sup>En Argentina un flex sensor sale 4000\$ pesos mientras que una IMU como el MPU6050 sale 250\$, precios a fecha de Marzo de 2021.

más económicos pero muy difíciles de conseguir comercialmente por lo que habitualmente se fabrican de forma casera. Alternativamente a los sensores de flexión, Gest por el contrario hace uso de sensores conocidos como unidades de medición inercial o IMU (del inglés *inertial measurement unit*) capaces de medir e informar acerca de la velocidad y orientación de las manos y dedos usando una combinación de acelerómetros y giróscopos.

Los sensores de los guantes, ya sean IMUs o sensores de flexión, se conectan a un sistema embebido con un microprocesador que a su vez transmite la información a una computadora o aplicación de celular. Existen muchos tipos de microcontroladores con distintas características; para este tipo de proyectos es conveniente que el chip utilizado tenga capacidad de comunicación Wifi y/o Bluetooth para la transmisión de los datos recolectados por los sensores. El Esp32 de Espressif y el Arduino BLE Nano 33 son dos chips con estas características.

### 2.2.3. Procesamiento de datos

Actualmente, el avance del poder de cómputo de los microcontroladores hizo viable el procesamiento de la información directamente en el mismo dispositivo, esto es lo que se conoce como *edge processing*<sup>6</sup>. Una de las ventajas de este modelo es la posibilidad de evitar centralizar el procesamiento de la información del sistema distribuido en un servidor central, haciendo que la toma de decisiones se realice *in situ* en el dispositivo de IoT. De esta forma se elimina la necesidad de transmitir datos continuamente desde el dispositivo al servidor, reduciendo así el consumo de batería del dispositivo y la latencia total del sistema asociada a una operación. Además el sistema depende menos de la conectividad del lugar donde se encuentre el dispositivo.

El procesamiento de algoritmos de machine learning en sistemas embebidos puede resultar un desafío ya que habitualmente estos algoritmos requieren de una capacidad de cómputo significativa y por consiguiente de un mayor consumo energético. Por ejemplo, hoy

---

<sup>6</sup>Traducible como "procesamiento de frontera"

en día es habitual usar GPUs en lugar de CPUs para correr una red neuronal, dado que tienen mayor poder de cómputo. Muchas de las bibliotecas que existen para machine learning fueron desarrollados para computadoras y lenguajes de alto nivel como python y no para sistemas embebidos donde la batería, el poder de cómputo y la memoria disponible son un factor crítico.

En 2017 Google anunció Tensor Flow Lite[27], una versión liviana de Tensor Flow (una biblioteca ampliamente utilizada para el desarrollo de soluciones de machine learning) especialmente diseñada para su uso en sistemas embebidos. Existen numerosos casos de productos electrónicos que circulan en el mercado y hacen uso tensor flow lite, ya sea desplegado dentro de un microcontrolador o en un dispositivo móvil. Los parlantes inteligentes por ejemplo, tales como Alexa de Amazon, hacen uso de esta tecnología para clasificar sonidos, pudiendo diferenciar un comando dictado por voz de otros sonidos ambientes.

#### 2.2.4. Integración, despliegue y aprendizaje continuo

Una colección de prácticas que viene de las metodologías ágiles y es muy popular en el desarrollo de software hoy en día es la denominada DevOps, que combina el desarrollo de software con operaciones de IT para reducir el tiempo del ciclo de vida de desarrollo que provee a los clientes con delivery continuo de sistemas de calidad. En este proyecto el proceso de integración, despliegue y mejora continua es más complicado que en servicios de software tradicionales porque además de los cambios de la aplicación pueden incorporarse nuevos datos o puede modificarse el modelo de machine learning.

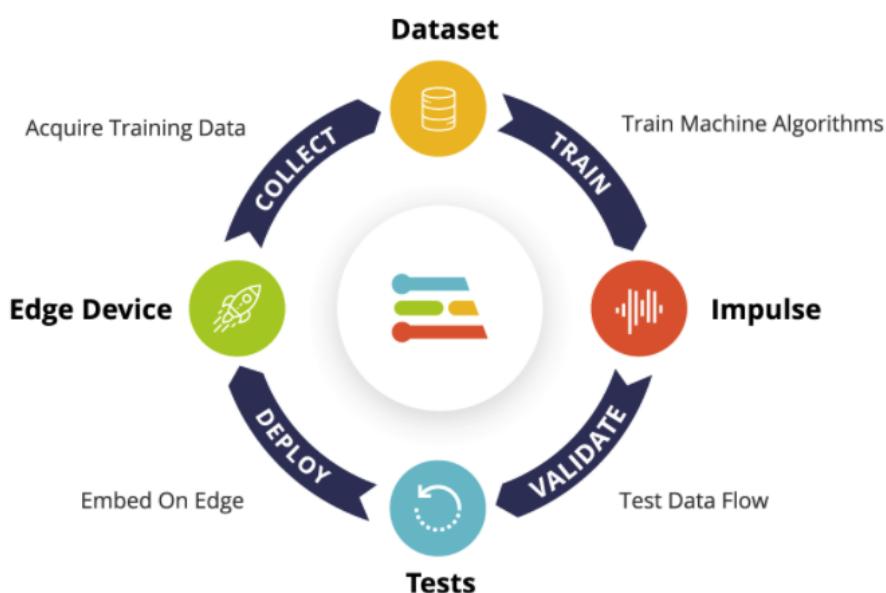
El mismo concepto de devops existe para machine learning, donde se recolecta data, se procesa, se hace la denominada extracción de características<sup>7</sup>, se alimenta el modelo de machine learning y se vuelve a implantar el algoritmo, combinando así el machine learning con operaciones de despliegue continuo. Como explica Martín Fowler en su artículo *Continuous Delivery for Machine Learning*[20], las tareas a realizar en el desarrollo de software

---

<sup>7</sup>también conocido como *features* en inglés

con machine learning son más complejas dado que están sujetas a cambios en tres ejes: el código, el modelo y los datos. El comportamiento de este tipo de aplicaciones suele ser complejo y difícil de predecir, además de ser difíciles de testear, de entender y de mejorar. Es por esto que Martin Fowler acuña el concepto de despliegue continuo de machine learning<sup>8</sup>, afirmando que una aplicación que hace uso de machine learning amerita tener un pipeline automatizado que lleve los cambios a producción de forma confiable.

Edge Impulse<sup>9</sup> es una herramienta que provee una solución en este sentido. Se trata de una plataforma de pipelines automatizados para aplicaciones con machine learning en sistemas embebidos. Está especialmente diseñado para integrar de forma simple herramientas usadas habitualmente por desarrolladores de sistemas embebidos e incluye una interfaz de alto nivel para incorporar algoritmos de machine learning livianos. Hace uso de la biblioteca de Tensor Flow Lite para entrenamiento, optimización y despliegue de modelos de deep learning en sistemas embebidos.



**Figura 2:** Esquema de funcionamiento del pipeline de Edge Impulse.

Edge Impulse permite automatizar el sistema de recolección de datos y transformación

<sup>8</sup>traducido del inglés *continuous delivery for machine learning*

<sup>9</sup><https://www.edgeimpulse.com/>

en un conjunto de datos útiles. Lo ideal es que se recolecten datos utilizando el mismo dispositivo que se utilizará de forma productiva, es decir, donde se desplegará el algoritmo de machine learning entrenado. Además, Edge Impulse permite cargar datasets de múltiples maneras: cargando un archivo en formato .json, a través de la red utilizando una api rest o a partir de los logs del dispositivo conectado a la computadora mediante un puerto serie. Una vez cargados los datos se puede dividir fácilmente en un set de entrenamiento y un set de validación. Permite además configurar el proceso de entrenamiento, por ejemplo se puede definir la cantidad de epícas, es decir la cantidad de veces que se actualizan los parámetros a partir del proceso de retro-propagación (*backpropagation*) en una red neuronal.

Es lo suficientemente flexible, permite al desarrollador configurar la cantidad de capas, o modificar hiperparámetros y agregar capas externas a la red o editar el código de entrenamiento dentro de la interfaz de usuario. Dado que Edge Impulse usa bibliotecas y la API de Tensor Flow, es posible extender el código de entrenamiento integrado con lógica propia de forma simple. Por ejemplo se puede ampliar el pipeline de entrenamiento y validación de los conjuntos de datos para agregar transformaciones. Una vez que nuestro algoritmo está entrenado podemos evaluar su precisión, visualizar los datos en gráficos e identificar problemas o sesgos en nuestro conjunto de datos mediante matrices de validación y otras herramientas.

Por último Edge Impulse permite automatizar el sistema de despliegue de nuevas versiones del algoritmo de machine learning en los dispositivos.

### 2.2.5. Rust embebido

Una de los puntos críticos de las aplicaciones embebidas en dispositivos con poder de cómputo y memoria limitados es el desempeño. Es posible desarrollar aplicaciones embebidas de alto desempeño utilizando C y C++, pero a costa de potenciales riesgos relacionados con el manejo de la memoria. Con el objetivo de lograr un mayor desempeño y mejores

niveles de seguridad en comparación con C++, Mozilla anunció en el año 2010 el proyecto de código abierto de desarrollo del lenguaje de programación Rust. Específicamente, Rust está diseñado para hacer frente a ciertos problemas con los que C++ es ineficiente, como el acceso seguro a la memoria y la concurrencia. La sintaxis es similar pero captura algunos de los beneficios de los lenguajes funcionales. El compilador de Rust está diseñado para detectar errores sutiles que pueden afectar al código de bajo nivel. Con un sistema de tipos estático, el compilador puede aplicar restricciones antes y después de una sentencia para evitar errores que pueden ocurrir en lenguajes de tipado dinámico. Es por esto que en los últimos tiempos Rust ha ganado popularidad dentro de las comunidades de desarrollo IoT. Se espera que el uso de Rust embebido aumente la calidad y corrección del software en dominios críticos para la seguridad. En particular, su enfoque en la estabilidad se considera una ventaja competitiva importante en sistemas de alto riesgo.

A pesar de todos los potenciales beneficios de utilizar el lenguaje de programación de Rust en este proyecto también existen ciertas desventajas. Es necesario invertir una cantidad de tiempo considerablemente mayor para lidiar con los errores de compilación que aseguran la prevención de errores. Además la comunidad de C/C++ es mucho más grande que la comunidad de Rust, en consecuencia el soporte para desarrollar en Rust es mucho menor, existen menos bibliotecas y menos frameworks. Por ejemplo la organización Espressif ofrece foros de consulta y ejemplos en C/C++ para utilizar sus productos de hardware pero no existe soporte para grabar código Rust en el dispositivo esp32. Por lo tanto, es totalmente incierto el uso de bluetooth o wifi a partir de Rust en esta familia de microcontroladores, pero es un desafío interesante para intentar implementar en este proyecto.

### **3. Alcance del proyecto**

El proyecto consiste en el desarrollo y la elaboración de un guante electrónico junto con una aplicación para celular que sirvan para traducir gestos de lengua de señas argentina.

El guante estará compuesto por un microcontrolador conectado con cinco sensores de movimiento, uno para cada dedo. El guante podrá conectarse a la aplicación a través de una conexión inalámbrica. Una de las funcionalidades de la aplicación será proporcionar al usuario información acerca del funcionamiento de los guantes, por ejemplo el nivel de batería, el funcionamiento de los sensores, el estado de la conexión, etc.

Para poder interpretar los gestos de la lengua de señas argentina realizados por un usuario final se desarrollará un algoritmo de machine learning implantado dentro del sistema embebido del guante. Concretamente este algoritmo será una red neuronal con los pesos de sus neuronas debidamente ponderados para clasificar los movimientos realizados usando el guante. Para entrenar la red será necesario realizar la recopilación y análisis de datos de movimientos de distintos gestos. Es por esto que la aplicación para celular tendrá dos modalidades de uso: recolección de datos e interpretación de gestos.

La modalidad de recolección de datos tiene por objetivo permitirle a un usuario recolectar mediciones de los sensores de un gesto específico para posteriormente entrenar el algoritmo de machine learning. El usuario seleccionaría un gesto que luego debería realizar con sus manos tantas veces como sea necesario para poder constituir un conjunto de datos de entrenamiento de calidad. La información recopilada por los sensores se almacenará temporalmente en la memoria del celular. Estos archivos estarán visibles para el usuario en un listado. El usuario podrá cargar los archivos al servidor y/o eliminarlos en caso de que sea pertinente. Será conveniente además que los archivos se eliminen automáticamente una vez que se hayan subido al servidor. Cabe destacar que la aplicación no permitirá que cualquier usuario pueda acceder a la funcionalidad de cargar nuevos movimientos al sistema porque ello podría generar inconvenientes en el modelo de predicción. Por ejemplo, si un usuario graba los movimientos que realiza y los asocia a una palabra equivocada luego la traducción no sería la correcta. Para evitar este inconveniente habrá dos tipos de usuarios: ordinario y autorizado. Solo el usuario autorizado podrá acceder a esta funcionalidad.

La modalidad de interpretación de gestos tiene por objetivo traducir gestos de la lengua de señas. Cuando el usuario usa los guantes en modo interpretación, cada guante realizará su propia interpretación del gesto realizado haciendo uso de la red neuronal embebida que procesará el flujo de mediciones de los sensores y enviará esta interpretación con las distintas probabilidades a la aplicación. Dicha aplicación procesará esa información proveniente de los guantes y realizará una interpretación definitiva. De esta forma se estará ahorrando energía al realizarse gran parte del procesamiento de los datos de los sensores en los mismos guantes, a diferencia de la modalidad de recolección de datos en la cual es necesario transferir constantemente los datos de los sensores a la aplicación.

### **3.1. Objetivo**

El objetivo del presente trabajo profesional es: diseñar, implementar e implantar un guante con sensores conectado a un microcontrolador para registrar datos de movimientos de las manos correspondientes a gestos de lengua de señas mediante el uso de una red neuronal embebida en el microcontrolador de los guantes, y además desarrollar una aplicación para celular que cumpla con:

1. comunicarse con los guantes y configurarlos
2. recopilar datos de los sensores de los guantes para poder alimentar el algoritmo de machine learning que clasificará gestos en tiempo real
3. visualizar información diversa de los guantes (por ejemplo el nivel de batería o si los sensores están bien conectados)
4. visualizar interpretaciones de las señas que realiza el usuario con los guantes puestos

#### **3.1.1. Subobjetivos**

Del objetivo se desprenden los siguientes subobjetivos:

1. Traducir un conjunto de palabras que conformen una frase en lengua de señas argentina.
2. Desarrollar en lenguaje de programación Rust los ítems a continuación enumerados:
  - a) Recolección de las mediciones de los sensores embebido en el microcontrolador.
  - b) Construcción de paquetes para transmitir datos a la aplicación mediante Bluetooth usando el protocolo GATT.
  - c) Integración de un complemento de la aplicación para determinar a qué gesto corresponde un movimiento realizado por las dos manos en base a las predicciones generadas por cada guante de forma independiente.

### 3.2. Historias de usuario

A partir del objetivo propuesto, se desprenden las siguientes historias de usuario:

1. Como usuario de la aplicación quiero poder conectar un guante electrónico a la aplicación para saber si está prendido y funcionando.
2. Como usuario de la aplicación quiero poder calibrar el guante para obtener mayor precisión en los movimientos medidos.
3. Como usuario de la aplicación quiero poder realizar un movimiento correspondiente a un gesto de lengua de señas utilizando los guantes electrónicos y ver en la pantalla de la aplicación la traducción en lengua castellana para poder expresar una idea.
4. Como usuario de la aplicación quiero poder autenticarme como administrador para validar que estoy autorizado a realizar ciertas tareas.
5. Como usuario autorizado quiero poder cargar nuevos datos de movimientos utilizando los guantes para aumentar el poder de traducción de la aplicación.

6. Como usuario autorizado quiero poder guardar los datos recolectados en archivos para poder visualizarlos luego.
7. Como usuario autorizado quiero poder eliminar un archivo cuando considero que el movimiento no se hizo adecuadamente para evitar corromper el set de datos de entrenamiento y con esto provocar traducciones equivocadas.
8. Como usuario autorizado quiero poder agregar los archivos de nuevos datos recolectados al modelo.
9. Como usuario autorizado quiero poder automatizar la eliminación de archivos cargados en el sistema para evitar tener que borrarlos de forma manual.

## Parte II

# Implementación del trabajo

## 4. Implementación del hardware

En esta sección se detallará cómo fue el trabajo de manipulación del hardware para construir los guantes electrónicos con sensores, explicando los motivos por los que se elegieron los distintos componentes de hardware que componen al prototipo de guante final, y los motivos por los cuales se descartaron otros componentes que fueron considerados.

### 4.1. Embebido



**Figura 3:** Node MCU - 32S v1.1

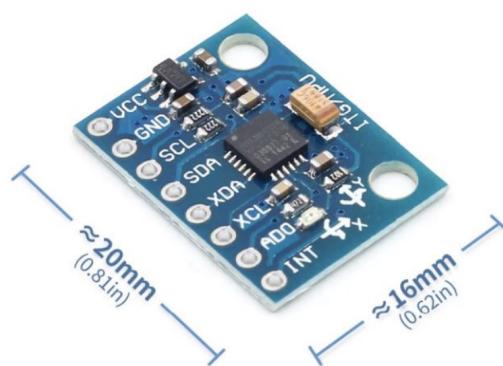
La placa de desarrollo empleada fue el Esp32. Se denominan Esp32 a una familia de chips SoC (system on a chip) de bajo costo y consumo de energía, con tecnología Wi-Fi y Bluetooth integrada. El Esp32 emplea un microprocesador Tensilica Xtensa LX6 con uno o dos núcleos, dependiendo del modelo. Fueron creados por Espressif Systems y vienen a sustituir otra familia de SoCs, el Esp8266 que cuenta con comunicación Wifi pero no así Bluetooth. Concretamente, el modelo del esp32 que hemos adquirido es el Nodemcu Esp32.

Fueron múltiples los motivos por los que nos decantamos por este chip, a saber:

- Dispone de capacidad de comunicación bluetooth, fundamental para poder comunicar el guante con la aplicación de celular de manera directa e inalámbrica
- El microprocesador tiene una frecuencia de reloj estándar de 160MHz y se puede configurar para que alcance 240MHz, lo cual puede incrementar la cantidad de mediciones por segundo obtenidas de los sensores.
- Bajo consumo de energía
- 520KiB de SRAM, suficiente para poder embeber el software necesario para controlar el guante
- Fácil adquisición en Argentina y precio económico

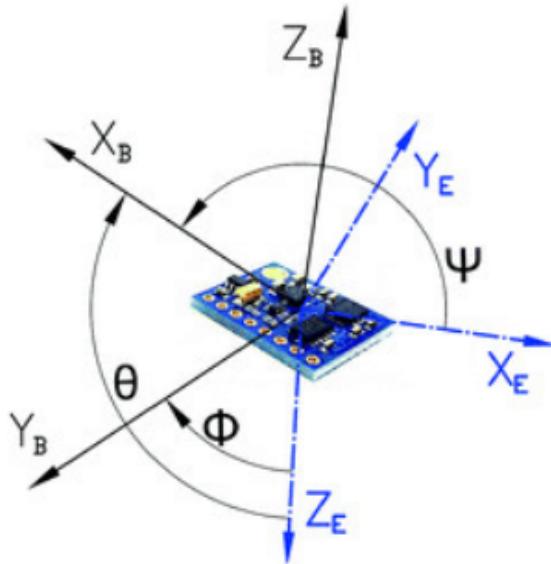
Además el Nodemcu-esp32 dispone de una gran cantidad de pines gpios (general purpose input output pins), varios pines de “ground” y una tensión de 3,3V.

## 4.2. Sensores



**Figura 4:** Sensor IMU MPU6050

Los sensores utilizados son los MPU6050. Se trata de unidades de medición inercial que cuentan con acelerómetro, giroscopio y sensor de temperatura. Estos sensores nos proporcionan información acerca de la orientación y del movimiento del MPU6050.



**Figura 5:** Esquema de vectores de las distintas mediciones provistas por el sensor IMU. En azul las aceleraciones y en negro la velocidad angular.

Colocando uno de estos sensores en cada uno de los dedos, podemos determinar la posición de los mismos, la inclinación y la dirección de los dedos en cada gesto de la lengua de señas. Son de fácil adquisición en el país y a un precio muy económico, en contraposición a otros sensores como los sensores de flexión, que no se consiguen en el país y tienen un costo más de veinte veces mayor a los de los MPU6050.

Alternativamente habíamos evaluado el uso de sensores de flexión, pero los terminamos descartando dada su difícil adquisición en Argentina y su elevado costo. Además consideramos en su momento que no era necesario el nivel de precisión que proveen los sensores de flexión (para medir el grado de flexión de los dedos) a la hora de entrenar el algoritmo de machine learning. Esta suposición se aseveró correcta.

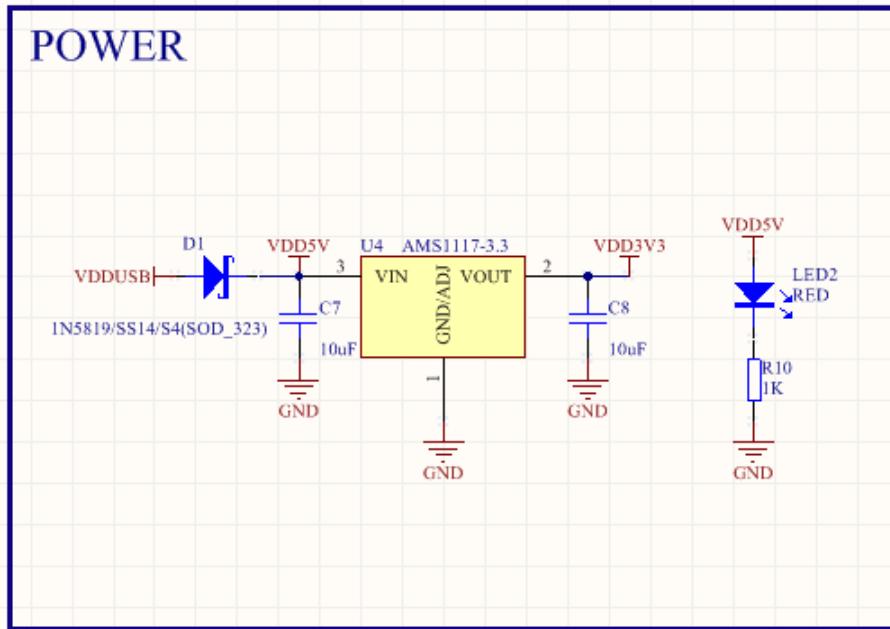
### 4.3. Autonomía del dispositivo

Un requerimiento importante del proyecto era que el dispositivo dispusiera de autonomía para brindar comodidad al usuario, suprimiendo así la necesidad de tener al guante conectado por cable a la computadora. Por esto es que estudiamos diversas alternativas de batería que fueran compatibles con el Esp32 y que proporcionaran autonomía para aproximadamente una hora de utilización.

#### Consideraciones

Los factores a considerar para determinar el tipo de batería y su modo de uso eran las siguientes:

- la tensión de entrada
- la autonomía
- la corriente
- cómo se conecta al dispositivo
- la intensidad de uso del sistema



**Figura 6:** Sección del esquema del Esp32 referido a la alimentación del dispositivo. En él vemos que internamente el esp32 se alimenta con una tensión de 3,3V. También se puede alimentar el dispositivo con 5V, dado que el mismo Esp32 dispone de un regulador de tensión interno AMS1117, que se encarga de reducir la tensión a 3,3V.

En cuanto a la compatibilidad con el Esp32 tuvimos que tener en cuenta que hay dos tensiones posibles con la que podemos alimentar el dispositivo: 3,3V y 5V. La ventaja de conectar una batería de 5V al esp32 es que el dispositivo mismo se encarga de regular la tensión, reduciéndola al 3,3V. Esto puede ser una vulnerabilidad también, dado que el AMS1117 regulador no tiene una reputación de ser muy robusto y una falla en el componente puede hacer que se arruine por completo el Esp32.

### Alternativas

| Batería   | Tensión de salida | Autonomía | Fácil de adquirir | Costo              | Recargable | Conexión directa |
|-----------|-------------------|-----------|-------------------|--------------------|------------|------------------|
| Ión litio | 3,7V              | 600 mAh   | Si                | 900 AR\$/u         | Si         | No               |
| LiFePO4   | 3,2V              | 600 mAh   | No                | 800 AR\$/u + envío | Si         | Si               |
| Coin      | 3,0V              | 190 mAh   | Si                | 400 AR\$/5u        | No         | No               |

**Cuadro 1:** Características principales de las diferentes baterías evaluadas.

Comercialmente las baterías no suelen venir con una tensión ni de 3,3V ni de 5,5V. Las más comunes son las de ión litio que vienen con una tensión de 3,7V, muy elevada para el pin de 3,3V que tiene un rango de (3,0V a 3,6V) y muy baja para el pin de 5V (con rango de 4V a 12V). Esto quiere decir que en caso de trabajar con una batería de ion litio es necesario agregar algún componente entre la batería y el esp32 que regule la tensión, ya sea con un step up para elevar la tensión o con un step down para reducirla.

Recientemente se ha empezado a popularizar un tipo de batería distinto, las baterías LiFePo4 (litio, hierro y fosfato). Proveen una tensión de 3,2V (dentro del rango del pin de 3,3V del esp32), con una curva de caída de tensión bastante plana, es decir que a medida que se consume la energía almacenada en la batería la tensión de salida se mantiene aproximadamente en 3,2V. Esto hace que en teoría podamos conectar la batería directamente al pin de 3,3V del Esp32, sin necesidad de algún step up o step down.

Una tercera alternativa fue utilizar baterías de tipo coin, no obstante las descartamos rápidamente debido a su muy limitada capacidad, además de no ser recargables.

## Experiencias

Inicialmente optamos por la batería de LiFePO4, considerando la ventaja que tenía el hecho de no requerir de un componente intermedio para regular la tensión, reduciendo el tamaño de nuestro dispositivo. Pudimos conseguir unas baterías de formato AA de un proveedor de Rio Cuarto, fue la única opción de compra que tuvimos en Argentina. Las baterías eran de marca Etinesan, como se muestra en la imagen.



**Figura 7:** Batería LiFePO4 marca Etinesan.

Sin embargo, la calidad de la batería de esta marca resultó ser mala; en la práctica si bien la tensión de salida de la batería era la estipulada, parecía ser que no estaba entregando la cantidad de corriente requerida por el Esp32. Tal es así que resultaba imposible establecer una conexión bluetooth con la aplicación, requisito fundamental para nuestro proyecto.

Debido a este problema optamos por descartar esta alternativa e intentar usar una más convencional batería de ión litio.



**Figura 8:** Paquete de las baterías LiIon utilizadas, marca Fulltotal

Para esta alternativa resultaba necesario regular la tensión. Pasar de 3,7V a 3,3V si bien es posible, no resultó ser viable debido a que se requiere de algún Low Dropout regulator

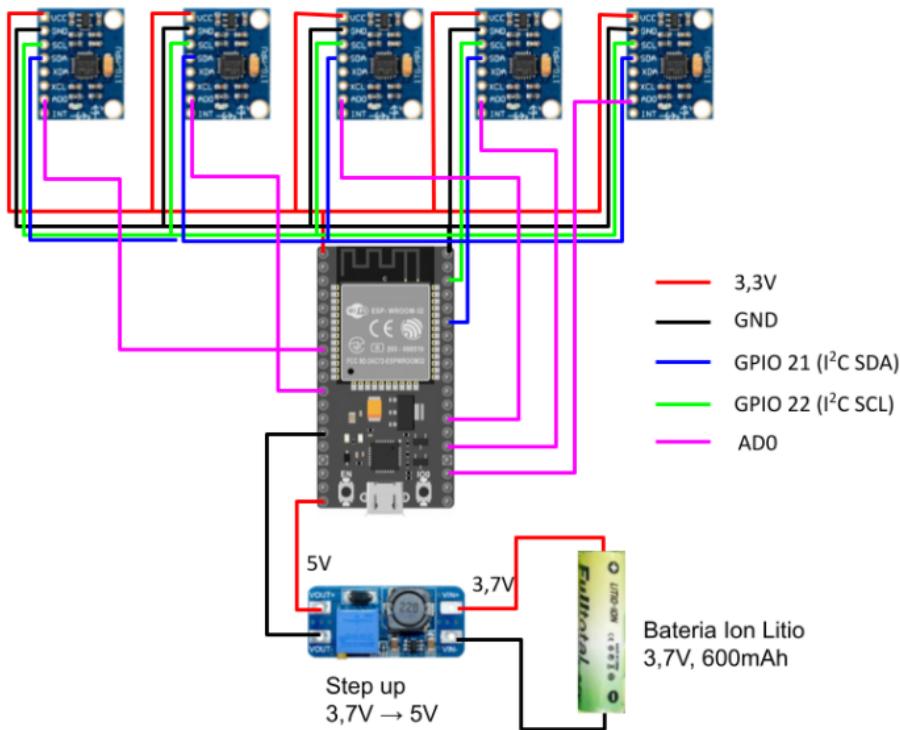
capaz de reducir la tensión en un escaso margen de 0,4V. Existen LDO's capaces de lograr esto pero que en Argentina no pudimos conseguir (por ejemplo el MCP1725T-3302E/MC), que hubiéramos tenido que importar con un considerable costo.

En cambio la opción del step up, elevando la tensión a 5V era más factible. El step up MT3608, de bajo costo y fácil adquisición, admite una tensión de entrada de 2V a 24V y una tensión de salida regulable de hasta 28V. Bastaba regular la pequeña perilla que tiene para poder ajustar la tensión de salida a 5V. Con este componente, pudimos resolver la cuestión de la autonomía del dispositivo, usando una batería de ión litio de 600mAh, en formato AA.



**Figura 9:** Fuente step up MT3608 DC-DC utilizada.

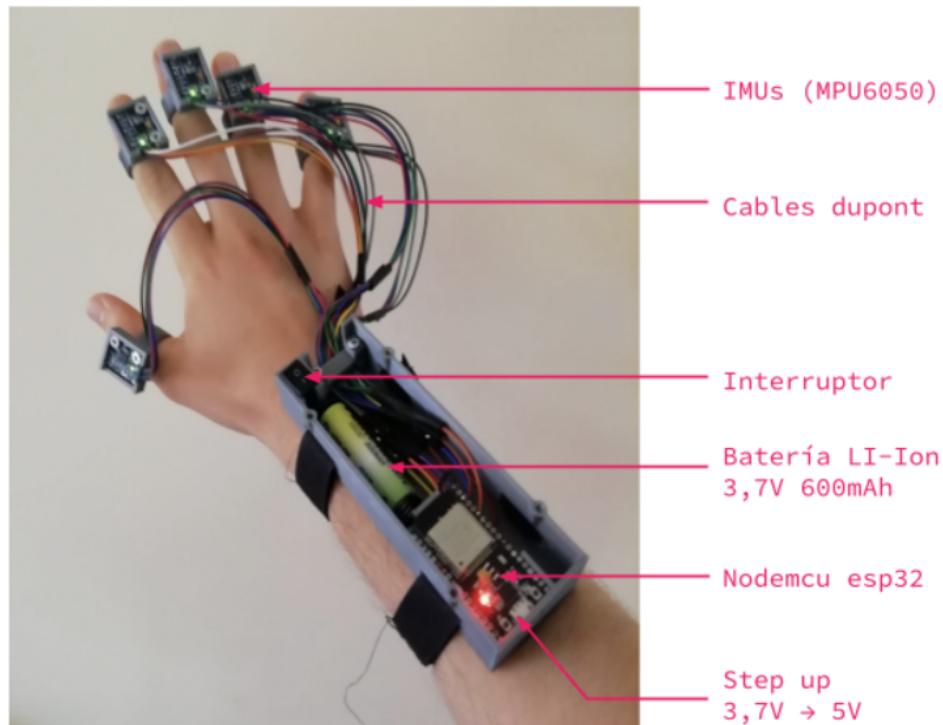
#### 4.4. Estructura del guante



**Figura 10:** Estructura general del guante.

En este diagrama vemos que el esp32 se alimenta mediante una batería de ión litio de 3,7V conectada a un step up que eleva la tensión a 5V y se conecta al pin de 5V del esp32. Posteriormente todos los sensores tienen conectados el pin de tensión al pin de 3,3V; el pin de ground al pin de ground (tierra) del esp32; el pin de SCL y el pin de SDA al correspondiente pin de SCL y SDA del esp32. Por último cada uno de los sensores tiene conectado un pin AD0 con un pin GPIO, de propósito general, que usaremos después para configurar la dirección de escritura del sensor en el bus I2C. En total cada sensor tiene conectados 5 cables al esp32. Un detalle menor a tener en cuenta es la existencia de un interruptor entre la batería y el step up que no figura en el diagrama.

#### 4.5. Proceso de fabricación del guante



**Figura 11:** Diagrama del guante armado y colocado

##### 4.5.1. Elementos necesarios

###### 4.5.1.1 Herramientas:

- Soldador
- Alambre de estaño
- Succiónador de estaño
- Herramienta para pelar cables
- Una prensa u otra herramienta para sostener los componentes que vamos a soldar
- Impresora 3D y plástico PLA

- Tornillos M3 y destornillador de cruz
- Tiras de velcro
- Tiras de termocontraíble el más fino posible, preferible antes que la cinta aislante que también se puede usar
- Multímetro

Conviene además tener:

- Una pinza
- Una máquina de coser
- Destornilladores de precisión

#### 4.5.1.2 Componentes electrónicos

- Placa de desarrollo con el esp32, como el Esp32 devkit v1
- 5 sensores MPU6050
- Step up Mt3608
- Porta pilas AA
- Batería Li Ion 3,7V AA
- 25 cables dupont de 30 cm
  - 5 cables con una terminación macho (de uno de los lados, por ende pueden ser macho - hembra o macho - macho, la otra terminación la vamos a sacar)
  - 20 cables sin terminación específica (pueden ser macho - macho, macho - hembra o hembra - hembra)

- 5 cables de menor tamaño con terminación hembra en uno de los extremos
- 2 remanentes de cables de unos 3cm de longitud

#### 4.5.2. Preparación de los componentes

##### 4.5.2.1 Impresión de la carcasa

Ir imprimiendo la caja a partir del siguiente modelo: <https://www.tinkercad.com/things/1FYqHGZh9bM-copy-of-wide-wrist-prototype/edit?sharecode=cW02jiwd44lIh8Mvp1XR8H3wInovRhg64IR6EUahRkM>  
Mientras se imprime vamos a ir preparando los componentes electrónicos.

##### 4.5.2.2 ESP32

A menos que hayamos comprado una placa de desarrollo sin los pines soldados, vamos a extraer los pines ya que no vamos a usar los terminales de los cables dupont para los que estos pines están previstos, para de este modo reducir el tamaño del guante. Entonces

1. Extraer las tiras de plástico que sostienen los pines con la ayuda de una pinza
2. Colocar la placa de desarrollo (ESP32) en una prensa o sujetarla con alguna otra herramienta de forma tal de que con el soldador en una mano derritamos de un lado el estaño y con una pinza en la otra mano extraigamos el pin que estamos trabajando, repitiendo este proceso para todos los pines.
3. Una vez extraídos todos los pines podemos eliminar el estaño que haya quedado en los agujeros con el succionador de estaño, calentando al estaño primero con el soldador.

##### 4.5.2.3 MPU6050

Estos sensores vienen sin los pines soldados, no obstante, en caso de que los hayamos soldado, efectuar lo mismo que se hizo con el Esp32.

#### 4.5.2.4 MT3608

El step up viene con un potenciómetro que tenemos que regular para que al estar conectado a la batería de 3,7V la tensión de salida sea de 5V.

1. Soldar los cables del portapilas a los terminales de entrada del step up.
2. Colocar la batería en el portapilas
3. Con el multímetro medir la tensión de salida del step up y al mismo tiempo regular el potenciómetro: girar en sentido horario para incrementar la tensión hasta alcanzar los 5V que veremos en el multímetro.

#### 4.5.3. Preparación del cableado

De los sensores vamos a utilizar 5 pines que deben estar conectados al esp32:

- Pin de tensión (power)
- Pin de tierra (ground)
- Pin de SDA
- Pin de SCL
- Pin de AD0

En el esp32 vamos a tener un único pin de tensión, de ground, de sda y de scl, por lo que han de ser compartidos por los 5 sensores (no así el pin de AD0, donde cada sensor va a tener uno asignado). Por este motivo es que vamos a tener que realizar una bifurcación en el cableado para poder derivar una terminación a cada uno de los sensores para los 4 pines mencionados.

Antes de proceder con esto, tenemos que seleccionar los cables que vamos a utilizar. Vamos a seleccionar los cables dupont de 30cm. Conviene elegir un color de cable para cada

uno de los pines para evitar confusiones y accidentalmente arruinar algún componente, por ejemplo:

- Pin de tensión → cable rojo o naranja
- Pin de ground → cable negro o gris
- Pin de SCL → cable azul o amarillo
- Pin de SDA → cable verde o blanco
- Pin de AD0 → cable marrón o morado

Una vez seleccionados los cables para los pines de tensión, ground, sda y scl, vamos a preparar las bifurcaciones.

1. Extraer los terminales de los cables dupont ya que no los vamos a usar
2. Pelar la punta del cable en uno de los extremos en los cinco cables y soldarlos en ese extremo
3. Agarrar un fragmento de cable de unos 3 o 4cm que nos va a servir para conectar estos cables que hemos preparado al pin correspondiente del esp32. Pelarlo en sus extremos y soldar una terminación adonde soldamos los demás cables. En definitiva tenemos que tener para este entonces un nodo del cual emanen los 5 cables dupont de 30 cm y el cable menor que se va a conectar al esp32.
4. Cortar un fragmento de termocontraible y cubrir la soldadura aplicándole calor para de este modo sellar la soldadura y evitar que sucedan cortocircuitos, además de reforzar la conexión dándole cierta rigidez estructural.

Repetir estos pasos hasta tener los cuatro cables bifurcados para los pines mencionados.

#### 4.5.4. Soldar los cables al ESP32

Antes de empezar, cabe mencionar que los cables deben ir por debajo del Esp32 y no por arriba.

##### 4.5.4.1 Pines AD0

Tenemos muchos cables que soldar. Podemos empezar soldando los cables que se van a conectar a los terminales del AD0 en el Esp32. Estos cables tienen que tener un terminal hembra para poder conectarlos al terminal macho del cable dupont de 30cm conectado al pin de AD0 de los sensores. Por ende, tenemos que agarrar un cable con terminal hembra, cortarlo con una longitud de unos 3 o 4 cm, pelar la punta del cable sin el terminal, pincelarlo con estaño y soldarlo al terminal del Esp32 adecuado. La convención seguida es:

| Dedo   | Pin |
|--------|-----|
| Middle | 26  |
| Ring   | 27  |
| Pinky  | 33  |
| Thumb  | 17  |
| Index  | 32  |

**Cuadro 2:** Convención de pines para los cables que se conectarán al terminal AD0 de los MPU6050.

##### 4.5.4.2 Pines SDA, SCL y Tensión

Los cables a conectarse a los pines de SDA, SCL y tensión son 3 de los 4 cables bifurcados que preparamos anteriormente. El cable de tierra lo vamos a ver más adelante porque es distinto.

El cable de alimentación va conectado al pin de 3,3V.

| Terminal | Pin |
|----------|-----|
| SDA      | 21  |
| SCL      | 22  |

**Cuadro 3:** Convención para los pines SDA y SCL. Esta convención no es arbitraria, los pines 21 y 22 son aquellos reservados en el Esp32 para este propósito.

Nuevamente, pelar el extremo a conectar, pincelarlo de estaño, insertarlo en el terminal adecuado y soldar, siempre tratando que no queden expuestos los cables de cobre al aire, es decir, intentando que el recubrimiento de plástico del cable quede pegado a la placa.

#### 4.5.4.3 Tierra (Ground)

En el Esp32 Devkit V1 hay un único pin de Ground. Necesitamos conectar dos cables a tierra:

1. El cable de tierra bifurcado que va a los sensores
2. El cable de tierra del terminal de salida del step up.

En vez de conectar todo al mismo terminal del esp32, es mucho más sencillo hacer lo siguiente:

1. Conectar el cable bifurcado al terminal de tierra de salida del step up, ya que hay mucha más superficie ahí para tener dos cables conectados.
2. Conectar un cable entre el terminal de tierra de salida del step up y el pin de ground del esp32.

#### 4.5.4.4 Pin de tensión de 5V

La única conexión que falta realizar es la del pin de 5V que se conecta al terminal de salida positivo del step up. Una vez realizada esa conexión ya tenemos casi todo listo (habría que

conectar los pines macho de los ad0 con los pines hembra conectados al esp32). Podemos insertar la batería al portapilas y todos los leds tanto del esp32 como de los sensores deberían encenderse.

## 5. Funcionalidades

Antes de explicar cómo fue la implementación del software tanto del sistema embebido como de la aplicación, en esta sección procedemos a detallar cuáles fueron las funcionalidades implementadas en el sistema para así poder entender mejor las arquitecturas de software planteadas.

Nuestro sistema cuenta con las siguientes funcionalidades:

1. Calibración del guante
2. Recolección de datos
3. Visualización de datos
4. Interpretación de gestos

A continuación detallamos los motivos detrás de cada una de estas funcionalidades y su modo de uso.

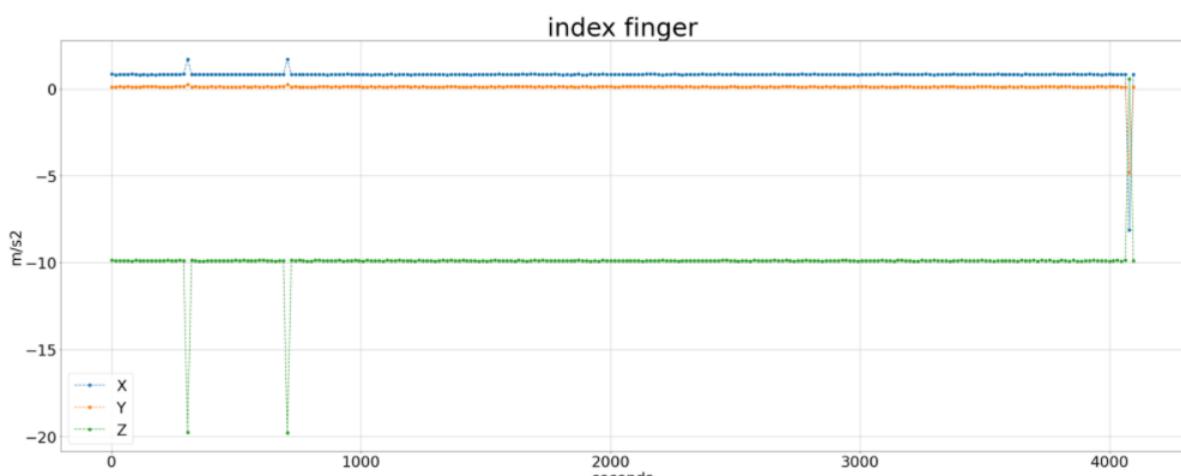
### 5.1. Calibración

Para prever que los sensores puedan sesgar sus mediciones y con ello proveer mediciones erróneas es importante contar con un mecanismo de calibración que mejore la precisión de las mediciones. Al igual que muchos otros sensores los mpu6050 deben calibrarse antes de ser utilizados por primera vez para eliminar el error cero, esto hace referencia a la pequeñas variaciones que pueden tener las mediciones del sensor, por ejemplo medir una mínima inclinación cuando están en posición horizontal y la inclinación debería ser nula. Esto se puede corregir con un valor offset que se le suma o se le resta a los valores de aceleración o velocidad angular medidos en crudo.

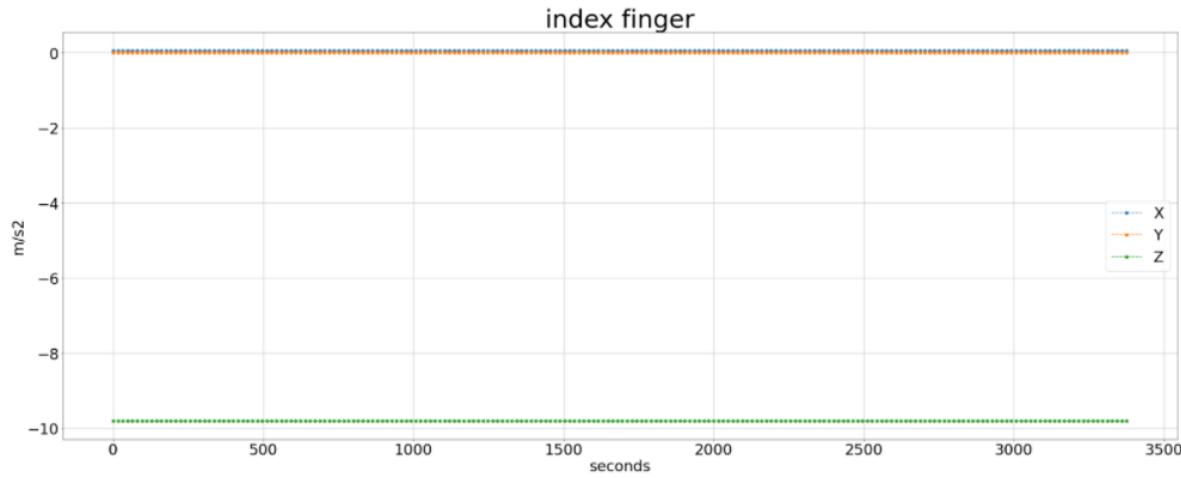
La calibración consta simplemente de una función de ajuste. Primero es necesario posicionar al sensor de una forma estática en la cual se sepan cuáles son los valores esperados de

las mediciones, generalmente cero salvo para la aceleración en el eje vertical que deberá ser igual al valor de la gravedad. Al mantener el sensor estático, la velocidad angular debería ser nula y el ángulo de inclinación calculada también. Se realizan un gran número de mediciones y se calcula el error medio de la muestra para cada eje. El error es la diferencia entre el valor esperado y el valor medido. Este valor se utiliza como offset en todas las siguientes mediciones.

La calibración generalmente se ejecuta al comienzo cuando se inicializa el sensor pero para este proyecto se decidió dejar al usuario el poder de decidir utilizar el guante sin calibrar o calibrar antes de usarlo. Para ello se implementó un botón de calibración en la aplicación que ejecuta el mecanismo de calibración para cada dedo. El usuario debe mantener los sensores estáticos en posición horizontal apoyando el guante con la palma hacia arriba sobre una mesa durante todo el proceso, lo cual se indica con un led azul. Es preferible que el usuario no tenga el guante puesto para la calibración por las pequeñas vibraciones que puede generar la mano humana. Cuando el led se apaga está indicando que el proceso de calibración termina para todos los sensores del guante y el usuario puede pasar a recolectar datos o traducir gestos. A continuación podemos ver las diferencias entre las mediciones de un sensor antes y después de ser calibrado.



**Figura 12:** Valores del acelerómetro del dedo índice descalibrado.



**Figura 13:** Valores del acelerómetro del dedo índice calibrado.

## 5.2. Recolección de datos

Como en todo proyecto que involucre inteligencia artificial, lo primero que se necesita es contar con los datos necesarios para alimentar a los algoritmos de inteligencia artificial y que puedan aprender. Debemos entonces implementar la funcionalidad de recolección de datos para grabar gestos, es decir obtener las mediciones de los sensores durante esos gestos, indicando qué tipo de gestos se están realizando. Una vez que se obtienen las mediciones de los sensores, alimentamos un pipeline de machine learning con el que vamos a obtener una red neuronal propiamente entrenada con los datos proporcionados y capaz de identificar con cierto grado de precisión a qué gesto corresponde el gesto que estemos realizando en tiempo real.

1. La recolección de datos se efectúa de la siguiente manera:
2. Encender y colocarse el guante
3. Conectar el guante mediante bluetooth a la aplicación de celular
4. En la aplicación, ir a “Recolección de datos”, seleccionar el gesto que se va a efectuar y darle al botón de grabar

5. Realizar el gesto con el guante del que se quieren recolectar datos.
6. Presionar el botón de stop en la aplicación para detener la recolección de datos.

Mientras se recolectan las mediciones, el guante va a estar transmitiéndolas mediante bluetooth a la aplicación, que se va a encargar de desserializar los mensajes recibidos y guardar en archivos las mediciones recolectadas.

### **5.3. Interpretaciones**

Una vez que se implanta el algoritmo de machine learning en el firmware del sistema embebido, se puede interpretar gestos realizados en tiempo real. Para hacer uso de esta funcionalidad es necesario:

1. Encender y colocarse el guante
2. Conectar el guante mediante bluetooth a la aplicación de celular
3. En la aplicación, ir a “Interpretaciones”
4. Presionar el botón de “Traducir”
5. Efectuar los gestos que uno pretende que la aplicación interprete.

## 6. Implementación del software del sistema embebido

En esta sección detallamos cómo se organiza el código embebido, cuál es su arquitectura y los motivos que llevaron a este diseño.

Sobre el lenguaje de programación utilizado, el código está realizado íntegramente en C++. Inicialmente habíamos optado por Rust, pero después de realizar una extensa investigación que detallamos en el anexo A, terminamos descartando esta opción. A modo de resumen, la razón detrás de esta decisión radica en que el Esp32 no soporta adecuadamente a Rust como lenguaje de programación y si bien es posible embeberle código en Rust, algunas funcionalidades del Esp32 clave para nuestro proyecto (tal y como la comunicación mediante bluetooth) no cuentan aún con soporte para este lenguaje.

A continuación enumeramos los paquetes que contienen el código de nuestro sistema embebido, detallando cuáles son las clases que los integran y el rol de cada una de ellas, así como cómo interactúan entre sí.

### 6.1. Communication/Ble

Se ubican las clases responsables de gestionar la comunicación entre el guante y la aplicación.

#### **BleCommunicator**

Clase encargada de comunicarse con la aplicación, contiene las características del servicio de bluetooth del guante, métodos para enviar mediciones e interpretaciones. Dispone además de callbacks a ejecutarse en caso de eventos asincrónicos como por ejemplo la conexión o desconexión de la aplicación.

### ServerCallbacks

Implementación de la clase `BleServerCallbacks` de la libería de espressif, dispone de los métodos `onConnect` y `onDisconnect` a ejecutarse cuando sucede una conexión o desconexión respectivamente.

### TasksControllerCallback

Clase encargada de reaccionar a los distintos comandos enviados desde la aplicación, siendo estos:

- `startdc`: iniciar recolección de datos
- `startint`: iniciar interpretaciones
- `stop`: detener (ya sea una recolección de datos o las interpretaciones)
- `calibrate`: calibrar el dispositivo

Debe su nombre a que estas operaciones (recolección de datos, interpretaciones, etc) se ejecutan en tasks (o hilos) separados, por lo que esta clase se encarga de iniciar o detener las tasks.

### BleSpecification

En esta carpeta además se ubica el archivo `BleSpecification.h` que contiene macros con los UUID (identificador único universal) del servicio del guante con sus características, que nos van a permitir identificar al dispositivo, sus servicios y características para así poder establecer la conexión e interactuar con la aplicación.

```
1 /** Service for reading measurements or retrieving interpretations from  
   the gloves. */  
2 #define LSA_GLOVES_SERVICE_UUID "7056f14b-02df-4dd8-86fd-0261c7b15c86"
```

```
3  
4 /** Characteristic for commanding the device. */  
5 #define CONTROLLER_CHARACTERISTIC_UUID "30b7db16-4567-42c5-acc4-2  
    b0270c1e14d"  
6  
7 /** Characteristic for reading measurements from the gloves. */  
8 #define DATA_COLLECTION_CHARACTERISTIC_UUID "47e62e53-e278-494d-a3f8-  
    ac00973ae0af"  
9  
10 /** Characteristic for receiving interpretations from the gloves. */  
11 #define INTERPRETATION_CHARACTERISTIC_UUID "079b8e74-101b-11ec-82a8  
    -0242ac130003"
```

## 6.2. TasksManager

Como explicamos más adelante, la programación concurrente fue necesaria para poder controlar el guante. El esp32 cuenta con dos núcleos en su procesador que permiten ejecutar código concurrente de manera eficaz, dado que en todo momento necesitamos tener un hilo de ejecución funcionando para poder controlar el guante desde la aplicación. Ese hilo está permanentemente escuchando los comandos enviados por el usuario desde la app. El otro hilo se ocupa de ejecutar los comandos recibidos, ya sea recolectar datos, interpretar gestos o calibrar el guante. Estos hilos se denominan Tasks, o tareas en inglés, de ahí el nombre del paquete, TasksManager, que contiene una única clase homónima. La clase TasksManager es la encargada de gestionar las tasks del software embebido, procesando los comandos recibidos desde la aplicación, iniciando o cancelando las tasks correspondientes a dichos comandos.

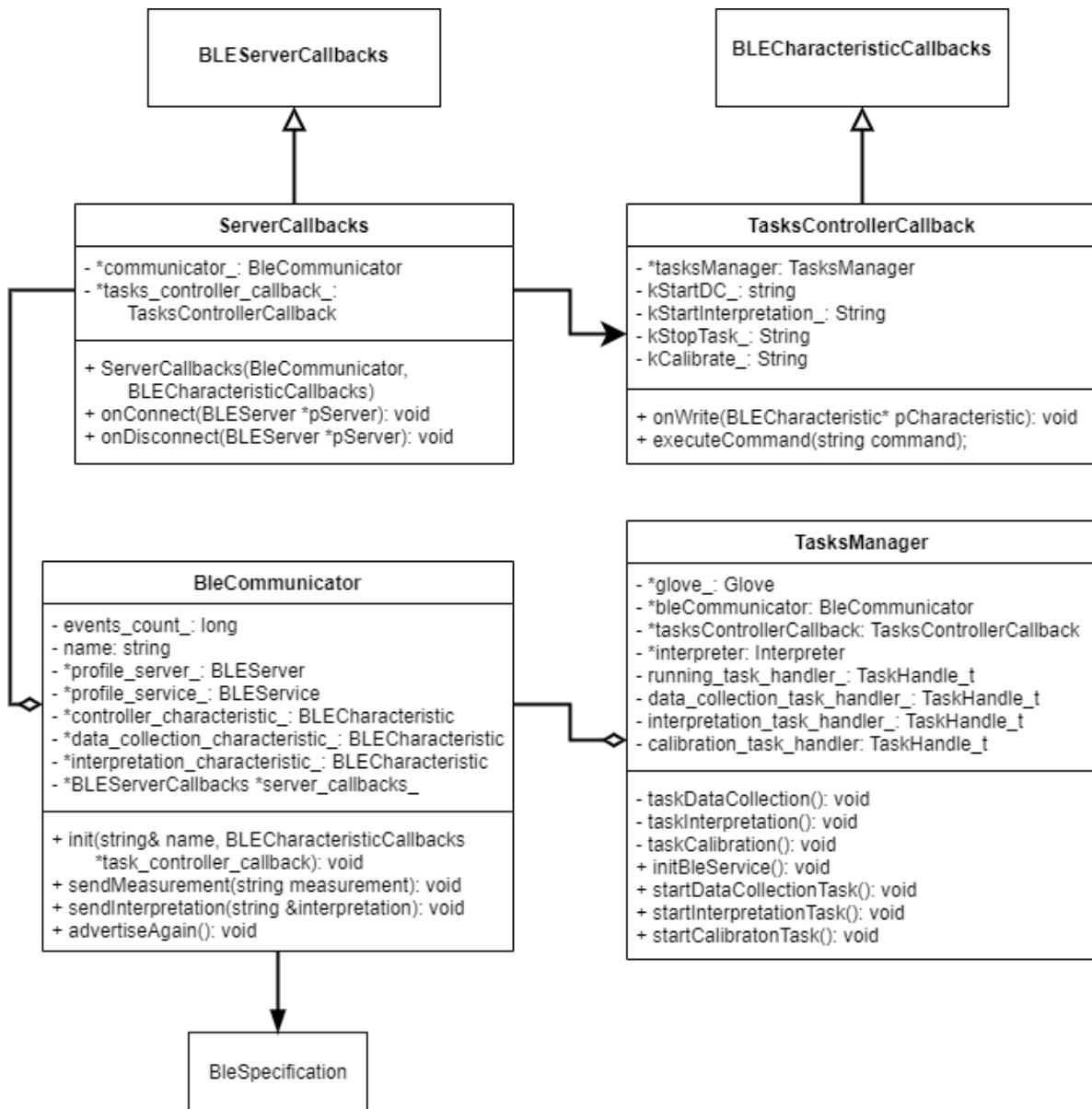
Los métodos públicos de la clase TasksManager son:

- void initBleService(): inicializa los servicios de bluetooth low energy

- `void startDataCollectionTask()`: inicializa la tarea de recolección de datos
- `void startInterpretationTask()`: inicializa la tarea de interpretación de gestos
- `void startCalibrationTask()`: inicializa la tarea de calibración
- `void stopRunningTask()`: mata la tarea que se esté ejecutando

Internamente contiene el código a ejecutarse en cada una de las distintas tasks.

TasksManager se asocia con la clase TasksControllerCallback, que hereda de la clase BleCharacteristicCallbacks (parte del framework de espressif para comunicación bluetooth). A su vez la clase BleCharacteristicCallbacks se asocia con la clase ServerCallbacks que hereda de la clase BLEServiceCallbacks (también parte del framework de espressif para la comunicación bluetooth). Estos vínculos entre clases son necesarios para poder controlar al guante desde la aplicación.

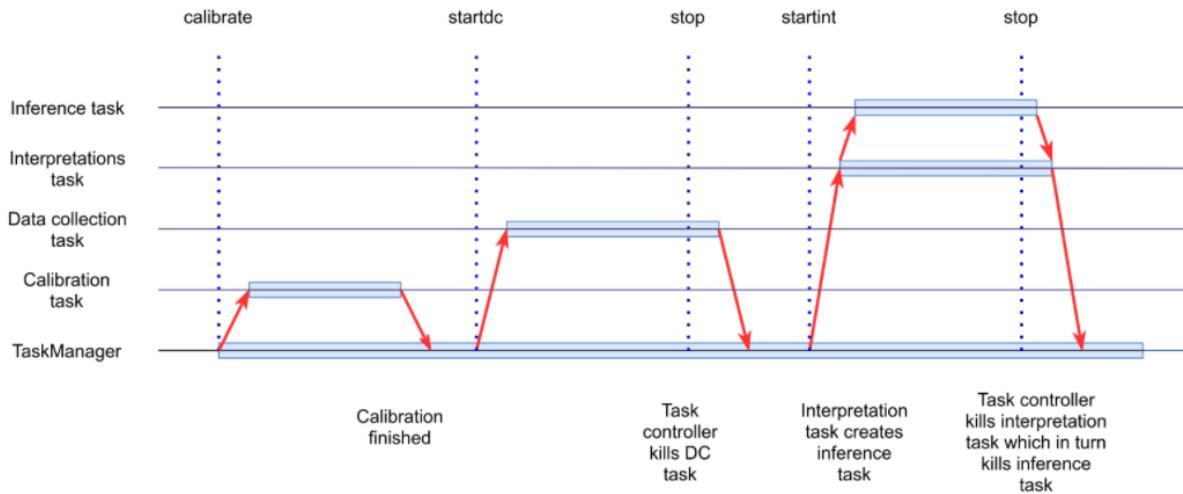


**Figura 14:** Diagrama de clases del paquete BLE, mostrando además la asociación con el TasksManager

Tal y como se muestra en la figura 14, la clase **TasksManager** se asocia con la clase **TasksControllerCallback**, que hereda de la clase **BLECharacteristicCallbacks** (parte del framework de espressif para comunicación bluetooth).

A su vez, la clase **BLECharacteristicCallbacks** se asocia con la clase **ServerCallbacks** que hereda de la clase **BLEServiceCallbacks** (también parte del framework de espressif para la comunicación bluetooth). Estos vínculos entre clases son necesarios para poder

controlar al guante desde la aplicación.



**Figura 15:** Tiempo de vida de las tasks en el embobido.

Como se muestra en la figura anterior, el TaskManager se mantiene siempre activo, ya que se encarga de ejecutar las operaciones que arriban desde la aplicación.

### 6.3. Glove

Contiene las clases que representan al guante propiamente dicho.

#### Mpu

Representa al MPU6050. Contiene lógica para modificar el estado del sensor, enviarle comandos y recibir mediciones. Contiene información acerca de a qué dedo está asociado el MPU.

#### Glove

Representa al guante, se compone de instancias de la clase de Mpu y contiene un método para iniciar la recolección de datos de los sensores y para calibrar los sensores. Además contiene información interna del guante.

**Finger**

Clase estática utilizada para representar a los dedos del usuario, dado que cada dedo va a tener un sensor conectado a un pin Ad0 distinto y es preciso identificarlo.

**Counter**

Contiene información acerca del número de medición obtenida en una sesión, así como el tiempo transcurrido desde el comienzo de la sesión hasta la obtención de la medición.

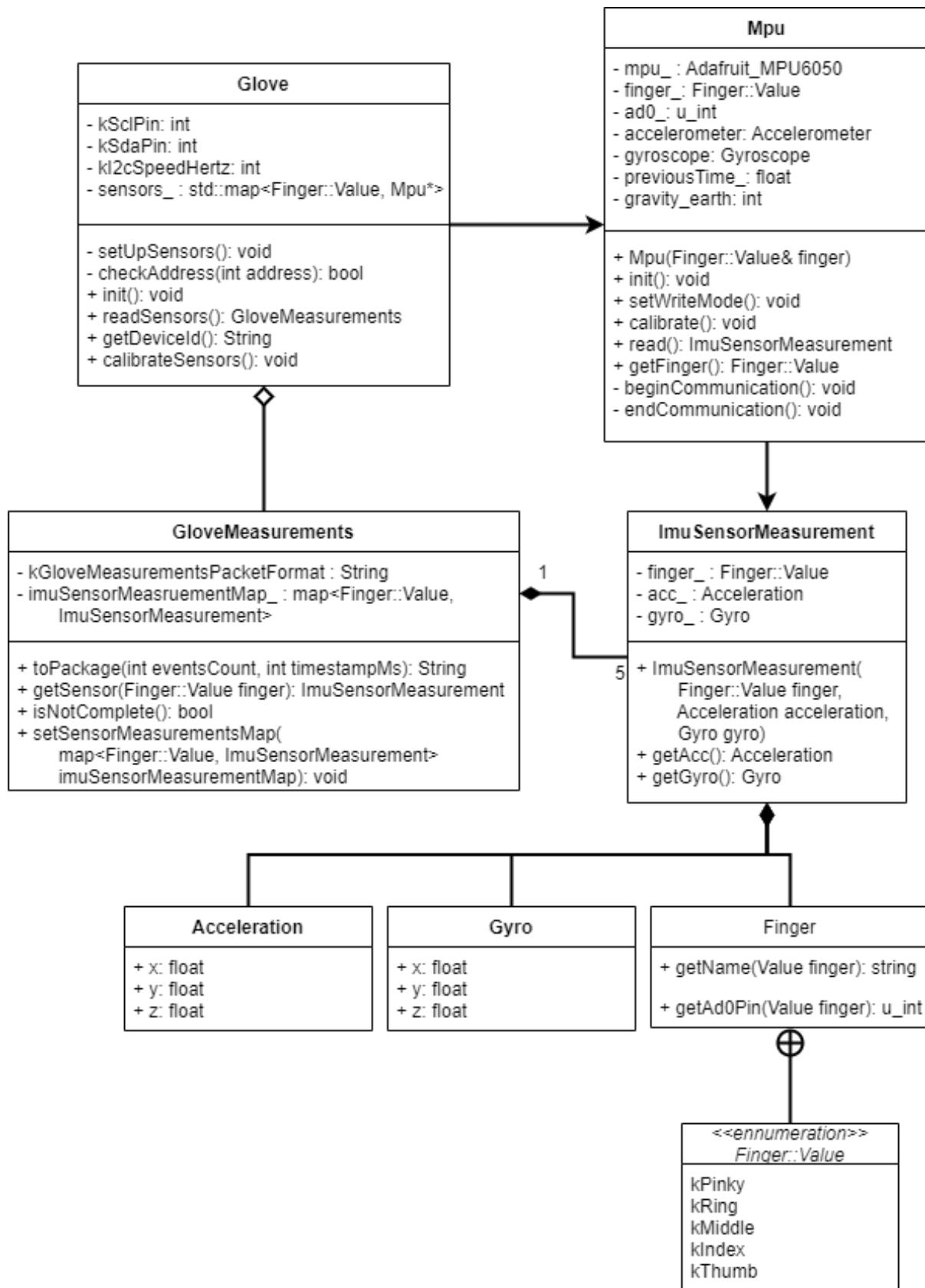


Figura 16: Diagrama de clases del paquete Glove con clases del paquete Sensors.

La clase Mpu se encarga de la comunicación entre el Esp32 y el sensor Mpu6050, mientras que la clase “Glove” se encarga de organizar la obtención de las mediciones de los sensores de todos los dedos. Es por esto que internamente esta última clase dispone del atributo sensors\_ que consiste en un mapa con punteros a instancias de Mpu, una por cada dedo. El método readSensors() devuelve una instancia de Glovemeasurements, que como mencionamos anteriormente contiene las mediciones de todos los sensores.

## 6.4. Sensors

Contiene la lógica para encapsular y empaquetar los datos obtenidos de los sensores del Mpu.

### Acceleration

La clase Acceleration representa una medición del acelerómetro. Como tal, contiene tres atributos internos: x, y, z, todos números de punto flotante.

### Accelerometer

Representa al sensor acelerómetro del Mpu. Esta clase no se encarga de realizar la lectura con el sensor (ese rol lo asume la clase Mpu), sino que contiene información asociada al sensor, como el rango del sensor y el error con el que corregir las mediciones.

### Gyro

La clase Gyro representa una medición del giroscopio del Mpu. Igual que la clase Acceleration, dispone de tres atributos internos x, y, z, números de punto flotante.

**Gyroscope**

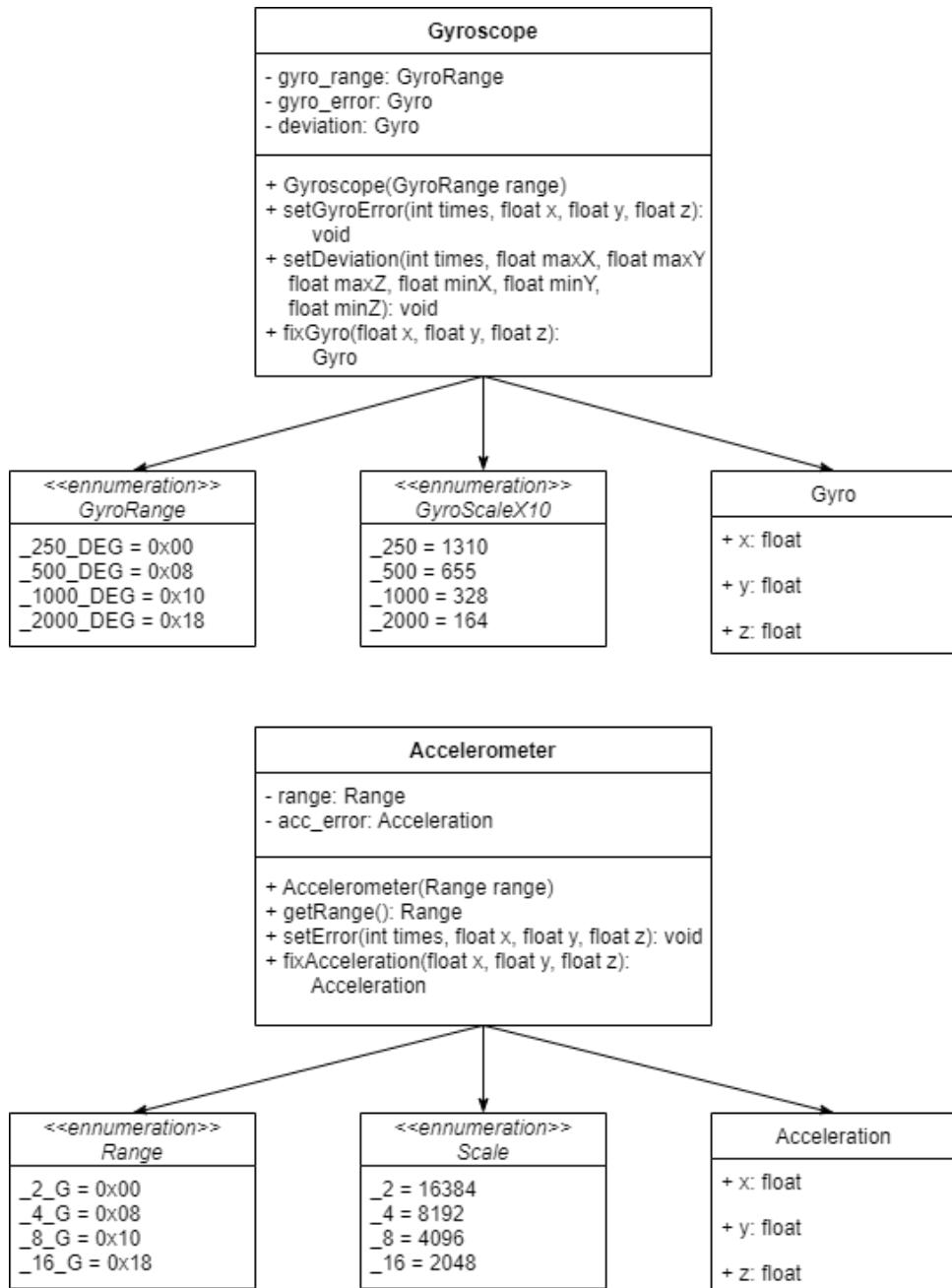
Similar a la clase Accelerometer, contiene información asociada al sensor giroscopio del Mpu. Esta información consiste en el rango y el error del sensor con el que poder corregir las mediciones.

**ImuSensorMeasurement**

Clase que agrupa las mediciones del acelerómetro y del giroscopio.

**GloveMeasurements**

Clase que representa una tanda de mediciones de todos los sensores IMU del guante, por ende, internamente contiene cinco instancias de ImuSensorMeasurement, una por cada dedo.



**Figura 17:** Diagrama de clases con las clases Accelerometer y Gyroscope.

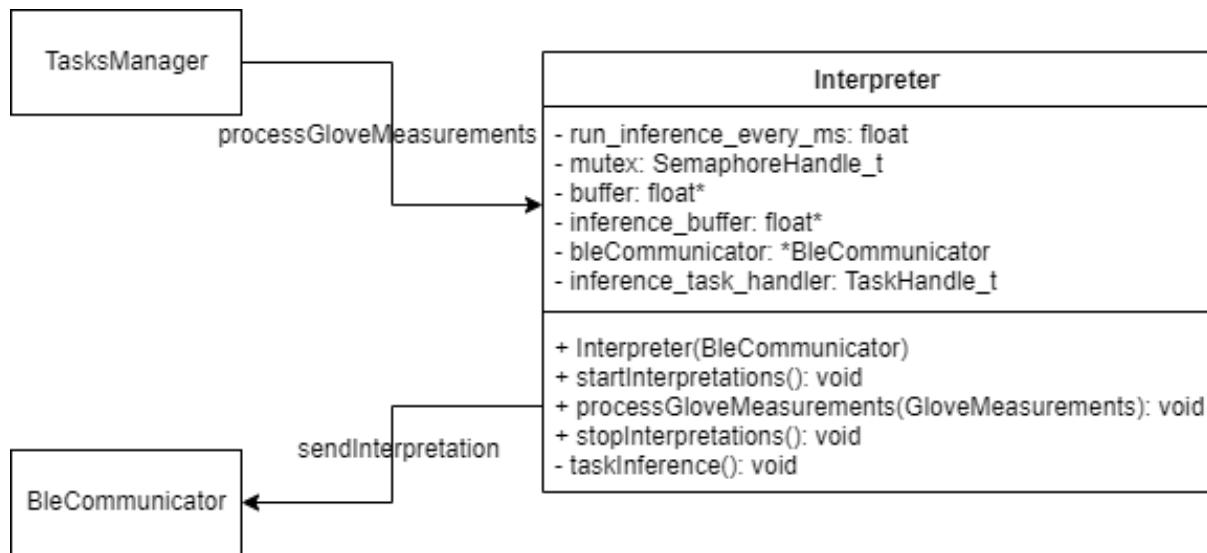
En la anterior figura podemos ver cómo las clases Accelerometer y Gyroscope encapsulan la información acerca de la configuración de los respectivos sensores, proveyendo además un método para corregir las mediciones de dichos sensores dado el rango y el error (o desvío) que acarrean.

Por otro lado, en la figura 16 podemos ver la arquitectura implementada para la obtención de los datos de los sensores de toda la mano. Esta información se encapsula dentro de la clase `GloveMeasurements`, que se compone de cinco instancias de la clase `ImuSensorMeasurement`, cada una de las cuales representa la medición de los sensores de la IMU de cada dedo. Es por esto que la clase `ImuSensorMeasurement` se compone de una instancia de `Acceleration`, `Gyro` (estas dos contienen los valores de los sensores) y por último de la `Finger`, esta clase representando a qué dedo corresponde la medición. Por último cabe mencionar que `Glovemeasurements` contiene el método `toPackage(int eventsCount, int timestampMs)`: `String` utilizado para serializar las mediciones para poder enviarlas a la aplicación mediante bluetooth, de forma tal que la clase `BleCommunicator` recibe un `glove measurement` y éste ya sabe cómo serializarse, ahorrándole esa responsabilidad al `BleCommunicator`.

También se puede ver en la figura 16 cómo funciona la comunicación entre el Esp32 y los sensores Mpu6050. La clase `Mpu` se encarga de la comunicación entre el Esp32 y el sensor `Mpu6050`, mientras que la clase `Glove` se encarga de organizar la obtención de las mediciones de los sensores de todos los dedos. Es por esto que internamente esta última clase dispone del atributo `sensors_` que consiste en un mapa con punteros a instancias de `Mpu`, una por cada dedo. El método `readSensors()` devuelve una instancia de `GloveMeasurements`, que como mencionamos anteriormente contiene las mediciones de todos los sensores.

## 6.5. Interpretation

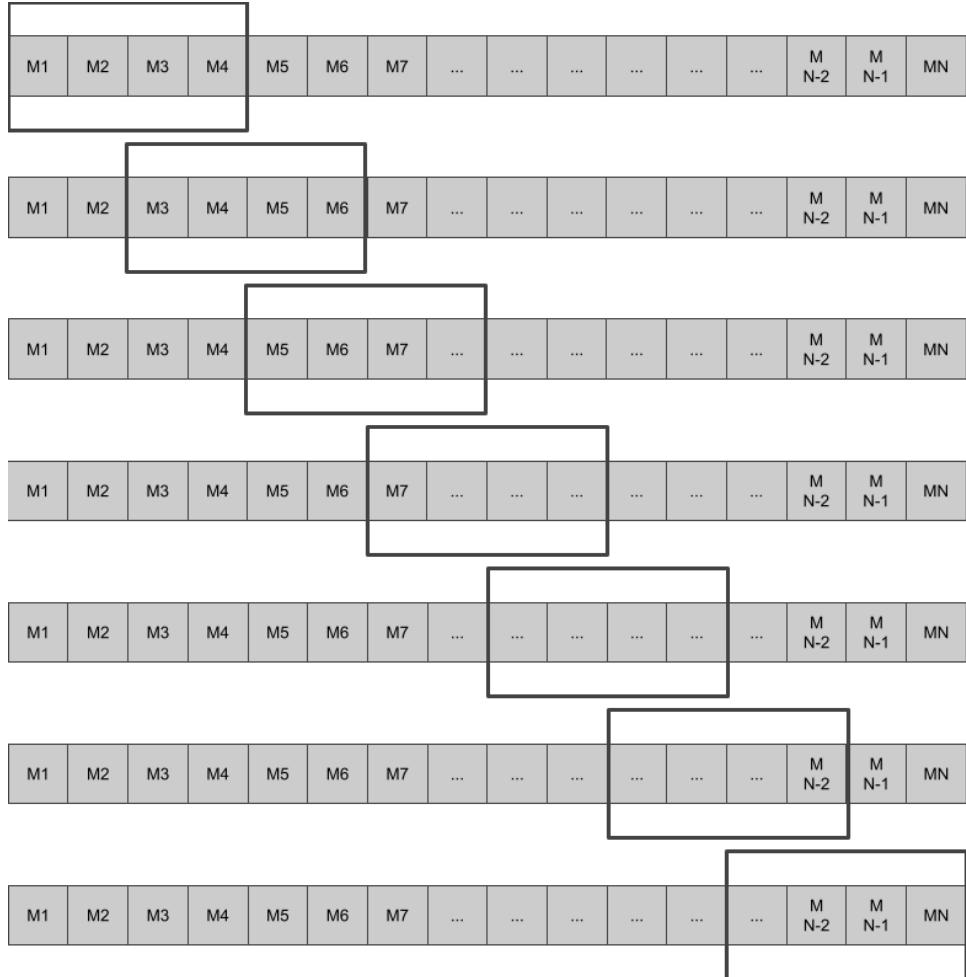
Este paquete contiene únicamente a la clase `Interpreter`. Aquí se halla la lógica para la interpretación de los gestos



**Figura 18:** Diagrama de clases del interprete.

Internamente, la clase Interpreter contiene una task; como mostrábamos en la figura 15, cuando se ejecuta la tarea de interpretación, tenemos 3 tasks corriendo en paralelo: la primera con el TasksManager, la segunda con la task de Interpreter y la tercera sería la task de Inference (inferencia) que es iniciada a partir de la segunda task.

Cuando se inicia la task de interpretación, se leen continuamente mediciones de los sensores de los dedos. Estas mediciones son almacenadas en un buffer en el heap con una modalidad de ventana deslizante, es decir que cada vez que llega una nueva medición del guante, se eliminan las mediciones más antiguas en el buffer, desplazando las mediciones restantes en el buffer y colocando la nueva medición en el espacio de memoria liberado.



**Figura 19:** El buffer en memoria actúa como una ventana deslizante. En esta figura se muestra cómo van pasando las mediciones en ese buffer a medida que van llegando.

La task de interpretación además inicia la task de inferencia ni bien arranca. Esta task de inferencia queda en espera hasta que transcurra el tiempo estipulado por el atributo `run_inference_every_ms`, configurado por defecto para ser 200ms. Cuando transcurre ese intervalo, en la task de inferencia se copian las mediciones contenidas en el buffer en ese momento y se las guarda en un segundo buffer de inferencia (`inference_buffer`) con el que se alimenta al motor de inferencia. Tener este mecanismo de dos buffers, permite que al mismo tiempo que se efectúa la inferencia de gestos, se pueda en paralelo seguir recibiendo mediciones. No olvidemos que la task de inferencia corre en paralelo con la task de interpretación que se encarga de recibir las mediciones de los sensores. Dicho de

otra manera, el motor de inferencia funciona en base a capturas de mediciones cada cierto intervalo de tiempo; no resulta conveniente que funcione el motor de inferencia por cada nueva medición que se recibe ya que esto sería muy ineficiente.

## 7. Programación de la aplicación

En esta sección abordamos lo relacionado con la aplicación mobile con la que el usuario interactúa para controlar el guante.

### 7.1. Flutter



La aplicación de celular está desarrollada utilizando Flutter. Se trata de un SDK creado por Google, pensado para desarrollar aplicaciones nativas (cross platform) para iOS y Android, aunque se puede programar programas en Flutter para otras plataformas (por ejemplo para desarrollo web). Está fuertemente orientado al diseño de interfaces y para ello provee una amplia gama de herramientas para facilitar el desarrollo front end.

En cuanto al backend, Flutter utiliza Dart, un lenguaje de programación también creado por Google orientado no solo al desarrollo de aplicaciones mobile sino también de aplicaciones web. Se espera que Flutter sea el framework principal para el desarrollo de aplicaciones para Fuschia, el nuevo OS en el que Google se encuentra trabajando.

Flutter compila directamente hacia el código final que interactúa con el procesador, saltándose los pasos intermedios, por lo que las aplicaciones resultantes tienen una mejor performance y mayor rendimiento. Además, Flutter usa sus propios Widgets y engine de renderizado llamado Skia Canvas desarrollado en C++.

## 7.2. Interfaz de usuario

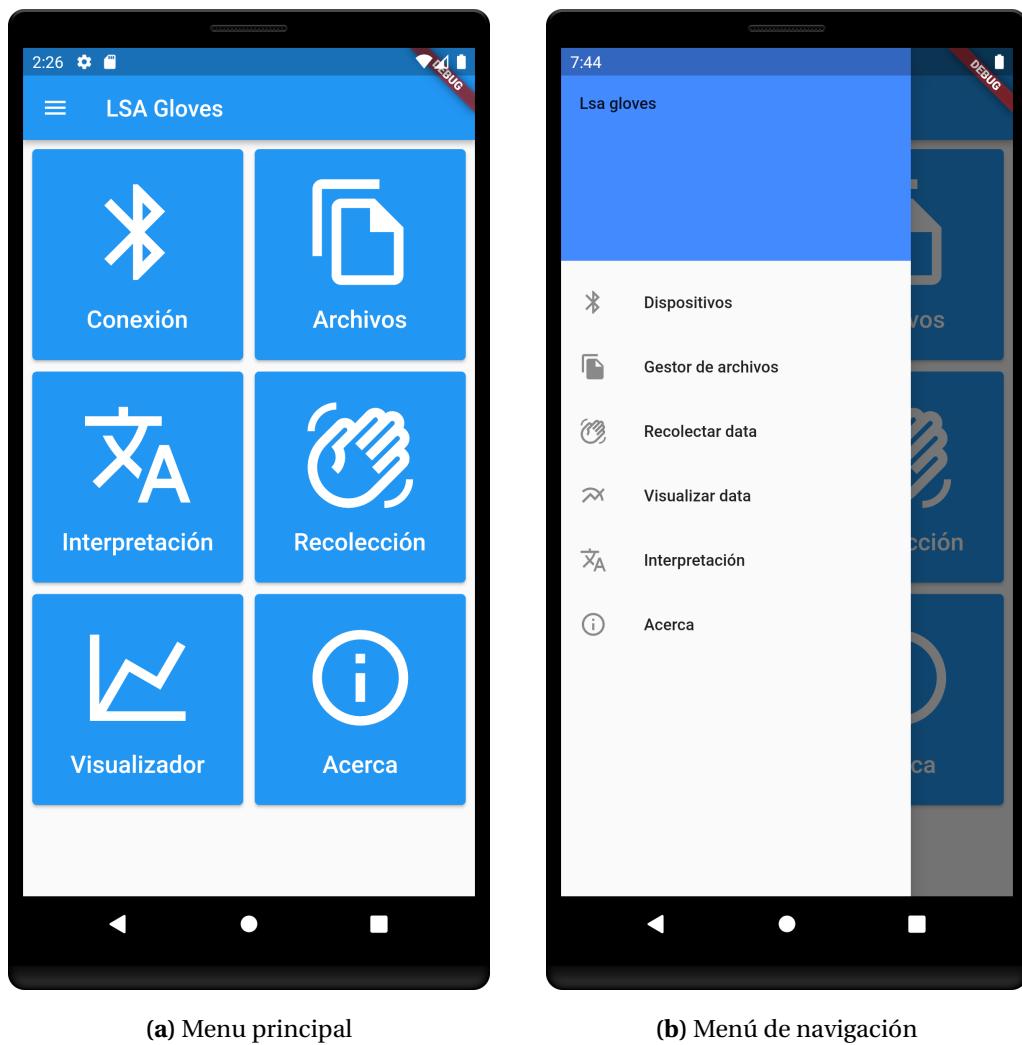
La aplicación permite:

- Conectarse o desconectarse con el guante
- Calibrar el guante
- Recolectar datos de gestos
- Gestionar los archivos generados al recolectar datos de gestos, con la posibilidad de subirlos a la nube al pipeline de machine learning de Edge Impulse
- Interpretar gestos
- Visualizar los datos de los sensores en tiempo real
- Leer un breve resumen sobre el proyecto

### 7.2.1. Navegación

Cuando el usuario accede a la aplicación, se encuentra con el siguiente menú principal que permite acceder a cada una de esas opciones.

Alternativamente, el usuario puede acceder a las funcionalidades utilizando el menú de navegación lateral, que se abre presionando el botón superior izquierdo con las tres líneas horizontales (popularmente denominado *hamburger*).

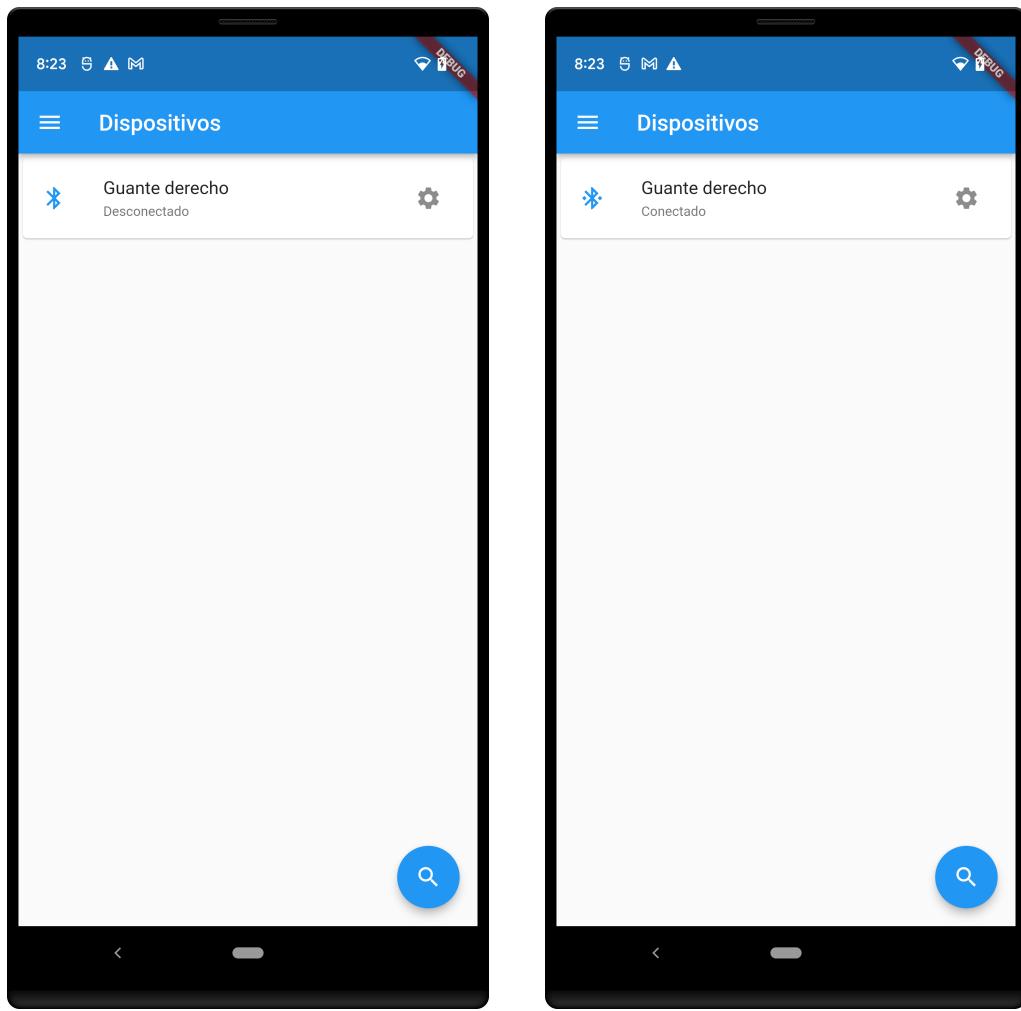


**Figura 20:** Las dos formas posibles con las que acceder a las distintas funcionalidades de la aplicación

### 7.2.2. Conexión y configuración

En el menú de conexión se van a listar los guantes disponibles que estén anunciándose mediante bluetooth. Para conectarse basta presionar sobre el elemento de la lista que se muestra.

Si no se halla ningún dispositivo, se debe presionar el botón de búsqueda para actualizar el listado.

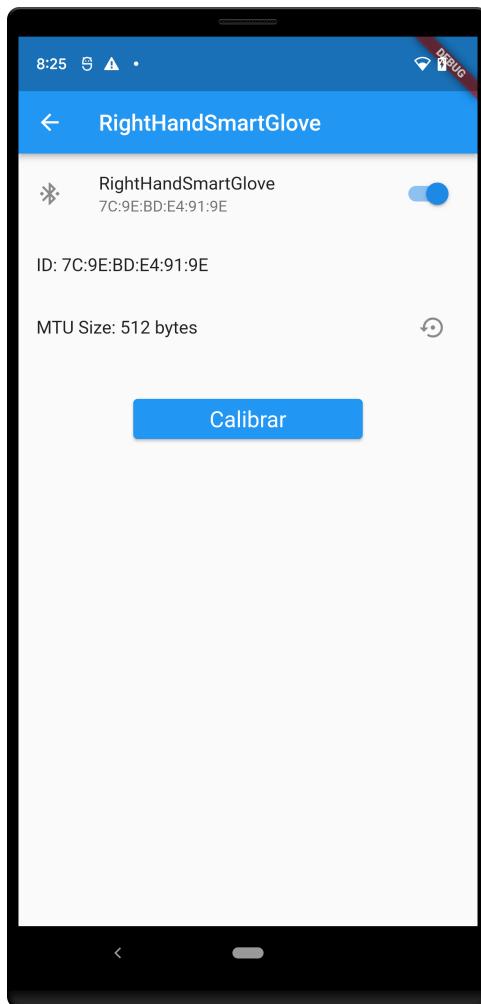


(a) Menú de conexión

(b) Dispositivo conectado

**Figura 21:** Pantalla de conexión en la que se muestra el estado del dispositivo (conectado o desconectado).

Además se dispone de una pantalla de configuración para cada uno de los dispositivos.

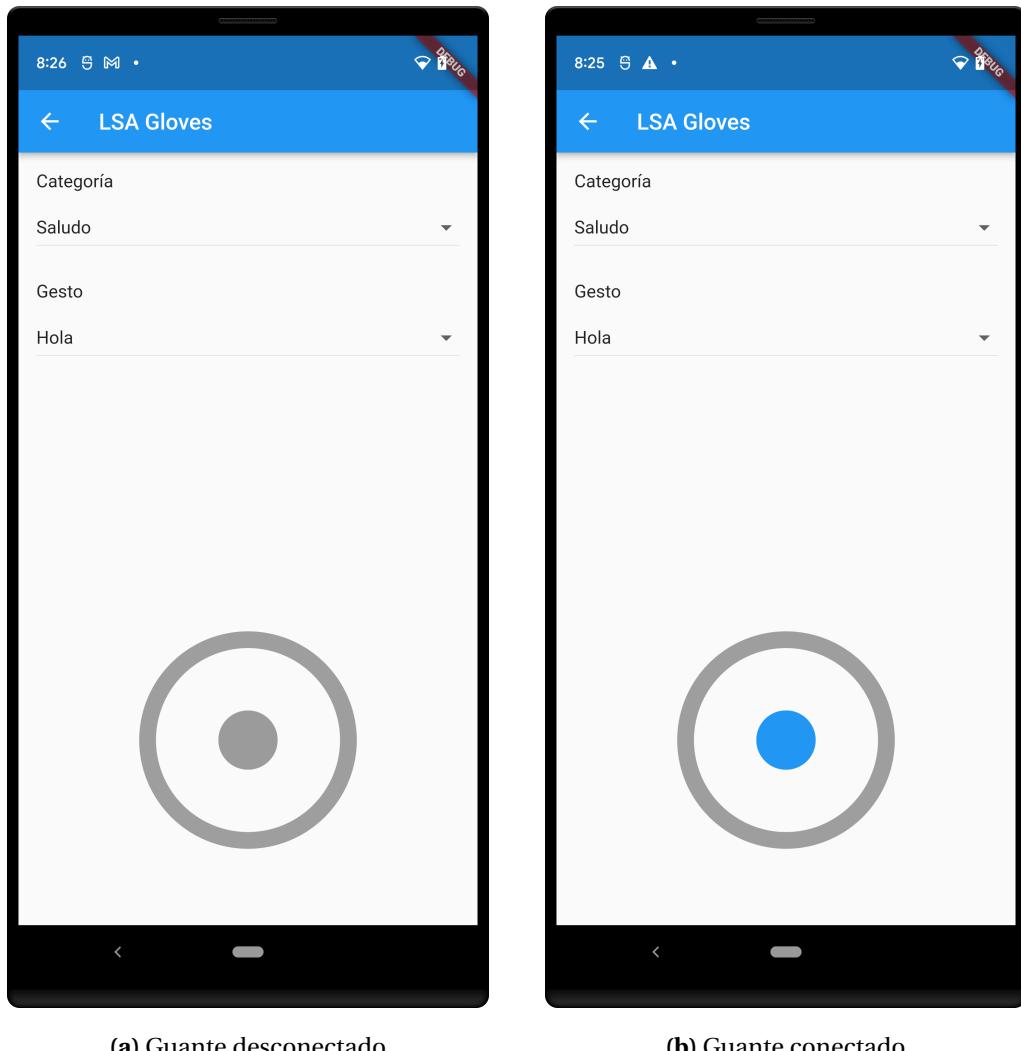


**Figura 22:** Menú de configuración

En esta pantalla se puede conectar / desconectar el guante y además se puede iniciar la tarea de calibración en el guante presionando el botón de "Calibrar". También se dispone de la posibilidad de modificar el MTU (*Maximum transmission unit*) empleado en la comunicación bluetooth<sup>10</sup>.

<sup>10</sup>Como se detalla más adelante, es necesario configurar el MTU a 512 bytes, de lo contrario la comunicación no funciona. Esta opción figura en la pantalla de configuración con propósitos de desarrollo.

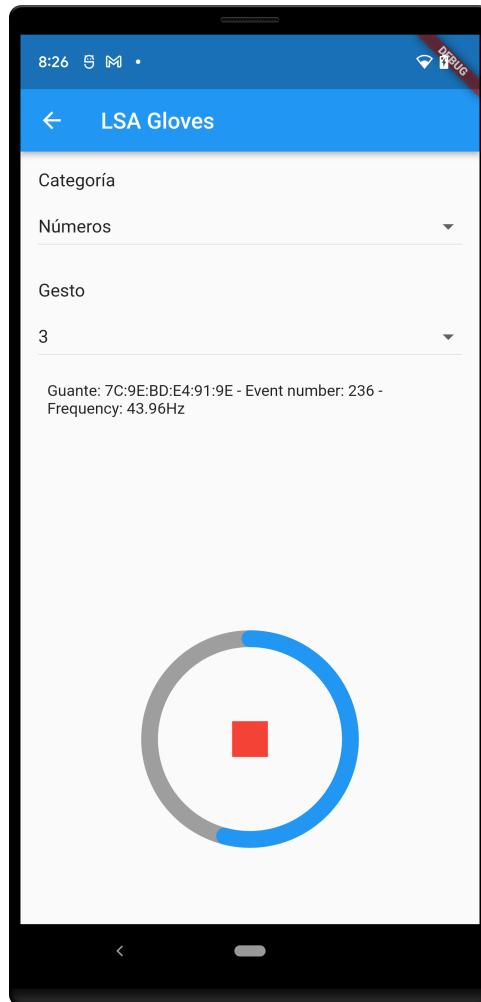
### 7.2.3. Recolección de datos



**Figura 23:** Pantalla de recolección de datos. El botón de grabar se habilita / deshabilita si el guante está conectado / desconectado.

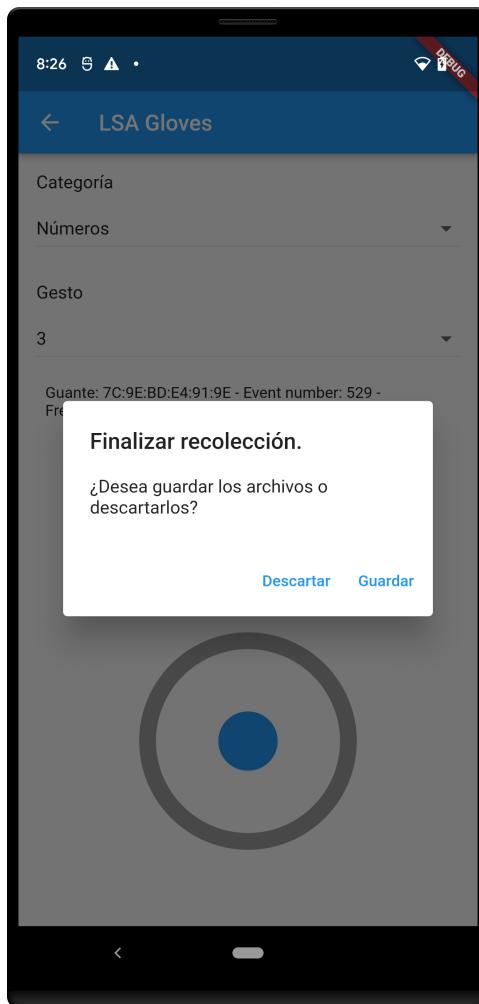
En la pantalla de recolección de datos, se debe seleccionar la categoría y el gesto específico que se quiere del cual se pretende recolectar datos de los sensores. Después, como se muestra en la figura 23 se dispone de un botón de grabar que está habilitado únicamente cuando el guante está conectado. Presionar el botón de grabar inicia una cuenta atrás de 3 segundos para darle tiempo al usuario (en caso de estar manipulando el mismo la aplicación y el

guante) para prepararse.



**Figura 24:** Recolección de datos en curso

Cuando se arranca la recolección de datos, vemos un círculo de progreso para tener notación del tiempo transcurrido, y además se muestra con propósitos de desarrollo información acerca de la cantidad de mediciones recibida y con qué frecuencia.

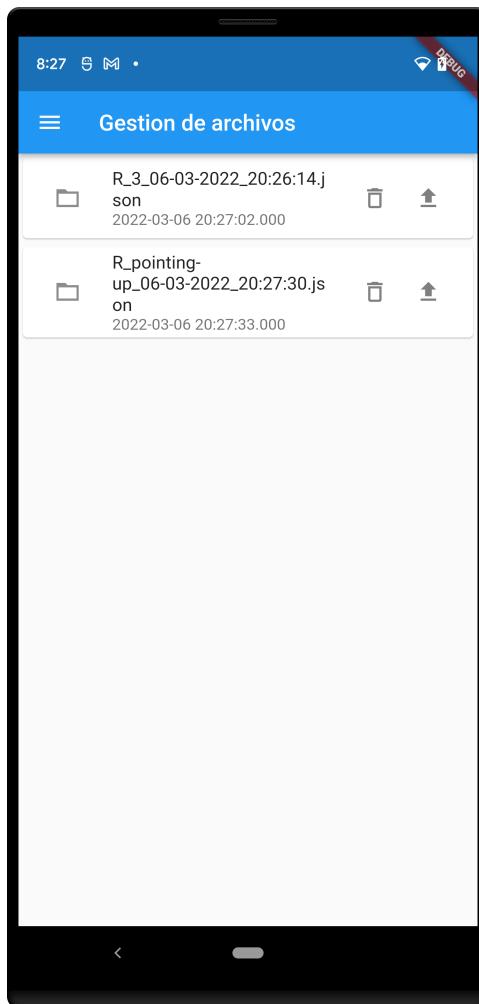


**Figura 25:** Finalizando la recolección de datos

Cuando se quiere detener la recolección de datos, se ofrece al usuario la posibilidad de descartar la recolección de datos mediante un diálogo (figura 25), caso contrario el archivo con los datos recolectados se guarda en disco.

#### 7.2.4. Gestión de archivos

Una vez realizadas una o más recolecciones de datos, el usuario puede gestionar los archivos generados. Se puede subir los archivos al pipeline de machine learning o bien eliminarlos.



**Figura 26:** Gestor de archivos

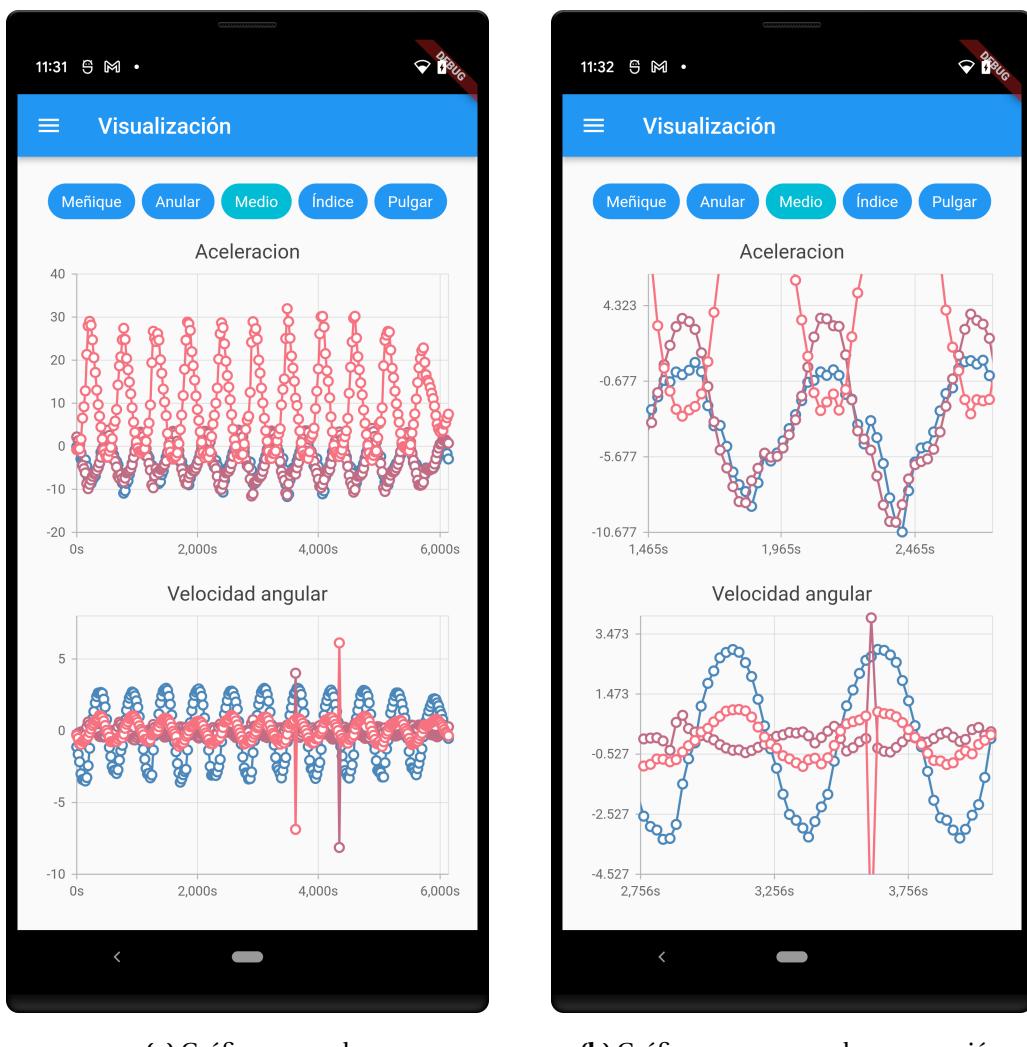
Para subir los archivos a la nube, basta con presionar el botón con el ícono de subir. Similarmente, para eliminar el archivo se debe presionar el ícono de la papelera. Los archivos una vez subidos se eliminan automáticamente.

Como mostramos a continuación, también se puede visualizar los datos de un archivo, para ello presionar sobre la tarjeta correspondiente a un archivo.

#### 7.2.5. Visualizador de datos recolectados

Con el fin de tener una primera impresión del resultado de nuestras recolecciones de datos, añadimos la funcionalidad de graficar los datos. De esta forma podemos descartar

rápidamente recolecciones que hayan sido defectuosas. Puede suceder que hayan muchas anomalías como por ejemplo las dos mediciones excéntricas que se ven en la velocidad angular de la figura 27. También puede suceder que se pierdan mediciones o que el guante se quede colgado o que directamente se desconecte en la mitad de la recolección de datos. Son posibilidades por lo que es imperioso realizar una validación de los datos recolectados y esta funcionalidad es una primera herramienta para este fin.

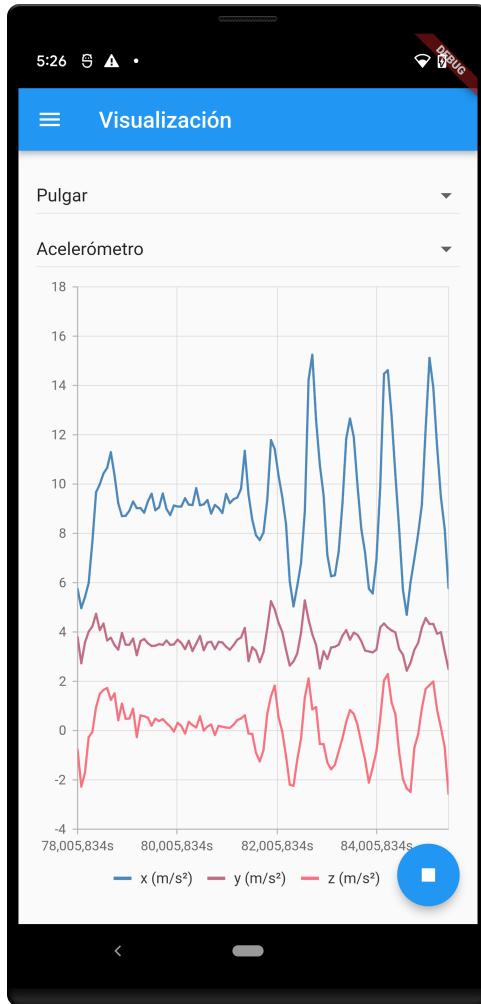


**Figura 27:** Pantalla de visualización de datos recolectados

Con el visualizador de datos podemos seleccionar el sensor del dedo del que queremos graficar los datos. También se cuenta con una funcionalidad de zoom tal y como se muestra

en la figura 27b, para poder ver con más detalle los datos en un periodo de tiempo.

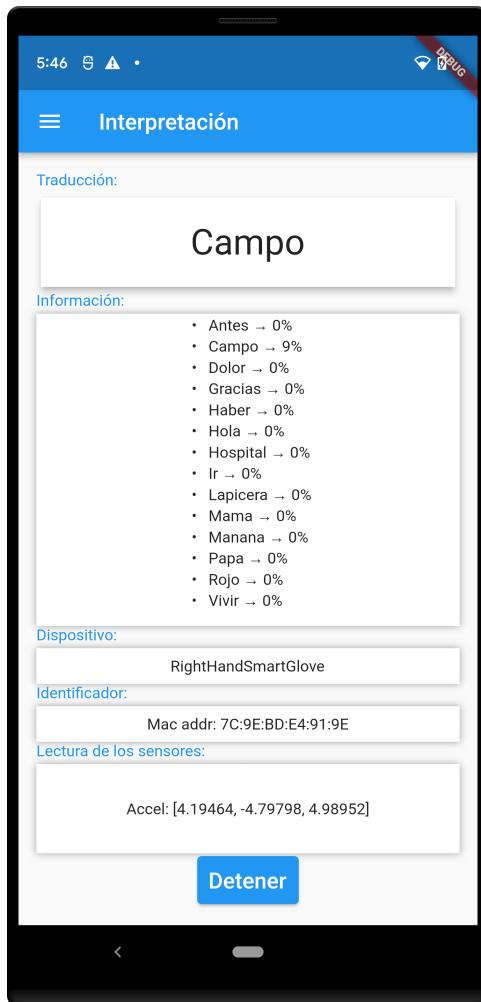
### 7.2.6. Visualizador de datos en tiempo real



**Figura 28:** Visualizador de datos en tiempo real

Otra herramienta que desarollamos con fines de desarrollo, para poder ver que los sensores estuvieran bien configurados y que no se nos estén mezclando sensores, es el visualizador de datos en tiempo real como se muestra en la figura 28. El usuario puede iniciar la recepción de datos de los sensores presionando el botón de play y ver cómo el gráfico se va actualizando en tiempo real con las mediciones recibidas. Se puede seleccionar el sensor de qué dedo y qué tipo de sensor, si acelerómetro o giroscopio.

### 7.2.7. Interpretaciones



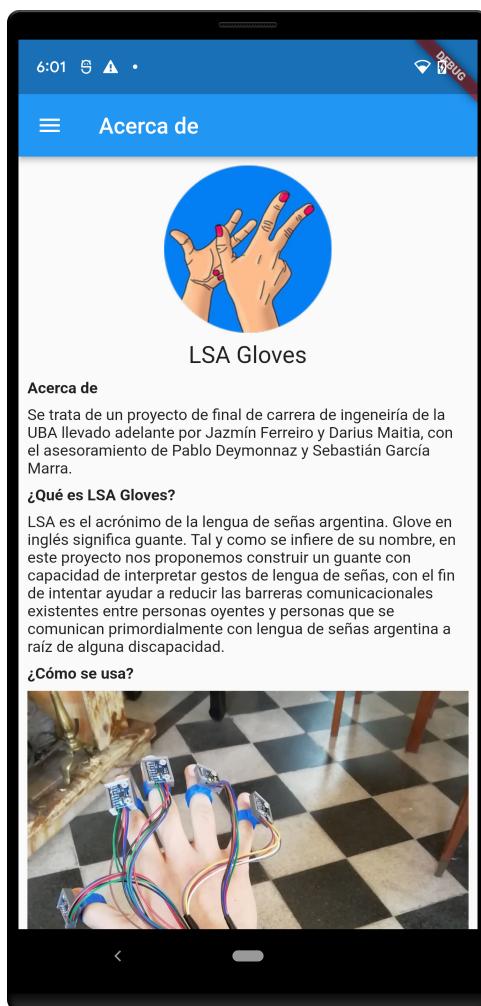
**Figura 29:** Pantalla de interpretación

La pantalla de interpretación nos permite ver las interpretaciones recibidas desde el guante que está ejecutando el algoritmo de machine learning. Cuando se infiere un gesto, este aparece en la sección superior de la pantalla. Abajo de ello se muestra los gestos con las probabilidades asociadas por el motor de inferencia. Después tenemos información acerca del guante, a saber el nombre del dispositivo y la dirección MAC del mismo. Por último se muestran los datos del acelerómetro de uno de los sensores; esto nos permite ver que se están recibiendo mediciones del guante correctamente.

Para iniciar o detener las traducciones, se debe presionar el botón inferior (Traducir / Detener).

### 7.2.8. Acerca de

Por último, menos importante pero también de utilidad para el usuario, hemos agregado una pantalla con información general acerca de la aplicación denominada "Acerca de", donde explicamos qué es la aplicación, por qué la hicimos, cómo se usa, e información de contacto.



**Figura 30:** "Acerca de"

### 7.3. Implementación del software de la app

El software de la aplicación se estructura en 6 paquetes:

- Connection/Ble: contiene toda la lógica para comunicarse con los guantes mediante Bluetooth
- Datacollection: contiene la lógica para almacenar los datos de los sensores de los guantes durante una sesión de recolección de datos
- Edgeimpulse: contiene el código para poder interactuar con la API de edge impulse y subir los archivos con datos de los sensores recolectados a la nube
- Glove: contiene clases y utilidades para representar el guante con sus sensores.
- Navigation: para manejar la navegación de la aplicación
- Pages: contiene todas las páginas o pantallas de la aplicación.

Por último, se dispone del archivo `main.dart` que no pertenece a ninguno de estos paquetes pero que representa la puerta de entrada a nuestra aplicación.

En esta sección haremos un análisis de las clases contenidas en cada uno de estos paquetes y cómo interactúan entre sí.

#### 7.3.1. Comunicación bluetooth

Lo central de la comunicación bluetooth se encuentra en el paquete `Connection/Ble` que contiene a las clases `BluetoothSpecification` y `BluetoothBackend`. En estas clases se encapsula toda la lógica de bajo nivel para comunicarnos con las aplicaciones, obteniendo los dispositivos conectados, sus servicios, características y estableciendo las configuraciones necesarias para el buen funcionamiento de la comunicación bluetooth con los guantes. Posteriormente, las clases que se dediquen por ejemplo a recolectar datos del guante o a recibir las interpretaciones, harán uso de estas clases para controlar a los guantes.

### 7.3.1.1 BluetoothSpecification

La clase `BluetoothSpecification` contiene puramente constantes estáticas que sirven para identificar e interactuar con los guantes; se dispone de UUIDs constantes para identificar a los servicios y a sus características, así como constantes con los comandos para iniciar ya sea la recolección de datos, las interpretaciones, las calibraciones o simplemente detener una tarea de estas que se esté ejecutando en el guante.

### 7.3.1.2 BluetoothBackend

El `BluetoothBackend` contiene la lógica necesaria para establecer una comunicación con los guantes y monitorear el estado de esas conexiones. Internamente se dispone de los siguientes atributos privados:

```
1 List<BluetoothDevice> _connectedDevices = [];  
2 Map<BluetoothDevice, BluetoothCharacteristic>  
    _controllerCharacteristics = Map();  
3 Map<BluetoothDevice, BluetoothCharacteristic>  
    _dataCollectionCharacteristics = Map();  
4 Map<BluetoothDevice, BluetoothCharacteristic>  
    _interpretationCharacteristics = Map();  
5 late Stream<List<BluetoothDevice>> connectedDevicesStream;  
6 late Stream<List<BluetoothDevice>> availableDevicesStream;
```

En Flutter, un Stream es un tipo de datos que varía indefinidamente a lo largo del tiempo y al cual uno se puede suscribir para actualizar la interfaz de usuario acorde a ello. En nuestro caso `connectedDevicesStream` y `availableDevicesStream` son Streams, porque en cualquier momento se pueden anunciar dispositivos a los que nos podemos conectar y también puede ser que se haya conectado o desconectado uno o más guantes. Resulta imperativo que nuestra aplicación pueda manejar esas situaciones de forma asíncrona, por ejemplo si se desconectan los guantes entonces debería deshabilitarse la funcionalidad de

recolección de datos o la de interpretaciones. `connectedDevicesStream` es un stream que monitorea periódicamente cada dos segundos los dispositivos que se encuentran conectados a la aplicación, mientras que `availableDevicesStream` se actualizará únicamente cuando el usuario presione el botón de búsqueda de dispositivos en la pantalla de conexión.

La lista `_connectedDevices` contiene una referencia a los dispositivos conectados. Sirve para poder detectar si hubo un evento de conexión o desconexión, ya que cada vez que se actualice el valor del stream `connectedDevicesStream` vamos a poder comparar con este listado y ver si tenemos más o menos dispositivos que antes.

Cuando se conecta un nuevo dispositivo, lo primero que se hace es descubrir cuales son sus servicios y características. Para obtener esta información se requiere de una comunicación ida y vuelta entre el guante y la aplicación, que puede demorarse un tiempo no despreciable, capaz de afectar a la experiencia de usuario. Por esto es que se dispone de los mapas de características `_controllerCharacteristics`, `_dataCollectionCharacteristics` y `_interpretationCharacteristics`. Estos mapas se actualizan ni bien sucede un evento de conexión o desconexión y de este modo logramos evitar tener que pedirle al guante la información de sus características cuando deseamos enviarle un comando desde la aplicación, directamente ya tenemos almacenada esa información en el bluetooth backend.

Se suceden además algunas configuraciones necesarias para el buen funcionamiento de la comunicación entre los guantes y la aplicación. Cuando se establece una nueva conexión sucede lo siguiente:

1. Se habilitan las notificaciones para las características (esta es la forma mediante la cual nos comunicamos, no utilizamos indicaciones)
2. Se modifica el MTU (*maximum transmission unit*) a 512 bytes.

Las notificaciones en bluetooth son un mecanismo de comunicación con una política de "dispara y olvida": si se pierde un paquete lo obviamos. Esto es diferente de las indicaciones en las que cuando se envía un paquete mediante indicaciones, el servidor (en este caso el

guante) se quedaría a la espera de recibir un ACK, es decir una confirmación de recepción de parte del cliente (la aplicación). Se puede establecer la analogía de que las indicaciones y notificaciones son similares a los protocolos de comunicación TCP y UDP en la capa de transporte de una conexión a internet. La decisión de utilizar exclusivamente notificaciones se justifica por el hecho de que priorizamos eficiencia por sobre integridad; no nos importa si perdemos un paquete de mediciones de entre los 50 que recibimos por segundo (no afecta demasiado a la recolección de datos), y el hecho de no tener que hacer acuso de recibo por parte de la aplicación permite que el guante no se deba quedar esperando a ese ACK, haciendo que el envío de mediciones sea más fluída.

Después de configurar las notificaciones, configuramos el MTU para que sea de 512 bytes en vez del default de 20 bytes. Esto nos permite recibir un paquete entero de mediciones del guante de una sola vez. Cuando se establece una conexión, automáticamente se configura el MTU para ser de 512 bytes, de no hacerse fallaría la comunicación dado que la aplicación espera recibir esa cantidad de bytes.

Cabe mencionar que la clase `BluetoothBackend` implementa la clase `ChangeNotifier`<sup>11</sup>, parte de una librería de Flutter para gestionar el estado de la aplicación. Esta librería nos permite tener una única instancia de la clase que viva en el mismo *lifespan* que la aplicación y que sea fácilmente accesible desde cualquier widget que compone el árbol de la aplicación. Un `ChangeNotifier` es asemejable a un `ViewModel` en Android nativo. Es por esto que en la función `main()` del archivo `main.dart` se tiene un `ChangeNotifierProvider`:

```
1 void main() {  
2     runApp(  
3         ChangeNotifierProvider(  
4             create: (_) => BluetoothBackend(),  
5             child: LsaGlovesApp(),  
6         )  
7     );  
}
```

<sup>11</sup><https://docs.flutter.dev/development/data-and-backend/state-mgmt/simplechangenotifier>

8 }

El ChangeNotifierProvider crea una instancia de BluetoothBackend que va a ser accesible desde todos los widgets hijos del ChangeNotifierProvider, en este caso, de todos los widgets de la aplicación dado que le estamos pasando la aplicación entera (LsaGlovesApp) como hijo. Para acceder a esta instancia de BluetoothBackend, basta con utilizar un Provider:

```
1 BluetoothBackend backend = Provider.of<BluetoothBackend>(context);
```

Para poder estar al tanto de cambios en la instancia de BluetoothBackend, podemos emplear un Consumer que va a ser notificado de esto (siguiendo un clásico patrón de listener - subscriber):

```
1 Consumer<BluetoothBackend>(builder: (context, backend, _) {  
2     ... //Código a ejecutarse ante un cambio en el bluetooth backend  
3 }
```

Por ejemplo, en la pantalla de recolección de datos de la figura 23, tenemos un Consumer del BluetoothBackend que va a ser notificado en caso de conexión o desconexión del guante, habilitando o deshabilitando el botón para iniciar la recolección de datos. Para notificar a los listeners, dentro de BluetoothBackend llamamos al método `notifyListeners()` que se llama únicamente cuando sucede un evento de conexión o desconexión dentro del método interno `_startMonitoringDevices()`:

```
1 void _startMonitoringDevices() {  
2     this.connectedDevicesStream.listen((newConnectedDevices) async {  
3         if (this._connectedDevices.length != newConnectedDevices.length) {  
4             developer.log("Connection event.", name: TAG);  
5             _assertAnyDisconnection(  
6                 this._connectedDevices, newConnectedDevices);  
7             this._connectedDevices = newConnectedDevices;  
8             _updateState(newConnectedDevices)  
9                 .then((_) => _configureCharacteristics())  
10                .then((_) => _requestMtu(newConnectedDevices))
```

```
11     .then((_) {
12
13         developer.log("Notifying listeners...", name: TAG);
14
15         notifyListeners();
16
17     });
18 }
```

### 7.3.2. Representación de las mediciones

#### 7.3.3. Recolección de datos

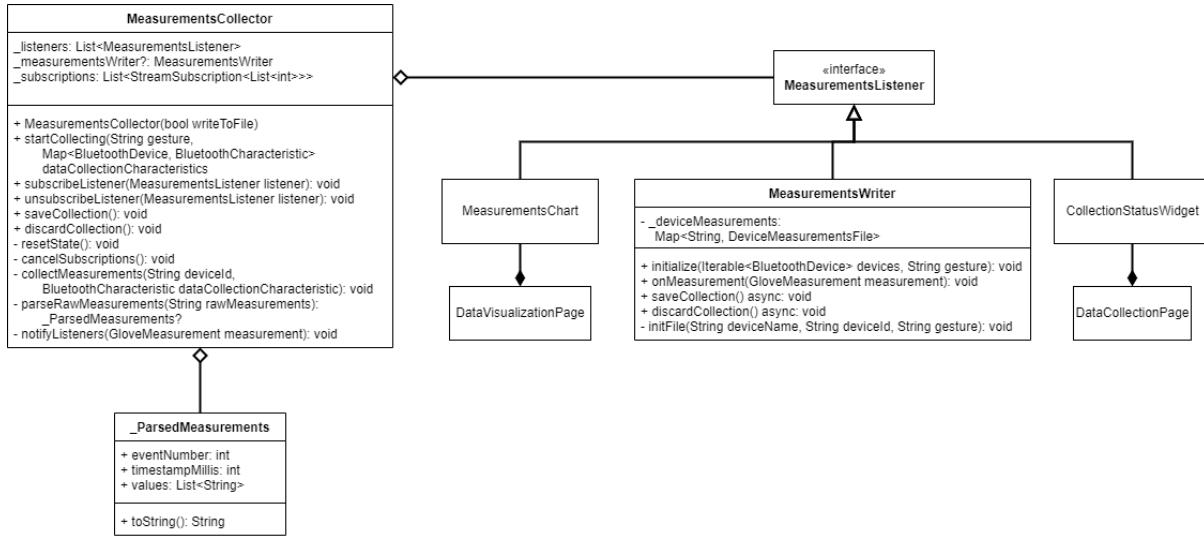
La lógica detrás de la recolección de datos se ubica primordialmente dentro del paquete `datacollection` que contiene los siguientes archivos:

- `measurements_collector.dart`
- `measurements_listener.dart`
- `measurements_writer.dart`
- `storage.dart`

Además la página de la figura 23 que permite el inicio de una sesión de recolección de datos se ubica en la carpeta `pages` y es la `ble_data_collection_page.dart`.

##### 7.3.3.1 Measurements\_collector.dart

La clase `Measurements_collector.dart` se encarga de recibir las mediciones del guante y de propagarlas a las clases interesadas como el `MeasurementsWriter` (encargada de escribir las mediciones en un archivo). Para esto, el `measurements collector` dispone de una lista de suscriptores a los que notificará cada vez que reciba una nueva medición. Los suscriptores implementan la interfaz `MeasurementsListener`.

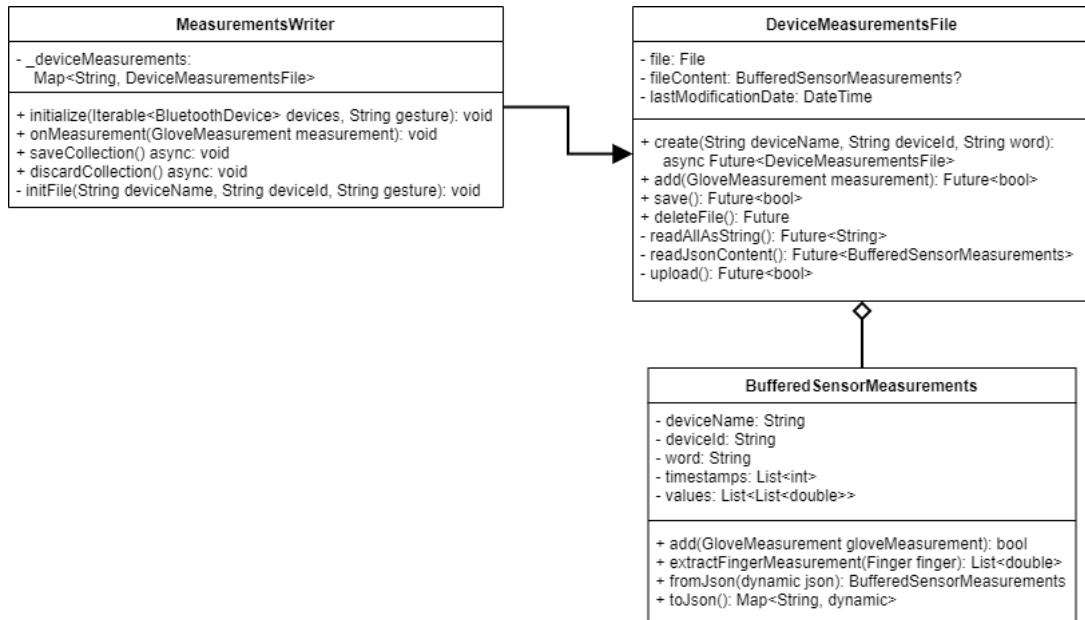


**Figura 31:** Diagrama de clases del MeasurementsCollector

La idea del MeasurementsCollector es desacoplar la tarea de recibir y parsear las mediciones del guante de la tarea de guardar las mediciones en un archivo. El desacoplamiento es necesario porque hay elementos de la aplicación que quisieran recibir las mediciones del guante y hacer algo con ellas, sin necesariamente guardarlas en un archivo. Un ejemplo de esto es el DataVisualizer de la figura 28. La pantalla de visualización de datos en tiempo real busca mostrar en pantalla los valores de las mediciones que se van recibiendo pero sin guardar en memoria esas mediciones. Otro ejemplo es el widget para mostrar el estado de la recolección de datos en la pantalla de recolección de datos. Este widget, llamado CollectionStatusWidget muestra la cantidad de mediciones recibidas y la frecuencia. Para esto necesita recibir las mediciones para procesarlas a su manera. Estos widgets se suscriben al MeasurementsCollector y cada vez que este recibe una medición, la propagará a sus suscriptores.

En caso de querer guardar en disco las mediciones recibidas, el MeasurementsCollector va a tener como suscriptor a una instancia de la clase MeasurementsWriter, encargada de escribir en un archivo las mediciones recibidas. De hecho, para mayor facilidad, la clase MeasurementsCollector recibe en su constructor un valor booleano que indica si se debe guardar

en disco o no. En caso de que se especifique que sí, internamente generará una instancia de MeasurementsWriter que automáticamente se suscribe al MeasurementsCollector para recibir las mediciones.



**Figura 32:** Diagrama de clases del MeasurementsWriter

Como se muestra en la figura 32 dispone de un mapa de DeviceMeasurementsFile asociado cada uno a un guante conectado, esto es porque la aplicación está preparada para recolectar información de más de un guante en simultáneo. De hecho el método initialize recibe por parámetro la lista de los dispositivos conectados y genera una instancia de DeviceMeasurementsFile por cada uno de ellos. Cada vez que se recibe un GloveMeasurement a través de onMeasurement(GloveMeasurement), se le solicita al DeviceMeasurementsFile apropiado que guarde esta medición.

Internamente, el DeviceMeasurementsFile se asocia con la clase BufferedSensorMeasurements que guarda las mediciones. Una vez la recolección de datos finalizada, se guardan las mediciones en un archivo. Esta es una limitación dado que las mediciones se mantienen en memoria hasta finalizarse la recolección de datos y una recolección de datos muy larga podría generar problemas de memoria. Una mejor implementación que se podría hacer

implicaría ir guardando periódicamente las mediciones en el archivo, liberando memoria cada vez.

#### 7.3.4. Gestión de los archivos

De la gestión de los archivos se encarga la clase `FileManager` ubicada en el archivo `storage.dart`. El `FileManager` se ocupa de crear los archivos y también de recuperarlos desde el espacio de almacenamiento reservado para la aplicación<sup>12</sup>. La pantalla de gestión de archivos de la figura 26 hace uso de esta clase para obtener los archivos a mostrar en la interfaz.

Una vez que se dispone de archivos con mediciones, queremos poder subirlos a la nube, es decir al pipeline de Edge Impulse. Para esto disponemos de la clase `EdgeImpulseApiClient` en el archivo `api_client.dart` ubicado en el paquete `edgeimpulse`. La clase `EdgeImpulseApiClient` dispone de toda la lógica necesaria para poder subir los archivos con el formato adecuado especificado por Edge Impulse<sup>13</sup>. Para esto es que está la función `uploadFile` en esa clase, que es invocada al presionar el botón de subir en la pantalla de gestión de archivos.

#### 7.3.5. Mediciones recibidas

Un único paquete de datos recibido mediante bluetooth desde el guante contiene las mediciones de los cinco sensores en un instante. A este paquete tenemos que parsearlo y procesarlo, generando instancias de clases más alto nivel que representan estas mediciones de los sensores. En esta sección detallamos cómo es el protocolo de comunicación y cuáles son las clases del lado de la aplicación que utilizamos para representar a estas mediciones.

---

<sup>12</sup>Lo que en Android nativo se conoce como "App specific storage"

<sup>13</sup><https://docs.edgeimpulse.com/reference/data-acquisition-format>

### 7.3.5.1 Formato de las mediciones

En el archivo `GloveMeasurements.cpp` del código del guante, tenemos la siguiente constante que diagrama el formato con el cual se envían las mediciones a la aplicación:

```
1 const std::string GloveMeasurements::kGloveMeasurementsPacketFormat =
2     "%d\n%lu\nP%.3f , %.3f , %.3f , %.3f , %.3f\nR%.3f , %.3f , %.3f , %"
3     ".3f , %.3f , %.3f\nM%.3f , %.3f , %.3f , %.3f , %.3f\nI%.3f , %.3f , %"
4     "%.3f , %.3f , %.3f , %.3f\nT%.3f , %.3f , %.3f , %.3f , %.3f ;";
```

Los caracteres `\n` sirven como delimitadores y el carácter `;` indica el final del paquete. Las secciones delimitadas son las siguientes:

1. número de evento
2. timestamp (ms)
3. mediciones de los dedos
  - a) meñique
  - b) anular
  - c) medio
  - d) índice
  - e) pulgar

donde cada una de las mediciones se subcompone en:

1. Acceleración eje X
2. Acceleración eje Y
3. Acceleración eje Z
4. Velocidad angular eje X

### 5. Velocidad angular eje Y

### 6. Velocidad angular eje Z

Un ejemplo de medición es el siguiente:

```
1      5                                     //Event count
2      22900                                //Timestamp (ms)
3      P -3.481,-7.149,6.627,-0.056,-0.018,-0.007 //Menique (pinky)
4      R -4.647,-6.421,5.717,-0.027,-0.495,-0.008 //Anular (ring)
5      M 7.362,-7.044,3.165,-0.019,-0.058,0.001    //Medio (middle)
6      I 8.025,-6.167,1.451,-0.030,-0.019,-0.004   //Indice (index)
7      T 9.313,-1.834,1.109,0.019,-0.091,0.012;     //Pulgar (thumb)
```

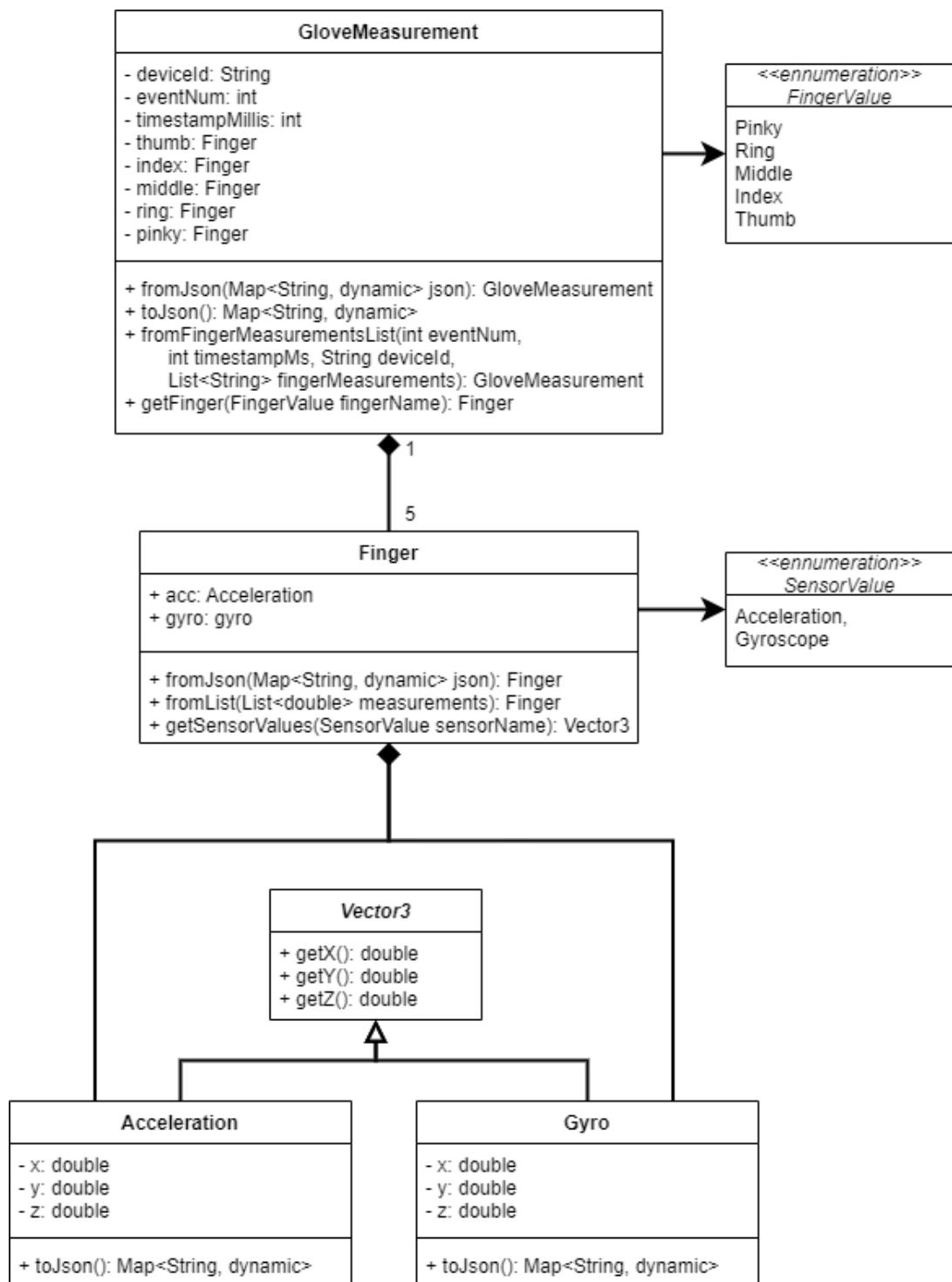
**Listing 1:** Ejemplo de envío de medición

Cuando arriba al guante, se descompone el mensaje en cada una de las mencionadas secciones, verificando que la cuenta de secciones sea la correcta, es decir que tienen que haber 7 secciones, caso contrario se descarta la medición ya que puede suceder que se reciba un paquete a la mitad. Cuando se tiene una medición del guante completa, la clase `GloveMeasurement` dispone de un método denominado `fromFingerMeasurementsList` que recibe por parámetro el número de evento, el timestamp en milisegundos, el id del dispositivo y la lista de mediciones de los sensores de los dedos, devolviendo una instancia de `GloveMeasurement`.

La recepción parcial de mediciones fue un problema al que debimos enfrentarnos y que explica la razón por la que utilizamos Strings en lugar de bytes crudos con Floats y demás. El protocolo Bluetooth utiliza un MTU (Maximum Transmission Unit) de 23 bytes por defecto. Un paquete recibido tendrá como mínimo 23 bytes, pero extender el MTU a 512 bytes como hicimos nosotros no garantiza que los paquetes recibidos tengan siempre ese tamaño. A veces se fragmenta y la forma de fragmentación se da en función de la calidad del enlace. Enviar Strings con el protocolo mencionado es una forma de agregar una capa de confiabilidad al protocolo, ya que podemos percatarnos fácilmente si un paquete llegó truncado o no.

### 7.3.5.2 Representación de las mediciones

En el paquete `glove` contamos con todas las clases que son utilizadas para representar las mediciones recibidas. Como se muestra en la figura 33 la arquitectura es parecida a la arquitectura del software del guante. Un `GloveMeasurement` se compone de instancias de `Finger` que a su vez se componen de instancias de `Gyro` y `Acceleration`. Los métodos `toJson()` son utilizados para serializar los datos al momento de generar el archivo con el formato requerido para subirlo a Edge Impulse. Los métodos `fromJson()` son utilizados para desserializar las mediciones al momento de recibirlas del guante a través del Bluetooth.

Figura 33: Diagrama de clases del `GloveMeasurement`

### 7.3.6. Navegación

Como mostramos en la figura 21 navegación se efectúa o bien desde la pantalla principal o bien utilizando el menú de navegación lateral. El menú de navegación lateral es un widget ubicado en el paquete `navigation`, denominado `navigation_drawer`. En Flutter la navegación es muy simple. Cada vez que queremos abrir una nueva página, se hace uso del Navigator, que es un widget propio de Flutter para gestionar la navegación a través de la aplicación. Por ejemplo el primer elemento del menú es un `ListTile` con el siguiente código:

```
1 ListTile(  
2   leading: Icon(Icons.bluetooth),  
3   title: const Text("Dispositivos"),  
4   onTap: () {  
5     Navigator.of(context).push(MaterialPageRoute(  
6       builder: (context) => BleGloveConnectionPage(),  
7       maintainState: false));  
8   },  
9 ),
```

### 7.3.7. Interpretaciones

Toda la lógica para las interpretaciones del lado de la aplicación está en la pantalla de interpretación `InterpretationPage` en el archivo `interpretation_page.dart`. La pantalla de interpretación se suscribe a los eventos recibidos a través de las características de interpretación obtenidas mediante el `BluetoothBackend`.

La inferencia de los gestos se realiza en el guante, por lo que a la aplicación le llega cual es el gesto que se cree que se está realizando y los porcentajes de estimación para cada uno de los gestos. Es preciso entonces serializar esta información en el guante y desserializarlo en la aplicación. El formato que utilizamos para enviar esta información es el siguiente:

```
1 [prediction]{category1:statistic}{category2:statistic2}...{categoryN:}
```

```
    statisticN}
```

donde N es la cantidad de categorías (o gestos) para las cuales la red neuronal está entrenada.

Una vez procesada esta información, la página de interpretación se ocupa de mostrarla acorde a lo que vimos en la figura 29.

## 7.4. Dependencias

Nuestra aplicación utiliza varias librerías externas que en Flutter son muy sencillas de agregar. En el archivo `pubspec.yaml` encontramos todas las librerías utilizadas. Basta con ejecutar en la terminal `flutter pub get` para traernos todas estas librerías. Las librerías utilizadas son:

- `assets_audio_player`: para la reproducción de audio durante las interpretaciones
- `flutter_blue`: para la comunicación bluetooth
- `percent_indicator`: para mostrar el progreso de una recolección de datos
- `simple_timer`: utilizado en el timer previo al inicio de una recolección de datos
- `animated_text_kit`
- `provider`: librería que nos proporciona utilidades para mantener el estado de la aplicación, como el `ChangeNotifierProvider` del que hablamos anteriormente.
- `synchronized`: brinda utilidades para garantizar exclusión mútua en operaciones concurrentes
- `syncfusion_flutter_charts`: librería utilizada para graficar las mediciones utilizadas
- `intl`
- `material_design_icons_flutter`: proporciona algunos íconos de material.io faltantes

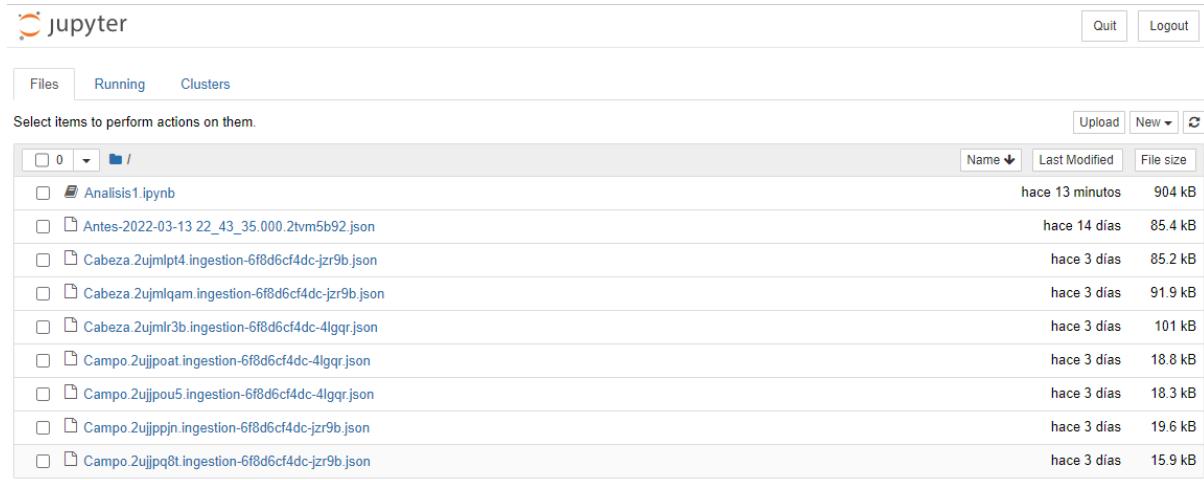
## 8. Visualizador de datos en Jupyter

Visualizar los datos reclectados es fundamental en cualquier proyecto de inteligencia artificial para poder asegurarnos que los datos proporcionados son válidos y no tienen problemas que podrían llegar a afectar negativamente el funcionamiento del algoritmo de machine learning. Edge Impulse proporciona algunas herramientas para visualizar los datos pero no son del todo satisfactorias dado que agrupa todas las mediciones en un único gráfico, no permitiendo plotear por sensor, por lo que la interpretación de esos datos se vuelve muy complicada. También habíamos implementado en la aplicación una feature para graficar las mediciones de una recolección de datos, pudiendo seleccionar dedo y sensor, pero dadas las limitaciones físicas de un dispositivo celular, se vuelve muy incómodo de usar. Se trata de una feature útil para hacer un primer análisis tentativo, pero no así algo más exhaustivo. Por eso es que desarrollamos una utilidad en Jupyter para visualizar los datos.

Jupyter Notebook es un entorno de trabajo interactivo que permite desarrollar código en Python de manera dinámica, a la vez que integrar en un mismo documento tanto bloques de código como texto, gráficas o imágenes. Es un SaaS utilizado ampliamente en análisis numérico, estadística y machine learning, entre otros campos de la informática y las matemáticas.

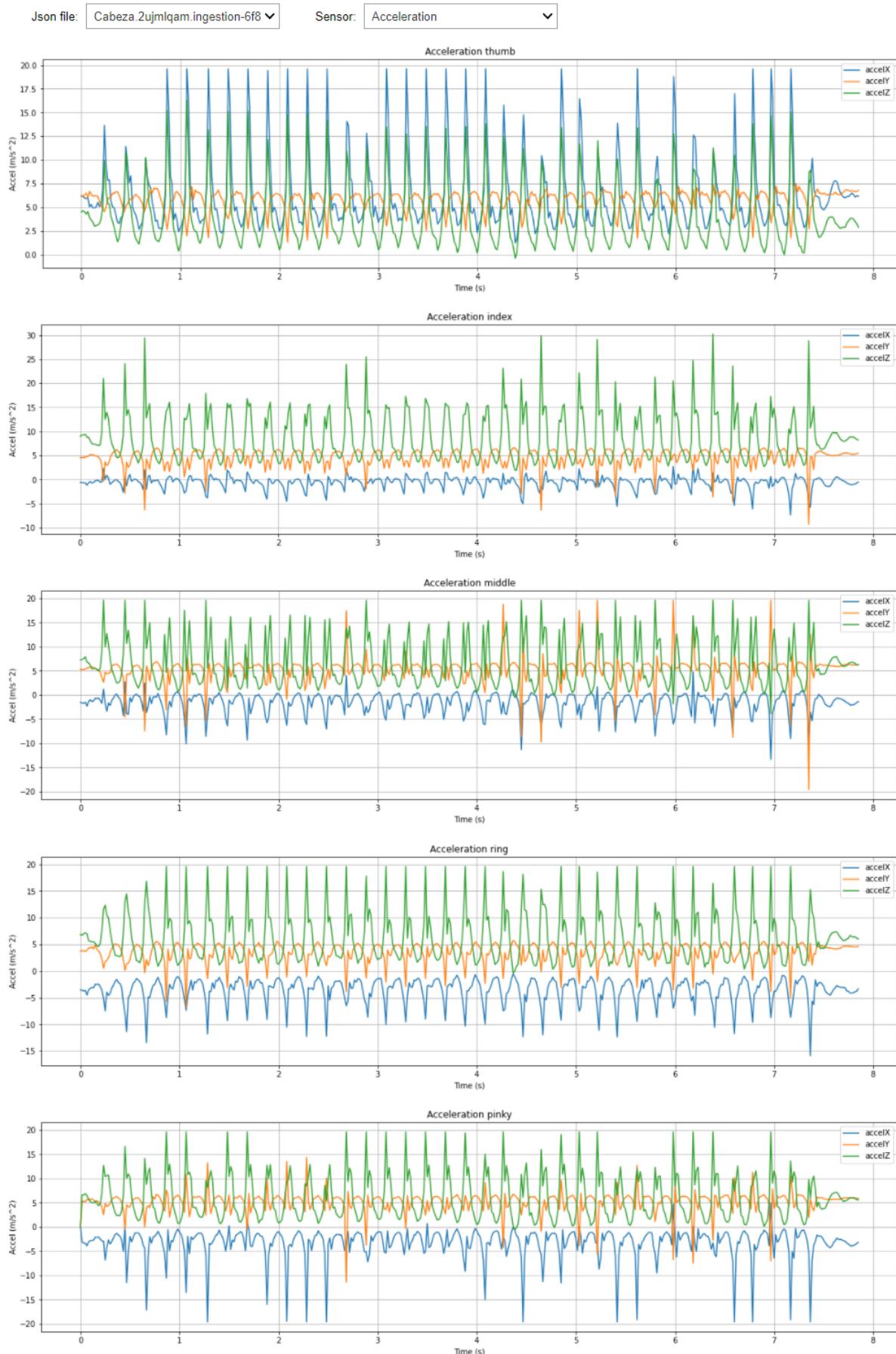
El repositorio con el código del jupyter notebook se halla en el anexo D.

Una vez instalado el repositorio, ejecutar `jupyter notebook` y abrir el entorno web indicado en la terminal (generalmente `http://localhost:8888`), tras lo cual veremos un panel como el de la figura 50. En este menú tenemos que importar los archivos que pretendemos visualizar.



**Figura 34:** Pantalla de entrada del notebook de Jupyter

Para visualizar los gráficos, abrir Analisis1.ipynb y ejecutar todas las celdas. La última celda contiene el visualizador de mediciones, que cuenta con dos dropdowns que nos permitirán seleccionar el archivo y el sensor que queremos visualizar. Cuando seleccionamos alguno de los dos parámetros, automáticamente se inicia un proceso de renderizado que nos muestra gráficamente los valores del sensor seleccionado en cada uno de los dedos para el archivo seleccionado.



**Figura 35:** Visualización de datos en Jupyter

## 9. Machine Learning embebido

El concepto de la inteligencia artificial es hoy en día un concepto muy amplio, dado que existen múltiples familias de algoritmos de machine learning con aplicaciones prácticas distintas. Los algoritmos de machine learning se pueden clasificar en algoritmos supervisados y no supervisados; en los algoritmos no supervisados como por ejemplo K-means<sup>14</sup>, el algoritmo solo dispone de los datos de entrada y busca destilar información de esos datos por si solo. En cambio, para algoritmos supervisados, a ellos se los entrena con un conjunto de datos etiquetados. El objetivo es enseñarle al algoritmo a reconocer patrones en datos que nunca antes vió a partir de los datos del set de entrenamiento que usamos.

Es justamente eso último lo que vamos a buscar en nuestro proyecto; necesitamos reconocer gestos en tiempo real a partir de las señales de los sensores del guante, podríamos entonces entrenar al algoritmo de inteligencia artificial para que sea capaz de reconocer patrones en esas señales y para esto podemos proveerle de datos etiquetados que le den una noción de "a qué se parecen" las señales para un determinado gesto.

Las redes neuronales son la tecnología que más se ajusta a las necesidades de nuestro proyecto. Se han popularizado ampliamente en los últimos años dado su grado de eficiencia para catalogar datos y son usadas extensivamente en el campo del reconocimiento de imágenes, del procesamiento de señales, y demás. No obstante, el uso de estos algoritmos generalmente requieren de una cantidad de recursos computacionales considerables, mayores a los que proporciona un sistema embebido como el Esp32. Afortunadamente en los últimos años han habido progresos tecnológicos en este sentido que nos han permitido llevar adelante nuestro proyecto.

---

<sup>14</sup>K-means es un algoritmo para agrupar datos en clusters

## 9.1. Tensorflow Lite

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos[14].

TensorFlow Lite, como se infiere de su nombre, es una versión liviana de TensorFlow y fue pensado específicamente para que pueda ser ejecutado en los microcontroladores de sistemas embebidos, minimizando el uso de recursos.

## 9.2. Redes Neuronales

La red neuronal surge inspirada por los estudios del funcionamiento de las neuronas biológicas. La unidad fundamental en una red neuronal se denomina perceptrón que es básicamente un nodo en la red. El modelo del perceptrón multiplica cada valor de entrada individual (input) por un peso que representa su importancia para la clasificación. Después todos los valores son sumados con un término. Finalmente, se toma esta suma de valores ponderados y se aplica una función de activación. La idea principal de la función de activación es eliminar la linealidad de la red. Sin una función de activación la red sería básicamente una concatenación de regresiones lineales. La elección de la función de activación es totalmente libre, puede utilizarse la función tangente o la función sigmoide. Pero empíricamente se ha demostrado que funciones mas simples logran mejores resultados.

En cada neurona se ejecuta una función sobre los datos de entrada antes propagarse la información a través del resto de unidades neuronales con el objetivo de activar, aumentar, disminuir, o eliminar el valor de esa característica en la red.

La función de activación que se utilizó fue la de ReLu (Rectified Linear Unit) que toma la parte positiva de cada valor.

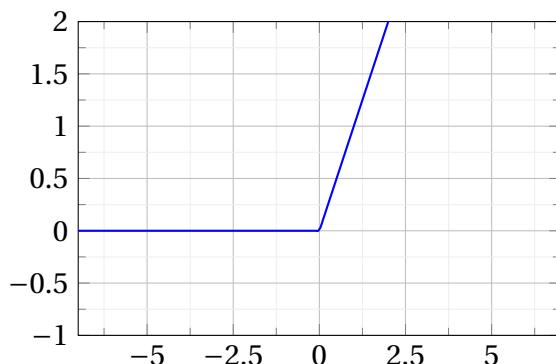


Figura 36: Función de activación Reactivación

Es una de las funciones de activación más utilizadas en redes neuronales profundas porque además empíricamente resultó tener menos errores que funciones logarítmicas. Lo que hace es descartar los resultados negativos, solo deja que se propaguen a través de las capas ocultas los resultados positivos.

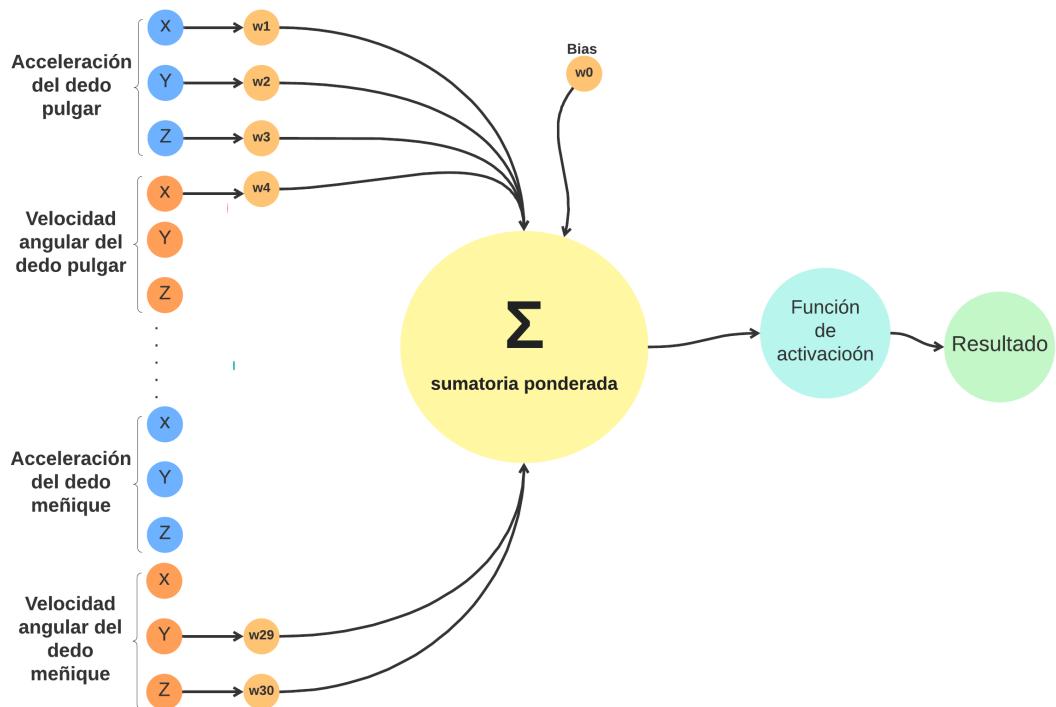
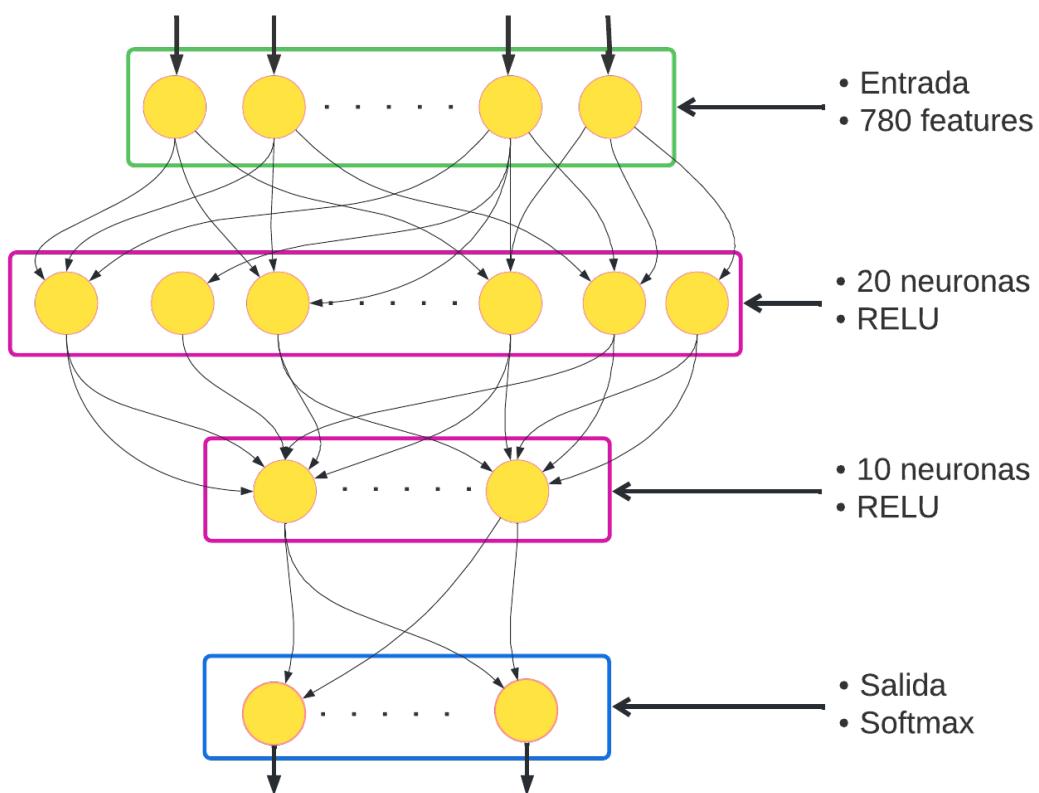


Figura 37: Neurona

A los conjuntos de neuronas que reciben los datos de entrada se lo llama capa de entrada.

El conjunto de neuronas que proporcionan el resultado de la clasificación se los nombra como capa de salida. Las redes neuronales mas sencillas están compuestas por tres capas: la capa de datos de entrada, una capa oculta y la capa de resultados. A la capa intermedia se le otorga el nombre de capa oculta porque ni los datos de entrada ni los resultados obtenidos son visibles. Los nodos de la capa oculta están conectados con todos los nodos de la capa de entrada. Podemos tener más de una capa oculta que conecte los resultados de la capa de entrada con el input de los neuronas de las siguientes capas. Cuando esto sucede la red toma el nombre de red neuronal densa.



**Figura 38:** Red neuronal utilizada

La red neuronal que se utilizó para este trabajo esta compuesta por la capa de entrada, una primer capa oculta de 20 nodos y una segunda capa oculta de 10 nodos antes de la capa de resultado.

```
1 import tensorflow as tf
```

```

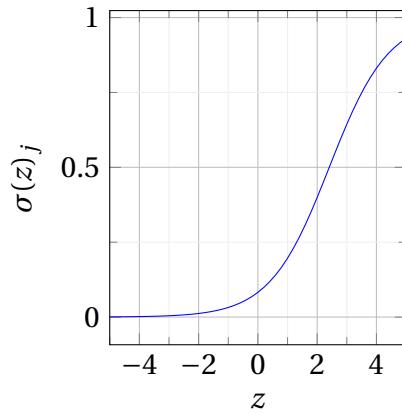
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense, InputLayer, Dropout, Conv1D,
4   Conv2D, Flatten, Reshape, MaxPooling1D, MaxPooling2D,
5   BatchNormalization, TimeDistributed
6
7 from tensorflow.keras.optimizers import Adam
8
9 model = Sequential()
10
11 model.add(Dense(20, activation='relu',
12   activity_regularizer=tf.keras.regularizers.l1(0.00001)))
13
14 model.add(Dense(10, activation='relu',
15   activity_regularizer=tf.keras.regularizers.l1(0.00001)))
16
17 model.add(Dense(classes, activation='softmax', name='y_pred'))

```

Para la última capa, el valor resultante debe estar dentro de un intervalo correspondiente con el set de palabras definidos, porque la última capa define el resultado, la probabilidad de que el gesto corresponda con cada palabra del set de resultados. La función de activación para esta capa es la función softmax, o función exponencial normalizada, que es una generalización de la función logística. Se emplea para comprimir un vector K-dimensional, de valores reales arbitrarios en un vector K-dimensional, de valores reales en el rango [0, 1], lo que nos permite filtrar un conjunto de valores que se encuentren por debajo de un valor máximo establecido.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

**Figura 39:** Función de activación Softmax



**Figura 40:** Gráfico de la función de activación Softmax

### 9.3. Evaluación del modelo

Para evaluar la performance del modelo utilizamos la matriz de confusión. El eje Y corresponde a los valores reales y el eje X a los obtenidos como solución luego de utilizar el modelo como clasificador. La idea es que la matriz empieza en cero y va sumando uno a cada casillero. La matriz expresa la confusión del modelo porque en cada celda se obtiene el numero de veces que se interpretó una etiqueta  $x_i$  cuando el valor real era  $y_i$ . Si  $x_i = y_i$ , entonces el modelo acerto en la clasificación, es decir que la mejor performance se corresponde con la matriz identidad, donde los numeros fuera de la diagonal son cero, es decir la confusión es nula. En general una matriz de identidad no se logra porque siempre existe cierto margen de error. Como la matriz en este caso toma valores porcentuales el objetivo es que los números de la diagonal sean mayores, lo que indica que las clasificaciones fueron correctas. Si un número fuera de la diagonal es muy alto podemos detectar alguna anomalía, como por ejemplo si tenemos un predictor naïve que siempre predice lo mismo vamos a tener valores positivos en una columna.

$$\text{presición} = \frac{\text{aciertos\_positivos}}{\text{aciertos\_positivos} + \text{falsos\_positivos}}$$

**Figura 41:** Cálculo de la precisión: qué tan acertada resulta ser la predicción.

|           | palabra A | palabra B | palabra C | palabra D |
|-----------|-----------|-----------|-----------|-----------|
| palabra A | 73.3      | 5.2       | 20.9      | 0.6       |
| palabra B | 2.0       | 91.0      | 7.0       | 0         |
| palabra C | 3.5       | 31.0      | 64.3      | 1.2       |
| palabra D | 0.77      | 0.82      | 0.71      | 96.4      |

**Cuadro 4:** Matriz de confusión

El color verde representa los aciertos positivos. El color gris representa los aciertos negativos. Si nos concentramos por ejemplo en la clasificación de la palabra B el color lila representa los falsos negativos, es decir las veces en las que se trataba de la palabra B pero se la clasificó como otra palabra. El color rojo representa los falsos positivos, es decir las veces en las que se lo clasificó como la palabra B siendo que el gesto correspondía a otra palabra.

Si la tasa es cercana a uno quiere decir que hay pocos falsos positivos entre las predicciones positivas reales. Si la tasa es cercana a cero quiere decir que muchos de los casos que salieron positivos son falsos.

Los problemas más comunes al entrenar un modelo son:

- **Subajuste** (underfitting): cuando el modelo funciona mal porque falla en entender la tendencia de los valores del set. Generalmente puede ser porque falta alimentar el set con mayor cantidad de datos.
- **Sobreajuste** (overfitting): cuando funciona muy bien para el set de entrenamiento pero funciona mal para casos no vistos anteriormente. En general en estos casos es porque falta variar mas los datos del set.

## 9.4. Alimentar el set de datos

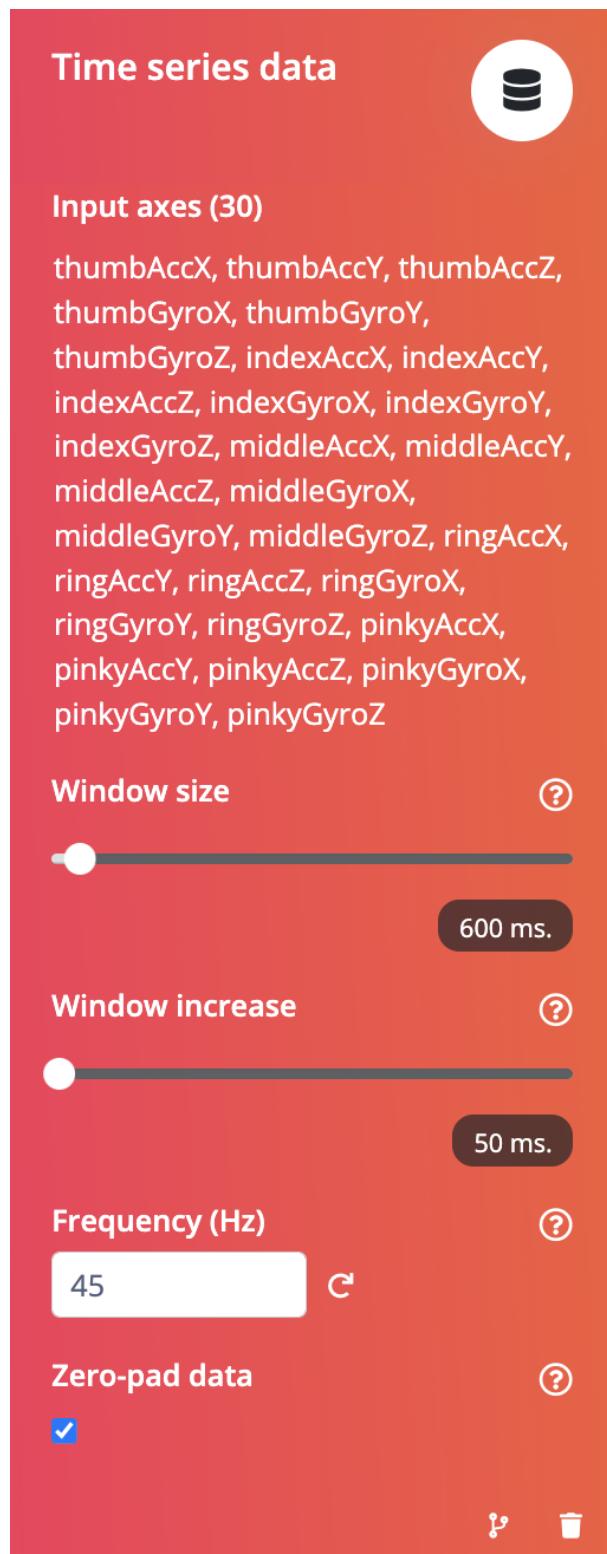
Uno de los puntos fundamentales de los algoritmos de machine learning es disponer de amplios conjuntos de datos para poder predecir o clasificar adecuadamente. En este proyecto nosotros nos encargamos de hacer la recolección de datos cargando los gestos a través de la aplicación. A diferencia de otros proyectos de machine learning donde los datos de entrada se componen de texto o fotos y se puede encontrar fácilmente en internet una amplia data para entrenar, como por ejemplo en artículos, twits, fotos de redes sociales, etc, nosotros no disponíamos de esta ventaja y fue necesario que carguemos todos los gestos para cada palabra utilizando el guante. Una vez realizado el gesto repetidas veces los enviábamos a Edge Impulse (el pipeline de machine learning) utilizando las funciones de la aplicación. Por cada gesto cargamos alrededor de 2 minutos de datos.

Uno de los principales desafíos que tuvimos al querer construir un set de datos fiable fue diferenciar cuando comienza y cuando termina un gesto. Es importante diferenciar el gesto que tiene un significado en lengua de señas argentinas de los movimientos que se realizan al comenzar el gesto o al finalizarlo, ya sea tener la mano en reposo o volverla a la posición inicial luego de realizar el gesto. Es muy fácil generar ruido en las interpretaciones si se toman estos movimientos para entrenar la red. Por ejemplo al querer realizar el gesto de "hospital" que se realiza haciendo una cruz sobre la frente, si la red neuronal toma en cuenta la aceleración y el giro de los dedos al subir la mano hacia la cabeza y al bajarla una vez terminado el gesto, cada vez que se suba la mano el algoritmo interpretará que se trata de hospital, aún cuando sea otro gesto que se realiza subiendo la mano. Esto genera ruido e interpretaciones erróneas. Para solucionarlo lo que hicimos fue recortar el flujo de datos, eliminando las mediciones iniciales y finales de un gesto utilizando la funcionalidad de Edge Impulse de recortar un set de datos de entrada como se muestra en la imagen.



**Figura 42:** Hospital

Una estrategia que implementamos que fue de muy buena ayuda fue agregar al set de gestos posibles un gesto de "reposo" que no iba a tener una traducción en la aplicación mobile, sino que se correspondería con el silencio. Grabamos los movimientos de la mano quieta en distintas posiciones para entrenarlo con varias posibilidades.



Como se puede ver en la imagen tenemos 30 valores de entrada para la red que se corresponde con la aceleración y la velocidad angular en cada eje para los 5 dedos. Edge Impulse permite configurar el tamaño inicial de la ventana, que nosotros definimos en 600 ms por ser el minimo slot de tiempo que tarda un gesto. También permite definir un tamaño para la ventana de crecimiento que nosotros definimos en 50 milisegundos, cerca del 0.1 del tamaño de la ventana. Tambien definimos la frecuencia en 45 Hz porque era el promedio de frecuencia de muestreo con la que recibiamos los datos medidos por los censores. Con las pilas totalmente cargadas la frecuencia resultaba ser algunos hz mayor que con las pilas descargadas. Por ultimo rellenar con ceros a la izquierda para completarlos marcos de tiempo sirvió para tener mejores resultados. A pesar de nuestros esfuerzos por llegar a una frecuencia alta, con una frecuencia de 45hz resultó totalmente suficiente para colecciónar datos de gestos precisos y con fluidos para entrenar la red.

**Figura 43:** Configuración de los datos de entrada

## 9.5. Bloque de procesamiento inicial

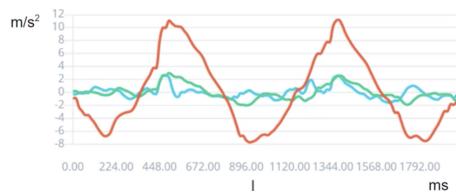
A medida que el modelo de aprendizaje automático crece en tamaño y complejidad, también lo hacen los requisitos computacionales. Se necesita mucha potencia de procesamiento para aprender automáticamente qué características son útiles para el clasificador y luego usar ese extracto de características en tiempo real. Se necesita almacenar, al menos, la cantidad de datos de entrada en la memoria y poder realizar operaciones matemáticas en cada uno de esos valores. Las secciones de extracción de características en un modelo de este tipo a menudo pueden tener varias capas de profundidad, lo que multiplica la complejidad computacional requerida para realizar la inferencia. Además, a medida que crecen los modelos de aprendizaje, generalmente se requieren muchos más datos para entrenar. Pero no tenemos un suministro interminable de datos de entrenamiento y una cantidad infinita de tiempo para entrenar. Si bien estos modelos de aprendizaje profundo pueden ser excelentes para grandes granjas de servidores, haciendo cosas como el procesamiento de lenguaje natural, no tenemos ese lujo en los sistemas integrados. A pesar de que podemos usar, y lo haremos, redes neuronales profundas en esta clase, se beneficiará enormemente al elegir las características apropiadas de los datos sin procesar. En última instancia, desea mantener su modelo de aprendizaje automático lo más pequeño y rápido posible. Como la mayoría de los algoritmos de aprendizaje automático requieren mucha memoria y potencia de procesamiento, y estos tienen un suministro limitado en la mayoría de los sistemas integrados. Una forma en que se puede hacer esto es eligiendo las funciones para el modelo a partir de un bloque de preprocesamiento.

Es posible combinar las muestras de diversas maneras para generar características únicas que ayuden a describir lo que sucede en el sistema. Por ejemplo, calculando la raíz cuadrada media de 125 muestras en cada eje. Este es un cálculo sencillo que da un solo número por eje, y nos da algo así como un promedio o media, para todos estos números en cada eje. De esta forma se logra tener tres dimensiones como entrada para el modelo, y tienen en cuenta dos

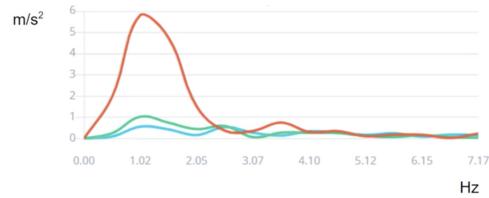
segundos de datos. Una cosa que se podría querer hacer es filtrar la media de cada conjunto de datos antes de calcular la raíz cuadrática media, de modo que la gravedad se elimine de la ecuación.

Es muy común el uso de una función espectral cuando se trata de datos de movimiento. La función espectral extrae las características de frecuencia y potencia de una señal. También se pueden aplicar filtros de paso bajo y paso alto para filtrar frecuencias no deseadas. Es recomendado para analizar patrones en una señal de movimientos o vibraciones de un acelerómetro.

Una técnica común cuando se observan datos de vibración o movimiento es tomar la transformada de Fourier de esos datos para obtener información sobre ellos en el dominio de la frecuencia. Es una forma de dividir una señal en sus diversos componentes de frecuencia. La transformada rápida de Fourier está optimizada para datos muestreados discretos.



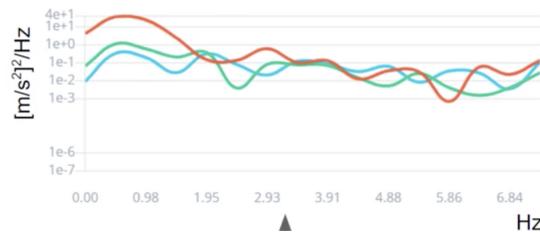
**Figura 44:** Aceleración en función del tiempo



**Figura 45:** Aceleración en función de la frecuencia

Un movimiento de izquierda a derecha se grafica en los diagramas anteriores. Si observa detenidamente el primer gráfico, se puede ver que se tarda aproximadamente un segundo en completar un ciclo. Eso significa que tiene una frecuencia de 1 hz. Debido a que esta es una característica destacada, se destacará en el dominio de la frecuencia. En el segundo gráfico se ve un pico en 1hz. De manera similar, si se moviera el dispositivo de un lado a otro más rápido, por ejemplo a 3hz, este pico aparecería a los 3hz en el gráfico de aceleración en función de la frecuencia. Se puede ir un paso más allá y calcular la función de densidad espectral. La transformada de Fourier puede ser algo engañosa a veces, ya que la amplitud en cada intervalo puede variar según el ancho de frecuencia de ese intervalo. La densidad

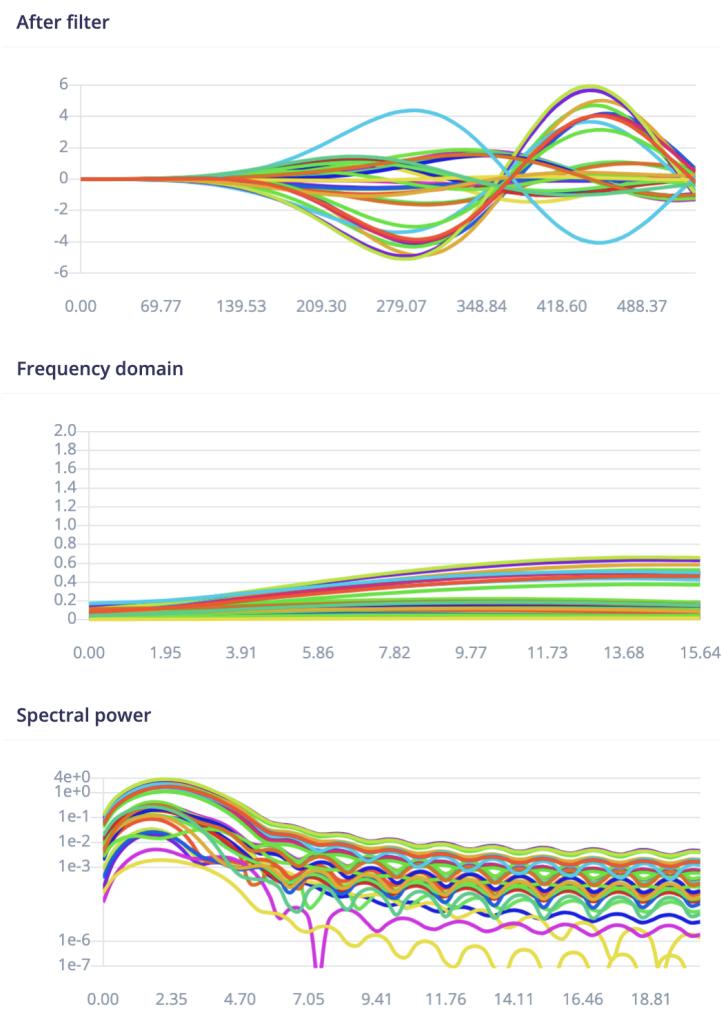
espectral ayuda al normalizar la amplitud al ancho de cada intervalo.



**Figura 46:** Densidad espectral

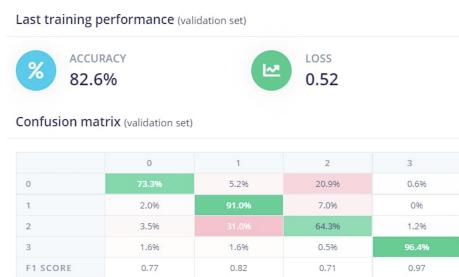
Edge Impulse permite agregar un bloque de densidad espectral y extraer algunas características importantes de él. El primero, es identificar la ubicación de amplitud y frecuencia de los picos más altos de cada eje. La segunda, es que suman todos los valores entre ciertos rangos. Estas dos acciones ayudan a describir la forma de la densidad de potencia sin necesidad de usar todos los valores resultantes de la función, lo que requeriría una entrada de mayor dimensión al modelo, y es lo que se quiere evitar.

Nosotros nos guiamos por estas recomendaciones y en las primeras pruebas para el entrenamiento utilizamos un bloque de densidad espectral. Para cada set de datos se pueden observar los gráficos resultantes del bloque de procesamiento. Por ejemplo para el gesto correspondiente a la palabra "Hola", la mano se mueve de derecha izquierda de forma repetitiva. El resultado del bloque de pre-procesamiento se puede observar en el siguiente gráfico.

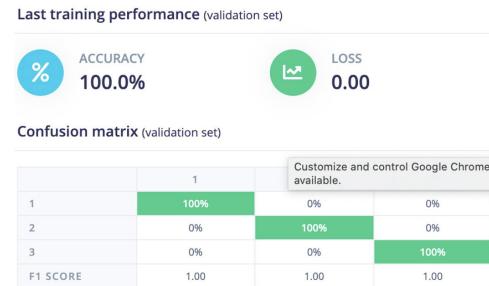


**Figura 47:** Gráficos de la densidad espectral del gesto de "hola"

Pero el resultado no fue lo que esperábamos. Con la función de densidad espectral el desempeño de la red no era muy acertado. La primera iteración consistió en el entrenamiento de un set de datos muy pequeño compuesto por los números del 1 al 3.



**Figura 48:** Matriz de confusión utilizando la función de densidad espectral



**Figura 49:** Matriz de confusión utilizando datos crudos

A pesar de que creíamos que el pre-procesamiento de densidad espectral iba a servirnos para reducir los recursos utilizados llegando a buenos resultados al comparar las matrices de confusión se puede observar que la precisión disminuye considerablemente en comparación a la utilización de datos crudos. Vemos en la primera matriz que el gesto correspondiente al valor 1 se interpretaba como dos en muchos casos y el gesto correspondiente a cero también se malinterpretaba como dos, resultando en un error del 0.5 en comparación a un error nulo cuando se usaban los datos en crudo. Aunque la segunda matriz de confusión marcaba una precisión del 100%, en la práctica no resultaba tan precisa pero sí que clasificaba de una forma mucho más acertada.

## 9.6. Entrenamiento

Una vez que obtuvimos un conjunto de datos se lo dividió en un conjunto de entrenamiento (60 %) y un conjunto de validación(20%). El resto de los datos conforman el set de test y solo se utilizan al final del proceso.

En el entrenamiento se trata de lograr que los pesos de cada nodo sean los ideales para predecir las etiquetadas correspondientes del set de entrenamiento. Para eso se multiplican los valores de entrada por los pesos de cada nodo. Luego se utiliza una función de error para poder identificar cual es la diferencia entre el resultado y el valor esperado.

Cuando comparamos el valor resultante con el que debería ser, el valor de la diferencia

se va a utilizar para “corregir” el modelo. Modificando los valores de la red en la dirección óptima. El proceso que realiza este ajuste se conoce como algoritmo de retro-propagación o backpropagation porque va ajustando los valores de cada capa empezando por la ultima capa hasta llegar a la capa de entrada. Se va a utilizar una función de pérdida para comparar cuantitativamente la diferencia entre el valor esperado y el resultante. La función de perdida siempre es una función compuesta por la función de activación.

$$L(f(x, W), y)$$

El algoritmo de retro-propagación va a usar los valores encontrados en el conjunto de entrenamiento  $f(x, W)$  y las etiquetas asociadas ( $y$ ) para actualizar los parámetros en el modelo de machine learning. ¿Pero cómo los ajusta? para tomar la mejor dirección de ajuste se basa en el gradiente de la función de pérdida. Si la función de activación  $f$  es una función que utiliza un vector de  $k$  valores y  $W$  es un vector con  $m$  parámetros, las derivadas para una posición del vector  $W$  se calcularían de la siguiente manera:

$$\frac{\partial L}{\partial W_j} = \sum_{i=1}^k \frac{\partial L}{\partial f_i} \frac{\partial f_i}{\partial W_j}$$

Al calcular las derivadas parciales para cada parámetro de  $W$  estamos calculando la famosa matriz jacobina para una capa de la red con  $m$  valores de entrada y  $k$  resultados.

$$J_f(W)_{ij} = \frac{\partial f_i}{\partial W_j}$$

En nuestro caso utilizamos la entropía cruzada como función de pérdida. Esta función de pérdida se suele utilizar para tareas de clasificación multi-clase. La función de entropía cruzada es una medida de la diferencia entre dos distribuciones de probabilidad para dos conjuntos de variables aleatorias o eventos. La fórmula consiste en calcular la esperanza relativa a la distribución de probabilidad real del logaritmo de la distribución de la probabilidad

obtenida. Se suele utilizar  $p$  para representar la distribución de probabilidades reales y  $q$  para representar la distribución de probabilidad de las variables aleatorias resultante.  $H(p, q) = -\text{E}_p[\log q]$ . En una red neuronal podemos considerar que la salida de un solo nodo define un valor de probabilidad, la función de entropía cruzada caracteriza la distancia entre la probabilidad de la salida real y la probabilidad de la salida esperada.

Para nuestro caso, el costo individual se puede expresar con la siguiente fórmula matemática.

$$\text{Error} = -\frac{1}{k} \sum_{i=1}^k y_i \log_2(f(x)_i) + (1 - y_i) \log_2(1 - f(x)_i)$$

Cuanto menor es el valor de la entropía cruzada, más cercanas están las dos distribuciones de probabilidad.

El framework de TinyML junto con Edge impulse nos permiten abstraernos de la complejidad matemática eligiendo la función de error 'categorical crossentropy' a partir de un conjunto de funciones de error utilizadas de manera frecuente.

```

1 # train the neural network
2 model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=[ 
    'accuracy'])
3 model.fit(train_dataset, epochs=30, validation_data=validation_dataset,
    verbose=2, callbacks=callbacks)

```

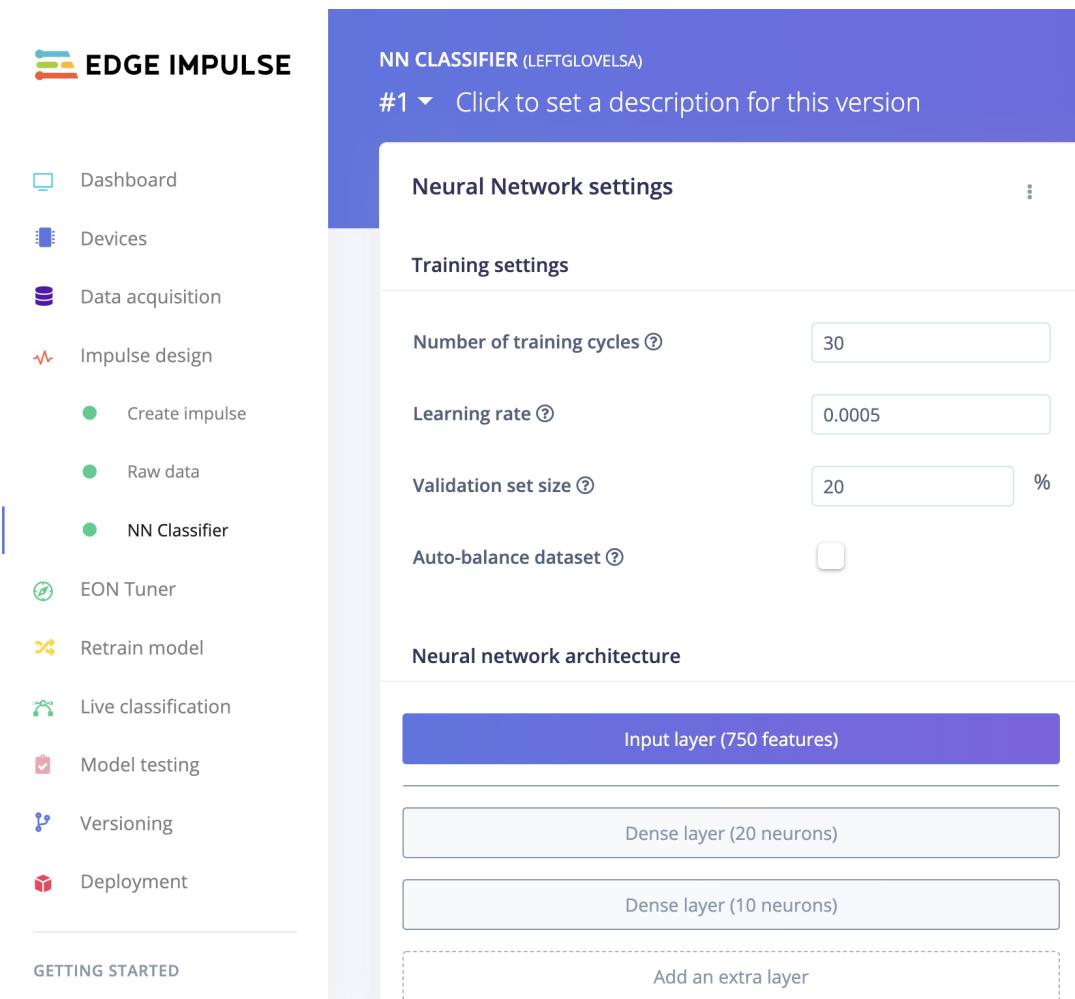
Cada vez que los datos pasan por todos los nodos de la red y se ejecuta la retro-propagacion se considera que termino una épica. El entrenamiento consiste en la repetición de varias épicas.

Luego del entrenamiento, el conjunto de datos de validación se utiliza para probar la performance del entrenamiento. Si resulta que no funciona bien se pueden modificar los pasos del entrenamiento o modificar los hiperparámetros, como es tamaño y la forma del modelo. Volvemos a entrenar con el mismo set de entrenamiento hasta que estemos felices

con la performance. Por último vamos a usar el set de test. Si nuestro algoritmo funciona bien con el set de validación pero no con el set de test entonces decimos que tiene el problema de overfitting.

Si tenemos muchos parámetros seguro necesitamos un set de validación más grande. validación cruzada: se toma al azar del conjunto de datos de entrenamiento un porcentaje como validación N veces, así se generan conjuntos aleatorios para entrenar al algoritmo varias veces alternando los datos de entrenamiento y validación.

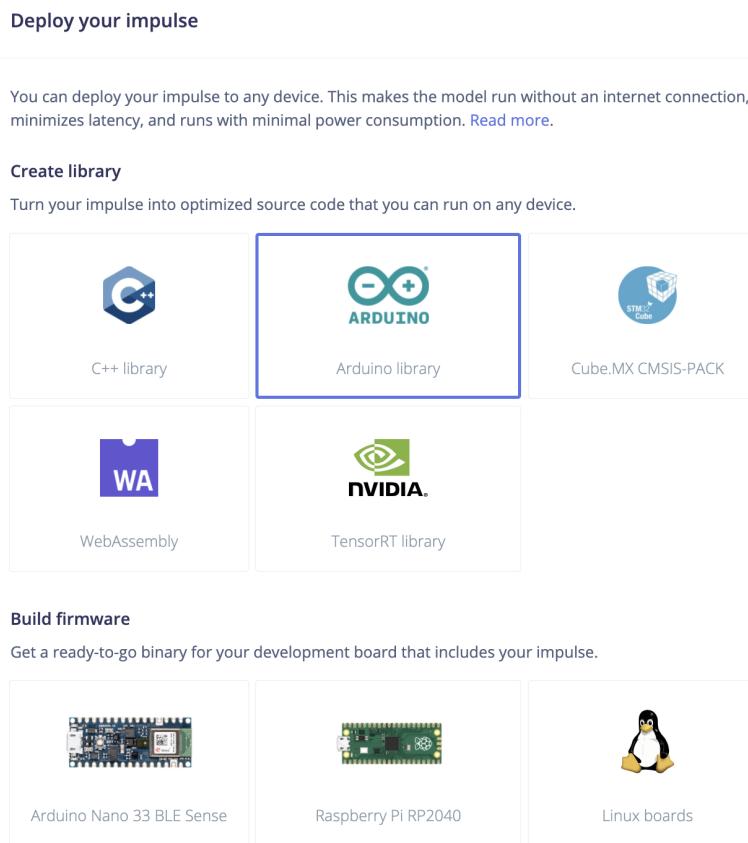
En edge impulse es muy fácil configurar los hiperparámetros de la red que estamos describiendo.



**Figura 50:** Configuración de los hiperparámetros de la red neuronal

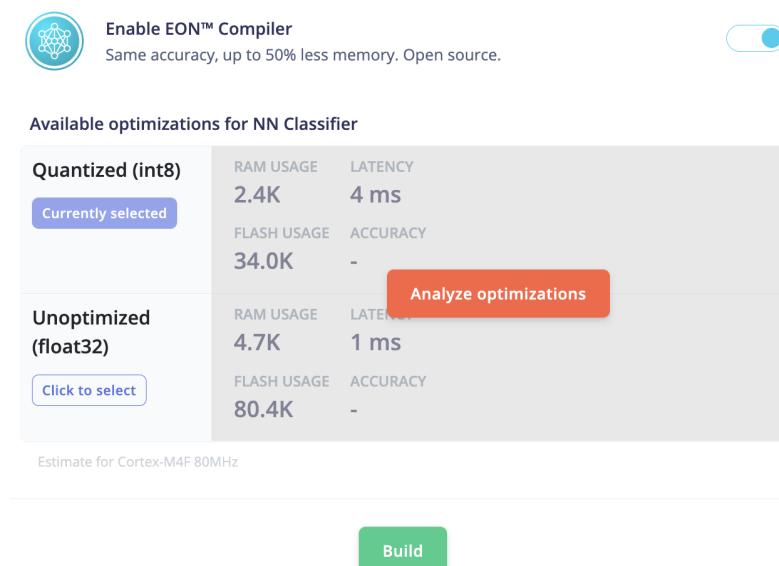
## 9.7. Depliegue continuo y aprendizaje continuo

Como mencionamos anteriormente Edge impulse es una herramienta que simplifica el ciclo de iteraciones de desarrollo de machine learning y que permite desplegar modelos de entrenamiento resultantes de cada iteración en nuestro dispositivo. La plataforma nos provee de diferentes alternativas para la creación de la librería de interpretación, algunas de ellas se generan listas para ser utilizadas en dispositivos con arquitecturas específicas, como por ejemplo Arduino nano 33 ble sens o Raspberry Pi RP2040 o hasta para arquitectura Linux por si uno desea probar los resultados en un ordenador. Como el dispositivo que nosotros utilizamos (esp32) no era una de las opciones posibles, utilizamos la opción de generar una librería Arduino en C que puede ejecutarse en diversos tableros arm y es compatible con el esp32.



**Figura 51:** Configuración de la librería

El último paso de cada ciclo de desarrollo consistía en construir la librería de arduino anteriormente seleccionada. Edge impulse simplifica el proceso, haciendo click en un botón podemos descargarnos la librería lista para ser embebida en el guante. La plataforma también dispone de un proceso de optimización que permite mejorar la performance en el dispositivo al reducir el uso de memoria flash y memoria ram pero también genera una pérdida de precisión en la ejecución. Nosotros decidimos asumir este riesgo.



**Figura 52:** Construcción de la librería

Nosotros utilizamos Platformio en conjunto con Visual Studio Code como entorno de desarrollo. Platformio es una herramienta "Open Source" de desarrollo especialmente para entornos IoT. Se basa en una aplicación de consola que se encarga de la configuración del proyecto y de manejar todos los requerimientos a nivel de librerías de forma centralizada posibilitando la migración entre plataformas. Platformio permite incorporar la librería generada por edge impulse muy fácilmente. En la raíz del proyecto se define un archivo de configuración platformio.ini donde se puede configurar el tablero de trabajo utilizado e incorporar librerías de arduino como dependencias externas tan solo definiendo la ruta específica como se puede ver a continuación:

```
1 [env:nodemcu-32s]
2 platform = espressif32
3 board = nodemcu-32s
4 framework = arduino
5 build_flags = -DCORE_DEBUG_LEVEL=0
6 lib_extra_dirs = ~/Documents/Arduino/libraries
7 lib_deps = RightGloveLSA_inferencing
8 monitor_filters = esp32_exception_decoder
9 board_build.partitions = huge_app.csv
```

## 9.8. Resultados

Luego de un eliminar el bloque de pre-procesamiento y tomar los datos crudos como puntos de entrada obtuvimos muy buenos resultados. Seguimos avanzando entrenando la red incorporando más palabras cada vez. Como nosotros fuimos las únicas dos personas que recolectamos datos para entrenar la red por lo tanto se podría decir que nuestro entrenamiento estaba sesgado a la forma y la velocidad en que nosotros hacíamos los gestos. Estamos ante un caso de sobre ajuste. El guante traduce bien pero ajustado únicamente a dos personas. Si otra persona que quiere usarlo intenta traducir con ciertas variaciones en los gestos o otra velocidad la precisión seria menor. Lo ideal hubiese sido tener un gran número de personas entrenando la red para disponer de un set de datos ampliamente variado para cada gesto. De esta manera sería mas probable que la traducción fuese acertada para un gran número de personas. En un comienzo tratamos de convocar a gente para entrenar pero no tuvimos buenos resultados.

## 10. Gestión del proyecto

Para gestionar las tareas de usuario a desarrollar se utilizo la metodología de rebanado vertical o también llamado vertical slicing en inglés, en el cual se definen las tareas lo mas

pequeñas posibles pero que atravesen toda la arquitectura de punta a punta, entregando el mínimo valor al usuario pero atravesando todas las componentes que interactúan en el proyecto. Es por eso que los features a desarrollar incorporaban tareas de desarrollo que involucraban el código del embebido y el código del guante.

El equipo realizaba reuniones semanales típicas del metodologías ágiles pero con la restricción de que estaba compuesto únicamente por dos desarrolladores. Se realizaban reuniones de planificación de las tareas del sprint, reuniones para refinar las tareas definidas. Junto con los tutores se realizaron reuniones para demostrar los avances de cada sprint y manifestar inquietudes y complicaciones. También se utilizó un tablero del tipo Kanban dentro de un proyecto de Github en el cual se definieron columnas para los estados de las tareas.

Las tareas realizadas a lo largo del proyecto pueden dividirse, además de las variadas pruebas de concepto, en tres grandes secciones. En la primera sección nos centramos en el diseño e implementación de un prototipo, en la segunda sección nos centramos en la funcionalidad para la recolección de datos para el entrenamiento. En la tercera sección nos centramos en el desarrollo de las traducciones. En esta última parte se realizaron varias iteraciones incrementales en las que se ejecutaron las siguientes tareas:

1. Realización campañas de recolección de datos específicos
2. Análisis la calidad de las mediciones a partir de gráficas
3. Identificación features y eliminar datos redundantes o erróneos
4. Creación un set de datos de entrenamiento, un set de datos de validación y un set de test
5. Creación de una red neuronal o variante de algoritmo supervisado
6. Entrenamiento de la red neuronal a partir del set de datos

7. Visualización y evaluación de los resultados obtenidos
8. Ajustar y calibración de la red
9. Despliegue embebido de la librería dentro de procesador ESP32 para clasificar las mediciones obtenidas de los sensores en palabras
10. Pruebas de la clasificación y analizar la eficiencia

se fueron incorporando cada vez mas palabras al set de traducción, para que el algoritmo de interpretación vaya creciendo de manera iterativa.

| #   | Tarea  | HP |
|-----|--|----|
| 0   | Pruebas de concepto  |    |
| 0.1 | Recopilar datos de un sensor IMU conectado al procesador ESP32   | 64 |
| 0.2 | Embeber un algoritmo de machine learning dentro del ESP32 y reconocer movimientos simples a partir de datos obtenidos por el sensor                          | 64 |
| 0.3 | Conectar cinco sensores MPU6050 al ESP32 y recopilar datos de ellos en simultáneo  | 8  |
| 0.4 | Establecer comunicación WIFI y Bluetooth low energy entre el ESP32 y un dispositivo  | 64 |
| 0.5 | Incorporar batería al sistema y estudiar su autonomía  | 24 |
| 0.6 | Configuración de entorno de aplicación android en Flutter  | 8  |
| 1   | Diseño de prototipos   |    |
| 1.1 | Diseño de prototipo en formato CAD para contención del ESP32 y la batería en dorso de la mano, tomando en consideración las conexiones cableadas pertinentes | 24 |
| 1.2 | Diseño de prototipo en formato CAD para contención de los sensores en los dedos de la mano, tomando en consideración las conexiones cableadas pertinentes    | 16 |
| 1.3 | Impresión de los prototipos y ensamblado   | 8  |
| 2   | Desarrollo de la app y programación de los embebidos   |    |
| 2.1 | Creación de pantalla para la conexión con los dispositivos   | 16 |

|     |   |      |
|-----|---|------|
| 2.2 | Creación de pantalla para configuración del guante (batería, sensores, etc)   | 24   |
| 2.3 | Creación de pantalla para selección de gestos y recopilación de datos   | 16   |
| 2.4 | Creación de pantalla para la traducción de gestos   | 16   |
| 2.5 | Creación de pantalla para la gestión de archivos de datos recopilados   | 32   |
| 2.6 | Diseño e implementación de sistema de comunicación entre el embebido y la app para la recepción de datos de sensores en la aplicación | 108  |
| 2.7 | Diseño e implementación de un sistema para subir los datos a un servidor en el que realizar el procesamiento de los datos.            | 72   |
| 2.8 | Creación de árbol de decisión en la aplicación para determinar el gesto realizado   | 108  |
| 2.9 | Grabar audio de cada palabra y cargarlo en la aplicación  | 8    |
| 3   | Recolección de datos, entrenamiento de la red neuronal y análisis de datos  |      |
| 3.1 | Inferencia de palabras del 1 al 3   | 200  |
| 3.2 | Inferencia de palabras del 1 al 5 y días de la semana   | 120  |
| 3.3 | Inferencia de palabras del 1 al 5 y días de la semana y tiempos   | 80   |
| 3.4 | Inferencia de palabras del 1 al 5 y días de la semana, tiempos y sujetos  | 80   |
| 3.5 | Inferencia de palabras del 1 al 5 y días de la semana, tiempos y sujetos y adjetivos  | 80   |
| 3.6 | Inferencia de palabras del 1 al 5 y días de la semana, tiempos sujetos, adjetivos y verbos  | 80   |
|     | TOTAL   | 1048 |

## 11. Desafíos

### 11.1. Comunicación con los sensores

#### 11.1.1. Problemática

La comunicación entre el Esp32 y los sensores MPU6050 se efectúa mediante el bus I2C, el cual nos permite obtener mediciones con una gran frecuencia. El funcionamiento del bus i2c funciona de la siguiente manera: cada sensor escribe en una dirección del bus i2c asignada de fábrica, especificada por el fabricante; en nuestro caso el MPU6050 tiene por defecto la dirección 0x68 para transmitir datos a través del bus i2c y si se le activa el pin de ad0 el sensor escribe en la dirección 0x69 del bus. Dado que necesitamos comunicarnos con 5 idénticos sensores en simultáneo y que disponemos únicamente de dos direcciones posibles para escribir en el bus i2c, nos encontramos limitados

#### 11.1.2. Solución

Leemos de a un sensor por vez de la siguiente manera:

1. Configuramos el sensor objetivo para que escriba en la dirección 0x68 del bus i2c, desactivando el pin AD0.
2. Configuramos los sensores restantes para que escriban en la dirección 0x69 del bus i2c, activando el pin AD0.
3. Ordenamos una operación de escritura a los sensores.

Con esta modalidad, logramos disponer de una medición válida en la posición 0x68, mientras que descartamos lo que sea que se haya escrito en la dirección 0x69, dado el hecho de que los 4 sensores restantes escribieron en simultáneo generando basura.

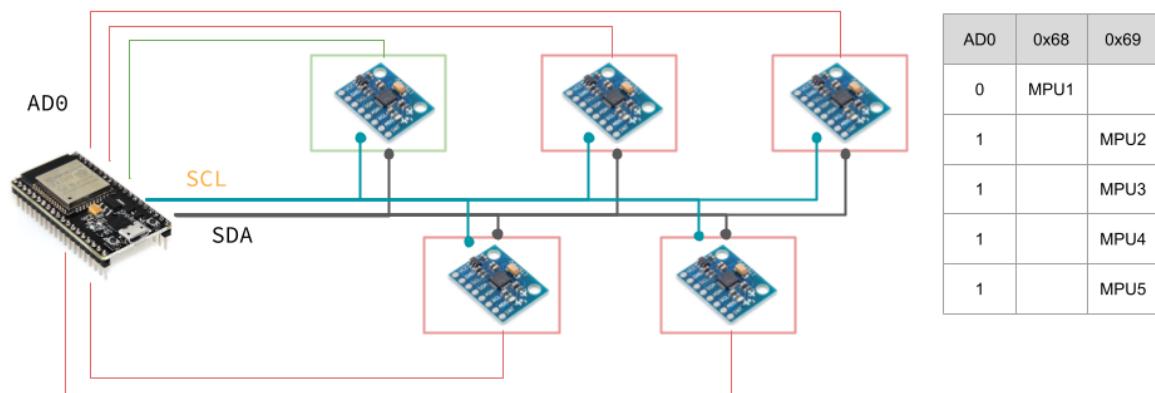


Figura 53: Configuración de los pines AD0 para la comunicación con el sensor 1

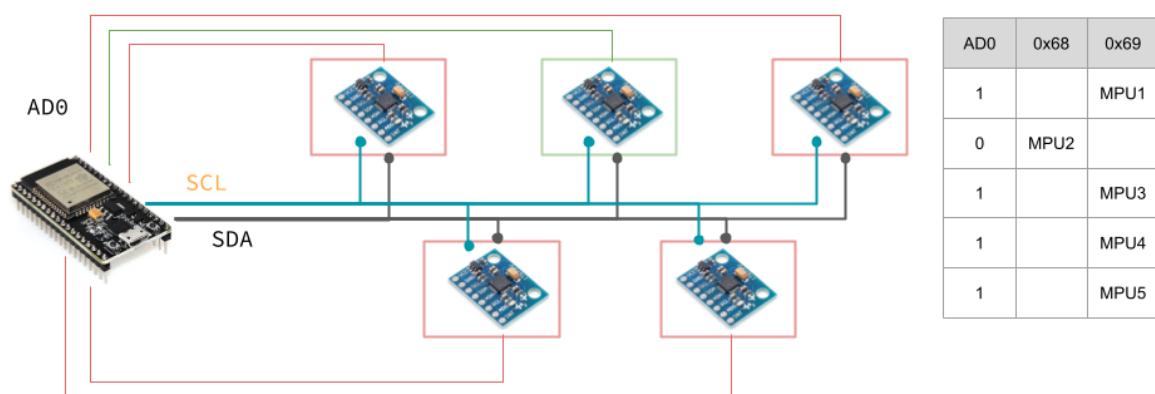


Figura 54: Configuración de los pines AD0 para la comunicación con el sensor 2

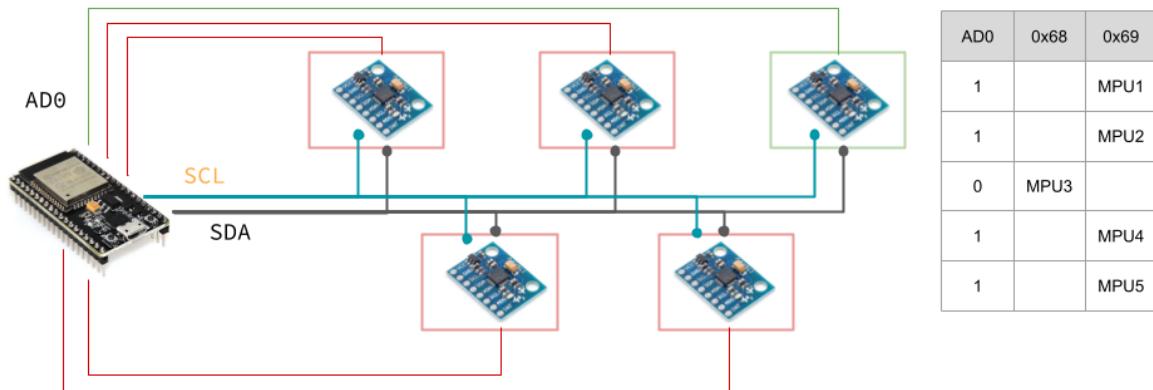


Figura 55: Configuración de los pines AD0 para la comunicación con el sensor 3

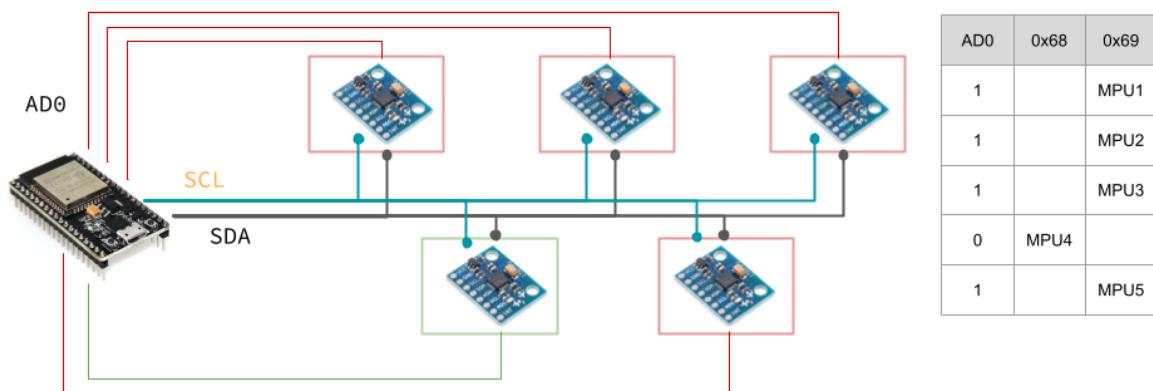
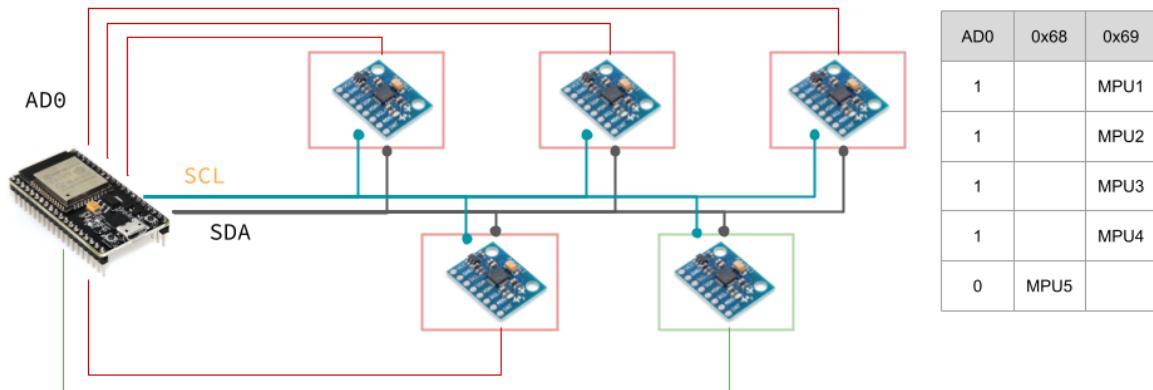


Figura 56: Configuración de los pines AD0 para la comunicación con el sensor 4



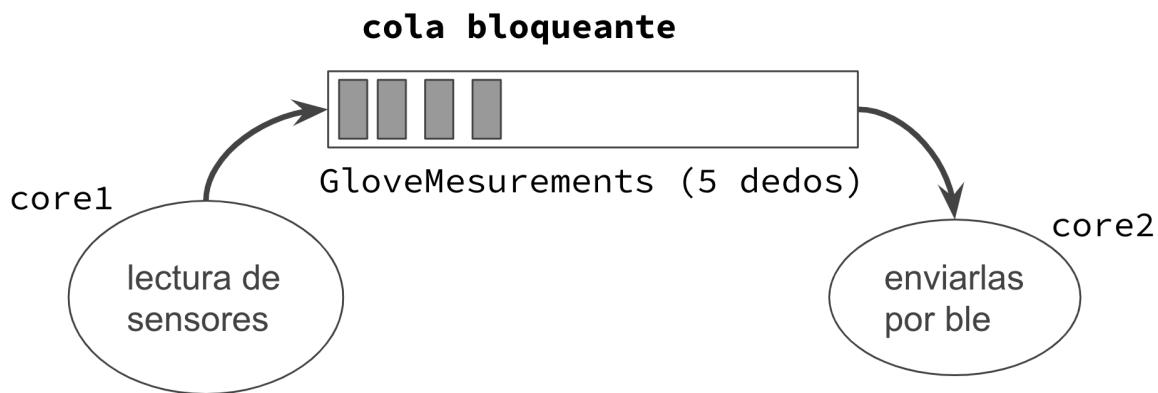
**Figura 57:** Configuración de los pines AD0 para la comunicación con el sensor 5

Además de establecer el anterior mecanismo,

## 11.2. Frecuencia

Uno de los problemas más importantes al que nos enfrentamos durante el desarrollo del trabajo estaba reaccionado con la frecuencia de muestreo. Una vez establecido el mecanismo anterior que nos permitía leer de cada sensor. Medimos la frecuencia de muestreo y resultó ser de 1Hz, lo cual era muy bajo para poder llegar a un flujo de datos que representaran un gesto continuo. En un primer diseño de la aplicación embebida decidimos aprovechar los dos cores del esp32 y destinar cada uno de ellos a una tarea específica. Un core iba a estar destinado a la lectura de los sensores mientras el otro core tenía como principal tarea el envío de datos a través de la red bluetooth. El primer proceso encolaba los datos de las mediciones de cada dedo en una cola bloqueante. El segundo proceso desencolaba uno por los elementos de la cola y al acumular cinco mediciones correspondientes una a cada dedo entonces enviaba los datos por bluetooth en un formato que la aplicación pudiera

deserializar.



**Figura 58:** Primer diseño del esquema de tareas del core del embebido

a fin de incrementar la frecuencia de obtención de mediciones, se realizaron las siguientes pruebas:

- Eliminar los logs de la aplicación
- No esperar acumular los datos de los cinco dedos en el segundo procesos y enviar cada uno de los elementos desencolados.
- Eliminar la cola bloqueante de mediciones y definir un solo ciclo principal que lea de cada sensor y luego envie los datos por bluetooth de manera secuencial.
- incrementar la frecuencia de comunicación del bus I2C a 400KHz (en lugar de los 100KHz por defecto)
- incrementar la frecuencia de toma de mediciones en los sensores MPU6050 a 400KHz.

Con la primera prueba aumento la frecuencia hasta 5Hz. Los logs consumían tiempo en cada ciclo y no son necesarios a menos que se este depurando un error, por lo implementamos una configuración que permitía deshabilitar los logs o habilitarlos. La segunda prueba

no aporto grandes diferencias en la frecuencia e incorporo un problema adicional. Al no esperar a acumular los datos de los cinco dedos podría pasar que se pierda algún paquete y no se pueda reconstruir en la aplicación las mediciones para todos los dedos. Por estas razones la segunda prueba fue descartada. La tercer prueba logró aumentar la frecuencia de 5Hz a 45Hz con lo cual interpretamos que encolar y desencolar elementos en la cola bloqueante tomaba bastante tiempo en los ciclos del dispositivo. Con las últimas dos pruebas se aprovechaba el hecho de que el bus i2c tiene una frecuencia de comunicación muy elevada, se pueden obtener mediciones con una frecuencia de 60Hz que es lo suficientemente buena para nuestro proyecto. No obstante, si bien realizamos múltiples recolecciones de datos a 60Hz, descubrimos que se incrementaban las anomalías en los datos (pudiendo corromper las interpretaciones de la red neuronal). Además, cualitativamente no mejoraba el rendimiento de las interpretaciones con respecto de la red neuronal entrenada con datos obtenidos a 45Hz, por lo que retrotrajimos este último cambio y volvimos a recolectar datos con 45Hz.

### 11.3. Memoria

Uno de los problemas a los que nos enfrentamos y que predecimos que podría llegar a ser una complicación, fue superar el tamaño de memoria flash del microcontrolador. Es lógico que a medida que la red tiene más palabras, el tamaño del código compilado con la red incorporada como una librería aumente proporcionalmente. Pudimos solucionar este problema modificando los tamaños definidos para los distintos tipos de memoria del microcontrolador. La memoria se puede clasificar en volátil y no volátil. La memoria que mantiene el código ejecutable no volátil porque se mantiene guardada incluso después de que el dispositivo se haya apagado. En cambio los valores de esas variables que se pierden luego de resetear el dispositivo se considera memoria volátil. Dentro de la memoria no volátil podemos diferenciar entre memoria flash y EEPROM. La memoria volátil está compuesta por

la memoria RAM que se divide en:

- DRAM (Data RAM): es la memoria para guardar los valores de las variables durante la ejecución. Es el tipo de memoria más común que se accede como heap.
- IRAM (Instrucción RAM): usualmente mantiene data ejecutable. Si se accede como memoria genérica está alineada con 32 bits.
- D/IRAM is RAM: puede ser usada para instrucciones o para data.

```
1 # Name,      Type, SubType, Offset,   Size, Flags
2 nvs,        data, nvs,     0x9000,    0x5000,
3 otadata,    data, ota,     0xe000,    0x2000,
4 app0,       app,  ota_0,   0x10000,   0x300000,
5 spiffs,     data, spiffs, 0x310000, 0xF0000,
```

**Listing 2:** board partition table

## 12. Conclusiones

### 12.1. Acerca del desarrollo del hardware

Hemos podido experimentar de primera mano que el desarrollo de hardware no es una tarea sencilla. Es necesario tener un mínimo de conocimientos de electrónica, como son por ejemplo las nociones de tensión, corriente, tierra, alimentación, consumo de batería, etc. Viene bien contar con la capacidad de leer los datasheets y los diagramas de los circuitos de los componentes electrónicos utilizados para saber qué tipo de componentes podemos conectar y a donde<sup>15</sup>.

La dificultad asociada al manejo del hardware se vio multiplicada en nuestro caso debido a las restricciones de tamaño que nos impusimos inicialmente, dado que pretendíamos disponer de un dispositivo lo más chico posible para incrementar la comodidad del usuario. Para esto, entendemos que si este proyecto estuviera en una etapa más avanzada en la que se estuviera evaluando la comercialización del producto, se tornaría indispensable contar en el equipo con una persona abocada pura y exclusivamente al diseño industrial, que pueda enfocarse en reducir el tamaño del dispositivo al mínimo, proveyendo al usuario una comodidad que a día de hoy no tiene con el producto. El guante actualmente se compone de una caja de plástico atada al antebrazo, con unos anillos de plástico conectados mediante cables, lo cual resulta incómodo si se pretende usar por muchas horas. No solo habría que intentar reducir aún más el tamaño del dispositivo, sino que también deberíamos evaluar el uso de otros materiales distintos al plástico, por ejemplo goma sintética o caucho o plástico flexible.

Más allá del diseño industrial, también es necesario reducir el tamaño del hardware ya que si bien dispositivos embebidos como el Esp32 son pequeños de por sí, los dispositivos de

---

<sup>15</sup>Por ejemplo, estudiando el esquema de la figura ?? de cómo se alimenta el Esp32 pudimos ver que hay un regulador de tensión conectado al pin de 5V, por lo que podíamos conectar la batería directamente ahí.

tipo "wearables" como los smartwatches, requieren de un hardware hecho a medida dadas las limitaciones de tamaño extremas. Creemos que un dispositivo como el nuestro entra en esa categoría y para comercializar un producto así sería necesario trabajar con alguna tercera parte que provea hardware con un diseño hecho a medida de las necesidades del producto.

## 12.2. Acerca de la programación de los embebidos

A la hora de programar un dispositivo embebido como es el Esp32, es necesario extremar el cuidado en cuanto al uso de los recursos, ya que se dispone de una memoria muy limitada y generalmente el microprocesador cuenta con uno o dos núcleos. Se cuenta con un sistema operativo muy limitado que es FreeRTOS, por lo que recae sobre el programador una responsabilidad mayor a la hora de gestionar los recursos.

En cuanto al tamaño del programa y las limitaciones en la memoria, si bien tuvimos problemas en un momento, pudimos solucionarlos modificando el particionamiento de la memoria disponible en el Esp32, pero de seguir creciendo el tamaño del programa, hubiera llegado un punto en el que no hubiéramos podido implantar el código en el embebido, por lo que se debe tener en consideración el tamaño del programa y no pensar que se tienen recursos infinitos.

Se debe además investigar sobre cada componente utilizado, por ejemplo cómo hacer para configurar programáticamente a los sensores, cómo comunicarnos con ellos, etc, ya que esto se logra mediante operaciones de muy bajo nivel, configurando los valores de los bits en determinadas secciones de memoria de los sensores. Respecto del Esp32, conviene tener noción de para qué sirven los pines que se tienen a disposición, qué son los pines GPIOs y cuáles son las capacidades de cada uno de ellos. Entender también cómo funciona el bus I2C, que es el mecanismo para la transmisión de data entre los componentes "esclavos" (en nuestro caso los sensores) y el componente "maestro" (el Esp32).

C y C++ son los lenguajes de programación por excelencia para los sistemas embebidos;

las librerías de soporte para estos hardwares vienen generalmente programados en estos lenguajes de programación. Rust se puede utilizar, pero todavía se está desarrollando la parte de Rust para sistemas embebidos, con lo que aveces no se cuenta con soporte para algunas arquitecturas de microprocesador (por ejemplo la arquitectura XTensa del Esp32). En esos casos uno se las tiene que rebuscar para hacerlo andar y puede que las complejidades asociadas a ello terminen opacando los beneficios del lenguaje.

El ciclo de desarrollo en la programación con embebidos es lento, ya que cada vez que probamos algo, hay que desplegar la nueva versión de código en el dispositivo, lo cual nos toma entre 30 y 60 segundos, y luego hay que probar los cambios, cosa que puede tomar varios minutos.

Debuggear se vuelve complicado; utilizamos logs para ver en qué estado está el programa, pero los logs afectan la performance del dispositivo, ya que el microcontrolador se tiene que encargar de transferir mediante usb serial los logs que van sucediendo. Esto consume ciclos de ejecución y dependiendo del baud rate configurado puede tomar más o menos tiempo. Al afectarse la performance, puede suceder que los logs generen que un bug que queremos reproducir no suceda, por lo que nos quedamos sin saber en qué momento falló el programa.

Por último, la interfaz "hombre-máquina" que hay con los dispositivos es prácticamente inexistente. Aparte de la conexión mediante USB y los logs mencionados que se utilizan para debugging, si no utilizamos eso, se cuenta con un led que podemos hacer parpadear de distintas maneras dado distintos estados, pero no más que eso. Se vuelve indispensable asociar mecanismos al embebido que nos puedan brindar más información de lo que sucede adentro del mismo. En nuestro caso solucionamos esta cuestión con la aplicación de celular, dado que con ella podemos ver si estamos recibiendo adecuadamente mediciones o no.

### 12.3. Acerca del desarrollo de la aplicación

Flutter, como framework para el desarrollo de aplicaciones mobile, tiene sus ventajas y desventajas.

Para empezar con las ventajas, podemos decir que Flutter acorta el ciclo de desarrollo, ya que los cambios que hacemos en el código se ven reflejados en la pantalla del dispositivo casi inmediatamente, gracias a la funcionalidad de flash build. No es necesario recompilar y deployar nuevamente todo el código como sucede con Android nativo.

Además, Flutter está muy orientado al desarrollo de UIs, con una gran cantidad de herramientas para sacar andando rápidamente una aplicación sencilla con una interfaz muy buena. Para esto provee de mucho widgets fáciles de utilizar que se insertan rápidamente en la interfaz. Un ejemplo de esto es el ListView, que es el equivalente al RecyclerView de Android nativo. En Android, un RecyclerView necesita que se le especifique un Adapter, un Layout Manager y los ViewHolders que componen al RecyclerView, debiéndose crear como mínimo tres clases nuevas e implementar un montón de métodos de determinada manera para que funcione adecuadamente. El ListView en Flutter, por el contrario, solamente necesita que se le especifique un ItemBuilder para saber cómo construir los widgets hijos.

Una gran ventaja de programar en Flutter es que la aplicación se puede desarrollar tanto para dispositivos Android como para dispositivos con IOs, aunque no pudimos probarlo en este último al no contar con dispositivos de Apple.

Se cuenta además con una gran cantidad de librerías que son muy fáciles de importar, como por ejemplo la librería que usamos para graficar los datos de las mediciones obtenidas, o la librería de FlutterBlue para la comunicación mediante Bluetooth.

En cuanto a las desventajas observadas, podemos decir que al ser un framework que todavía está en desarrollo hay algunos aspectos en los que todavía está limitado en comparación con Android nativo, particularmente en lo que refiere a la interacción con el hardware subyacente. En nuestro caso, para poder comunicarnos con el guante mediante bluetooth,

tuvimos que usar la ya mencionada librería de FlutterBlue que es en realidad la librería de una tercera persona que todavía se hallaba en desarrollo y que contaba con algunos bugs e issues sin resolver, sin hablar siquiera de la limitadísima documentación de esta librería.

Otra cuestión que observamos es que al estar el código de la vista no en archivos .xml como sucede en Android sino en medio del código de la app en Dart, se puede desordenar muy fácilmente el código, acabando con un código muy extenso y difícil de leer. Más aún por la forma en cómo se implementan las interfaces de usuario en Flutter, en la cual se van anidando los widgets que tienen el comportamiento programado ahí mismo. Es por esto que se hace necesario extremar las buenas prácticas de programación y tratar de estructurar cuidadosamente el código de la aplicación para evitar el desorden.

## 12.4. Acerca del machine learning embebido

Resulta asombroso que hoy en día podamos implantar una red neuronal de unos escasos 1,5MB en un microcontrolador y que funcione tan bien. Implantar una red neuronal en un sistema embebido tiene muchísimas aplicaciones prácticas para proyectos de IoT de toda índole.

No obstante, los 1,5MB que la red neuronal se deben en parte a que tenemos un conjunto de gestos a interpretar limitado. Habría que ver cómo evoluciona el tamaño a medida que se le agregan gestos; eventualmente el tamaño de la red neuronal superará los 4MB disponibles del Esp32, por lo que habrá que considerar usar un módulo de memoria RAM externo.

Se debe considerar que la red neuronal y la recolección de datos deben ejecutarse en paralelo<sup>16</sup>, por lo que es conveniente disponer de un microcontrolador con al menos dos núcleos, para que en uno corra un hilo con la red neuronal y en otro el hilo para la recolección de los datos. No obstante esto no es un impedimento, el motor de inferencia con la red neuronal y el hilo para la recolección de datos se alternaron los ciclos de ejecución de uno de

---

<sup>16</sup>A menos que nuestro caso de uso permita que se alterne la recolección de datos con las inferencias.

los dos núcleos (ya que el otro se encargaba de escuchar comandos desde la aplicación) y no hubo mayor problema.

## 12.5. Acerca del entrenamiento de la red neuronal

Más allá de que se haya tratado de un proyecto de machine learning para sistemas embebidos, nos topamos con los problemas habituales de los proyectos de machine learning. En nuestro caso la cantidad de gestos que le enseñamos a la red neuronal fue limitado, y se halla muy ajustado a los movimientos de las manos que efectuamos nosotros. Así como en la lengua hablada hay distintas tonalidades, velocidades de habla, y demás, lo mismo sucede con la lengua de señas argentina. Es esperable que otra persona haga los gestos de alguna forma un tanto distinta, por lo que habría que ver si la red neuronal es capaz de interpretar bien sus gestos. Posiblemente no, para esto necesitaríamos contar con un set de datos muchísimo mayor, provisto por una gran cantidad de personas diferentes. En caso de un emprendimiento con este producto, probablemente habría que destinar presupuesto para obtener este tipo de datos diversos de calidad, contratando personas para efectuar los gestos bajo una supervisión adecuada por parte de una persona capacitada en la recolección de datos.

Por otra parte, el hecho de que la matriz de confusión refleje una alta precisión, esto no necesariamente se refleja en la práctica. Una precisión de alrededor del 100 % en la matriz de confusión generalmente es un indicativo de que se está incurriendo en un problema de overfitting, es decir que la red neuronal se está adecuando demasiado a los datos provistos y que no va a ser capaz de asimilar ligeras variaciones en los gestos.

Es conveniente también disponer de un set de datos para testing, para así validar la precisión de la red neuronal sin tener que probar a mano una nueva red neuronal a cada cambio o incremento del set de datos, reduciendo así el ciclo de desarrollo.

## 12.6. Acerca de la funcionalidad

Con este proyecto hemos podido mostrar que usando únicamente sensores IMU y una red neuronal embebida se pueden interpretar gestos hechos con las manos y brazos; para ello fundamentalmente debemos proporcionar los datos adecuados para entrenar la red.

No obstante, hay varios aspectos fundamentales en la lengua de señas que nosotros dejamos de lado pero que han de ser considerados si el día de mañana se quisiera que un producto como éste ayude fehacientemente a personas que se comunican con lengua de señas argentina. Para empezar nos limitamos al uso de un único guante, el cual registra los gestos de un único brazo, cuando en realidad los gestos realizados en LSA involucran generalmente a las dos manos. No solo eso, sino que también estamos dejando de lado las expresiones faciales, clave para expresar sentimientos y estados de ánimo; un gesto puede variar en su significado en base a estas expresiones faciales.

Además, debemos traducir de un idioma a otro con todas las complejidades que eso conlleva, dado que tenemos dos gramáticas distintas. En la actualidad no nos ocupamos de ello, traduciendo gesto a gesto, lo cual es una limitación.

También sucede que dado el contexto en el que se realiza una oración en lengua de señas, un gesto puede significar una cosa u otra. Por ejemplo, señalar "tú" o "el" o "ella" implica el mismo gesto de señalar con el dedo índice, pero orientado de manera distinta directo hacia una persona o hacia otra, por lo que sería difícil hacer la distinción.

Desde una perspectiva técnica, hay algunos gestos que son muy parecidos entre sí y difíciles de distinguir. Esto requeriría de una mayor cantidad de datos para poder ayudar a la red a distinguir entre los gestos. También está la cuestión de la temporalidad de la seña y de la velocidad de la misma, es decir qué tan rápido está efectuando el usuario la seña ya que esto afecta a las interpretaciones. Una forma de resolverlo sería recolectar más datos más diversos, con personas que realicen el gesto a mayor o menor velocidad.

Por último, tal y como mencionamos acerca de las conclusiones sobre el desarrollo del

hardware, el guante a día de hoy es incómodo y entorpece los gestos, por lo que será necesario mejorar el prototipo para que esto no suceda.

## Referencias

- [1] Espressif api docs. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/index.html>.
- [2] Freertos. <https://www.freertos.org/features.html>.
- [3] How many people use asl in the united states? why estimates need updating. sign language studies.
- [4] rust-embedded. <https://docs.rust-embedded.org/book/>.
- [5] Minority report 's gesture-based user interface, 2002. <https://www.youtube.com/watch?v=PJqbivkm0Ms>.
- [6] Créase el programa nacional de detección temprana y atención de la hipoacusia., 07 2010.
- [7] Gest: work with your hands, 2016. <https://www.kickstarter.com/projects/asdffilms/gest-work-with-your-hands>.
- [8] Plexus vr-ar gloves, 2016. <https://medium.com/@qbittech/plexus-vr-ar-gloves-1738d50ce4ed>.
- [9] Mocap pro gloves, stretch sensor-enabled motion capture gloves, 2020. <https://stretchesense.com/solution/gloves/>.
- [10] Sigberto A. Viesca Brandon Garcia. Real-time american sign language recognition with convolutional neural networks. 2016.
- [11] Gustavo Menna Jose Luis Riccardo Camara de diputados: Lorena Matzen, Gonzalo Pedro Antonio del Cerro. Instituyase en el ambito de las fuerzas de seguridad de la nacion la lengua de seÑas argentina - lsa -. 11 2019.

- [12] David J. Sturman David Zeltzer. A survey of glove-based input. *IEEE Computer Graphics Applications*, 1994.
- [13] Ministerio de salud. Resolución 1209/2010, 2010.
- [14] Jeffrey Dean. Tensorflow: Open source machine learning, 2015.
- [15] Espressif Systems. *ESP32 Hardware Design Guidelines*, 2021. version 3.2.
- [16] Espressif Systems. *ESP32 Series Datasheet*, 2021. version 3.2.
- [17] Espressif Systems. *ESP32 Technical Reference Manua*, 2021. Version 4.6.
- [18] Bolt et al. Put-that-there: Voice and gesture at the graphics interface, 1982.
- [19] Flex sensor. Flex sensor — Wikipedia, the free encyclopedia, 2020. "[https://en.wikipedia.org/wiki/Flex\\_sensor](https://en.wikipedia.org/wiki/Flex_sensor)".
- [20] Martin Foulder. Continuous delivery for machine learning. 2019.
- [21] InvenSense Inc. *MPU-6000 and MPU-6050 Product Specification*, 2013. Revision 3.4.
- [22] InvenSense Inc. *MPU-6000 and MPU-6050 Register Map and Descriptions*, 2013. Revision 4.2.
- [23] Gladys Massé. Indec. estudio nacional sobre el perfil de las personas con discapacidad 2018. 2018.
- [24] Allan D Murray. Instructables circuits. 2017.
- [25] ONU. Desarrollo social inclusivo, 2015.
- [26] D. Situnayake P. Warden. *TinyML: Machine Learning with TensorFlow on Arduino, and Ultra-Low Power Micro-Controllers*. O'Reilly Media, Sebastopol, 2019.

[27] TensorFlow team. Announcing tensorflow lite. 11 2017. <https://developers.googleblog.com/2017/11/announcing-tensorflow-lite.html>.

## Apéndice A Rust y el Esp32

Es muy común que la programación en sistemas embebidos se efectúe en C o en C++, debido al hecho de que le proporcionan al programador control sobre las funciones de hardware subyacentes, así como un mayor control en cuanto al manejo de la memoria del sistema. No obstante, hacer esto resulta complejo con dichos lenguajes de programación; es frecuente causar leaks de memoria y otros problemas asociados a la programación concurrente, como son los deadlocks, race conditions, etc... Rust es un moderno lenguaje de programación que proporciona un nivel de control sobre el sistema similar al de C o C++ pero previniendo en tiempo de compilación muchos de los problemas mencionados anteriormente.

Por esto es que inicialmente nos propusimos realizar el desarrollo del software del microcontrolador en Rust, para lo cual invertimos una gran cantidad de horas de trabajo. Sin embargo nos topamos con varios problemas que nos hicieron decantar por la programación en C++.

### A.1 Compilación cruzada

Primero se realizaron varias pruebas de concepto. Aunque requiere una configuración previa y un entendimiento amplio del tema pudimos embeber código en Rust dentro del Esp32. Para lograrlo es necesario realizar una cross-compilation (compilación cruzada), es decir que se compila el código para que sea ejecutable en una plataforma distinta a aquella en la que el compilador se ejecuta. En nuestro caso el compilador se ejecutaba dentro de un sistema operativo linux o ios(aarch64-apple-darwin) y luego se ejecutable en el esp32 (xtensa-esp32-elf). La plataforma de xtensa-esp32-elf no es una opción listada dentro las arquitecturas mas comunes que el compilador de Rust contempla<sup>17</sup>, es necesario utilizar una herramienta particular llamada rustup. Rustup es lo que se llama un toolchain, un conjunto

---

<sup>17</sup><https://docs.rust-embedded.org/faq.html>

de herramientas informáticas que se ejecutan en cadena. Rustup además instala el lenguaje de programación Rust desde canales de lanzamiento oficiales, lo que permite mantenerlo actualizado en versiones estables y también utilizar compiladores beta o nightly. Rust llama nightly a una versión del lenguaje que se lanza automáticamente cada noche y que tiene funcionalidades inestables o de prueba. Para poder utilizar rust en la arquitectura de xtensa es necesario instalar la versión menos estable, es decir, la versión nightly. Podríamos suponer que tener una versión del leguaje no estable es una de las razones por la cual no prosperó esta implementación. Para poder compilar código que no tiene una versión estable de rust no se puede utilizar el típico gestor de paquetes de cargo, se debe utilizar una variante llamada xargo que afortunadamente puede instalarse utilizando cargo.

Para lograr una compilación cruzada con esa plataforma como "objetivo" es necesario especificarlo dentro de un archivo config situado en la carpeta oculta de cargo A continuacion mostramos la configuracion que fue utilizada

```
1 [target.xtensa-esp32-none-elf] # esp32
2 runner = "xtensa-esp32-elf-gdb -q -x openocd.gdb"
3 rustflags = [
4     "-C", "link-arg=-Wl,-Tlink.x",
5     "-C", "link-arg=-nostartfiles",
6     "-C", "link-arg=--sysroot=/xtensa-rust-quickstart/target/sysroot/
7         lib/rustlib/xtensa-esp32-none-elf",
8 ]
9 [build]
10 target = "xtensa-esp32-none-elf" # esp32
11 rustc = "rustc"
```

**Listing 3:** cargo config

Se pueden identificar dos configuraciones definidas entre corchetes: build y target. En la configuración del build se especifica para que arquitectura objetivo se quiere compilar y construir el código ejecutable. En el comienzo del archivo se define como se va a ejecutar la

compilación para la arquitectura especificada después del punto. El comando que se va a ejecutar en la compilación junto con algunos flags:

- -Wl,-Tlink.x: para utilizar el enlazador predeterminado LLD
- -nostartfiles: indicación para que no se utilice los archivos de inicio del sistema estándar durante el proceso de link o vinculación. Las bibliotecas del sistema estándar se usan normalmente, a menos que se use
- -sysroot: especificamos el path completo de donde se buscaran los scripts de GCC y GNU.

Con esta configuración fue posible construir código ejecutable utilizando el comando

```
1 cargo xbuild
```

**Listing 4:** build

## A.2 Pruebas de concepto

La primera prueba de concepto que se realizó consistió en un programa cuyo objetivo era hacer parpadear la led del nodemcu cada cierto intervalo de tiempo. Esta es la prueba más sencilla que se puede realizar para identificar que el dispositivo está ejecutando el código desplegado. Es equivalente al típico programa 'Hola mundo' que suele realizarse en la introducción de cualquier lenguaje estándar.

La siguiente prueba de concepto que se realizó consistió en un programa que pueda imprimir un logs con cierto formato por el puerto usb serial. Aunque parece sencillo, en un entorno no standar involucra cierta dificultad porque se debe poder transformar la señal de forma correcta. Un error muy frecuente suele ser tener una diferencia de frecuencia entre el dispositivo que envía los datos y el ordenador que los trata de decodificar. Además como la señal que contiene los comentarios loggeados se envía a través del puerto serie como

recurso físico compartido quizás por mas de un proceso, fue necesario tener en cuenta la sincronización de los distintos procesos para hacer uso del recurso de manera ordenada sin causar un deadlock.

### A.3 Dependencias

Para configurar estas dos pruebas de concepto fue necesario definir el siguiente cargo.toml. Al igual que sucede cuando se utiliza rust en un entorno estandar, el archivo cargo.toml es un archivo de manifiesto que se utiliza para describir el paquete y sus dependencias. En este caso, en lugar de cargo, será Xargo quien lo interprete. La unica condición es que no podemos tener como dependencias librerías estándar o paquetes que hagan uso del sistema operativo.

A continuación detallaremos la configuración del archivo cargo.toml que fue utilizado para esta primera versión.

```
1 [package]
2 name = "esp_glove_rs"
3 version = "0.0.1"
4 edition = "2018"
5
6 [dependencies]
7 xtensa-lx-rt = "0.5.0"
8 xtensa-lx = "0.3.0"
9 panic-halt = "0.2.0"
10 esp32-hal = { path = "esp32-hal", version = "0.2.0", features = ["alloc"]
11   }
12
13 [dependencies.nalgebra]
14 default-features = false
15 version = "0.24.1"
```

```
16 [profile.dev]
17 codegen-units = 1
18
19 [[example]]
20 name = "esp32"
21 required-features = ["xtensa-lx-rt/lx6", "xtensa-lx/lx6"]
22
23 [[example]]
24 name = "logger"
25 required-features = ["xtensa-lx-rt/lx6", "xtensa-lx/lx6"]
```

**Listing 5:** cargo toml

En el comienzo definimos el nombre del paquete (esp glove rs) y la version (0.0.1). Luego las dependencias (xtensa -lx-rt, xtensa -lx, panic -halt, esp32 -halt) con su correspondiente versión y por último las dos pruebas de concepto anteriormente mencionadas como programas de ejemplos que se pueden utilizar como base o para probar el correcto funcionamiento (esp32, logger). Cabe mencionar que para estos dos ejemplos se necesita la característica específica de xtensa para compilar.

También definimos una configuración específica para un entorno de desarrollo que utiliza 1 cargogen-units. Esta configuración controla el indicador -C de unidades de generación de código que controla en cuántas "unidades de generación de código" se dividirá un crate. Más unidades de generación de código permiten que se procese más de un crate en paralelo, lo que posiblemente reduzca el tiempo de compilación, pero puede producir un código más lento. El valor predeterminado es 256 para compilaciones incrementales y 16 para compilaciones no incrementales.

## A.4 Hardware Abstraction Layer

En el apartado anterior mencionamos que una de las dependencias utilizadas fue esp-hal. Pero ¿En qué consiste esta dependencia? y ¿porque es crucial en la compilación de este programa embebido? HAL es el acrónimo de hardware abstraction layer. Para trabajar en un embebido es necesario identificar los registros de memoria que nos dan acceso a los periféricos.

Los periféricos son los circuitos digitales que nos permiten una interacción con el mundo «exterior» al microcontrolador. Su función es la de poder habilitar o deshabilitar las salidas digitales, leer sensores analógicos, comunicación con terminales digitales o sacar señales analógicas de una conversión digital. El acceso a esos periféricos puede estar escrito en distintos niveles.

- Board crate: es un paquete con el mas alto nivel de abstracción para trabajar con un tablero.
- HAL crate: es un paquete que contiene una capa de abstraccion al hardware, por sus siglas en ingles Hardware abstraction layer.
- MicroArquitecture crate en conjunto con un peripheral access crate: dos paquetes que en conjunto definen el acceso a los periféricos para un hardware específico. Es el mas bajo nivel de abstracción

Nosotros utilizamos como dependencia un crate del tipo HAL que funciona exponiendo una interfaz a las estructuras físicas en crudo. Por ejemplo podemos acceder a distintos periféricos como por ejemplo:

- pines de entrada salida (GPIO)
- Serial communication

- I2C
- Reloj, timers
- conversion analogica digital

## A.5 Panic halt

El minimo codigo embebido no estandar necesita definir una funcion de panic

### A.5.1 Despliegue

Para grabar el codigo se ejecuta el siguiente comando:

```
1 cargo espflash --chip esp32 --features="xtensa-lx-rt/lx6,xtensa-lx/lx6"  
      /dev/tty.usbserial-0001
```

**Listing 6:** flash

le indicamos primero cual es el chip que utilizamos (esp32) luego las features y por ultimo cual es el puerto tty que en mi caso se encontraba bajo el path /dev/tty.usbserial-0001

## A.6 Rust y lectura de sensores MPU6050

Leer de los sensores utilizando Rust fue posible. Luego de un estudio de la configuración de los sensores fue posible construir un programa no estándar que ejecutara en un ciclo sin fin la lectura de la temperatura, la aceleración y la velocidad angular en cada eje de coordenadas y las mostrara por pantalla. A continuación se puede observar el código fuente de la función principal:

---

```
1 #[no_std]  
2 #[no_main]  
3 #[feature(alloc_error_handler)]  
4 #[feature(default_alloc_error_handler)]
```

```

5  use xtensa_lx::timer::delay;
6  use esp32_hal::*;
7  extern crate alloc;
8  use esp32_hal::alloc::{Allocator, DEFAULT_ALLOCATOR};
9  use core::ops::Deref;
10 #[global_allocator]
11 pub static GLOBAL_ALLOCATOR: Allocator = DEFAULT_ALLOCATOR;
12
13 mod mpu;
14 #[macro_use]
15 mod logger;
16 use core::fmt::Write;
17
18 const CORE_HZ: u32 = 40_000_000;
19 main() -> ! {
20     // take the peripherals
21     let peripherals = target::Peripherals::take().expect("Failed to obtain
22     peripherals");
23     let mut timg0 = peripherals.TIMG0;
24     let mut timg1 = peripherals.TIMG1;
25
26     // openocd disables the watchdog timers on halt on startup
27     disable_timg_wdts(&mut timg0, &mut timg1);
28
29     //General Purpose Input/Output pins
30     let pins = peripherals.GPIO.split();
31     let mut led = pins.gpio2.into_push_pull_output();
32
33     let (dport, dport_clock_control) = peripherals.DPORT.split();
34
35     let logger = logger::Logger::new(dport_clock_control, peripherals.RTCCNTL,
36                                     peripherals.APB_CTRL, peripherals.UART0, pins.gpio1, pins.gpio3);
37     let serial_port_logger = alloc::sync::Arc::new(
38         CriticalSectionSpinLockMutex::new(logger));
39     let mut mpu = mpu::Mpu::new(serial_port_logger.clone(),
40                                peripherals.I2C0, pins.gpio21, pins.gpio22, dport);
41     mpu.init().unwrap();
42
43     mpu.set_accel_range(mpu::device::AccelRange::G8).unwrap();
44     let acc_range = mpu.get_accel_range().unwrap();
45     serial_port_logger.deref().lock(|logger| {
46         println!(logger, "Accelerometer range set to: +-{:?}", acc_range);
47     });

```

```

48     mpu.set_gyro_range(mpu::device::GyroRange::D500).unwrap();
49     let gyro_range = mpu.get_gyro_range().unwrap();
50     serial_port_logger.deref().lock(|logger| {
51         println!(logger, "Gyro range set to: +-{:?} deg/s", gyro_range);
52     });
53
54
55     mpu.set_filter_bandwidth(mpu::device::BANDWITH::_21_HZ).unwrap();
56     let filter_bandwidth = mpu.get_filter_bandwidth().unwrap();
57     serial_port_logger.deref().lock(|logger| {
58         println!(logger, "Filter bandwidth set to: {:?}", filter_bandwidth);
59     });
60
61     loop {
62         let temp = mpu.read_temperature_celcius().expect("read error");
63         serial_port_logger.deref().lock(|logger| {
64             println!(logger, "temperature: {:.3} C", temp);
65         });
66         let acc = mpu.read_acceleration().expect("acc error");
67         serial_port_logger.deref().lock(|logger| {
68             println!(logger, "acc: x={:.3} y={:.3} z={:.3}", acc.x, acc.y, acc.z);
69         });
70         let gyro = mpu.read_gyro_radians().expect("gyro error");
71         serial_port_logger.deref().lock(|logger| {
72             println!(logger, "gyro: x={:.3} y={:.3} z={:.3}", gyro.x, gyro.y, gyro.z);
73         });
74         led.set_high().expect("led error");
75         delay(CORE_HZ); // timer
76     }
77 }
```

En las primeras líneas se puede observar que como mencionamos anteriormente, fue necesario incorporar anotaciones para definir un proyecto que no use las librerías estándar ni una función de entrada main. También fue necesaria una configuración específica para la función de alocación de memoria utilizando un crate de rust. Se incorporó el uso de la librería de esp32 HAL como dependencia para tener acceso a todas las abstracciones de la capa física.

```
#![no_std]
#![no_main]
#![feature(alloc_error_handler)]
#![feature(default_alloc_error_handler)]

use xtensa_lx::timer::delay;
use esp32_hal::*;

extern crate alloc;

use esp32_hal::alloc::{Allocator, DEFAULT_ALLOCATOR};

use core::ops::Deref;
```

A continuación se puede ver que se importan dos módulos propios, el logger y la librería del sensor, que detallaremos más adelante

```
mod mpu;
#[macro_use]
mod logger;
use core::fmt::Write;
```

También fue necesario definir la frecuencia del core. El reloj por default tiene como fuente de frecuencia un oscilador de cristal que en la mayoría de los casos es de 40mhz pero puede ser de 2mhz dependiendo del tablero.

```
const CORE_HZ: u32 = 40_000_000
```

Después definimos variables de los periféricos del dispositivo que nos servirán para las configuraciones siguientes. Fue necesario deshabilitar los temporizadores de vigilancia (watchdog) para evitar un reseteo del sistema al iniciar

```
let peripherals = target::Peripherals::take().expect("Failed to obtain
peripherals");
let mut timg0 = peripherals.TIMG0;
let mut timg1 = peripherals.TIMG1;
disable_timg_wdts(&mut timg0, &mut timg1);
```

Luego se puede ver que se inicializa el logger para lo cual fue necesario hacer uso del pin general numero 1, el pin general numero 3 y el puerto uart. También fue necesario el uso de un mutex para poder sincronizar el orden de los logs por el puerto serie que es un recurso compartido por los distintos core del dispositivo.

```
let logger = logger::Logger::new(dport_clock_control, peripherals.RTCCNTL,
                                 peripherals.APB_CTRL, peripherals.UART0, pins.gpio1, pins.gpio3);
let serial_port_logger = alloc::sync::Arc::new(
    CriticalSectionSpinLockMutex::new(logger));
```

Luego se configuro el sensor MPU con un rango de aceleración igual a 8 grabedades, un rango de gyroscoipo igual a 500 grados y una amplitud de 21 hz para el filtro de frecuencia. No]ótese el uso intencional de unwrap en estos casos para cortar la ejecución en caso de que alguna configuración falle.

```
let mut mpu = mpu::Mpu::new(serial_port_logger.clone(),
                             peripherals.I2C0, pins.gpio21, pins.gpio22, dport);
mpu.init().unwrap();
mpu.set_accel_range(mpu::device::AccelRange::G8).unwrap();
let acc_range = mpu.get_accel_range().unwrap();
mpu.set_gyro_range(mpu::device::GyroRange::D500).unwrap();
let gyro_range = mpu.get_gyro_range().unwrap();
mpu.set_filter_bandwidth(mpu::device::BANDWIDTH::_21_HZ).unwrap();
let filter_bandwidth = mpu.get_filter_bandwidth().unwrap();
```

Si nos adentramos en la librería del MPU definida por nosotros también se puede observar que se escribe en los registros especificados por datasheet del sensor para cada configuración.

Por ejemplo para configurar el filtro de amplitud de frecuencia es necesario escribir el valor deseado los tres bits menos significativos en el registro de direccion 0X1A

```
/*
Wite byte 0000xxx Register 0x1a to set filter bandwith

----- / -- / -- / EXT_SYNC_SET / DLPF_CFG /
----- */

pub fn set_filter_bandwidth(&mut self, bandwith: BANDWITH) -> Result<(), Error> {
    self.write_bits(CONFIG::ADDR,
                    CONFIG::DLPF_CFG.bit,
                    CONFIG::DLPF_CFG.length,
                    bandwith as u8)?;
    Ok(())
}
```

Para configurar el rango de la aceleración y definir una aceleración acorde es necesario escribir en el registro de dirección 0x1C con una mascara de bits 00011000 para no pisar los demás bits del registro

```
/*
Set accel range, and update sensitivy accordingly

Wite byte 000xx000 Register 0x1c to set filter bandwith

----- / XA_ST / YA_ST / ZA_ST / AFS_SEL / -- / -- / -- /
----- */

pub fn set_accel_range(&mut self, range: AccelRange) -> Result<(), Error> {
    self.write_bits(ACCEL_CONFIG::ADDR,
                    ACCEL_CONFIG::FS_SEL.bit,
                    ACCEL_CONFIG::FS_SEL.length,
                    range as u8)?;

    self.acc_sensitivity = range.sensitivity();
}
```

```
    Ok(())
}
```

La misma mascara de bits se utiliza para la configuración del giroscopio pero en este caos la dirección de memoria correspondiente al registro del giroscopio es 0x1B

```
/***
Set gyro range, and update sensitivy accordingly
Write byte 000xx000 Register 0x1b to set filter bandwith
-----
| XG_ST | YG_ST | ZG_ST |     FS_SEL    | -- | -- | -- |
-----
***/
```

```
pub fn set_gyro_range(&mut self, range: GyroRange) -> Result<(), Error> {
    self.write_bits(GYRO_CONFIG::ADDR,
                    GYRO_CONFIG::FS_SEL.bit,
                    GYRO_CONFIG::FS_SEL.length,
                    range as u8)?;

    self.gyro_sensitivity = range.sensitivity();
    Ok(())
}
```

y por ultimo se puede observar el ciclo principal en el cual se lee de los distintos registros del sensor y se toma el control sobre el puerto serie de manera sincronizada utilizando un lock para imprimir los logs. Al final de cada ciclo se agrega un delay. Por ultimo se prende la led de luz para que sea posible observar la velocidad de las lecturas fácilmente al ver el dispositivo.

```
loop {
    let temp = mpu.read_temperature_celcius().expect("read error");
    serial_port_logger.deref().lock(|logger| {
```

```
    uprintln!(logger, "temperature: {:.3} C", temp);  
});  
  
let acc = mpu.read_acceleration().expect("acc error");  
serial_port_logger.deref().lock(|logger| {  
    uprintln!(logger, "acc: x={:.3} y={:.3} z={:.3}", acc.x, acc.y, acc.z);  
});  
  
let gyro = mpu.read_gyro_radians().expect("gyro error");  
serial_port_logger.deref().lock(|logger| {  
    uprintln!(logger, "gyro: x={:.3} y={:.3} z={:.3}", gyro.x, gyro.y, gyro.z);  
});  
led.set_high().expect("led error");  
delay(CORE_HZ); // timer  
}  
}
```

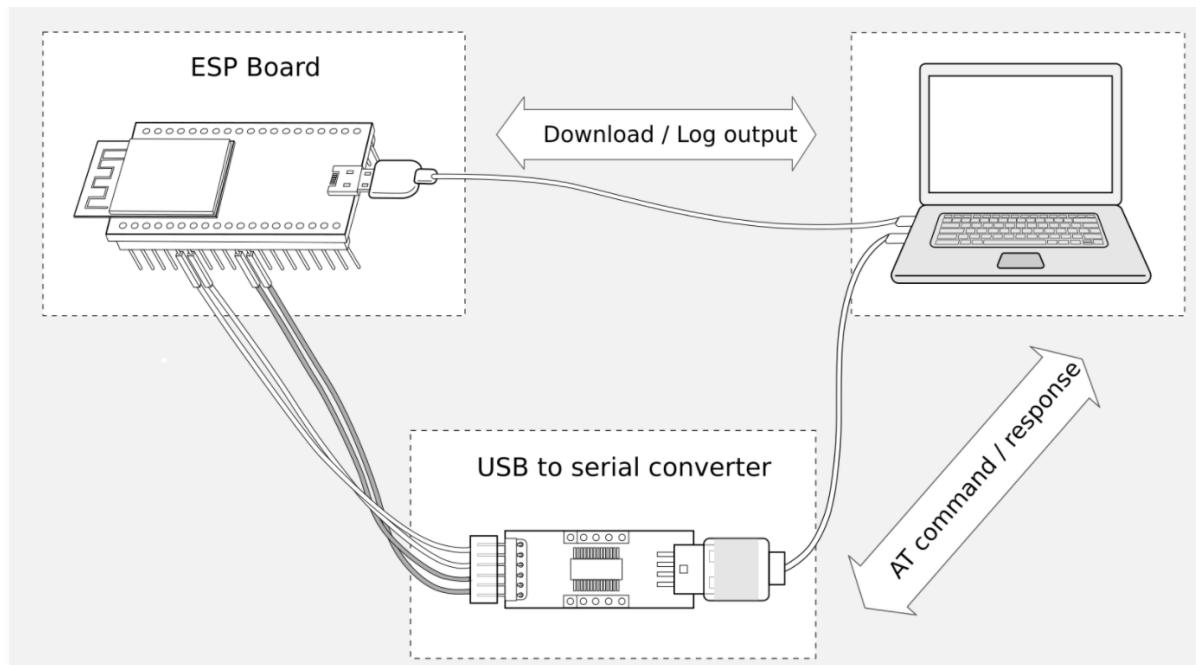
## A.7 Rust y wifi/bluetooth

Al querer utilizar redes inalámbricas para poder enviar las mediciones de los sensores a la implicación móvil nos encontramos con varios impedimentos. En principio quisimos utilizar una librería de código abierto esp-rs/esp-wifi la cual todavía estaba en un estado inmaduro y no compilaba con las ultimas actualizaciones de espressif. Aun al solucionar nosotros mismos los errores de compilación para lograr embeberlo en nuestro dispositivo las pruebas no funcionaron. Aun cuando no existía ningún error de ejecución no lográbamos enviar datos a través de la red. Nosotros podríamos haber creado nuestra propio driver de wifi en rust pero para adecuarlos a la arquitectura de extensa tendríamos que haber emulado el ciclo principal con manejo de interrupciones a partir de la capas de mas bajo nivel, siguiendo los protocolos de TCP/IP. Consideramos que esta alternativa iba a demorar más el proyecto y decidimos descartarla.

## A.8 Comandos AT

Para facilitar la conectividad wifi o bluetooth de los dispositivos existe una solución alternativa compuesta por un conjunto de comandos, llamados comandos AT. Anteriormente se utilizaban para configurar y parametrizar modems pero luego se convirtieron en un estándar. Todos los comandos comienzan por los caracteres AT que significan “atención” y de ahí surge su denominación. Utilizando los comandos AT es posible lograr una conectividad con una red inalámbrica ya sea bluetooth o wifi para transmitir datos de forma sencilla y reduciendo los costos de desarrollo. Por ejemplo podríamos conectarnos a una red wifi enviando al dispositivo el string AT+CWJAP="ssid","password".

Parece una buena idea utilizarlo, pero para este proyecto no fue posible implementarlo, a continuación detallamos las razones. Para poder utilizarlo se debe implantar un soporte lógico (firmware) desarrollado por spressif en el módulo ESP-WROOM-32. El módulo esp32 funciona como un esclavo (slave) que recibe los comandos AT desde un microcontrolador anfitrión (host). El esp32 los interpreta y le envía una respuesta al MCU host. Expressif lo que propone es utilizar un host externo y conectarlo al tablero esp32 utilizando la comunicación UART (Universal Asynchronous Receiver/Transmitter) es una forma de comunicación física a través de un puerto serie. Solo se necesitan dos cables para crear la comunicación entre dos dispositivos. Un pin de transmisión Tx en el dispositivo anfitrión se debe conectar con el pin de recepción Rx en el esclavo y viceversa, el pin de Tx en el esclavo se debe conectar con el pin de recepción Rx en el anfitrión. En la comunicación UART no hay una señal de reloj. Sino que se usa un bit de inicio y de fin. Como una primera aproximación a esta solución, se realizó una prueba siguiendo las recomendaciones de Espressif. Se utilizó una computadora como host y el tablero esp32 como esclavo. Utilizando un cable USB conectado al puerto serie es posible enviar comandos AT desde el ordenador al tablero esp32 pero para recibir las respuestas adecuadas es necesario el uso de un convertidor entre señales USB y señales TTL.



**Figura 59:** Conección física de los componentes para pruebas de comandos AT

Nosotros seguimos las recomendaciones de espressif que se ven en la imagen para probar los comandos AT pero como convertidor utilizamos un Arduino. Para configurar el Arduino de forma tal de que funcione como un convertidor serial utilizamos la librería Software Serial con los numero 6 y 7 pines utilizados para la conexión.

```
#include<SoftwareSerial.h>
SoftwareSerial ATCOM(6,7); //RX, TX
int incomingByte = 0;
void setup()
{
    Serial.begin(115200);
    Serial.println("Ready....");
    ATCOM.begin(115200);
}

void loop() {
    if(Serial.available()){

    }
}
```

```
String outData = Serial.readStringUntil('\n');
Serial.println("Serial: " + outData);
ATCOM.println(outData);
delay(30);
}

if(ATCOM.available()){
    Serial.print("ESP32: ");
    String inData = ATCOM.readStringUntil('\n');
    Serial.println(inData);
}
}
```

Estas pruebas fueron exitosas. Logramos enviar comandos AT y recibir respuestas de éxito. En lugar de utilizar una computadora como anfitrión se podría utilizar otro esp32, un arduino, un STM32 o cualquier otro microcontrolador pero para implementarlo en el guante no era viable incorporar otro dispositivo porque haría el aparato mucho más grande e incómodo.

Una alternativa posible a la comunicación UART para envío y recepción de comandos AT es utilizar la comunicación SDIO (Secure Digital Input Output) es un tipo de comunicación segura de entrada/salida utilizando el bus SD. Este protocolo se puede utilizar al comunicar una tarjeta extraíble SD pero la idea era utilizar los comandos AT usando el esp32 como anfitrión y esclavo a la vez para no tener que incorporar más elementos. Para lograrlo era necesario utilizar 4 a 6 pines (dependiendo si se utiliza el modo de 1 bit o de 4 bits para la comunicación): Gpio 14 para el reloj, Gpio 15 para los comandos, Gpio 2 y Gpio 4 para los datos. En nuestro caso varios de estos pines ya estaban siendo utilizados para la comunicación con los sensores.

En conclusión la utilización de comandos AT fue descartada debido a la complejidad que resulta su implementación en tablero integrado como es el esp32 y a que no estábamos dispuestos a agrandar el guante con otro dispositivo que lo hiciera mucho más incómodo y

difícil de utilizar.

### A.8.1 Propuesta alternativa

Una alternativa posible hubiese sido utilizar como anfitrión al microcontrolador Blue Pill STM32F103C8 conectarlo con el ESP-WROOM-32 únicamente sin utilizar el tablero esp 32 completo o al modulo de wifi ESP8266.

|         |       |
|---------|-------|
| ESP8266 | STM32 |
| VCC     | 3.33V |
| GND     | G     |
| CH PD   | 3.33V |
| Tx      | Pa3   |
| Rx      | Pa2   |

Creemos que conectándolo los componentes de esta forma hubiese sido posible utilizar los comandos AT y por ende utilizar el WiFi de una forma sencilla, lo cual nos hubiese permitido continuar con el proyecto en lenguaje Rust.

## Apéndice B Gestación del proyecto

El equipo de trabajo se constituyó en Noviembre del 2020, arrancando a trabajar en Febrero de 2021 y presentando la propuesta a la comisión curricular de ingeniería informática recién en Agosto del 2021. En todo ese tiempo se hizo mucho trabajo de investigación, barajándose múltiples ideas de proyecto, hasta que nos decantamos por esta. En este anexo vamos a relatar cómo fue todo ese proceso que consumió entre un tercio y un cuarto del esfuerzo total abocado al trabajo profesional.

### B.1 Motivaciones

La motivación que determinó en gran medida las decisiones que tomamos a lo largo de la planificación del proyecto fue la de que queríamos aprender a trabajar con tecnologías con las que no estábamos familiarizados y que nos resultaran de interés. Tal es así que inicialmente partimos de la base de que queríamos trabajar con sistemas embebidos en un proyecto de IoT.

### B.2 Ideas de proyectos y consideraciones

Cuando arrancamos a trabajar con esto, lo primero que hicimos fue un brainstorming de ideas. En este proceso tiramos cualquier cosa que se nos ocurriá, no importaba si era factible o no, sino que debía constituir una base a partir de la cual ir definiendo una idea.

1. Dispositivo para la recolección de datos meteorológicos
2. Robot delivery en espacios controlados (por ejemplo un hotel) que predice el tiempo de envío
3. Collar ubicador de mascotas

4. Dispositivo en heladera / alacena que detecta productos y genera recetas
5. Dispositivo para controlar y monitorear la salud de una pecera o de una laguna
6. Dispositivo para detectar y recolectar malezas
7. Medidor de contaminación
8. Algo relacionado a la salud (ejemplo: medidor de temperatura)
9. Sistema de IoT para un cultivo más eficiente en una huerta
10. Proyecto abocado al deporte, como por ejemplo el boxeo

Fuimos descartando ideas, algunas por su complejidad, por disponer de una curva de aprendizaje muy grande. Otras ideas por ser muy básicas (por ejemplo el dispositivo para la recolección de datos meteorológicos es el "holo mundo" de un proyecto de IoT). Otros porque por cuestiones logísticas se complica mucho conseguir los componentes adecuados (por ejemplo para hacer algo relacionado con la salud, los sensores a conseguir están estrictamente regulados por la ANMAT y por ende se hacen muy difíciles de conseguir).

Después, el IoT es un campo muy extenso y para determinar a qué proyecto abocarnos, era importante que definiéramos un foco a poner en el proyecto y que tuviéramos en claro cuales eran los factores y desafíos típicos de un proyecto de IoT.

A continuación enumeramos unos cuantos factores técnicos que tomamos en cuenta:

- Conectividad
- Duración de la batería
- Librerías asociadas al lenguaje
- Ciberseguridad
- Procesamiento de imágenes

- Concurrencia
- Gestión de recursos
- Procesamiento de múltiples datos
- Interoperabilidad entre stacks, mantenibilidad
- Costo unitario
- Tamaño del producto
- Performance
- Consumo energético
- Flexibilidad para modificar el proyecto (sin que incurra en un incremento exagerado del costo de desarrollo)
- Tiempo para la elaboración de un prototipo
- Tiempo para sacar el producto a mercado
- Mantenibilidad
- Seguridad

Además de los factores técnicos, también tuvimos en cuenta que el proyecto tuviera cierta implicación social y si además se contaba con un stakeholder que nos pudiera asesorar, mejor. En este sentido el proyecto de desarrollar un sistema aplicado al deporte, particularmente al boxeo, era el que mejor se ajustaba. Se trataba de un proyecto que habían iniciado unas alumnas de la facultad bajo el asesoramiento de nuestro tutor Pablo Deymonnaz pero que había sido abandonado por ellas. Ya existía de aquel entonces un contacto que era el presidente de la federación argentina de boxeo.

Al poco tiempo de decidir encarar ese proyecto, este mismo se cayó por el motivo de la pandemia de coronavirus de 2020, dado que debido a esto los gimnasios de boxeo se hallaban todos cerrados, al igual que la federación. El presidente no se dispuso a atendernos por estos motivos que eran entendibles. Para este proyecto habíamos estado barajando la idea de desarrollar un guante de boxeo con sensores para poder monitorear los golpes, construyendo un sistema de análisis de datos para poder brindarle a los stakeholders una perspectiva científica sobre el entrenamiento realizado. Hay que decir que ya existen sistemas parecidos para el entrenamiento de alto rendimiento pero su costo es demasiado elevado y resulta prohibitivo para las federaciones argentinas, dado su escaso presupuesto. No obstante, la idea del guante con sensores siguió presente, aunque hubo que reformularla.

A estas alturas, que fue durante la primera mitad de Abril de 2021 podemos decir que volvimos a la posición de inicio en la que no teníamos una idea definida. Además en paralelo a la ideación de un proyecto, estuvimos dedicándole mucho esfuerzo a la investigación sobre embeber Rust en el Esp32 sin mucho éxito, ya que nos estaba dando muchas complicaciones y para aquel entonces usar Rust embebido era una idea central que teníamos para el proyecto. Estas cosas hicieron que reiniciáramos el proceso de generación de ideas de proyectos, pero ya con una mayor madurez en cuanto a cómo realizar ese proceso, e identificando los errores que habíamos cometido. Hasta ese entonces, no tuvimos en claro cuáles eran los alcances esperados para un trabajo práctico profesional de la facultad de ingeniería y siempre que barajábamos una idea nos excedimos en las expectativas de lo que debíamos realizar. Era frecuente que nos desmotivaramos al ver que un proyecto potencial ya existiera en el mercado. También teníamos la idea preconcebida de que el resultado del proyecto tenía que ser algo muy innovador que se usara en la vida real. Esto ponía la vara demasiado alta por lo que debíamos ser más realistas con lo que buscábamos. Lo cierto es que está bueno que un proyecto de TP tenga una aplicación práctica y más aún que se use, pero lo cierto es que la casi totalidad de los trabajos profesionales después no se terminan usando, incluso los que

son hechos para adentro de la facultad. Es más importante y realista poner el foco en que el proyecto tenga un interés tecnológico, desarrollando un MVP<sup>18</sup> y sin excedernos con las expectativas de perfección del producto.

Es con estas premisas que la idea del guante para lengua de señas se fue gestando. Habíamos visualizado algunos proyectos de guantes realizados para realidad virtual, investigamos el estado del arte del machine learning embebido y de a poco fuimos atando cables para darle forma al proyecto actual.

---

<sup>18</sup>minimum viable product

## Apéndice C Minutas de reunión

En un inicio del trabajo se desarrollaron reuniones que no fueron registradas en minutas en las que se conversó de temas como utilizar rust en el código del embobido, debates sobre la utilización de redes inalámbricas en rust, la utilización de comandos AT, desafíos en el mecanismo de lectura de 5 sensores del bus i2c y por última una comparación de las distintas pilas a utilizar.

**Fecha:** 02 Septiembre 2021

**Medio:** Videollamada

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

**desarrollos realizados:** se realizaron mejoras en la comunicación bluetooth low energy de recolección de datos de sensores. Se creó la pantalla de recolección de datos. Se realizaron algunos diseños de prototipo sin buenos resultados. Se compartieron links a dos posibles diseños

**objetivos del siguiente sprint:** terminar el diseño del prototipo. Arreglar errores en la deserialización de mensajes. Crear pantalla de traducción.

**otros temas tratados:** frente a los problemas traídos a la mesa por los prototipos realizados los tutores recomendaron generar un prototipo que no dependa de la caja, que se pueda sacar y poner fácilmente.

---

**Fecha:** 24 Septiembre 2021

---

**Medio:** Videollamada

---

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

---

Se realizo una muestra de dos prototipos con diferente diseño y también una demostración de las funcionalidades de la aplicación: realizar una conexión por bluetooth, recopilación de datos, interpretación (con datos falsos de interpretaciones) y generación de archivos con los datos recopilados.

**Avances:** En líneas generales se logró muy buenos avances en lo que refiere a la comunicación entre la app y los dispositivos. No obstante, quedan algunas cuestiones por aclarar, por ejemplo la cuestión de la frecuencia de muestreo, dónde si la frecuencia de toma de mediciones es muy elevada podemos terminar generando archivos muy grandes, por lo que se debería averiguar cuál es una frecuencia razonable que permita entrenar adecuadamente la red neuronal sin incurrir en dicho problema (si es que es un problema). Otra cuestión es el porcentaje de pérdida de paquetes a través de la comunicación bluetooth, si éste resulta ser demasiado elevado entonces puede llegar a ser un inconveniente.

**objetivos del siguiente sprint:** Se fijó como objetivo para la siguiente reunion intentar recolectar un pequeño set de datos de prueba y subirlos al pipeline de Edge Impulse. Pero ello es necesario primero: validar las mediciones de los sensores entre los guantes y la app, asegurándonos que las mediciones tengan sentido, y segundo realizar pruebas de frecuencias respecto de la recolección de mediciones y posterior envío mediante bluetooth a la App. También se va a dedicar tiempo a resolver algunos pequeños problemas del hardware; por ejemplo Jazmin deberá soldar los sensores e imprimir nuevas cajas para los sensores, mientras Darius posiblemente debará reemplazar un MPU que no parece estar andando adecuadamente y tal vez añadir un interruptor para poder prender y apagar el guante sin tener que sacar la batería.

---

**Fecha:** 14 de Octubre 2021

---

**Medio:** Videollamada

---

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

---

**Avances:** Se realizaron varios avances, en cuanto al hardware: Jazmín soldó los sensores y Darius terminó el guante izquierdo. En cuanto a la aplicación se mejoró la comunicación entre el esp32 y la app, se agregó la funcionalidad de calibración que se dispara desde la aplicación. Se realizó una modificación del código de la aplicación para desacoplar el modelo y la vista que permite simplificar y centralizar la lógica relacionada con la comunicación del bluetooth en una única clase.

**problemas planteados:** se trajo a la mesa un problema con el que se toparon los alumnos, una muy baja frecuencia en la recolección de datos, cerca de 5hz. No tenemos pruebas empíricas de la frecuencia necesaria para lograr interpretar los gestos pero los profesores manifestaron que una frecuencia de 5hz seguramente sea insuficiente. Un objetivo adecuado sería de alrededor de 100Hz.

**objetivos del siguiente sprint:** Con el fin de resolver el problema, se plantearon realizar las siguientes pruebas para mejorar la frecuencia

- Averiguar si se pueden eliminar los delays al cambiar de address: hacer pruebas con menos sensores.
- Averiguar porque el i2c en 400 lee mediciones erróneas.
- Modificar el modelo de datos y enviar dedo por dedo.
- Tratar de reducir el procesamiento del micro y enviar data en crudo
- Probar dividir los sensores en dos buses i2c

---

**Fecha:** 28 de Octubre 2021

---

**Medio:** Videollamada

---

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

---

**Avances:** Se desarrolló una vista en la aplicación con gráficos de los datos que se van recibiendo por bluetooth. Se hizo una demostración de la misma. Se realizaron distintas pruebas en el código del guante para mejorar la frecuencia: por un lado se eliminó la cola asincrónica que enviaba los datos desde la task de lectura a la task de bluetooth, para realizar la recolección y envío de datos en la misma task, esto mejoró la frecuencia hasta 45hz. Las pruebas de i2c con velocidad 400hz no ayudaron porque se leen datos erróneos. La tercera prueba consistió en enviar las mediciones de cada sensor (en lugar de 5 sensores). Se inició una ultima prueba que quedó sin terminar que consiste en tener un periodo de recolección de datos y otro periodo de envío, es decir, acumular mediciones por un periodo de tiempo y luego enviarlas al final.

**otros temas planteados:** se plantearon otras posibles pruebas a realizar para mejorar la frecuencia que implican más esfuerzo. Una opción es probar tener dos buses i2c. Reemplazar el envío por bluetooth por red wifi.

**objetivos del siguiente sprint:**

- Continuar con la última prueba planteada.
- Realizar pruebas con bus i2c 400hz utilizando una librería.
- Decidir una configuración para definir que mediciones tomar (solo aceleración, solo giro, las dos etc).
- Recolectar data de tres gestos.
- Interactuar con Edge impulse para tratar de entrenar la red

---

**Fecha:** 11 de Noviembre 2021

---

**Medio:** Videollamada

---

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

---

**Avances:** Elevación de la frecuencia a 60Hz Implantación de la librería de edge impulse en el guante y posterior envío de las interpretaciones al guante que identifica movimientos continuos y repetitivos en forma horizontal y vertical de la mano. Visualización de datos después de efectuar una collection **Discusiones:** El problema del bluetooth y de la pérdida de paquetes pareciera pasarle solo a un integrante del equipo, a Jazmín pero no a Darius. De este último punto se resolvió en intentar identificar la causa del error, ya que puede ser alguna cuestión de interferencias, o el celular no funcionando correctamente, o el hardware (aunque muy posiblemente no sea que esté quemado). Una buena prueba para descartar la hipótesis del celular que no funciona es la de probar el guante de Jazmín con otro celular, y también probar el guante de Darius con el celular de Jazmín.

**objetivos del siguiente sprint:** Habiendo logrado alcanzar un MVP con la implantación de la librería de edge impulse, y dado el hecho de que tenemos que realizar una presentación para el Viernes 19, nos fijamos como objetivo pausar momentáneamente el desarrollo para concentrarnos en avanzar con la documentación, para preparar la presentación y también para arreglar algunas cosas que quedaron pendientes, como por ejemplo el problema del guante de Jazmín.

---

---

**Fecha:** 07 de Febrero 2022

---

**Medio:** Videollamada

---

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

---

**avances:** La casi totalidad del esfuerzo que se dedicó al TP se enfocó en la recolección de datos y entrenamiento de la red neuronal que es implantada en el microprocesador del guante, validando después el buen funcionamiento de las interpretaciones con los datos nuevos.

**debate:** Se conversó acerca de qué gestos realizar, a lo cual Sebastián (y estando Pablo de acuerdo) recomendó que nos hiciéramos asesorar por las personas capacitadas en el área de lengua de señas con las que nos comunicamos al inicio del proyecto, con el fin de:

Determinar qué o cuáles frases queremos “interpretar” con el guante. Evaluar de manera más realista el alcance y las limitaciones del producto en sí para un usuario real.

Se hizo un repaso por las historias de usuario y se discutió acerca de alguna que otra feature que podría quedarse afuera del proyecto. Sobre la funcionalidad de interpretar gestos con las dos manos se resolvió que con una única mano ya se podría validar el concepto del guante como herramienta para interpretar gestos de lengua de señas, por lo que no añadiría demasiado valor y podríamos dejarlo afuera. De igual manera, respecto a la historia de usuario número cuatro de la propuesta de TPP: “Como usuario de la aplicación quiero poder autenticarme como administrador para validar que estoy autorizado a realizar ciertas tareas.”

El equipo planteó que no suma mucho valor al proyecto, que la dificultad del trabajo nos dimos cuenta de que no pasa por ahí y que podíamos asumir “buena fe” de parte de los usuarios (dado que el motivo de esta historia de usuario era impedir que personas no capacitadas contribuyeran a constituir el set de datos para el entrenamiento del modelo).

**objetivos del siguiente sprint:** Avanzar con el informe final. Actualizar el diagrama de gantt, marcando los ítems que están resueltos y los que no.

|                       |   |
|-----------------------|---|
| <b>Fecha:</b>         | 7 de Marzo 2022                                       |
| <b>Medio:</b>         | Videollamada  |
| <b>Participantes:</b> | Sabrina, Enrique Díaz Beltran, María, Jazmín Ferreiro |

**temas conversados:** Enrique diaz Beltra contactó al equipo tras haber visto un video publicado en la red social de linkedin en el que se mostraba el uso del guante traduciendo los días de la semana. El pidió reunirse para hacer una propuesta junto con Sabrina, una chica sordomuda y María, una interprete de lengua de señas peruanas. Ellos se mostraron muy interesados por el guante como producto. Ellos pertenecían a una sociedad llamada Professional Growth Center que se dedica al desarrollo personal y está orientado a la inclusión social, ofrsen clases y seminarios de lengua de señas peruanas entre otras cosas. Sabrina preguntó si el guante podía traducir palabras con gran velocidad, si respetaba la gramática de la lengua de señas y si estaba a la venta. También consultaron si era un producto patentado y si podría en un futuro desarrollarse un guante para traducir braille a partir del tacto. Jazmín comentó que la velocidad de traducción depende de la velocidad del entrenamiento. La gramática de la lengua se respeta bastante al traducir palabra por palabra y que de momento no estaba a la venta al ser solo un proyecto final de carrera. En cuanto a la propuesta de leer braille Jazmín comentó que era una buena idea pero estaba fuera del alcance planteado para este proyecto. Sabrina mencionó que el guante le serviría para desempeñarse con mayor naturalidad en un mundo donde la mayoría de las personas son oyentes y no saben lengua de señas. Le parece que sería una buena idea que se incorporen este tipo de productos en hospitales, para hacer una consulta con un doctor sin depender de una intérprete o en instituciones donde se debe hacer un trámite.

También mencionó que estaba orgullosa de que en latinoamérica (ella era oriunda de Perú) haya personas que estén innovando con un objetivo de inclusión. Enrique diaz Beltra propuso consultó si el quipo podía dar una charla abierta difundiendo el avance del proyecto y una demostración del funcionamiento del producto sobre la plataforma de facebook live. Ellos propusieron hacer la difusión de un flyer promocionando la charla. La propuesta fue aceptada y se pactó una fecha para la mencionada charla virtual.

---

---

**Fecha:** 09 de Marzo 2021

---

**Medio:** Videollamada

---

**Participantes:** Pablo Deymonaz, Sebastian García, Jazmín Ferreiro, Darius Maitia

---

**Debate:** En esta reunión Jazmín comentó sobre la reunión que tuvo con las personas interesadas en el proyecto. Darius expreso sus discrepancias. Argumento que le pareció que la intención de esta compañía era hacerse publicidad a costa del trabajo de otro y que el equipo no ganábamos nada. La posición al respecto tanto de Pablo como de Sebastian era diferente. Nos remarcaron que un punto muy importante de la reunión que tuvo Jazmín, es que obtuvimos una validación que nos estaba faltando, dado que nos plantearon un montón de funcionalidades que podría tener nuestro producto como lee braile o traducir frases a gran velocidad. Plantearon limitaciones también (por ejemplo cómo hace la persona hablante para comunicarse con una persona sorda? siendo que la comunicación que plantea este guante es para el sentido inverso). Plantearon incluso que patentáramos el producto, osea que en última instancia se podría llegar a comercializar, lo que indica que hay una necesidad que nuestro producto puede cubrir y que se puede hacer un negocio con ello. En cuanto a la presentación planteada, nosotros tenemos que poner los términos y condiciones de la charla, no necesariamente tenemos que mostrarlo como un producto terminado listo para la venta, sino que más bien el espíritu de la presentación tiene que ser de difusión: explicar qué hicimos, por qué motivación, de qué forma, dejando abierta la posibilidad a que otra persona continúe el proyecto, explicando cómo pensamos que sería la mejor manera. Además, este tipo de eventos sirven para abrir puertas, establecer vínculos, etc, que uno nunca sabe a donde lo pueden llevar. No estaría bueno que todo el trabajo realizado se pierda en el olvido dentro de los muros de la facultad. Esta presentación no es mandatoria para nuestra recibida, pero puede llegar a sumar puntos para darle más legitimidad a la presentación final de nuestro TP.

**Sobre la muestra:**la presentación en la facultad, tiene que durar unos 25 minutos, explicar la motivación, cómo fuimos resolviendo las cosas, qué problemas nos encontramos, qué dificultades, en qué queda este proyecto y conclusiones. **Sobre el informe:**El informe tiene que tener un espíritu similar, es importante que quede bien explicada la motivación que hubo detrás de este proyecto, incluir las partes relacionadas con nuestras tareas de investigación que hicimos cuando escribimos en la propuesta, etc... para que así una persona que lea el informe pueda entender las razones. Se indicó también que era importante que adjuntáramos en un anexo las minutas de reunión, porque la documentación de cómo fue el proceso también es importante. Se definió el 14 de Abril como fecha de presentación

## Apéndice D    Repositorios

1. <https://github.com/Gloveland/lساAnalytics>
2. <https://github.com/Gloveland/app>
3. [https://github.com/Gloveland/esp32\\_glove](https://github.com/Gloveland/esp32_glove)
4. [https://github.com/Gloveland/esp32\\_glove\\_rs](https://github.com/Gloveland/esp32_glove_rs)
5. <https://github.com/Gloveland/mpu6050>
6. [https://github.com/Gloveland/adxl\\_calibration](https://github.com/Gloveland/adxl_calibration)