



UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA

IE-0624 LABORATORIO DE MICROPROCESADORES

Laboratorio 5

José Antonio Ramos Pereira, B86485

Grupo 01

Prof. MSc. Marco Villalta Fallas.

https://github.com/JAR1224/Laboratorio_5_Jose_Ramos

Índice de contenidos

1.	Introducción	1
1.1.	Resumen	1
1.2.	Conclusiones	1
2.	Nota Teórica	2
2.1.	Microcontrolador	2
2.2.	Componentes Electrónicos Complementarios	5
2.3.	Diseño del circuito	5
2.4.	Temas de laboratorio	5
3.	Desarrollo y análisis	7
3.1.	Programa	7
3.2.	Funcionamiento	13
4.	Conclusiones	15
4.1.	Recomendaciones	15
5.	GIT	17

Índice de diagramas

2.1. Obtenido de [1]	2
2.2. Obtenido de [2]	3
2.3. Obtenido de [1]	4
2.4. Obtenido de [1]	5
3.1. Obtener Datos	10
3.2. Entrenar	11
3.3. Clasificación	12
3.4. Obtener Datos	13
3.5. Entrenar	14
3.6. Clasificación	15

1. Introducción

1.1. Resumen

La elaboración del proyecto consistió en plantear una solución utilizando una tarjeta de desarrollo Arduino Nano Ble 33 para detectar gestos (puño, salto, rodilla). La aplicación consistía en obtener los datos necesarios para entrenar una red neuronal usando la biblioteca de código abierto TensorFlow y crear una aplicación que detecta los gestos del usuario y los manda serialmente a la PC.

Los retos principales para la elaboración de la solución fueron los siguientes:

1. Obtener los datos en una ventana similar para todas las muestras utilizadas para el entrenamiento de la red neuronal
2. Encontrar una configuración de una red neuronal que logre identificar los gestos con un alto grado de precisión.
3. Cuando el usuario está usando la aplicación final, poder ordenar los datos provenientes del giroscopio de forma que sea similar a la ventana en la que fueron entrenados.

Para poder superar los retos, se realizaron los siguientes pasos:

1. Acomodar los datos provenientes del giroscopio según un valor de activación según la intensidad del movimiento.
2. A punta de prueba y error, probar distintos números de neuronas, optimizadores y tasa de aprendizaje hasta obtener un resultado satisfactorio.

1.2. Conclusiones

Se concluyó que por medio de las metodologías mencionadas anteriormente, se pudo implementar una solución al problema planteado que cumple con todos los requisitos de diseño requeridos por el enunciado.

2. Nota Teórica

2.1. Microcontrolador

Características generales

Las características generales son las siguientes [1]:

- **NINA B306 Module**
 - **Processor**
 - 64 MHz Arm® Cortex-M4F (with FPU)
 - 1 MB Flash + 256 KB RAM
 - **Bluetooth® 5 multiprotocol radio**
 - 2 Mbps
 - CSA #2
 - Advertising Extensions
 - Long Range
 - +8 dBm TX power
 - -95 dBm sensitivity
 - 4.8 mA in TX (0 dBm)
 - 4.6 mA in RX (1 Mbps)
 - Integrated balun with 50 Ω single-ended output
 - IEEE 802.15.4 radio support
 - Thread
 - Zigbee
 - **Peripherals**
 - Full-speed 12 Mbps USB
 - NFC-A tag
 - Arm CryptoCell CC310 security subsystem
 - QSPI/SPI/TWI/I²S/PDM/QDEC
 - High speed 32 MHz SPI
 - Quad SPI interface 32 MHz
 - EasyDMA for all digital interfaces
 - 12-bit 200 ksp/s ADC
 - 128 bit AES/ECB/CCM/AAR co-processor
 - **LSM9DS1 (9 axis IMU)**
 - 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
 - $\pm 2/\pm 4/\pm 8/\pm 16$ g linear acceleration full scale
 - $\pm 4/\pm 8/\pm 12/\pm 16$ gauss magnetic full scale
 - $\pm 245/\pm 500/\pm 2000$ dps angular rate full scale
 - 16-bit data output
 - **MPM3610 DC-DC**
 - Regulates input voltage from up to 21V with a minimum of 65% efficiency @minimum load
 - More than 85% efficiency @12V

Características Generales 2.1: Obtenido de [1]

Diagrama de bloques CortexM4F

Block Diagram

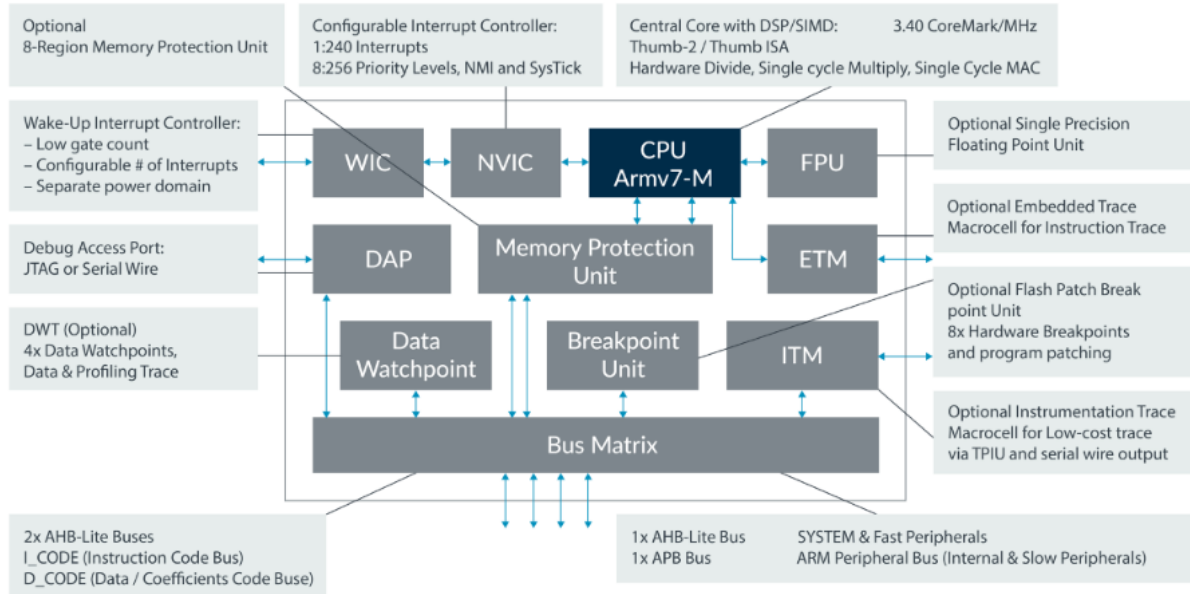


Figure 2: Cortex-M4 processor components

Diagrama de bloques 2.2: Obtenido de [2]

Diagrama de pines

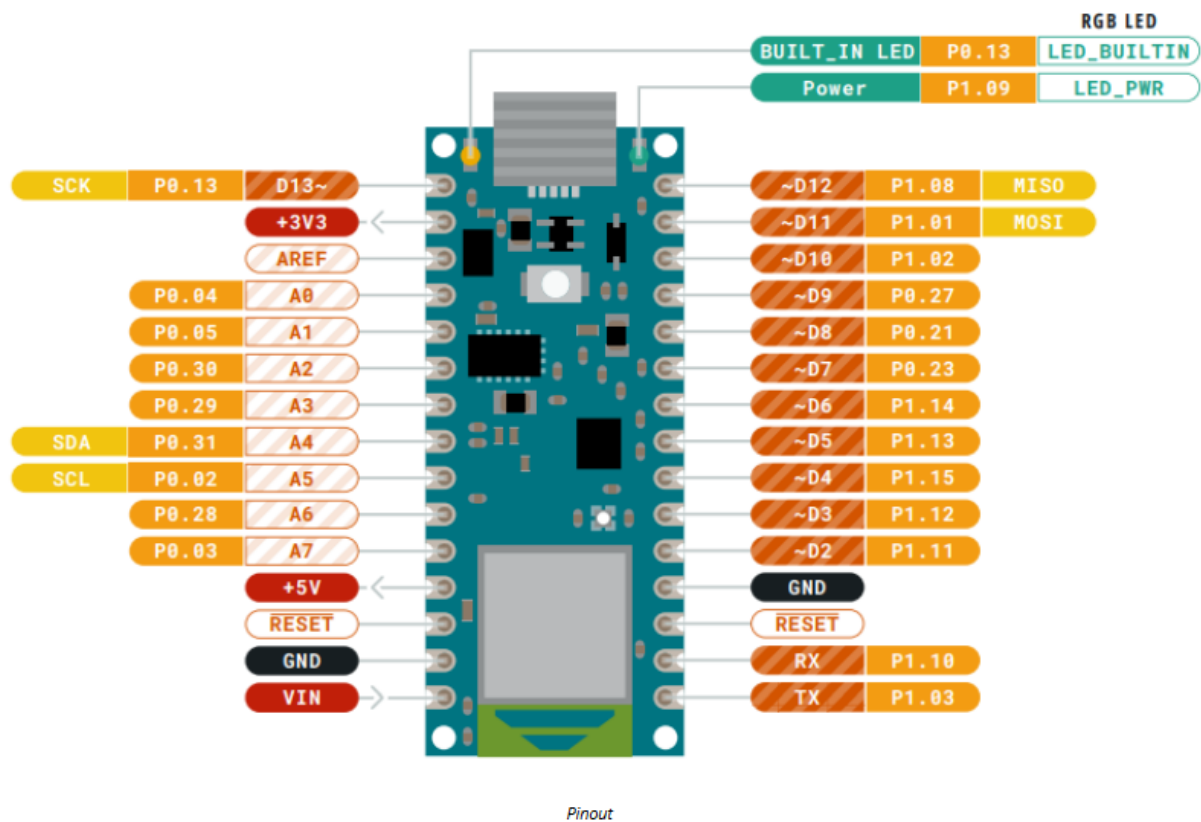


Diagrama de Pines 2.3: Obtenido de [1]

Características eléctricas

1.2 Ratings

1.2.1 Recommended Operating Conditions

Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C (40 °F)	85°C (185 °F)

1.3 Power Consumption

Symbol	Description	Min	Typ	Max	Unit
PBL	Power consumption with busy loop		TBC		mW
PLP	Power consumption in low power mode		TBC		mW
PMAX	Maximum Power Consumption		TBC		mW

2 Functional Overview

Características eléctricas 2.4: Obtenido de [1]

2.2. Componentes Electrónicos Complementarios

Para este laboratorio no se usaron componentes electrónicos complementarios

2.3. Diseño del circuito

NA

Esquemático

NA

2.4. Temas de laboratorio

Aprendizaje automático

El aprendizaje automático es un subconjunto de la inteligencia artificial que se centra en el desarrollo e implementación de algoritmos que permiten a los sistemas informáticos aprender a partir de datos y realizar predicciones o tomar decisiones sin necesidad de ser programados

explícitamente. En su esencia, el aprendizaje automático se basa en modelos estadísticos y técnicas computacionales para identificar patrones y relaciones dentro de conjuntos de datos.

Estos modelos se entrenan utilizando grandes cantidades de datos etiquetados o no etiquetados, donde los algoritmos ajustan iterativamente sus parámetros internos para optimizar el rendimiento y minimizar los errores. El proceso implica extraer características significativas de los datos, seleccionar algoritmos apropiados y utilizar técnicas de optimización para afinar los modelos. El aprendizaje automático abarca diferentes enfoques, incluyendo el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo, cada uno diseñado para abordar tareas y dominios problemáticos específicos. [3]

Redes neuronales

Las redes neuronales son un tipo de modelo de aprendizaje automático inspirado en el funcionamiento del cerebro humano. Consisten en un conjunto interconectado de unidades llamadas neuronas artificiales o nodos, que procesan y transmiten información.

Cada neurona recibe entradas ponderadas, las procesa a través de una función de activación y produce una salida. La información fluye a través de múltiples capas de neuronas, desde la capa de entrada hasta la capa de salida, a través de conexiones llamadas pesos. Durante el entrenamiento, los pesos se ajustan para minimizar el error entre las salidas actuales y las salidas deseadas. Este proceso se logra mediante algoritmos de optimización, como el descenso del gradiente, que actualizan los pesos de las conexiones en función de la magnitud del error.

Las redes neuronales pueden tener diferentes arquitecturas, como redes neuronales convolucionales para el procesamiento de imágenes o redes neuronales recurrentes para el procesamiento de secuencias. [3]

TensorFlow y TensorFlow Lite

TensorFlow es una biblioteca de software de código abierto ampliamente utilizada en el campo del aprendizaje automático. Desarrollada por Google, TensorFlow proporciona un

conjunto de herramientas y funciones para construir y entrenar redes neuronales. La biblioteca se basa en el concepto de gráficos computacionales, donde los cálculos se representan como un flujo de nodos interconectados que realizan operaciones matemáticas en tensores, que son arreglos multidimensionales de datos. TensorFlow ofrece una interfaz flexible que permite crear modelos de aprendizaje automático desde cero o utilizar modelos preentrenados para realizar tareas específicas. [4]

TensorFlow Lite, por otro lado, es una versión optimizada de TensorFlow diseñada específicamente para dispositivos móviles y sistemas integrados con recursos limitados. TensorFlow Lite permite ejecutar modelos de aprendizaje automático en dispositivos con capacidades de procesamiento y memoria más limitadas, lo que permite la implementación de inferencia en tiempo real en dispositivos móviles y otros dispositivos IoT. TensorFlow Lite utiliza técnicas de optimización, como la cuantización y la compresión de modelos, para reducir el tamaño del modelo y mejorar la eficiencia computacional sin comprometer significativamente la precisión. [4]

3. Desarrollo y análisis

3.1. Programa

Descripción del programa

La aplicación en su totalidad consiste de 3 programas.

1) Obtención de datos

Este programa, realizado para la plataforma Arduino Nano Ble 33, se encarga de capturar datos del usuario y mandarlos a la PC para poder ser usados en el entrenamiento de una red neuronal en la detección de tres gestos: puño, salto y rodilla.

El reto principal para este programa era capturar los datos de cada gesto dentro de una ventana similar entre sí para poder facilitar el entrenamiento de la red más adelante.

Para solucionar este reto, se utilizaron dos arreglos y un valor de activación según intensidad de actividad. La funcionalidad del programa es la siguiente:

Se lee continuamente los datos del giroscopio y se guardan en un buffer que puede almacenar hasta 20 datos. Estos datos se sobrescriben constantemente mientras no llegue un valor lo suficientemente alto para sobrepasar el valor de activación. La función de este primer arreglo es poder tener acceso a varios datos antes de la llegada del primer dato real del gesto. Cuando llega un valor del giroscopio que sobrepasa el valor de activación predeterminado, se almacenan en un segundo arreglo los siguientes 120 datos. Por último, los datos que se mandan a la PC son aquellos que son conformados por la concatenación del primer y segundo arreglo, y son guardados en archivos con formato .npy, que son arreglos numpy.

De esta forma, se asegura que los datos para cada gesto van a estar ajustados de forma similar dentro de una ventana de igual tamaño para todas las muestras.

2) Entrenamiento

Este programa fue realizado para correr en la PC con la ayuda de una gpu NVIDIA GeForce GTX 1650 Super, usando la biblioteca de código abierto TensorFlow y el manejador de ambientes Anaconda.

El reto principal para este programa era elegir una configuración para la red neuronal que fuese efectiva y que pudiese ser exportada a un modelo de TensorFlow Lite. Este paso fue resuelto a prueba y error.

Lo primero que hace el programa es cargar los datos de entrenamiento y colocarlos en arreglos de numpy. Estos estaban conformados por 300 muestras en total, 100 para cada gesto. Luego, se cambia el orden de los datos de forma aleatoria para hacer el entrenamiento más efectivo. Antes de entrenar la red, se separan los datos totales en 3 partes. El 60 % de los datos se usan para entrenar, un 20 % se usa para validar y el resto para pruebas.

Seguidamente, se configura la red neuronal. En este caso se utilizó una red simple de 3 capas, una de entrada, una oculta y otra de salida. La capa de entrada tiene 50 neuronas, la oculta tiene 15 y la de salida tiene 3 (una para cada gesto). Se usó el optimizador Adam, la métrica de pérdida **mae** y una tasa de aprendizaje de 0.0001. Adicionalmente, se entrenó la red por 150 epochs. El tamaño del modelo terminó siendo de 172 kB.

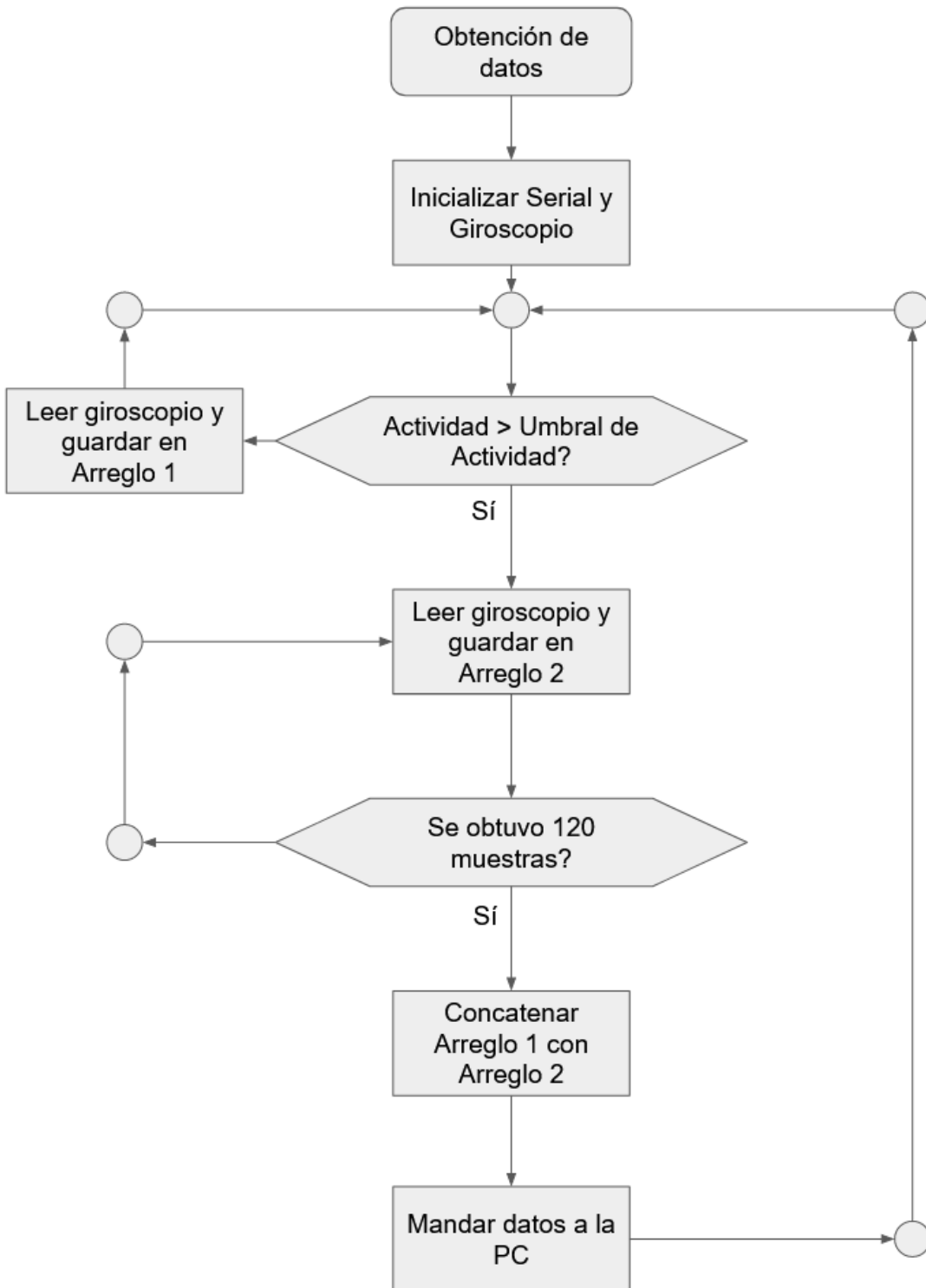
Después de entrenar la red, se exporta el modelo usando TensorFlow Lite y se guarda en

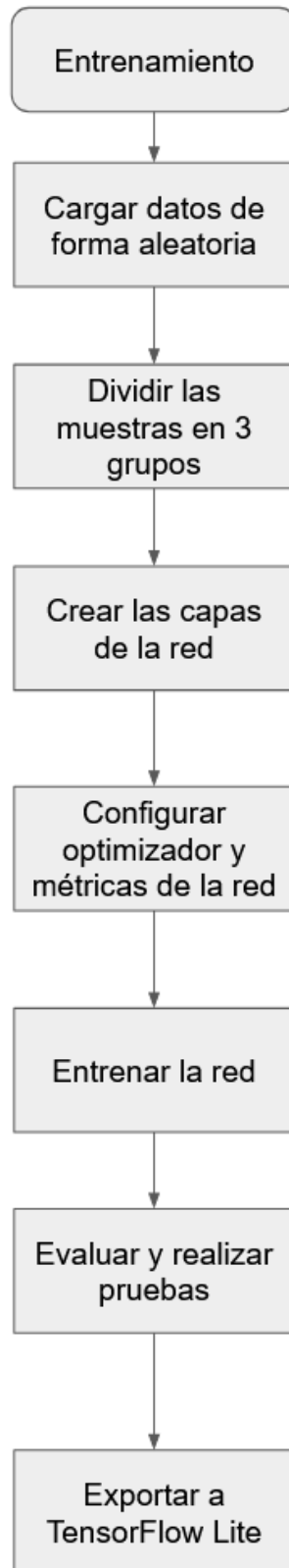
un archivo .h para ser cargado en el Arduino Nano Ble 33.

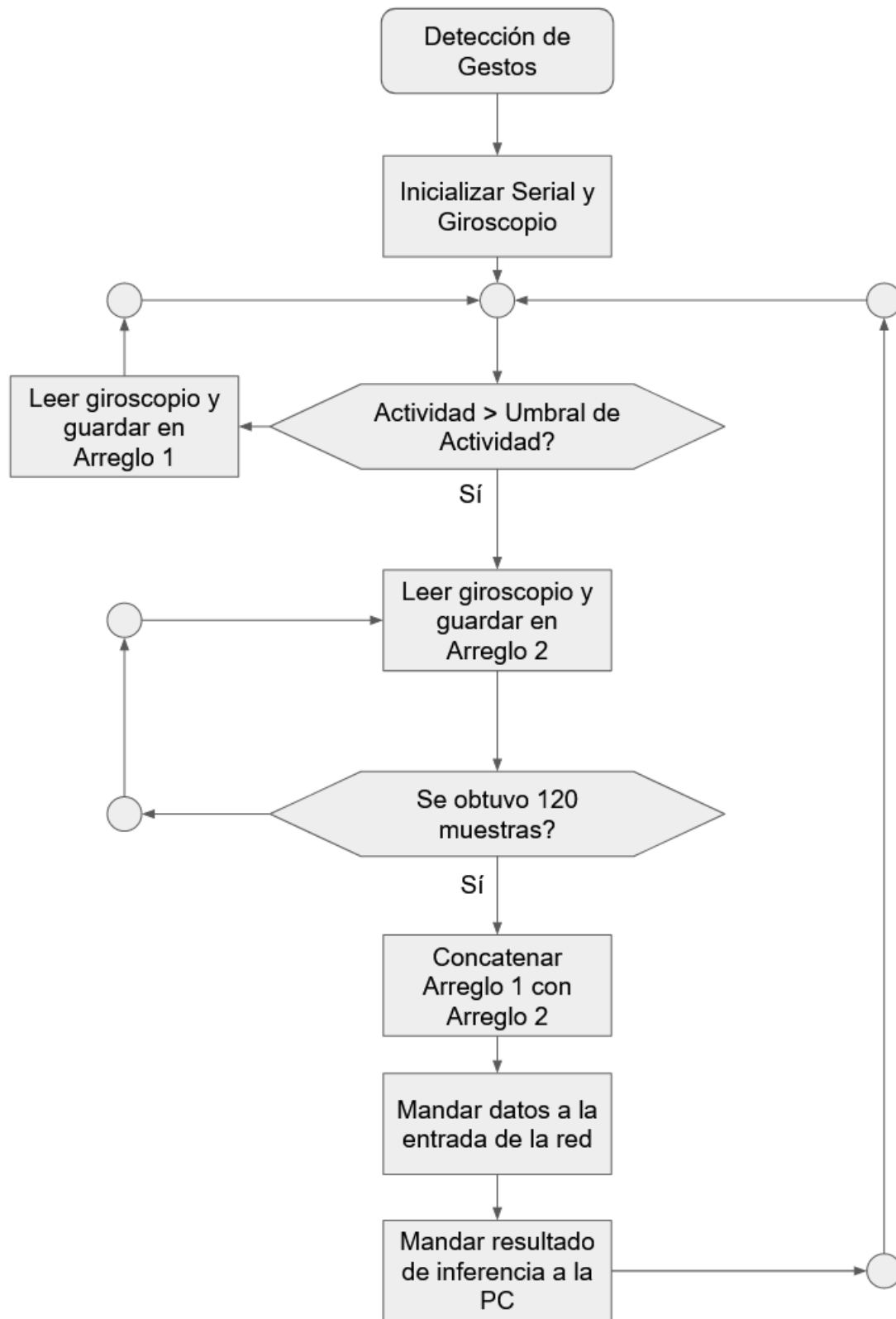
3) Detección de gestos

Este programa es muy similar al de obtención de datos. Pero una vez obtenidos los 2 arreglos, en vez de mandarlos a la PC, se mandan como entrada a la red neuronal y se manda a la PC el resultado de la inferencia.

Diagrama de bloques



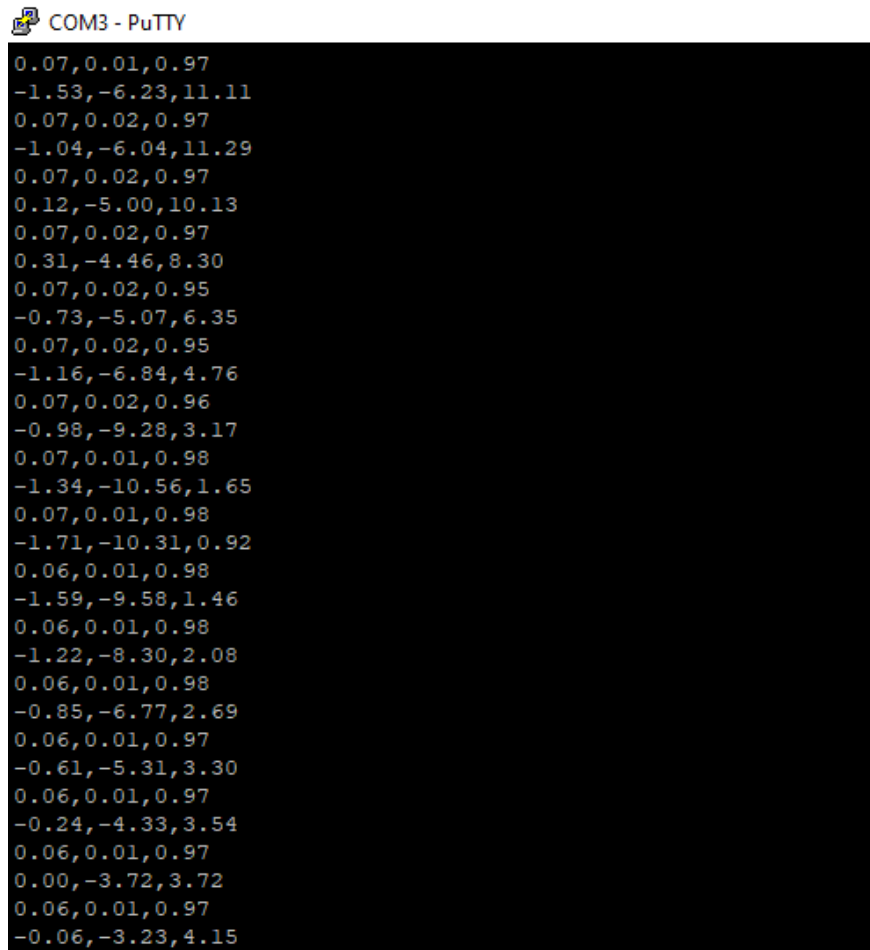




3.2. Funcionamiento

1) Obtener datos

La siguiente imagen muestra los datos seriales del giroscopio mandados a la PC tras detectar actividad significativa.



```
COM3 - PuTTY
0.07,0.01,0.97
-1.53,-6.23,11.11
0.07,0.02,0.97
-1.04,-6.04,11.29
0.07,0.02,0.97
0.12,-5.00,10.13
0.07,0.02,0.97
0.31,-4.46,8.30
0.07,0.02,0.95
-0.73,-5.07,6.35
0.07,0.02,0.95
-1.16,-6.84,4.76
0.07,0.02,0.96
-0.98,-9.28,3.17
0.07,0.01,0.98
-1.34,-10.56,1.65
0.07,0.01,0.98
-1.71,-10.31,0.92
0.06,0.01,0.98
-1.59,-9.58,1.46
0.06,0.01,0.98
-1.22,-8.30,2.08
0.06,0.01,0.98
-0.85,-6.77,2.69
0.06,0.01,0.97
-0.61,-5.31,3.30
0.06,0.01,0.97
-0.24,-4.33,3.54
0.06,0.01,0.97
0.00,-3.72,3.72
0.06,0.01,0.97
-0.06,-3.23,4.15
```

Obtener Datos 3.4: Obtener Datos

El siguiente video es una demostración:

<https://www.youtube.com/watch?v=DnDSBX10E0o>

2) Entrenar

La siguiente imagen muestra el entrenamiento de la red y los resultados.

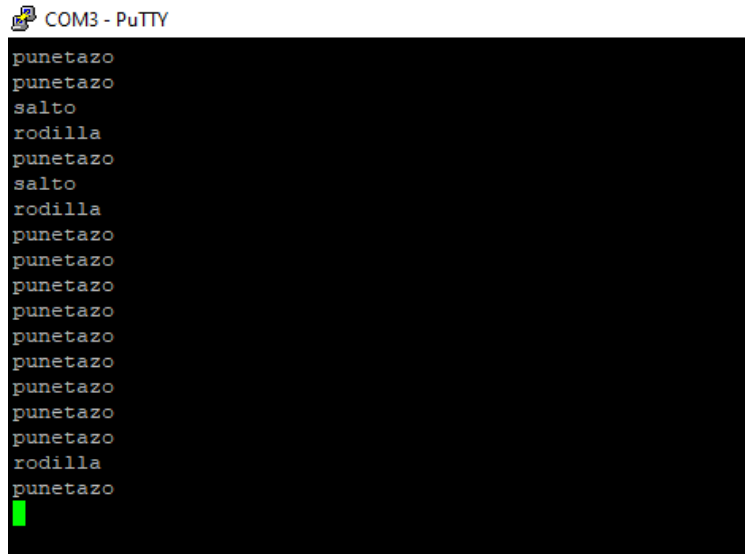
```
Anaconda Prompt
Epoch 132/150
180/180 [=====] - 0s 2ms/step - loss: 0.0888 - accuracy: 0.9889
Epoch 133/150
180/180 [=====] - 0s 2ms/step - loss: 0.0876 - accuracy: 0.9889
Epoch 134/150
180/180 [=====] - 0s 2ms/step - loss: 0.0815 - accuracy: 0.9889
Epoch 135/150
180/180 [=====] - 0s 2ms/step - loss: 0.0818 - accuracy: 0.9833
Epoch 136/150
180/180 [=====] - 0s 2ms/step - loss: 0.0717 - accuracy: 0.9889
Epoch 137/150
180/180 [=====] - 0s 2ms/step - loss: 0.0735 - accuracy: 0.9889
Epoch 138/150
180/180 [=====] - 0s 2ms/step - loss: 0.0698 - accuracy: 0.9944
Epoch 139/150
180/180 [=====] - 0s 2ms/step - loss: 0.0773 - accuracy: 0.9889
Epoch 140/150
180/180 [=====] - 0s 2ms/step - loss: 0.0727 - accuracy: 0.9889
Epoch 141/150
180/180 [=====] - 0s 2ms/step - loss: 0.0649 - accuracy: 0.9889
Epoch 142/150
180/180 [=====] - 0s 2ms/step - loss: 0.0724 - accuracy: 0.9889
Epoch 143/150
180/180 [=====] - 0s 2ms/step - loss: 0.0680 - accuracy: 0.9889
Epoch 144/150
180/180 [=====] - 0s 2ms/step - loss: 0.0764 - accuracy: 0.9889
Epoch 145/150
180/180 [=====] - 0s 2ms/step - loss: 0.0577 - accuracy: 0.9889
Epoch 146/150
180/180 [=====] - 0s 2ms/step - loss: 0.0600 - accuracy: 0.9889
Epoch 147/150
180/180 [=====] - 0s 2ms/step - loss: 0.0578 - accuracy: 0.9944
Epoch 148/150
180/180 [=====] - 0s 2ms/step - loss: 0.0615 - accuracy: 0.9889
Epoch 149/150
180/180 [=====] - 0s 2ms/step - loss: 0.0564 - accuracy: 0.9944
Epoch 150/150
180/180 [=====] - 0s 2ms/step - loss: 0.0590 - accuracy: 0.9889
60/60 [=====] - 0s 2ms/step - loss: 0.1364 - accuracy: 0.9500
60/60 [=====] - 0s 2ms/step - loss: 0.1149 - accuracy: 0.9833
```

Entrenar 3.5: Entrenar

Se observa en las últimas 2 líneas, que en las muestras de prueba y validación se obtuvo un 95 y 98 % de precisión respectivamente. Esto demuestra que la red fue entrenada efectivamente.

3) Clasificación automática

La siguiente imagen muestra los datos seriales de la inferencia a la PC tras realizar la detección automático de gestos.



```
COM3 - PuTTY
punetazo
punetazo
salto
rodilla
punetazo
salto
rodilla
punetazo
punetazo
punetazo
punetazo
punetazo
punetazo
punetazo
punetazo
punetazo
rodilla
punetazo
```

Clasificación 3.6: Clasificación

El siguiente video es una demostración:

<https://www.youtube.com/watch?v=yM8YHjF9WPw>

4. Conclusiones

Se concluye que la solución propuesta para este quinto laboratorio cumple con los requerimientos del enunciado y se comporta según el diseño teórico realizado.

4.1. Recomendaciones

Una recomendación para este laboratorio es implementar un algoritmo más avanzado que permita que la detección de gestos funcione independientemente de la forma en la que se coloque el dispositivo en la cadera. Sin esto, la detección correcta de gestos depende mucho

de si el dispositivo se colocó muy similarmente a la forma en la que se colocó a la hora de entrenar la red neuronal.

Bibliografía

- [1] “Nano ble33 data sheet.” <https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf>, 2007. (Accessed on 25/03/2023).
- [2] “Cortex m4f data sheet.” <https://developer.arm.com/documentation/102832/latest/>, 2007. (Accessed on 25/03/2023).
- [3] “Machine learning.” https://en.wikipedia.org/wiki/Machine_learning, 2007. (Accessed on 25/03/2023).
- [4] “Tensor flow.” <https://www.tensorflow.org/?hl=es-419>, 2007. (Accessed on 25/03/2023).

5. GIT

https://github.com/JAR1224/Laboratorio_5_Jose_Ramos