



**Reconocimiento de gestos,
Implementación de redes
neuronales en ESP32 y sensor
MPU-6050**

**COSENTINO FACUNDO E.
ZELAYA SERGIO S.**

UNIVERSIDAD TECNOLÓGICA NACIONAL
INGENIERÍA ELECTRÓNICA

Enviado a Universidad Tecnológica Nacional, Facultad
Regional Tucuman (FRT) en forma de cumplimiento de los
requisitos para la obtención del título de
Ingeniero en Electrónica

Director de Proyecto: **ING. OVEJERO CESAR**
Comité Examinador: **ING. EGEA RUBEN,**
ING. POLI FABIO, ING. GALVEZ OSCAR

Dedicatoria

Quiero dedicar mi primera tesis de grado a las personas que han sido fundamentales en mi vida y que me han ayudado a llegar hasta aquí. A mi familia, quienes me han brindado su apoyo incondicional. A mi abuelo, quien no está físicamente presente, pero cuyo legado y enseñanzas me han marcado para siempre, le dedico este trabajo con todo mi amor y gratitud. Finalmente, quiero mencionar a Messi, un ejemplo de perseverancia y dedicación en el fútbol, quien me ha enseñado que para alcanzar nuestros sueños hay que trabajar duro y nunca rendirse

Facu

Dedico esta tesis a mis padres, hermanos, a mis abuelos que los llevo en mi corazón y a mis amigos y compañeros; por su amor incondicional, apoyo y paciencia durante este camino académico. Su confianza en mí siempre me impulsó a alcanzar mis metas y a nunca rendirme. Gracias por ser mi fuente de inspiración y motivación en cada paso que he dado. Este logro es también suyo, ya que sin su presencia en mi vida, no habría sido posible.

Sergio

Agradecimiento

Queremos expresar nuestro más profundo agradecimiento a nuestras familias, quienes siempre nos brindaron su apoyo incondicional tanto económico como emocional, lo cual fue fundamental para alcanzar la culminación de este proyecto.

Asimismo, agradecemos a nuestra alma máter por brindarnos la oportunidad de cursar la carrera de Ingeniería en Electrónica, la cual nos permitió crecer tanto personal como profesionalmente en el transcurso de los 6 años de formación académica.

De manera especial, queremos expresar nuestra gratitud al Ing. Ovejero Cesar, nuestro tutor, por su inmensa paciencia y predisposición, tan bueno como persona como profesional. Gracias a él, pudimos aprender y superarnos a lo largo de esta travesía, gracias a su ardua tarea de enseñanza y motivación constante, desde los primeros años con conceptos fundamentales en informática y programación básica hasta los años finales en materias como Técnicas Digitales III en programación de microcontroladores de 32bits.

Además, queremos agradecer al Ing. Egea Ruben, Secretario de Planeamiento del rectorado de UTN, por su invaluable colaboración en todo momento. Su constante disposición para ayudarnos en cuestiones administrativas y logísticas, así como su incansable labor para facilitarnos los laboratorios y equipos de medición, fueron esenciales para el desarrollo exitoso de este proyecto. Su generosidad y amabilidad serán siempre recordadas con gratitud por nosotros.

Agradecemos a todas las personas que de alguna u otra forma hicieron posible este logro, y esperamos poder retribuir de alguna manera en el futuro todo lo que nos brindaron.

A todos, muchas gracias.

Introducción

La implementación de redes neuronales de base radial en el microcontrolador ESP32 programado con Python y MicroPython es un proyecto que combina el uso de la inteligencia artificial y la electrónica para crear un sistema que sea capaz de reconocer la posición de la mano en función de los vectores gravitacionales medidos por un sensor MPU6050.

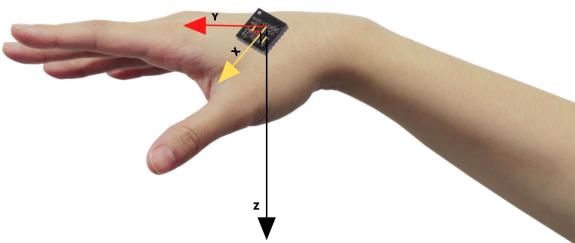


Figura 1: Posición de Reposo

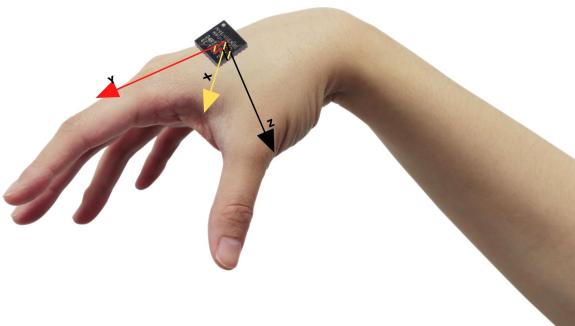


Figura 2: Posición inclinada

Para comprender un poco más sobre los conceptos involucrados, podemos definir brevemente algunos de ellos. En primer lugar, las redes neuronales son sistemas computacionales que se basan en el funcionamiento del cerebro humano y que se utilizan para resolver problemas de clasificación, predicción y control en diversas áreas. Las redes neuronales de base radial son un tipo de red neuronal que utiliza funciones radiales para determinar el grado de similitud entre la entrada y los patrones de entrenamiento.

En este proyecto en particular, el sensor MPU6050 es un dispositivo que combina un acelerómetro y un giroscopio y se utiliza para medir la aceleración de un objeto en tres ejes. Estos datos son procesados por la red neuronal de base radial para determinar la posición de la mano.

Aplicaciones potenciales, algunas de las cuales son:

-
- Interfaz de usuario sin contacto: Con el sistema de reconocimiento de la posición de la mano, es posible crear una interfaz de usuario sin contacto para dispositivos electrónicos, como smartphones, tablets o computadoras. Esto permitiría a los usuarios interactuar con los dispositivos de forma más natural y sin la necesidad de tocar la pantalla.
 - Control de juegos: Este proyecto podría utilizarse para crear una forma de controlar juegos de computadora o consolas mediante gestos. Los jugadores podrían controlar sus personajes y acciones del juego con movimientos naturales de la mano.
 - Rehabilitación: El sistema de reconocimiento de posición de la mano también podría utilizarse en la rehabilitación física de pacientes que han sufrido una lesión en las manos o brazos. Los terapeutas podrían utilizar el sistema para monitorizar el progreso de sus pacientes y ajustar los ejercicios y tratamientos en consecuencia.
 - Automatización industrial: En la industria, este proyecto podría utilizarse para controlar robots industriales mediante gestos, permitiendo una mayor precisión y seguridad en la ejecución de tareas.

En resumen, este proyecto combina el uso de la inteligencia artificial y la electrónica para crear un sistema que es capaz de reconocer la posición de la mano en función de los vectores gravitacionales medidos por un sensor MPU6050. La implementación se realiza en un microcontrolador ESP32 programado con Python y MicroPython, lo que lo convierte en un proyecto interesante y útil en diversas aplicaciones.

Índice general

| | |
|---|-----------|
| Indice de Figuras | 8 |
| Indice de Algoritmos | 11 |
| Abreviaciones | 12 |
| 1. Fundamentacion | 13 |
| 2. Redes Neuronales | 14 |
| 2.1. Generalidades | 14 |
| 2.1.1. Reseña Histórica | 14 |
| 2.1.2. De la neurona biológica a la artificial. | 16 |
| 2.1.3. Definiciones rígurasas de una red neuronal. | 17 |
| 2.2. Ventajas que ofrecen las redes neuronales. | 18 |
| 2.3. Redes neuronales y computadoras digitales. | 18 |
| 2.4. Arquitectura de una red Neuronal | 19 |
| 2.5. Estructura de una red Neuronal | 21 |
| 2.6. Entrenamiento de redes neuronales | 22 |
| 2.7. Redes de función radial (RBF) | 23 |
| 2.7.1. Activación de las neuronas de la red de base radial. | 23 |
| 2.7.2. Aprendizaje de las redes neuronales de base radial | 25 |
| 3. Sensor MPU6050 | 27 |
| 3.1. ¿Porqué elegimos el sensor MPU6050? | 27 |
| 3.1.1. Especificaciones del MPU | 28 |
| 3.2. Funcionamiento y Pin Out | 29 |
| 3.2.1. Funcionamiento del MPU | 29 |
| 3.2.2. Características generales del MPU | 31 |
| 3.2.3. Pin Out del MPU | 32 |
| 3.3. Comunicación con el MPU | 32 |
| 3.3.1. Definición y conceptos | 32 |
| 3.3.2. Protocolo I2C | 33 |
| 3.4. Pruebas MPU6050 | 37 |
| 3.5. Calibracion | 38 |
| 3.6. Curva de Aceleración | 42 |
| 3.6.1. Read and Send data: Recepción de datos MPU6050 y transmisión de datos a la PC | 42 |
| 3.6.2. Plot Python: Recepcion de los datos desde ESP32 para graficas | 44 |
| 3.6.3. Pruebas | 47 |

| | |
|---|------------|
| 4. Pantalla OLED | 50 |
| 4.1. Tecnologías OLED | 50 |
| 4.1.1. Controlador SSD1306 | 52 |
| 4.1.2. Visualización de gráficos en pantalla OLED | 53 |
| 4.1.3. Pineado módulo pantalla OLED | 54 |
| 4.1.4. Especificaciones Display Oled 128x32 SSD1306 | 54 |
| 4.2. Pruebas de pantalla SSD-1306 | 55 |
| 4.3. Logo en pantalla | 56 |
| 4.4. Implementación MPU6050 y SSD1306 | 59 |
| 5. Microcontrolador ESP32-wroom | 63 |
| 5.1. Descripción y Características | 63 |
| 5.1.1. Wifi y Bluetooth en un solo chip | 63 |
| 5.1.2. Especificaciones | 64 |
| 5.2. Pin Out y conexiones básicas | 65 |
| 5.2.1. Pin Out ESP32wroom | 65 |
| 5.2.2. Conexionado básico | 65 |
| 5.3. Diagrama circuital completo | 66 |
| 5.3.1. Partición y secciones del circuito | 66 |
| 5.3.2. Diagrama Final | 68 |
| 5.3.3. Diseño de PCB | 69 |
| 5.3.4. Armado y montaje final | 70 |
| 6. Adquisición de los datos | 72 |
| 6.1. Data Set | 72 |
| 6.1.1. Definición | 72 |
| 6.1.2. Tipos de Datasets | 72 |
| 6.2. Creación del conjunto de datos | 73 |
| 6.2.1. ¿Por qué necesito un conjunto de datos? | 73 |
| 6.2.2. Procesamiento de Datos | 73 |
| 6.2.3. Objetivo a largo plazo: La estrategia perfecta | 75 |
| 6.3. Normalizar en Data Science | 75 |
| 6.3.1. Métodos más usados | 76 |
| 6.4. Lectura y Envio de datos | 79 |
| 7. Practica de Redes Neuronales | 91 |
| 7.1. Muestreo de datos para las distintas posiciones de la mano | 91 |
| 7.2. Implementación en el ESP32 | 103 |
| 7.3. Medición de tasa de acierto | 105 |
| 8. Metodología y planificación | 108 |
| Apendice | 110 |

Índice de figuras

| | |
|---|----|
| 1. Posición de Reposo | 4 |
| 2. Posición inclinada | 4 |
| 2.1. Neuronal biológica | 16 |
| 2.2. Neuronal artificial | 17 |
| 2.3. Red Neuronal de una Capa | 19 |
| 2.4. Red Neuronal de dos Capas | 20 |
| 2.5. Esquema básico de una red neuronal | 21 |
| 2.6. Red Neuronal de Base Radial. Arquitectura | 24 |
| 2.7. Arquitectura de la red neuronal | 26 |
| 3.1. Ejes del MPU | 27 |
| 3.2. Pin Out del MPU | 32 |
| 3.3. Esquema básico del I2C | 33 |
| 3.4. Condición inicial del bus | 34 |
| 3.5. Comunicación del bus | 34 |
| 3.6. Condición de parada del bus | 34 |
| 3.7. Secuencia del registro del ángulo para el módulo de brújula CMPS03 | 36 |
| 3.8. Prueba de Sensor MPU-6050 | 37 |
| 3.9. Librerías en ESP32 | 37 |
| 3.10. Salida del algoritmo 3.1 | 38 |
| 3.11. Valor Offset en Z | 41 |
| 3.12. Valor Offset en X | 41 |
| 3.13. Valor Offset en Y | 41 |
| 3.14. Calibración Completa | 42 |
| 3.15. Diagrama en flujo de 'Read and Send data' | 44 |
| 3.16. Diagrama en flujo de 'Plot Python' | 47 |
| 3.17. Run 'Read and Send data' | 48 |
| 3.18. Run 'Plot Python' | 48 |
| 3.19. Curva con posición Z=1 X=0 Y=0 | 48 |
| 3.20. Curva con posición Z=0 X=1 Y=0 | 48 |
| 3.21. Curva con posición Z=0 X=0 Y=1 | 49 |
| 3.22. Curva con varias posiciones | 49 |
| 4.1. Estructura pantalla OLED | 51 |
| 4.2. Proceso emisión de luz | 52 |
| 4.3. Matriz de la pantalla OLED | 53 |
| 4.4. Pines pantalla OLED | 54 |
| 4.5. Imagen UTN | 54 |
| 4.6. Esquematico ESP32 y Pantalla OLED | 55 |
| 4.7. Librerías en ESP32 | 55 |

| | |
|---|-----|
| 4.8. Print en pantalla OLED | 56 |
| 4.9. Imagen a convertir | 56 |
| 4.10. Salida de imagen UTN | 59 |
| 4.11. Circuito ESP32 MPU6050 SSD1306 | 59 |
| 4.12. Salida del algoritmo 4.2 | 62 |
| 5.1. Pin Out ESP32 wroom | 65 |
| 5.2. Configuración básica del ESP32 wroom | 65 |
| 5.3. Conexionado de la fuente de alimentación | 66 |
| 5.4. Conexionado para las comunicaciones USB-UART | 66 |
| 5.5. Configuración usada para conectar el ESP32wroom | 67 |
| 5.6. Periféricos usados en el circuito final | 67 |
| 5.7. Circuito final implementado con ESP32wroom | 68 |
| 5.8. PCB vista de cara inferior | 69 |
| 5.9. PCB vista de cara superior | 69 |
| 5.10. Vista 3D del proyecto terminado con los componentes montados | 70 |
| 5.11. Plaqueta física terminada preparada para la toma de datos con conexión a PC | 70 |
| 5.12. Plaqueta lista y entrenada para su funcionamiento | 71 |
| 5.13. Adición de guante para posterior prueba de funcionamiento | 71 |
| 5.14. Trabajo final terminado y funcionando correctamente . . | 71 |
| 6.1. Proceso del Data Set | 74 |
| 6.2. Procesamiento de Datos | 74 |
| 6.3. El proceso ideal para nuestro proyecto | 75 |
| 6.4. A modo de exemplificar, se toman distintos valores próximos a 30, considerando un período de 50 partes (horas, días, etc). | 76 |
| 6.5. Misma gráfica pero tomando como referencia los máximos y los mínimos en el rango de 0 a 1. | 77 |
| 6.6. Ejemplo de gráfica tomando muestras alejadas a la media. | 77 |
| 6.7. La normalización estándar para los datos anteriores no es una buena elección | 78 |
| 6.8. Diagrama en flujo de Python-Made-DataSet | 79 |
| 6.9. 1er parte Diagrama en flujo de Read and Save data . . | 82 |
| 6.10. 2da parte Diagrama en flujo de Read and Save data . . | 83 |
| 6.11. Posición 1 | 89 |
| 6.12. Posición 2 | 89 |
| 6.13. Posición 3 | 89 |
| 6.14. Posición 4 | 90 |
| 6.15. Posición 5 | 90 |
| 6.16. Gráfica de los 5 grupos de datos tomados para cada posición de la mano, tener en cuenta las referencias para cada uno | 90 |
| 7.1. Primer muestreo: Mano horizontal | 93 |
| 7.2. Segundo muestreo: Mano hacia derecha | 96 |
| 7.3. Tercer muestreo: Mano hacia izquierda | 98 |
| 7.4. Cuarto muestreo: Mano hacia abajo | 101 |
| 7.5. Quinto muestreo: Mano hacia abajo | 103 |

| | |
|---|-----|
| 7.6. Gráfico correspondiente a la tasa de acierto | 107 |
| 8.1. Cronograma | 109 |

Algoritmos

| | | |
|------|--|-----|
| 3.1. | Test-esp32-mpu6050 | 37 |
| 3.2. | Calibracion de MPU6050 | 38 |
| 3.3. | Read and Send data | 42 |
| 3.4. | Plot Python | 44 |
| 4.1. | Test-esp32(ssd1306 | 55 |
| 4.2. | Implementacion ssd1306 y mpu6050 | 59 |
| 6.1. | Python-Made-Dataset | 80 |
| 6.2. | Read and Save Database | 83 |
| 7.1. | Muestreo 1 | 91 |
| 7.2. | Muestreo 2 | 94 |
| 7.3. | Muestreo 3 | 96 |
| 7.4. | Muestreo 4 | 99 |
| 7.5. | Muestreo 5 | 101 |
| 7.6. | Código final del micro | 103 |
| 7.7. | Tasa de acierto | 106 |

Abreviaciones

| Abreviación | Expansión |
|-------------|------------------------------------|
| ML | Machine Learning |
| AI | Artificial Intelligence |
| ANN | Artificial Neural Network |
| RBF | Radial Base Función |
| RBFNN | Radial Base Función Neural Network |

Capítulo 1

Fundamentacion

La implementación de redes neuronales de base radial en microcontroladores es una técnica que se ha utilizado con éxito en diversas aplicaciones prácticas. Esta técnica combina la inteligencia artificial con la electrónica para crear sistemas capaces de resolver problemas de clasificación, predicción y control en diversas áreas. En el contexto de este proyecto, se busca utilizar redes neuronales de base radial en un microcontrolador ESP32 programado con Python y MicroPython para reconocer la posición de la mano en función de los vectores gravitacionales medidos por un sensor MPU6050.

Las redes neuronales de base radial son un tipo de red neuronal que utiliza funciones radiales para determinar el grado de similitud entre la entrada y los patrones de entrenamiento. En este proyecto en particular, la utilización de redes neuronales de base radial permite determinar la posición de la mano con precisión y eficiencia. Esto es útil en diversas aplicaciones prácticas como el control de robots y sistemas de seguridad.

El uso de microcontroladores como el ESP32 programado con Python y MicroPython permite implementar soluciones compactas y eficientes en términos de recursos, lo que resulta útil en aplicaciones donde se requiere un procesamiento de datos en tiempo real. En este proyecto, se utiliza el sensor MPU6050, que es un dispositivo para medir la aceleración de un objeto en tres ejes. La utilización de este sensor permite medir los vectores gravitacionales y procesarlos mediante la red neuronal de base radial para determinar la posición de la mano.

En resumen, la implementación de redes neuronales de base radial en microcontroladores para reconocer la posición de la mano en función de los vectores gravitacionales medidos por un sensor MPU6050 es una técnica eficiente y precisa que permite resolver problemas de clasificación y predicción en diversas aplicaciones prácticas. La utilización de microcontroladores como el ESP32 programado con Python y MicroPython permite implementar soluciones compactas y eficientes en términos de recursos, lo que resulta útil en aplicaciones donde se requiere un procesamiento de datos en tiempo real.

Capítulo 2

Redes Neuronales

2.1. Generalidades

Las redes neuronales son más que otra forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos. Si se examinan con atención aquellos problemas que no pueden expresarse a través de un algoritmo, se observará que todos ellos tienen una característica en común: la experiencia. El hombre es capaz de resolver estas situaciones acudiendo a la experiencia acumulada. Así, parece claro que una forma de aproximarse al problema consiste en la construcción de sistemas que sean capaces de reproducir esta característica humana. En definitiva, las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es “un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona”.

2.1.1. Reseña Histórica

1936 - Alan Turing. Fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas (Un Cálculo Lógico de la Inminente Idea de la Actividad Nerviosa - Boletín de Matemática Biofísica 5: 115-133). Ellos modelaron una red neuronal simple mediante circuitos eléctricos.

1949 - Donald Hebb. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Aun hoy, este es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal. Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También intentó encontrar semejanzas entre el aprendizaje y la actividad nerviosa. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

1950 - Karl Lashley. En sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida encima de él.

1956 - Congreso de Dartmouth. Este Congreso frecuentemente se menciona para indicar el nacimiento de la inteligencia artificial.

1957 - Frank Rosenblatt. Comenzó el desarrollo del Perceptron. Esta es la red neuronal más antigua; utilizándose hoy en día para aplicación como identificador de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado en el entrenamiento. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente.

1959 - Frank Rosenblatt: Principios de Neurodinámica. En este libro confirmó que, bajo ciertas condiciones, el aprendizaje del Perceptron converge hacia un estado finito (Teorema de Convergencia del Perceptron).

1960 - Bernard Widroff/Marcian Hoff. Desarrollaron el modelo Adaline (ADAptive LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.

1961 - Karl Steinbeck: Die Lernmatrix. Red neuronal para simples realizaciones técnicas (memoria asociativa).

1969 - Marvin Minsky/Seymour Papert. En este año casi se produjo la muerte abrupta de las Redes Neuronales; ya que Minsky y Papert probaron (matemáticamente) que el Perceptrón no era capaz de resolver problemas relativamente fáciles, tales como 7 el aprendizaje de una función no-lineal. Esto demostró que el Perceptron era muy débil, dado que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real.

1974 - Paul Werbos. Desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (backpropagation); cuyo significado quedó definitivamente aclarado en 1985.

1977 - Stephen Grossberg: Teoría de Resonancia Adaptada (TRA). La Teoría de Resonancia Adaptada es una arquitectura de red que se diferencia de todas las demás previamente inventadas. La misma simula otras habilidades del cerebro: memoria a largo y corto plazo.

1985 - John Hopfield. Provocó el renacimiento de las redes neuronales con su libro: Computación neuronal de decisiones en problemas de optimización.

1986 - David Rumelhart/G. Hinton. Redescubrieron el algoritmo de aprendizaje de propagación hacia atrás (backpropagation).

A partir de 1986, el panorama fue alentador con respecto a las investigaciones y el desarrollo de las redes neuronales. En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen (sobretodo en el área de control) y las empresas que lanzan al mercado productos nuevos, tanto hardware como software (sobre todo para simulación).

2.1.2. De la neurona biológica a la artificial.

Neuronal Biológica

Las diferentes configuraciones y algoritmos que se diseñan para las redes neuronales artificiales están inspiradas en la organización del complejo sistema neuronal del cerebro humano. No obstante conviene aclarar que esta inspiración no supone que las ANN lleguen a emular al cerebro como algunos optimistas lo desean ya que entre otras limitaciones el conocimiento sobre el modo de funcionamiento y comportamiento del cerebro es bastante simple y reducido. De hecho los diseñadores de redes artificiales van más lejos del conocimiento biológico actual y prueban nuevas estructuras que presentan un comportamiento adecuado y útil. El sistema nervioso humano constituido por células llamadas neuronas presenta una estructura muy compleja. El número estimado de neuronas es de 10^{11} y las interconexiones entre ellas son del orden de 10^{15} .

Cada neurona comparte muchas características con otras células del cuerpo humano pero tiene propiedades particulares y especiales para recibir, procesar y transmitir señales electroquímicas a través de todas las interconexiones del sistema de comunicación del cerebro.

La Figura 2.1 muestra la estructura de un par de neuronas biológicas. Del cuerpo de la neurona se extienden las dendritas hacia otras neuronas donde reciben las señales transmitidas por otras neuronas. El punto de contacto o de conexión se llama sinapsis y estas entradas son dirigidas al núcleo donde se suman. Algunas de las entradas tienden a excitar a la célula y otras sin embargo tienden a inhibir la célula. Cuando la excitación acumulada supera un valor umbral, las neuronas envían una señal a través del axón a otras neuronas.

La mayoría de los modelos de las ANN presenta este funcionamiento básico de la neurona aun cuando el comportamiento real de una célula nerviosa tiene muchas complejidades y excepciones.

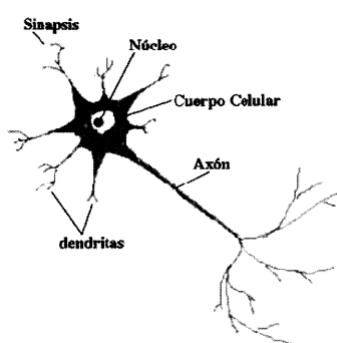


Figura 2.1: Neuronal biológica

Neuronal Artificial

La neurona artificial fue diseñada para “emular” las características del funcionamiento básico de la neurona biológica. En esencia, se aplica un conjunto de entradas a la neurona, cada una de las cuales representa una salida de otra neurona. Cada entrada se multiplica por su “peso”

o ponderación correspondiente análogo al grado de conexión de la sinapsis. Todas las entradas ponderadas se suman y se determina el nivel de excitación o activación de la neurona. Una representación vectorial del funcionamiento básico de una neurona artificial se indica según la siguiente expresión de la ecuación 2.1.

$$NET = \mathbf{X} \cdot \mathbf{W} \quad (2.1.1)$$

Normalmente la señal de salida NET suele ser procesada por una función de activación F para producir la señal de salida de la neurona OUT. La función F puede ser una función lineal, o una función umbral o una función no lineal que simula con mayor exactitud las características de transferencia no lineales de las neuronas biológicas.

La Figura 2.2 representa una neurona artificial con una función de activación F.

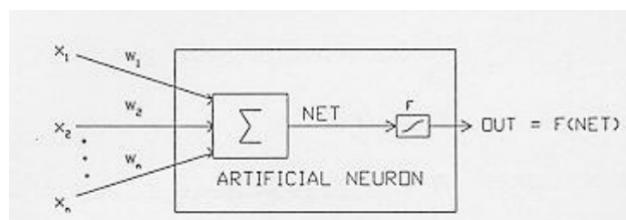


Figura 2.2: Neuronal artificial

Las funciones F más utilizadas son la función Sigmoid y Tangente hiperbólica expresadas en la siguiente tabla.

| | |
|----------------------|----------------------|
| Sigmoide | $OUT=1/(1+e^{-NET})$ |
| Tangente hiperbólica | $OUT=1/(1+e^{-NET})$ |

Este tipo de modelo de neurona artificial ignora muchas de las características de las neuronas biológicas. Entre ellas destaca la omisión de retardos y de sincronismo en la generación de la salida. No obstante, a pesar de estas limitaciones las redes construidas con este tipo de neurona artificial presentan cualidades y atributos con cierta similitud a la de los sistemas biológicos.

Todo este recorrido desde la neurona biológica a la artificial se la debemos al profesor Xabier Basogain Olabe del Dpto. de Ingeniería de Sistemas y de la Escuela Superior de Ingeniería de Bilbao, UPV-EHU.¹⁾

2.1.3. Definiciones rigurosas de una red neuronal.

Existen numerosas formas de definir a las redes neuronales; desde las definiciones cortas y genéricas hasta las que intentan explicar más detalladamente qué son las redes neuronales.

Por ejemplo:

- 1) Una nueva forma de computación, inspirada en modelos biológicos.
- 2) Un modelo matemático compuesto por un gran número de elementos

¹Link al documento

procesales organizados en niveles.

3) Un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.

4) Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

2.2. Ventajas que ofrecen las redes neuronales.

Debido a su constitución y a sus fundamentos, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas.

Entre las ventajas se incluyen:

Aprendizaje Adaptativo. Capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.

Auto-organización. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.

Tolerancia a fallos. La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.

Operación en tiempo real. Los cálculos neuronales pueden ser realizados en paralelo; para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.

Fácil inserción dentro de la tecnología existente. Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes.

2.3. Redes neuronales y computadoras digitales.

Para entender el potencial de la computación neuronal, sería necesario hacer una breve distinción entre sistemas de computación neuronales y digitales: los sistemas neurológicos no aplican principios de circuitos lógicos o digitales. Un sistema de computación digital debe ser síncrono o asíncrono. Si fuera asíncrono, la duración de los impulsos neuronales debería ser variable para mantener uno de los valores binarios por períodos de tiempo indefinido, lo cual no es el caso. Si el principio fuera síncrono, se necesitaría un reloj global o maestro con el cual los pulsos

estén sincronizados. Éste tampoco es el caso. Las neuronas no pueden ser circuitos de umbral lógico, porque hay miles de entradas variables en la mayoría de las neuronas y el umbral es variable con el tiempo, siendo afectado por la estimulación, atenuación, etc. La precisión y estabilidad de tales circuitos no es suficiente para definir ninguna función booleana. Los procesos colectivos que son importantes en computación neuronal no pueden implementarse por computación digital. Por todo ello, el cerebro debe ser un computador analógico. Ni las neuronas ni las sinapsis son elementos de memoria biestable. Todos los hechos fisiológicos hablan a favor de las acciones de las neuronas como integradores analógicos, y la eficiencia de la sinapsis cambia de forma gradual, lo cual no es característico de sistemas biestables. Los circuitos del cerebro no implementan computación recursiva y por lo tanto no son algorítmicos. Debido a los problemas de estabilidad, los circuitos neuronales no son suficientemente estables para definiciones recursivas de funciones como en computación digital. Un algoritmo, por definición, define una función recursiva.

2.4. Arquitectura de una red Neuronal

La capacidad de cálculo y potencia de la computación neuronal proviene de las múltiples conexiones de las neuronas artificiales que constituyen las redes ANN. La red más simple es un grupo de neuronas ordenadas en una capa como se muestra en la Figura 2.3. Los nodos circulares sólo son distribuidores de las entradas y no se consideran constituyentes de una capa.

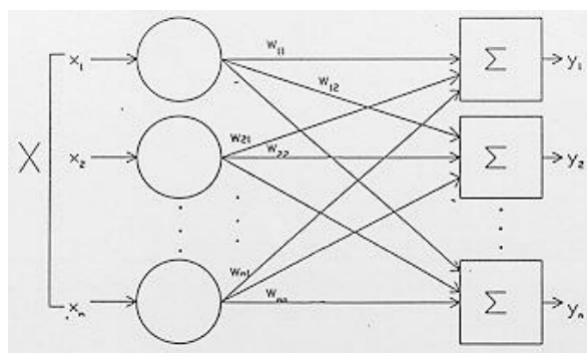


Figura 2.3: Red Neuronal de una Capa

Cada una de las entradas está conectada a través de su peso correspondiente a cada neurona artificial. En la práctica existen conexiones eliminadas e incluso conexiones entre las salidas y entradas de las neuronas de una capa. No obstante la figura muestra una conectividad total por razones de generalización. Normalmente las redes más complejas y más grandes ofrecen mejores prestaciones en el cálculo computacional que las redes simples. Las configuraciones de las redes construidas presentan aspectos muy diferentes pero tienen un aspecto común, el ordenamiento de las neuronas en capas o niveles imitando la estructura de capas que presenta el cerebro en algunas partes. Las redes multicapa se forman con un grupo de capas simples en cascada. La salida de una capa es la entrada de la siguiente capa. Se ha demostrado que las redes multicapa presentan

cualidades y aspectos por encima de las redes de una capa simple. La Figura 2.4 muestra una red de dos capas.

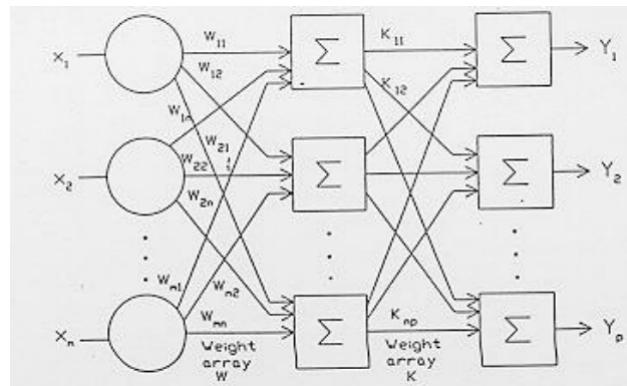


Figura 2.4: Red Neuronal de dos Capas

2.5. Estructura de una red Neuronal

A continuación, se puede ver un esquema de una red neuronal:

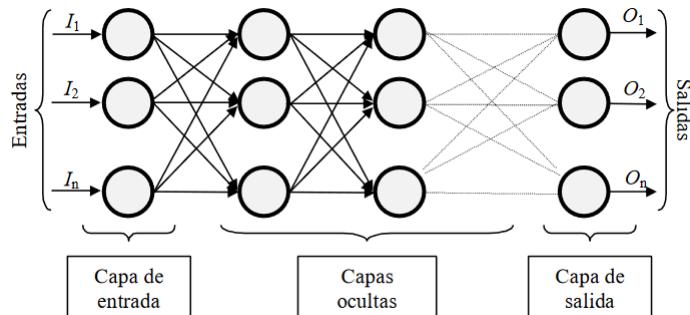


Figura 2.5: Esquema básico de una red neuronal

La misma está constituida por neuronas interconectadas y arregladas en tres capas (esto último puede variar). Los datos ingresan por medio de la capa de entrada, pasan a través de la capa oculta y salen por la capa de salida. Cabe mencionar que la capa oculta puede estar constituida por varias capas.

Función de entrada (input function)

La neurona trata a muchos valores de entrada como si fueran uno solo; esto recibe el nombre de entrada global. Por lo tanto, ahora nos enfrentamos al problema de cómo se pueden combinar estas simples entradas (in_{i1}, in_{i2}, \dots) dentro de la entrada global, gin_i . Esto se logra a través de la función de entrada, la cual se calcula a partir del vector entrada. La función de entrada puede describirse como sigue:

$$Input_i = (in_{i1}w_{i1}) * (in_{i2}w_{i2}) * \dots * (in_{in}w_{in})$$

Donde * representa al operador apropiado (por ejemplo: máximo, sumatoria, productoria, etc.), n al número de entradas a la neurona N_i y w_i al peso. Los valores de entrada se multiplican por los pesos anteriormente ingresados a la neurona. Por consiguiente, los pesos que generalmente no están restringidos cambian la medida de influencia que tienen los valores de entrada. Es decir, que permiten que un gran valor de entrada tenga solamente una pequeña influencia, si estos son lo suficientemente pequeños.

Función de activación (activation function)

Una neurona biológica puede estar activa (excitada) o inactiva (no excitada); es decir, que tiene un estado de activación. Las neuronas artificiales también tienen diferentes estados de activación; algunas de ellas solamente dos, al igual que las biológicas, pero otras pueden tomar cualquier valor dentro de un conjunto determinado. La función activación calcula el estado de actividad de una neurona; transformando la entrada global (menos el umbral, θ_i) en un valor (estado) de activación, cuyo rango normalmente va de (0 a 1) o de (1 a 1). Esto es así, porque una neurona puede estar totalmente inactiva (0 o 1) o activa (1). La función activación, es una función de la entrada global (g_{ini}) menos el umbral (θ_i).

Para explicar porque se utilizan estas funciones de activación se suele emplear la analogía a la aceleración de un automóvil. Cuando un auto inicia su movimiento necesita una potencia elevada para comenzar a acelerar. Pero al ir tomando velocidad, esta demanda un menor incremento de dicha potencia para mantener la aceleración. Al llegar a altas velocidades, nuevamente un amplio incremento en la potencia es necesario para obtener una pequeña ganancia de velocidad. En resumen, en ambos extremos del rango de aceleración de un automóvil se demanda una mayor potencia para la aceleración que en la mitad de dicho rango.

Función de salida (output function)

El último componente que una neurona necesita es la función de salida. El valor resultante de esta función es la salida de la neurona i_{out-i} ; por ende, la función de salida determina que valor se transfiere a las neuronas vinculadas. Si la función de activación está por debajo de un umbral determinado, ninguna salida se pasa a la neurona subsiguiente. Normalmente, no cualquier valor es permitido como una entrada para una neurona, por lo tanto, los valores de salida están comprendidos en el rango $[0, 1]$ o $[-1, 1]$. También pueden ser binarios 0, 1 o -1, 1.

2.6. Entrenamiento de redes neuronales

Una de las principales características de las ANN es su capacidad de aprendizaje. El entrenamiento de las ANN muestra algunos paralelismos con el desarrollo intelectual de los seres humanos. No obstante aun cuando parece que se ha conseguido entender el proceso de aprendizaje conviene ser moderado porque el aprendizaje de las ANN está limitado.

El objetivo del entrenamiento de una ANN es conseguir que una aplicación determinada, para un conjunto de entradas produzca el conjunto de salidas deseadas o mínimamente consistentes. El proceso de entrenamiento consiste en la aplicación secuencial de diferentes conjuntos o vectores de entrada para que se ajusten los pesos de las interconexiones según un procedimiento predeterminado. Durante la sesión de entrenamiento los pesos convergen gradualmente hacia los valores que hacen que cada entrada produzca el vector de salida deseado.

Los algoritmos de entrenamiento o los procedimientos de ajuste de los valores de las conexiones de las ANN se pueden clasificar en dos grupos: Supervisado y No Supervisado.

Entrenamiento Supervisado: estos algoritmos requieren el emparejamiento de cada vector de entrada con su correspondiente vector de salida. El entrenamiento consiste en presentar un vector de entrada a la red, calcular la salida de la red, compararla con la salida deseada, y el error o diferencia resultante se utiliza para realimentar la red y cambiar los pesos de acuerdo con un algoritmo que tiende a minimizar el error.

Las parejas de vectores del conjunto de entrenamiento se aplican secuencialmente y de forma cíclica. Se calcula el error y el ajuste de los pesos por cada pareja hasta que el error para el conjunto de entrenamiento entero sea un valor pequeño y aceptable.

Entrenamiento No Supervisado: los sistemas neuronales con entrenamiento supervisado han tenido éxito en muchas aplicaciones y sin

embargo tienen muchas críticas debido a que desde el punto de vista biológico no son muy lógicos. Resulta difícil creer que existe un mecanismo en el cerebro que compare las salidas deseadas con las salidas reales. En el caso de que exista, ¿de dónde provienen las salidas deseadas?

Los sistemas no supervisados son modelos de aprendizaje más lógicos en los sistemas biológicos. Desarrollados por Kohonen (1984) y otros investigadores, estos sistemas de aprendizaje no supervisado no requieren de un vector de salidas deseadas y por tanto no se realizan comparaciones entre las salidas reales y salidas esperadas. El conjunto de vectores de entrenamiento consiste únicamente en vectores de entrada. El algoritmo de entrenamiento modifica los pesos de la red de forma que produzca vectores de salida consistentes. El proceso de entrenamiento extrae las propiedades estadísticas del conjunto de vectores de entrenamiento y agrupa en clases los vectores similares.

(Agradecemos enormemente a la colaboración de la Universidad Tecnológica Nacional Facultad Regional Rosario por todo el contenido brindado sobre Redes Neuronales en el trabajo “Redes Neuronales: Conceptos Básicos y Aplicaciones.” elaborado por Damián Jorge Matich (2001)²)

2.7. Redes de función radial (RBF)

Se encontró que las redes neuronales de base radial son las que mejor se adecuan a nuestro objetivo, debido a que son redes multicapa con conexiones hacia delante, que se caracterizan porque están formadas por una única capa oculta y cada neurona de esta capa posee un carácter local, en el sentido de que cada neurona oculta de la red se activa en una región diferente del espacio de patrones de entrada. Este carácter local viene dado por el uso de las llamadas funciones de base radial (RBF), generalmente la función gaussiana, como funciones de activación. Como se puede apreciar en la imagen anterior, las neuronas de la capa de salida de las redes de base radial simplemente realizan una combinación lineal de las activaciones de las neuronas ocultas. La capa de entrada la componen un conjunto de neuronas que reciben las señales del exterior, transmitiéndolas a la siguiente capa sin realizar ningún procesado sobre dichas señales. Las neuronas de la capa oculta reciben las señales de la capa de entrada y realizan una transformación local y no lineal sobre dichas señales.

2.7.1. Activación de las neuronas de la red de base radial.

Si la red tiene “p” neuronas en la capa de entrada, “m” neuronas en la capa oculta y “r” neuronas en la capa de salida, las activaciones de las neuronas de salida para el patrón de entrada “n”:

$$X(n) = (x_1(n), x_2(n), \dots, x_p(n),) \quad (2.7.1)$$

²Link al documento

son denotadas como $y_k(n)$, vienen dadas por la siguiente ecuación:

$$y_k(n) = \sum w_{ik} \theta_i(n) \pm u_k \text{con} : k = 1, 2, \dots \quad (2.7.2)$$

donde: w_{ik} es el peso de la conexión de la neurona oculta i a la neurona de salida k ; u_k es el umbral de la neurona de salida k ; $\theta_i(n)$ son las activaciones de las neuronas ocultas para el patrón de entrada $X(n)$.

Las funciones de base radial θ_i determinan las activaciones de las neuronas ocultas de la red en función de un vector de entrada a la red $X(n)$ y vienen dadas por expresiones que dependen de los centros de la función de base radial, la desviación o amplitud de la función de base radial y la distancia del vector de entrada $X(n)$ al centro C_i .

Las entradas x_1, x_2, \dots, x_m conforman un vector de entrada x , y son aplicadas a todas las neuronas en una capa oculta. Según la topología de la red que se muestra en la siguiente figura:

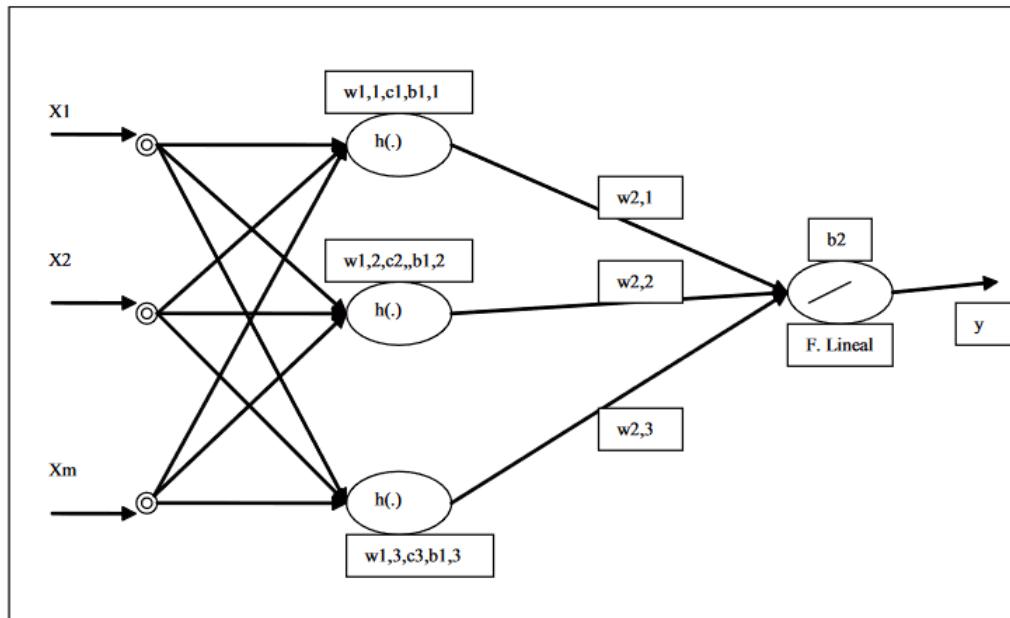


Figura 2.6: Red Neuronal de Base Radial. Arquitectura

Cada neurona de la capa oculta calcula la siguiente función exponencial (radial):

$$h_i = e^{(-D_i^2/\sigma^2)}$$

Donde:

$$D_i^2 = (x - u_i)^T (x - u_i) = \text{Distancia Euclídea}$$

x y u = vectores columna.

T = indica la transpuesta del vector.

Los pesos de cada neurona de capa oculta son asignados desde los valores de un vector de entrenamiento de entrada. La neurona de salida produce la suma de pesos lineal de estos:

$$y = \sum h_i w_i + b_{ij}$$

Donde:

x = un vector de entrada.

y = un vector de salida.

$w_{1,j}$ y $w_{2,j}$ = vector peso de la neurona i de la capa oculta y j de la capa de salida.

$b_{1,j}$ y $b_{2,j}$ = umbrales de la neurona i de la capa oculta y j de la capa de salida.

u_i = centros de las neuronas ocultas

Las Redes de Base Radial son aproximadores universales de carácter local.

2.7.2. Aprendizaje de las redes neuronales de base radial

Se pueden dar dos casos: Híbrido y totalmente supervisado. Para nuestro proyecto en particular vimos necesario y más efectivo realizarlo de forma híbrida.

En el caso híbrido: se tiene la primera fase no supervisada y la segunda supervisada.

Fase no supervisada:

Los centros de las funciones de base radial se determinan mediante un algoritmo clasificación no supervisado que permita dividir el espacio de patrones de entrada en clases. El número de clases es el número de neuronas ocultas en la red de base radial. Se puede utilizar el algoritmo de K medias, o cualquier otro, e incluso los mapas autoorganizados de Kohonen. Las amplitudes o desviaciones se calculan de manera que cada neurona oculta se active de una región del espacio de entrada y de manera que el solapamiento de las zonas de activación de una neurona a otra sea lo más ligero posible, para supervisar así la interpolación. Entre otras, se pueden usar las heurísticas de los vecinos más cercanos, la media uniforme de las distancias euclídeas del centro “ C_i ” a los centros más cercanos, o la media geométrica de las distancias centro C_i a los “ p ” centro más cercanos o incluso valores determinados que permitan un valor de salida predeterminado para luego aplicar la función de base radial.

Fase supervisada:

Se busca minimizar el error entre el valor de salida de la red y el de la salida deseada que corresponde a su respectivo par p de entrada. Para ello se puede seguir el método de mínimos cuadrados o el de la pseudoinversa.

Usando el método de la pseudoinversa aplicamos la siguiente ecuación:

$$W = G^+ S = (G^T G)^{-1} G^T S$$

Donde: W es la matriz de pesos y umbrales de la red,

G^+ es la matriz pseudoinversa de G ,

G^T es la matriz transpuesta de G ,

G es la matriz que contiene los valores de las funciones de base radial (salida de la capa oculta),

S la matriz que tiene todas las salidas deseadas.

Según lo visto podemos decir que una red neuronal con muchas capas ocultas y neuronas puede ser capaz de aprender patrones muy complejos en los datos, pero también puede ser más propensa a sobreajustar (aprender demasiado de los datos de entrenamiento y no generalizar bien a nuevos datos). En cambio, una red neuronal con menos capas y neuronas puede tener una capacidad de aprendizaje más limitada, pero puede ser más resistente al sobreajuste.

Por lo tanto, por cuestiones de diseño y procurando contener la mayor cantidad de datos bien identificados, decidimos utilizar el siguiente formato para entrenar nuestra red neuronal, la cual respeta una estructura como la de la siguiente figura:

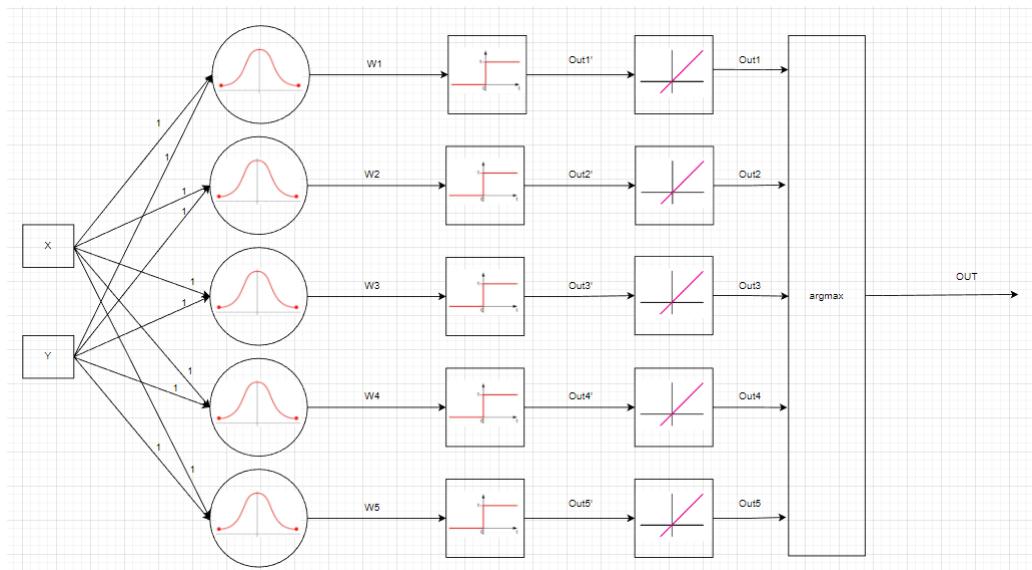


Figura 2.7: Arquitectura de la red neuronal

Capítulo 3

Sensor MPU6050

3.1. ¿Porqué elegimos el sensor MPU6050?

El MPU-6050 es un sensor de movimiento que posee un acelerómetro y un giroscopio en los 3 ejes (x,y,z) con una alta precisión. Posee ADC internos de 16Bit y se maneja por I2C desde cualquier microcontrolador. Dentro de una de las ventajas que cabe mencionar, El MPU-6050 posee conversores analógicos digitales por cada uno de los ejes de cada uno de los sensores para obtener los valores simultáneos con un rango de hasta 2000 °/s para el giroscopio y hasta +/- 16g para el acelerómetro.

El módulo Acelerómetro MPU tiene un giroscopio de tres ejes con el que podemos medir velocidad angular y un acelerómetro también de 3 ejes con el que medimos los componentes X, Y y Z de la aceleración. La dirección de los ejes está indicada en el módulo el cual hay que tener en cuenta para no equivocarnos en el signo de las aceleraciones. (Figura 3.1)

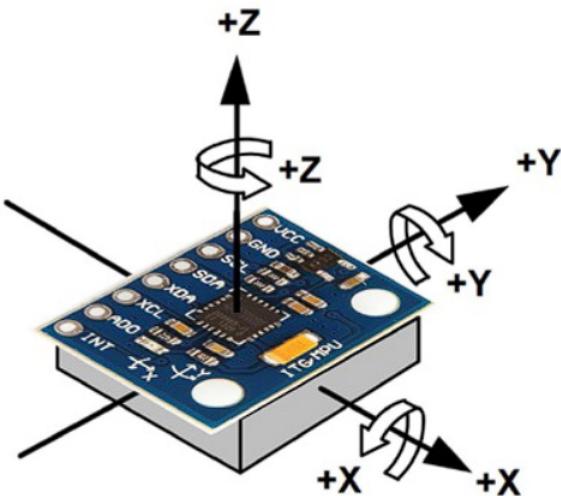


Figura 3.1: Ejes del MPU

Otra de las ventajas es que el MPU-6050 elimina los problemas de alineación del eje transversal que puede arrastrarse hacia arriba en porciones discretas. El sensor es muy preciso, ya que contiene una conversión hardware de 16 bits de A/D por cada canal, para la digitalización de las salidas del acelerómetro. Para ello capta los canales x, y y z al mismo tiempo.

3.1.1. Especificaciones del MPU

Este dispositivo es el primer integrado de rastreo de movimiento de 6 ejes del mundo. Muchas de las especificaciones de este módulo que se dan a continuación fueron el motivo de la elección del mismo:

- El MPU6050 tiene un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un procesador de movimiento digital integrado en un solo chip.
- Funciona con la fuente de alimentación de 3V-5V.
- MPU6050 utiliza el protocolo I2C para la comunicación y la transferencia de datos.
- Este módulo tiene incorporado un ADC de 16 bits que proporciona una gran precisión.
- MPU6050 puede ser interconectado con otros dispositivos IIC como los magnetómetros.
- MPU6050 también tiene un sensor de temperatura incorporado.
- El bus de sensores I2C le ayuda a recoger datos directamente de la brújula externa de 3 ejes, que utiliza para proporcionar una salida completa de 9 ejes de MotionFusion.
- Para los usuarios, la MPU6050 elimina la necesidad de selección, calificación e integración a nivel de sistema de dispositivos discretos.
- Usando su puerto I2C, los sensores no iniciales como el sensor de presión pueden ser interconectados.
- MPU6050 consta de tres ADC de 16 bits para digitalizar las salidas del giroscopio 0 y tres ADC de 16 bits para digitalizar las salidas del acelerómetro.
- Se dispone de una gama de giroscopios y de una gama de acelerómetros programables por el usuario para el seguimiento de recesión de movimientos tanto rápidos como lentos.
- Se dispone de un búffer FIFO de 1024 bytes en el chip que ayuda a reducir el consumo de energía del módulo.
- La necesidad de agrupar frecuentemente la salida del sensor se reduce al mínimo con la ayuda del DMP en el chip.
- El MPU6050 también tiene un oscilador en el chip con una variación de $\pm 1\%$.
- El MPU6050 tiene filtros de paso bajo para el giroscopio.
- El pin de referencia VLOGIC se utiliza para establecer los niveles lógicos de la interfaz I2C.

- El rango programable por el usuario del giroscopio presente en el MPU6050 es de ± 250 , ± 500 , ± 1000 y $\pm 2000^{\circ}/\text{seg}$.
- La imagen, el vídeo y la sincronización del GPS son compatibles con la clavija de sincronización externa del giroscopio.
- Este giroscopio ha mejorado el rendimiento del ruido de baja frecuencia.
- El giroscopio necesita 3,6mA de corriente para funcionar.
- El filtro de paso bajo del giroscopio es programable digitalmente.
- El acelerómetro presente en el MPU6050 opera en 500A de corriente.
- El rango programable a escala completa de este acelerómetro es de $\pm 2g$, $\pm 4g$, $\pm 8g$ y $16g$.
- El acelerómetro también puede detectar la orientación.
- Las interrupciones programables por el usuario están presentes para el acelerómetro.
- Entre los ejes del acelerómetro y del giroscopio hay una sensibilidad mínima en los ejes cruzados.
- Para comunicarse con todos los registros se utiliza el modo rápido de 400kHz I2C.
- El DMP presente en el MPU6050 soporta el procesamiento e movimiento en 3D y los algoritmos de reconocimiento de gestos.
- Se proporciona una lectura de ráfagas para el procesador del sistema. Después de leer los datos del FIFO, el procesador del sistema entra en modo de suspensión de baja potencia mientras que el MPU recoge más datos.
- Las interrupciones programables admiten características como el reconocimiento de gestos, el desplazamiento, el zoom, el desplazamiento, la detección de pulsaciones y la detección de shack.
- El MPU6050 también tiene una entrada de reloj externo opcional de 32,768kHz o 19,2Mhz.

3.2. Funcionamiento y Pin Out

3.2.1. Funcionamiento del MPU

El sensor MPU-6050 es una pequeña pieza tecnológica de procesamiento de movimiento. El cual mediante la combinación de un MEMS (Sistemas Microelectromecánicos) giroscopio de 3 ejes y un MEMS acelerómetro de 3 ejes en la misma pastilla de silicio junto con un DMP (Movimiento Digital Processor), es capaz de procesar los algoritmos de

movimientos complejos de 9 ejes (MotionFusion) en una placa. Las piezas integran el algoritmo MotionFusion para 9 ejes que pueden incluso acceder a magnetómetros externos u otros sensores a través de un bus I2C auxiliar maestro, permitiendo reunir un conjunto completo de dispositivos sensores de datos, sin la intervención del procesador del sistema. El MPU-6050 es un 6 DOF (grados de libertad = Degrees of Freedom) o un sensor IMU de seis ejes, lo que significa que da seis valores de salida.

El procesador digital de movimiento (DMP) incorporado se encuentra dentro de la MPU-6050 y descarga el cálculo de los algoritmos de procesamiento de movimiento desde el procesador host. Los datos resultantes pueden ser leídos de los registros de la DMP, o pueden estar tamponados en un FIFO. El DMP tiene acceso a uno de los pines externos de la MPU, que pueden ser utilizados para la generación de interrupciones. El propósito del DMP es descargar los requisitos de temporización y la potencia de procesamiento del procesador anfitrión.

El sensor MPU-6050 es muy preciso, ya que contiene una conversión hardware de 16 bits de A/D por cada canal, para la digitalización de las salidas del acelerómetro. Para ello capta los canales x, y y z al mismo tiempo.

El módulo MPU6050 está compuesto por los siguientes bloques y funciones:

- Un sensor giroscópico de 3 ejes MEMS con tres ADC de 16 bits y acondicionamiento de señal.
- Un sensor acelerómetro MEMS de 3 ejes con tres ADC de 16bits y acondicionamiento de señal.
- Un motor de procesador de movimiento digital en el chip.
- Interfaces primarias de comunicación digital 12C.
- Interfaces primarias de comunicación digital 12C.
- Interfaces auxiliares I2C para la comunicación con sensores externos como el magnetómetro.
- Reloj interno.
- Registros de datos para almacenar los datos de los sensores.
- Memoria FIFO que ayuda a reducir el consumo de energía.
- Interrupciones programables por el usuario.
- Un sensor de temperatura de salida digital.
- Autotest para el giroscopio y el acelerómetro.
- LDO y Bias.
- Bomba de carga.
- Registros de estado

3.2.2. Características generales del MPU

Giroscopio:

- Sensores de velocidad angular de los ejes X, Y y Z de salida digital (giroscopios) con un rango de escala completa programable por el usuario de ± 250 , ± 500 , ± 1000 y ± 2000 \circ/seg .
- La señal de sincronización externa conectada al pin FSYNC admite sincronización de imagen, video y GPS.
- Los ADC integrados de 16 bits permiten el muestreo simultáneo de giroscopios.
- La estabilidad mejorada de la temperatura de polarización y sensibilidad reduce la necesidad de calibración por parte del usuario.
- Rendimiento de ruido de baja frecuencia mejorado.
- Filtro de paso bajo programable digitalmente.
- Corriente de funcionamiento del giroscopio: 5mA.
- Corriente de espera: $5 \mu \text{A}$.
- Factor de escala de sensibilidad calibrado de fábrica.

Acelerómetro:

- Acelerómetro de triple eje de salida digital con un rango de escala completa programable de ± 2 g, ± 4 g, ± 8 g y ± 16 g.
- Los ADC integrados de 16 bits permiten el muestreo simultáneo de acelerómetros sin necesidad de un multiplexor externo.
- Corriente de funcionamiento normal del acelerómetro: 500 A.
- Corriente de modo de acelerómetro de baja potencia: 10 A a 1,25 Hz, 20 A a 5 Hz, 60 A a 20 Hz, 110 A a 40 Hz.
- Detección de orientación y señalización.
- Detección de toques.
- Interrupciones programables por el usuario.
- Interrupción de caída libre.
- Interrupción de alta G.
- Cero movimiento/interrupción de movimiento.
- Autoevaluación del usuario.

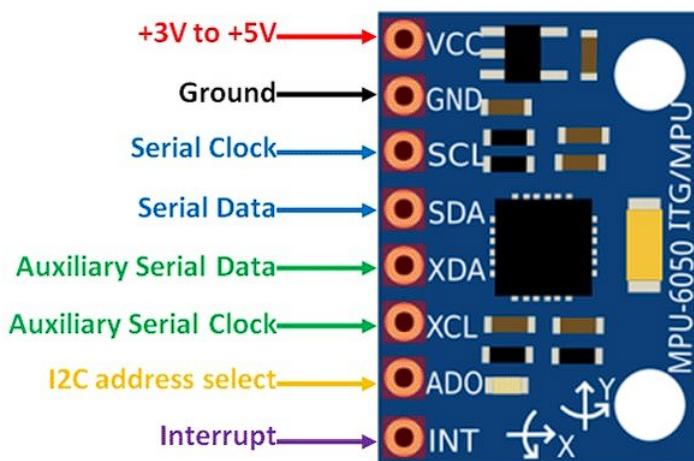


Figura 3.2: Pin Out del MPU

3.2.3. Pin Out del MPU

| Nombre Pin | Nro. Pin | Descripción |
|----------------------------|----------|---|
| Vcc | 1 | Pin de voltaje de la fuente de alimentación |
| GND | 2 | Pin GND del modulo |
| Reloj serie | 3 | es el reloj de serie I2C |
| Datos serie | 4 | es el pin de datos en serie de I2C. |
| Auxiliar de Datos serie | 5 | es el pin de datos de la serie maestra de I2C. Este pin se usa para conectar sensores externos. |
| Auxiliar de Reloj serie | 6 | es el reloj serial maestro de I2C. Esta clavija se usa para conectar sensores externos. |
| I2C selección de dirección | 7 | es el pin LSB de dirección de esclavo de I2C. |
| Interrupción | 8 | es el pin de salida digital de interrupción. |

3.3. Comunicación con el MPU

3.3.1. Definición y conceptos

Tal cual mencionamos anteriormente el MPU utiliza el protocolo de comunicación conocido como I2C, abreviatura de Inter-IC (Inter Integrated Circuits), un tipo de bus diseñado por Philips Semiconductors a principios de los 80s, que se utiliza para conectar circuitos integrados (ICs).

Básicamente el I2C es un bus con múltiples maestros, lo que significa que se pueden conectar varios chips al mismo bus y que todos ellos pueden actuar como maestro, sólo con iniciar la transferencia de datos. Este bus se utiliza dentro de una misma placa de un dispositivo. Es un estándar que facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de nteligencia, sólo requiere de dos líneas de señal y un común o masa lo que permite el intercambio de información

entre muchos dispositivos a una velocidad aceptable, de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 3,4 MHz.

La metodología de comunicación de datos del bus I2C es en serie y sincrónica. Una de las señales del bus marca el tiempo (pulsos de reloj) y la otra se utiliza para intercambiar datos.

Direccionamiento de dispositivos en el bus I2C

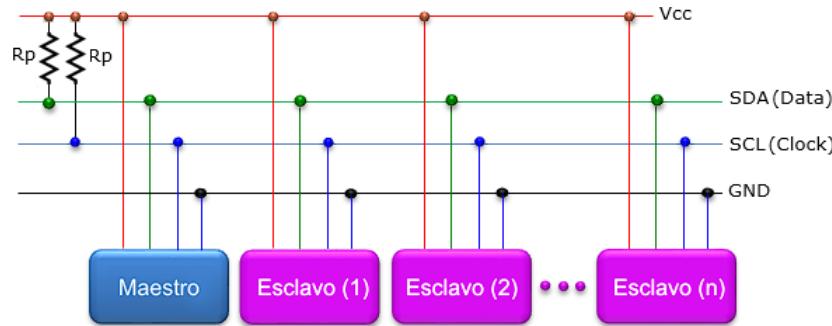


Figura 3.3: Esquema básico del I2C

Descripción de las señales

- SCL (System Clock) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (Masa) común de la interconexión entre todos los dispositivos nenganchados al bus.

Las líneas SDA y SCL son del tipo drenaje abierto, es decir, un estado similar al de colector abierto, pero asociadas a un transistor de efecto de campo (o FET). Se deben polarizar en estado alto (conectando a la alimentación por medio de resistores pull-up) lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas. En principio, el número de dispositivos que se puede conectar al bus no tiene límites, aunque hay que observar que la capacidad máxima sumada de todos los dispositivos no supere los 400 pF. El valor de los resistores de polarización no es muy crítico, y puede ir desde 1K8 (1.800 ohms) a 47K (47.000 ohms). Un valor menor de resistencia incrementa el consumo de los integrados pero disminuye la sensibilidad al ruido y mejora el tiempo de los flancos de subida y bajada de las señales. Los valores más comunes en uso son entre 1K8 y 10K.

3.3.2. Protocolo I2C

Habiendo varios dispositivos conectados sobre el bus, es lógico que para establecer una comunicación a través de él se deba respetar un protocolo. Digamos, en primer lugar, lo más importante: existen dispositivos maestros y dispositivos esclavos. Sólo los dispositivos maestros pueden iniciar una comunicación.

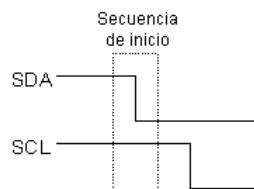


Figura 3.4: Condición inicial del bus

La condición inicial, de bus libre, es cuando ambas señales están en estado lógico alto. En este estado cualquier dispositivo maestro puede ocuparlo, estableciendo la condición de inicio (start). Esta condición se presenta cuando un dispositivo maestro pone en estado bajo la línea de datos (SDA), pero dejando en alto la línea de reloj (SCL). El primer byte que se transmite luego de la condición de inicio contiene siete bits que componen la dirección del dispositivo que se desea seleccionar, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura). Si el dispositivo cuya dirección corresponde a la que se indica en los siete bits (A0-A6) está presente en el bus, éste contesta con un bit en bajo, ubicado inmediatamente luego del octavo bit que ha enviado el dispositivo maestro. Este bit de reconocimiento (ACK) en bajo le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos.



Figura 3.5: Comunicación del bus

Si el bit de lectura/escritura (R/W) fue puesto en esta comunicación a nivel lógico bajo (escritura), el dispositivo maestro envía datos al dispositivo esclavo. Esto se mantiene mientras continúe recibiendo señales de reconocimiento, y el contacto concluye cuando se hayan transmitido todos los datos. En el caso contrario, cuando el bit de lectura/escritura estaba a nivel lógico alto (lectura), el dispositivo maestro genera pulsos de reloj para que el dispositivo esclavo pueda enviar los datos. Luego de cada byte recibido el dispositivo maestro (quien está recibiendo los datos) genera un pulso de reconocimiento.

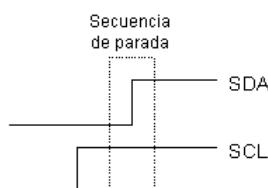


Figura 3.6: Condición de parada del bus

El dispositivo maestro puede dejar libre el bus generando una condición de parada (o detención; stop en inglés).

Si se desea seguir transmitiendo, el dispositivo maestro puede generar otra condición de inicio en lugar de una condición de parada. Esta nueva

condición de inicio se denomina *inicio reiterado* y se puede emplear para direccionar un dispositivo esclavo diferente o para alterar el estado del bit de lectura/escritura.

Direccionamiento de dispositivos en el bus I2C

Lo más común en los dispositivos para el bus I2C es que utilicen direcciones de 7 bits, aunque existen dispositivos de 10 bits. Este último caso es raro.

Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I2C, ya que un número de 7 bits puede ir desde 0 a 127. Cuando se envían las direcciones de 7 bit, de cualquier modo la transmisión es de 8 bits. El bit extra se utiliza para informarle al dispositivo esclavo si el dispositivo maestro va a escribir o va a leer datos desde él. Si el bit de lectura/escritura (R/W) es cero, el dispositivo maestro está escribiendo en el esclavo. Si el bit es 1 el maestro está leyendo desde el esclavo. La dirección de 7 bit se coloca en los 7 bits más significativos del byte y el bit de lectura/escritura es el bit menos significativo.

El hecho de colocar la dirección de 7 bits en los 7 bits más significativos del byte produce confusiones entre quienes comienzan a trabajar con este bus. Si, por ejemplo, se desea escribir en la dirección 21 (hexadecimal), en realidad se debe enviar un 42, que es un 21 desplazado un bit hacia arriba. También se pueden tomar las direcciones del bus I2C como direcciones de 8 bit, en las que las pares son de sólo escritura y las impares son de sólo lectura. Para dar un ejemplo, el integrado de brújula magnética CMPS03 es fijado en fábrica en la dirección 0xC0 (\$C0). La dirección 0xC0 se utiliza para escribir en él y la dirección 0xC1 es para leer de él.

Protocolo de programación para el bus I2C

Lo primero que ocurre en un bus I2C es que el dispositivo maestro envía una secuencia de inicio. Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transacción. Éstos quedan atentos para ver si se trata de una solicitud para ellos. A continuación el dispositivo maestro envía la dirección de dispositivo. El dispositivo esclavo que posee esa dirección continuará con la transacción, y los otros ignorarán el resto de los intercambios, esperando la próxima secuencia de inicio.

Habiendo direccionado ya el dispositivo esclavo, lo que debe hacer ahora el maestro es enviar la ubicación interna o número de registro desde el que desea leer o al que va a escribir. La cantidad depende, obviamente, de qué dispositivo es y de cuántos registros internos posee. Algunos dispositivos muy simples no tienen ninguno, pero la mayoría sí los poseen.

Siguiendo con el ejemplo del CMPS03, éste posee 16 ubicaciones internas, numeradas desde el 0 al 15. Otro dispositivo, el medidor ultrasónico de distancia SRF08, tiene 36 registros.

Una vez que el maestro ha enviado la dirección del dispositivo en el bus I2C y la dirección del registro interno del dispositivo, puede en-

viar ahora el byte o bytes de datos. El dispositivo maestro puede seguir enviando bytes al esclavo, que normalmente serán puestos en registros con direcciones sucesivas, ya que el esclavo incrementa automáticamente la dirección del registro interno después de recibir cada byte. Cuando el maestro ha terminado de escribir datos en el esclavo, envía una secuencia de parada que concluye la transacción.

Lectura desde un dispositivo esclavo:

Esta operación es algo más complicada, pero no demasiado. Antes de leer datos desde el dispositivo esclavo, primero se le debe informar desde cuál de sus direcciones internas se va a leer. De manera que una lectura desde un dispositivo esclavo en realidad comienza con una operación de escritura en él. Es igual a cuando se desea escribir en él: Se envía la secuencia de inicio, la dirección de dispositivo con el bit de lectura/escritura en bajo y el registro interno desde el que se desea leer. Ahora se envía otra secuencia de inicio nuevamente con la dirección de dispositivo, pero esta vez con el bit de lectura/escritura en alto. Luego se leen todos los bytes necesarios y se termina la transacción con una secuencia de parada.

Volviendo al ejemplo del módulo de brújula CMPS03, veamos cómo se lee el registro de ángulo:

1. Enviar una secuencia de inicio.
2. Enviar 0xC0 (La dirección de dispositivo del CMPS03 con el bit de lectura/escritura en bajo)
3. Enviar 0x01 (dirección interna del registro de ángulo en valor 0-255)
4. Enviar una secuencia de inicio (inicio reiterado)
5. Enviar 0xC1 (La dirección de dispositivo del CMPS03 con el bit de lectura/escritura en alto)
6. Leer un byte de dato desde el CMPS03.
7. Enviar la secuencia de parada.

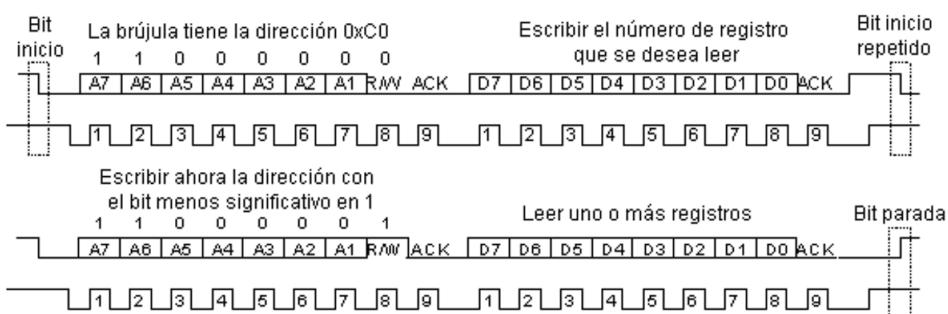


Figura 3.7: Secuencia del registro del ángulo para el módulo de brújula CMPS03

3.4. Pruebas MPU6050

Para obtener los datos del sensor MPU-6050 con nuestro microcontrolador se utilizará Thonny que es el entorno de desarrollo integrado de Python el cual nos permite programar el ESP32. Utilizamos el esquema de la imagen 3.8

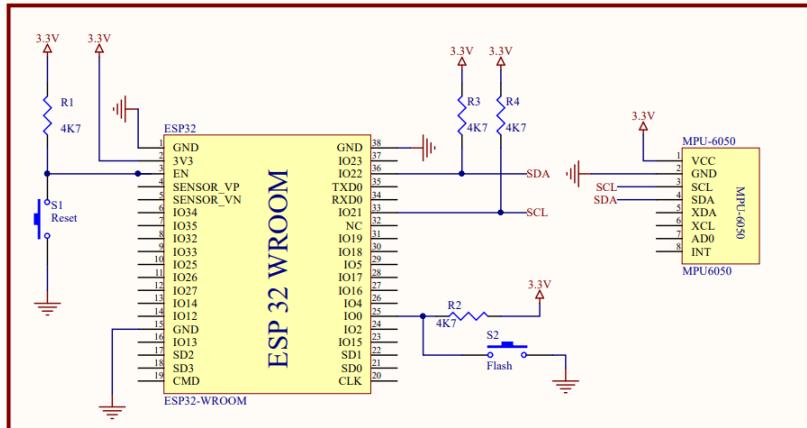


Figura 3.8: Prueba de Sensor MPU-6050

Para la implementación del sensor MPU-6050 utilizamos la librería de Peter Hinch, Micropython developer, Ver aquí el perfil y Ver aquí para descargas. Las librerías en nuestro microcontrolador quedarían de la siguiente manera

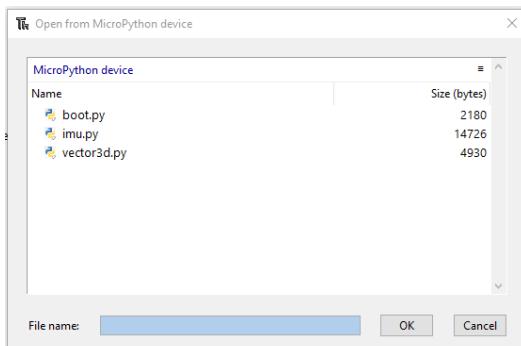


Figura 3.9: Librerías en ESP32

Pondremos el sensor en la posición de reposo, es decir Z=1, X=0, Y=0, nuestro programa es el siguiente

```

1 """
2 Programa: Test MPU6050
3
4 Propósito: Verificar funcionamiento de sensor MPU6050
5
6 Programadores: Facundo Cosentino, Sergio Zelaya
7 Fecha: 30/07/2022
8 """
9
10 # Importamos MPU_6050 de imu
11 from imu import MPU6050
12 # Importamos time para los delay
13 import time

```

```

14 # Importamos Pin e I2C de machine
15 from machine import Pin, I2C
16
17
18 #Configuramos comunicacion I2C
19 I2c = I2C(0,sda=Pin(21),scl=Pin(22),freq=400000)
20 #Configuramos MPU6050
21 imu = MPU6050(I2c)
22
23 while True:
24     #Leemos Aceleracion en X
25     Gx = imu.accel.x
26
27     #Leemos Aceleracion en Y
28     Gy = imu.accel.y
29
30     #Leemos Aceleracion en Z
31     Gz = imu.accel.z
32
33     #Impresion de los valores
34     print("Gx:",round(Gx, 2))
35     print("Gy:",round(Gy, 2))
36     print("Gz:",round(Gz, 2))
37     print(" ")
38     #Esperamos un 1 segundo
39     time.sleep(1)

```

Algoritmo 3.1: Test-esp32-mpu6050

y tenemos la siguiente salida

```

MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
    Gx: 0.01
    Gy: -0.06
    Gz: 0.92

    Gx: 0.01
    Gy: -0.07
    Gz: 0.91

    Gx: 0.01
    Gy: -0.07
    Gz: 0.93

```

Figura 3.10: Salida del algoritmo 3.1

3.5. Calibracion

Podemos ver que en la imagen 3.10 de la sección 3 los valores no son los deseados para en dicha posición, entonces procedemos a hacer una calibración del sensor. Usaremos el circuito de la figura 3.8, el objetivo es medir el valor de offset de cada eje en las posiciones de calibración, y sumarle ese offset a nuestro programa general. El algoritmo que implementado fue el siguiente

```

1 '''
2 Programa: Calibracion de MPU6050
3

```

```

4 Proposito: Obtener los valores medios de offset para
5 la aceleracion en cada eje
6
7 Programador: Cosentino Facundo, Sergio Zelaya
8 fecha: 30/07/2022
9 '''
10 '''
11 '''
12 -----
13 Librerias a usar
14 -----
15 '''
16 # Importamos MPU_6050 de imu
17 from imu import MPU6050
18
19 # Importamos time para los delay
20 import time
21
22 # Importamos Pin e I2C de machine
23 from machine import Pin, I2C
24
25 '''
26 -----
27 Configuracion de modulos
28 -----
29 '''
30 # Configuramos comunicacion I2C
31 I2c = I2C(0,sda=Pin(21),scl=Pin(22),freq=400000)
32
33 # Configuramos MPU6050
34 imu = MPU6050(I2c)
35
36 '''
37 -----
38 Definicion de Funciones
39 -----
40 '''
41 """ Funcion para error en X
42 def errorX(n):
43     error = 0                      # Inicializamos error
44     suma = 0                        # Inicializamos sumatoria
45
46     print("Calculando...")          # letrero
47
48     for i in range (n):            # iteramos n ciclos
49         vinst = imu.accel.x        # leemos eje X
50         error = 1 - vinst          # calculamos el error
51         suma = suma + error      # acumulamos el error
52         time.sleep(0.01)           # tiempo de 10ms
53
54     promedio = suma / n           # el valor promedio
55     return promedio, suma         # devolvemos el promedio
56                               # y la sumatoria
57
58 """ Funcion para error en Y
59 def errorY(n):
60     error = 0                      # Inicializamos error
61     suma = 0                        # Inicializamos sumatoria
62
63     print("Calculando...")          # letrero

```

```

64     for i in range (n):           #iteramos n ciclos
65         vinst = imu.accel.y      #leemos eje Y
66         error = 1 - vinst       #calculamos el error
67         suma = suma + error   #acumulamos el error
68         time.sleep(0.01)        #tiempo de 10ms
69
70     promedio = suma / n        #el valor promedio
71     return promedio, suma     #devolvemos el promedio
72                                         #y la sumatoria
73
74
75 ##### Funcion para error en Z
76
77 def errorZ(n):
78     error = 0                  #Inicializamos error
79     suma = 0                   #Inicializamos sumatoria
80
81     print("Calculando...")     #letrero
82
83     for i in range (n):        #iteramos n ciclos
84         vinst = imu.accel.z    #leemos eje Y
85         error = 1 - vinst      #calculamos el error
86         suma = suma + error   #acumulamos el error
87         time.sleep(0.01)        #tiempo de 10ms
88
89     promedio = suma / n        #el valor promedio
90     return promedio, suma     #devolvemos el promedio
91                                         #y la sumatoria
92
93 #### Funcion para Imprimir los datos
94
95 def printdata(eje,error,sumatoria,muestras):
96 #si no ingresamos el valor de eje como
97 #un string salta un error
98     if type(eje) is str:
99         #Mensaje de exito
100        print("El valor medio del error en el eje: ",eje)
101        print("es de: ",error)
102        print("siendo la sumatoria: ",sumatoria)
103        print("en : ",muestras,"muestras")
104    else:
105        #Mensaje de error
106        print("Error: el valor de eje debe ser 'str' ")
107
108 '''
109 -----
110 Funcion principal
111 -----
112 '''
113 #Medimos el tiempo en el calcular el error
114 start=time.ticks_us()
115 #Calculamos el error en un determinado eje
116 errorz,sumatoria=errorZ(100)
117 end=time.ticks_us()
118 #Mostramos los datos obtenidos
119 printdata('Z',errorz,sumatoria,100)
120 #Tiempo de proceso
121 print("Proceso de",end-start,"uS")

```

Algoritmo 3.2: Calibracion de MPU6050

tenemos la siguiente salida para la calibración del eje Z, donde no le afecte la graves a los otros dos ejes es decir, Z=1, X=0, Y =0

```
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Calculando...
El valor medio del error en el eje: Z
es de: 0.07965576
siendo la sumatoria: 7.965576
en : 100 muestras
Proceso de 1198936 uS

>>>
```

Figura 3.11: Valor Offset en Z

tenemos la siguiente salida para la calibración del eje X, en la posición Z=0, X=1, Y =0

```
>>> %Run -c $EDITOR_CONTENT

Calculando...
El valor medio del error en el eje: X
es de: -0.06112061
siendo la sumatoria: -6.112061
en : 100 muestras
Proceso de 1201684 uS

>>>
```

Figura 3.12: Valor Offset en X

tenemos la siguiente salida para la calibración del eje Y, en la posición Z=0, X=0, Y =1

```
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Calculando...
El valor medio del error en el eje: Y
es de: 0.03150146
siendo la sumatoria: 3.150146
en : 100 muestras
Proceso de 1201863 uS

>>>
```

Figura 3.13: Valor Offset en Y

Agregamos estos valores en los respectivos ejes del programa 3.1

```
23     while True:
24         #Leemos Aceleracion en X y compensamos
25         Gx = imu.accel.x-0.06112061
26
27         #Leemos Aceleracion en Y y compensamos
28         Gy = imu.accel.y+0.03150146
29
```

```

30      #Leemos Aceleracion en Z y compensamos
31      Gz = imu.accel.z+0.07965576
32
33      #Impresion de los valores
34      print("Gx:",round(Gx, 2))
35      print("Gy:",round(Gy, 2))
37      print("Gz:",round(Gz, 2))
38      print(" ")

```

y obtenemos la salida del algoritmo, para cada eje

```

Gx: 1.0 Gy: 0.03 Gz: -0.01
Gy: 0.05 Gy: 1.0 Gz: -0.03
Gz: 0.0 Gz: -0.06 Gz: 1.0

```

Figura 3.14: Calibración Completa

3.6. Curva de Aceleración

3.6.1. Read and Send data: Recepción de datos MPU6050 y transmisión de datos a la PC

En subsección veremos un algoritmo que escribimos separa poder visualizar las variables X, Y, Z en distintas posiciones de la mano para poder nosotros mismos por medio de gráficas reconocer patrones, que seria lo que es secciones posteriores haremos con nuestra red neuronal. El algoritmo 'read and send data' esta basado en el diagrama en flujo de la imagen 3.15 , con el algoritmo descripto podremos obtener los datos del sensor, procesarlos desde ESP32 con micropython y envíalos a la PC.

El algoritmo implementado es el siguiente

```

1 """
2 Programa: Armado de trama de datos MPU6050
3
4 Proposito: Armar trama de datos con el formato
5 'GX,GY,GZ\n' para receptionar con nuestro programa
6 de python "PLOT-PYTHON" en el repositorio
7 "PLOT-MPU6050" y poder ver de manera grafica los datos
8
9 Programadores: Facundo Cosentino, Sergio Zelaya
10 Fecha: 2/08/2022
11 """
12
13 #Importamos MPU_6050 de imu
14 from imu import MPU6050
15 #Importamos time para los delay
16 import time
17 #Importamos Pin e I2C de machine
18 from machine import Pin, I2C, UART
19
20
21 #Configuramos comunicacion I2C
22 I2c = I2C(0,sda=Pin(21),scl=Pin(22),freq=400000)
23 #Configuramos MPU6050

```

```

24 imu = MPU6050(I2c)
25 #Configuracion UART 9600baudios y el modulo 2
26 uart= UART(2,9600)
27
28 #Variable para guardar el tiempo anterior
29 tiempoant=0
30 #Variable para contar el numero de muestras
31 muestras=0
32 #Variable para guardar la trama
33 trama=' '
34
35 while True:
36
37     #Esperamos una bandera para empezar a enviar datos
38     flag = uart.read()
39     #convertimos la bandera de byte a str
40     flag = str(flag)
41     #inicializamos las muestras en 0
42     muestras=0
43
44     #leeremos 150 muestras luego de tener
45     #la bandera en '1'
46     while flag.find('1') == 2 and muestras<=150:
47         #Medimos el tiempo en el que estamos
48         tiempoact = time.ticks_ms()
49
50         #Vemos si ya paso 100ms entre la ultima medida
51         if(tiempoact-tiempoant >= 100):
52             #Actualizamos el tiempo
53             tiempoant = time.ticks_ms()
54
55         #Leemos Aceleracion en X
56         Gx = imu.accel.x-0.06112061
57
58         #Leemos Aceleracion en Y
59         Gy = imu.accel.y+0.03150146
60
61         #Leemos Aceleracion en Z
62         Gz = imu.accel.z+0.07965576
63
64         #Damos formato al dato
65         datax = str(int(Gx*100))
66         datay = str(int(Gy*100))
67         dataz = str(int(Gz*100))
68
69         #Armamos la trama de datos
70         trama=datax+','+datay+','+dataz
71         #Enviamos Trama de dato
72         uart.write(trama)
73         #Enviamos salto de linea
74         uart.write('\n')
75         #Incrementamos el numero de
76         #muestras recibidas
77         muestras = muestras + 1
78
79         #Imprimimos el tiempo que tarda
80         #print(time.ticks_ms()-tiempoant)
81
82         #Imprimimos el numero de muestras

```

```
83     print(muestras)
```

Algoritmo 3.3: Read and Send data

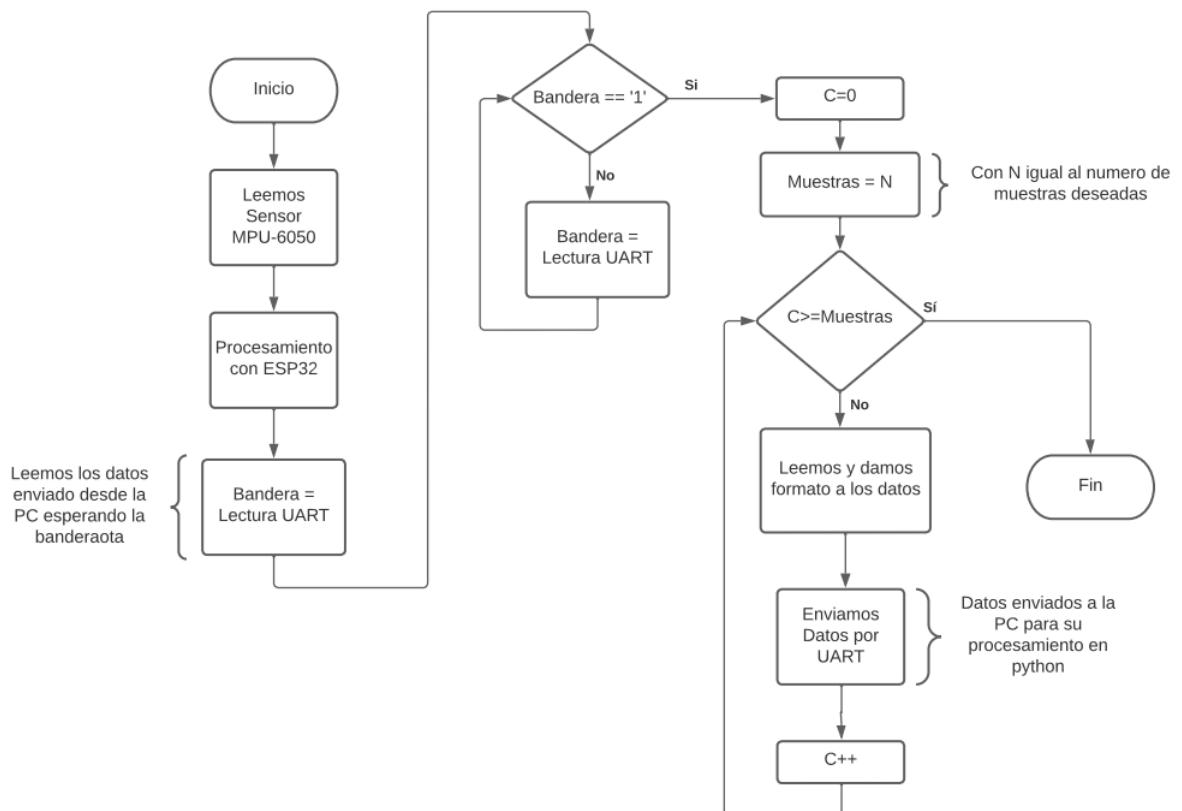


Figura 3.15: Diagrama en flujo de 'Read and Send data'

3.6.2. Plot Python: Recepcion de los datos desde ESP32 para graficas

En subsección veremos un algoritmo capaz de separar y poder visualizar las variables X, Y, Z, enviadas desde el algoritmo 'Read and Send data' y graficar los datos mediante la librería matplotlib. El algoritmo 'Plot Python' esta basado en el diagrama en flujo de la imagen 3.16 , con el algoritmo descripto podremos recibir los datos enviados del ESP32, procesarlos en python en nuestra PC y Mostrar las curvas.

El algoritmo implementado es el siguiente

```

1  """
2 Programa: PLOT-PYTHON
3
4 Proposito: Recepcion de datos de sensor MPU-6050
5 y graficas de los ejes X,Y,Z
6
7 Programadores: Facundo Cosentino, Sergio Zelaya
8 Fecha: 2/08/2022
9 """
10
11 """
12 -----
13 Librerias a Utilizar
14

```

```
15 -----
16 '''
17 #Libreria para manejo de UART
18 import serial
19 #Libreria para manejo de arrays
20 import numpy as np
21 #Libreria para hacer graficas
22 import matplotlib.pyplot as plt
23
24 '''
25 -----
26 Configuracion de comunicacion UART
27 -----
28 '''
29 #Elejimos el puerto y lo configuramos
30 ser = serial.Serial()
31 ser.port = 'COM5'
32 ser.baudrate = 9600
33 ser.open()
34
35 '''
36 -----
37 Declaracion de variable
38 -----
39 '''
40 #Cantidad de muestras
41 muestras=150
42
43 #Array avacio para eje X, Y, Z
44 x = np.ones(muestras).reshape(muestras,1)
45 y = np.ones(muestras).reshape(muestras,1)
46 z = np.ones(muestras).reshape(muestras,1)
47
48 #Base de tiempo
49 t = np.arange(0,muestras/10, 0.1, dtype=float)
50 t = t.reshape(muestras,1)
51 #Numero de muestras
52 tm=np.arange(0,muestras, 1, dtype=float)
53 tm=tm..reshape(muestras,1)
54
55 '''
56 -----
57 Funcion para descomponer String
58 -----
59 '''
60 def descomponer(cadena): #ingresa cadena
61
62 #Remplazamos los 'b por ' '
63 cadena = cadena.replace("b'"," ")
64 #Remplazamos los ' por '
65 cadena = cadena.replace("'", " ")
66 #Remplazamos los \n por '
67 cadena = cadena.replace("\n", " ")
68 #Remplazamos los espacion en blanco
69 #logrando borrarlos de la cadena
70 cadena = cadena.replace(" ", "")
71
72 #convertimos este string en un entero
73 return int(cadena)
74
```

```
75  '''
76 -----
77 Programa principal
78 -----
79 '''
80 while True:
81
82     entrada=input('Ingrese 1 para proceder: ')
83
84     ser.flushInput()    #Vaciamos el buffer serial
85     ser.write(b'1')    #Mandamos una bandera para
86                         #saber que debe empezar
87                         #a transmitir datos
88
89     if(entrada == '1'):
90
91         #Iteramos la cantidad de muestras
92         for i in range(muestras):
93
94             #Leemos el puerto serie
95             dato = ser.readline()
96             #convertimos el dato tipo byte
97             #en un tipo string
98             cadena = str(dato)
99
100            #separamos los datos X Y Z
101            ex,ey,ez = cadena.split(sep=',')
102
103            #--- Convertimos X Y Z a tipo int ---
104
105            cadena=str(ex)
106            ex = descomponer(cadena)
107            cadena=str(ey)
108            ey = descomponer(cadena)
109            cadena=str(ez)
110            ez = descomponer(cadena)
111
112            #---Armamos los arreglos ---
113            x[i,0]=ex/100    #Armamos el arreglo X
114            y[i,0]=ey/100    #Armamos el arreglo Y
115            z[i,0]=ez/100    #Armamos el arreglo Z
116
117
118 '''
119 -----
120 Mostramos los datos
121 -----
122 '''
123     print("-----Datos en eje X-----")
124     print(x)
125     print("-----Datos en eje Y-----")
126     print(y)
127     print("-----Datos en eje Y-----")
128     print(z)
129
130 '''
131 -----
132 Graficamos los datos
133 -----
134 '''
```

```

135 #Titulo de la grafica
136 plt.title('Curva de Aceleracion')
137 #label de absicas
138 plt.xlabel('Muestras')
139 #label de ordenadas
140 plt.ylabel('Aceleracion')
141 #plot x en funcion de muestra
142 plt.plot(tm,x,label="X")
143 #plot y en funcion de muestras
144 plt.plot(tm,y,label="Y")
145 #plot z en funcion de muestras
146 plt.plot(tm,z,label="Z")
147 #activamos el grid
148 plt.grid(alpha=0.4,color="SteelBlue",linestyle="-",
149 linewidth=1.5)
150 #activamos los carteles
151 plt.legend()
152 #mostramos las curvas
153 plt.show()
154 break

155 else:
156     #programa cancelado
157     print('Programa Cancelado')
158     break

```

Algoritmo 3.4: Plot Python

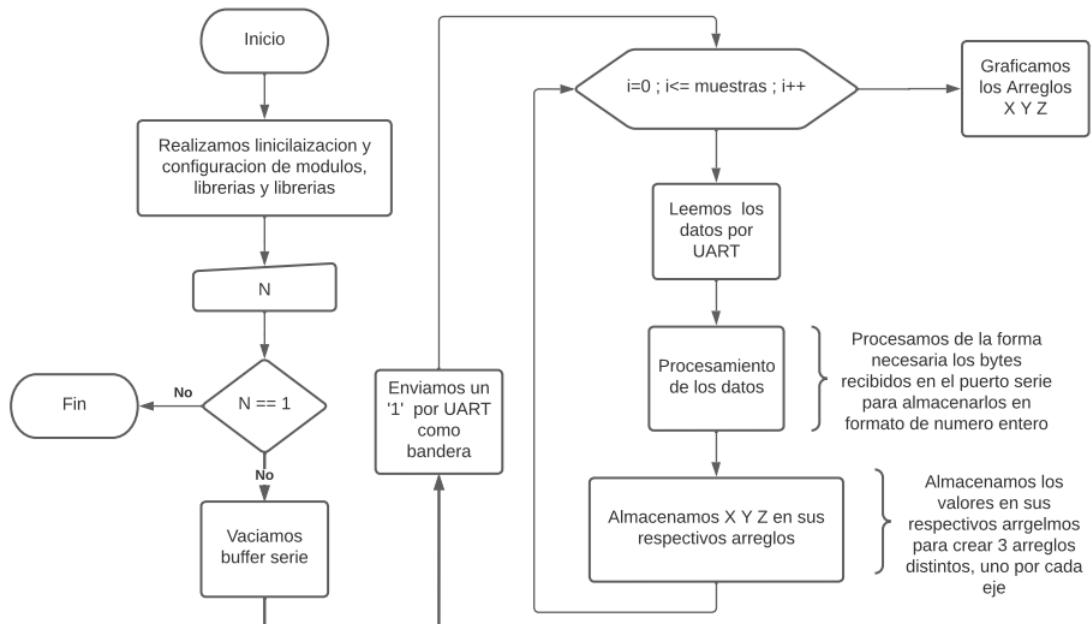


Figura 3.16: Diagrama en flujo de 'Plot Python'

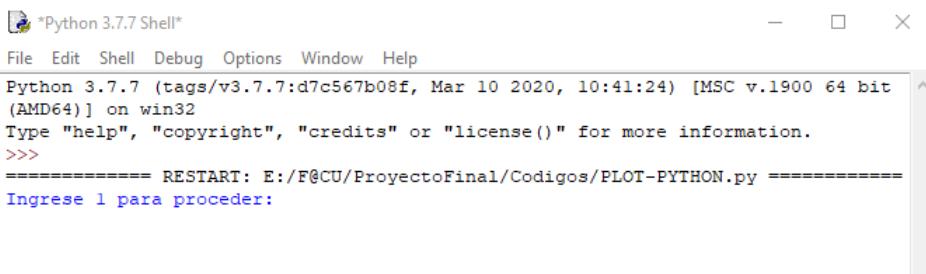
3.6.3. Pruebas

Para realizar las pruebas de ambos algoritmos en conjunto, ejecutaremos los algoritmo y deberíamos ver las siguientes pantallas en thonny y en IDLE respectivamente.



```
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

Figura 3.17: Run 'Read and Send data'



```
*Python 3.7.7 Shell*
File Edit Shell Debug Options Window Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:/F@CU/ProyectoFinal/Codigos/PLOT-PYTHON.py =====
Ingrese 1 para proceder:
```

Figura 3.18: Run 'Plot Python'

Colocamos el sensor en posiciones Z=1 X=0 Y=0, Z=0 X=1 Y=0, Z=0 X=0 Y=1 y movemos las variables para obtener las curvas. ver imágenes eje x

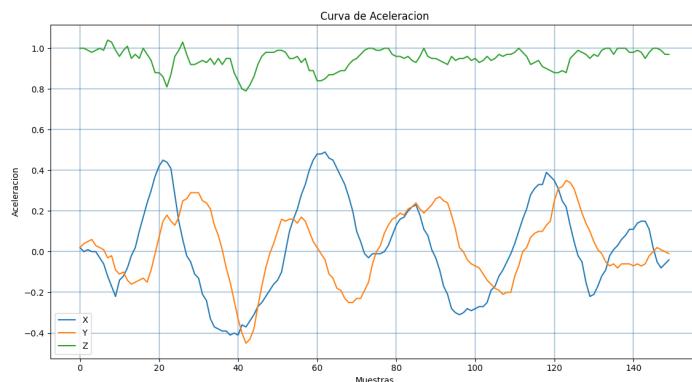


Figura 3.19: Curva con posición Z=1 X=0 Y=0

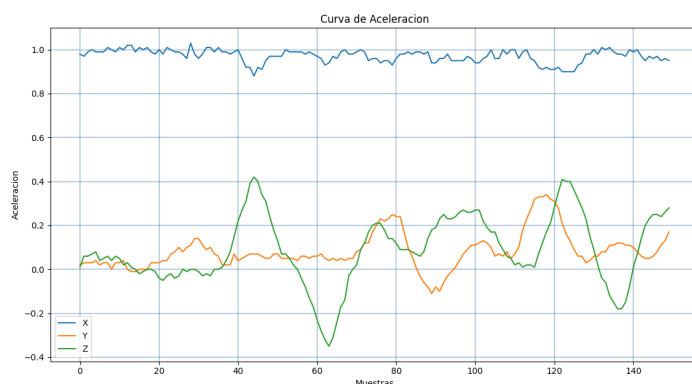


Figura 3.20: Curva con posición Z=0 X=1 Y=0

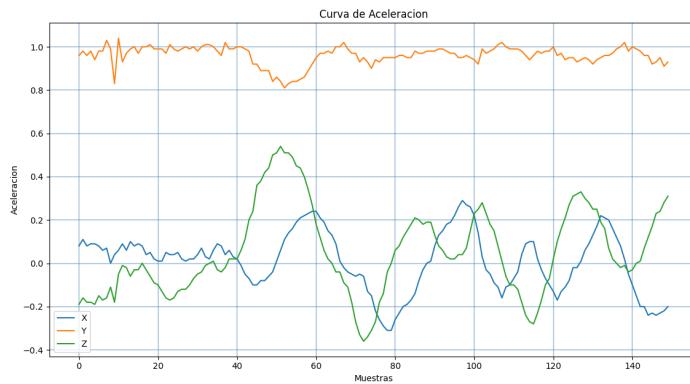


Figura 3.21: Curva con posición Z=0 X=0 Y=1

Ahora haremos una prueba con movimientos en todas las posiciones para ver una combinación de estos 3 ejes

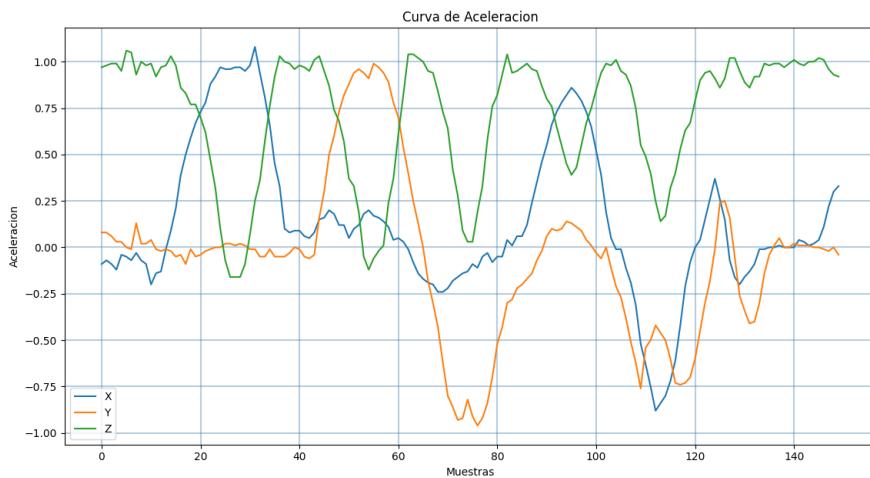


Figura 3.22: Curva con varias posiciones

De esta manera vimos como implementar los distintos algoritmos, todos para diferentes tareas pero con un fin en común, conectar MPU6050 con ESP32 y poder obtener los datos, reconocer nosotros mismo el patrón viendo la curva, una adaptación posterior en este proyecto, que veremos en secciones posteriores es para creación de los datos, dataset, necesarios para el entrenamiento y prueba de nuestra red neuronal.

Capítulo 4

Pantalla OLED

4.1. Tecnologías OLED

El comienzo del siglo XXI ha visto aparecer los diodos OLED (LEDs orgánicos), fabricados con materiales polímeros orgánicos semiconductores. Aunque la eficiencia lograda con estos dispositivos está lejos de la de los diodos inorgánicos, y son biodegradables, su fabricación promete ser considerablemente más barata que la de aquellos, siendo además posible depositar gran cantidad de diodos sobre cualquier superficie empleando técnicas de pintado para crear pantallas en color. El OLED (organic light-emitting diode: "diodo orgánico de emisión de luz") es un diodo basado en una capa electro-luminiscente que está formada por una película de componentes orgánicos que reaccionan a un determinado estímulo eléctrico, generando y emitiendo luz por sí mismos.

No se puede hablar realmente de una tecnología OLED, sino más bien de tecnologías basadas en OLED, ya que son varias las que hay, dependiendo del soporte y finalidad a la que vayan destinados. Su aplicación es realmente amplia, mucho más que cualquier otra tecnología existente. Pero, además, las tecnologías basadas en OLED no solo tienen una aplicación puramente como pantallas reproductoras de imagen, sino que su horizonte se amplía al campo de la iluminación, privacidad y otros múltiples usos que se le pueda dar.

Las ventajas de esta nueva tecnología son enormes, pero también tiene una serie de inconvenientes, aunque la mayoría de estos son totalmente circunstanciales y desaparecerán, en unos casos, conforme se siga investigando en este campo, y en otros, conforme vaya aumentando su uso y producción. Una solución tecnológica que pretende aprovechar las ventajas de la eficiencia alta de los LEDs típicos (hechos con materiales inorgánicos principalmente) y los costes menores de los OLED (derivados del uso de materiales orgánicos) son los sistemas de iluminación híbridos (orgánicos/inorgánicos) basados en diodos emisores de luz.

Un OLED está compuesto por dos finas capas orgánicas: una capa de emisión y una capa de conducción, que a la vez están comprendidas entre una fina película que hace de terminal ánodo y otra igual que hace de cátodo. En general estas capas están hechas de moléculas o polímeros que conducen la electricidad. Sus niveles de conductividad eléctrica se encuentran entre el nivel de un aislador y el de un conductor, y por ello se los llama semiconductores orgánicos. La elección de los materiales

orgánicos y la estructura de las capas determinan las características de funcionamiento del dispositivo: color emitido, tiempo de vida y eficiencia energética.

Un OLED consta de las siguientes partes:

Sustrato (plástico transparente, vidrio, lámina) - El sustrato soporta el OLED.

Ánodo (transparente): el ánodo elimina electrones (añade "agueros" de electrones) cuando una corriente fluye a través del dispositivo.

Capas orgánicas - Estas capas están hechas de moléculas orgánicas o polímeros.

Capa conductora - Esta capa está hecha de moléculas de plástico orgánico que transportan "agueros" desde el ánodo. Un polímero conductor utilizado en los OLED es la polianilina.

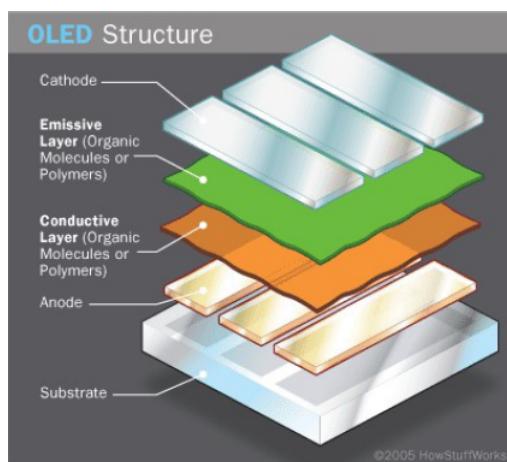


Figura 4.1: Estructura pantalla OLED

Los OLEDs emiten la luz de una manera similar a los LED, a través de un proceso llamado electrofotofluorescencia y el proceso es el siguiente:

- La batería o la fuente de alimentación del dispositivo que contiene el OLED aplica un voltaje a través del OLED.
- Una corriente eléctrica fluye desde el cátodo hasta el ánodo a través de las capas orgánicas (una corriente eléctrica es un flujo de electrones). El cátodo da electrones a la capa emisiva de moléculas orgánicas. El ánodo elimina electrones de la capa conductora de moléculas orgánicas. (Este es el equivalente a dar agujeros de electrones a la capa conductora).
- En el límite entre las capas emisiva y conductora, los electrones encuentran huecos de electrones. Cuando un electrón encuentra un hueco, el electrón llena el hueco (cae en un nivel de energía del átomo que falta un electrón). Cuando esto sucede, el electrón abandona la energía en forma de un fotón de luz (véase Cómo Funciona la Luz).
- El OLED emite luz.
- El color de la luz depende del tipo de molécula orgánica en la capa emisora. Los fabricantes colocan varios tipos de películas orgánicas en el mismo OLED para hacer presentaciones en color.

- La intensidad o el brillo de la luz depende de la cantidad de corriente eléctrica aplicada: cuanto más corriente, más brillante es la luz.

Los OLEDs emiten la luz de una manera similar a los LED, a través de un proceso llamado electrofotoforescencia y el proceso es el siguiente:

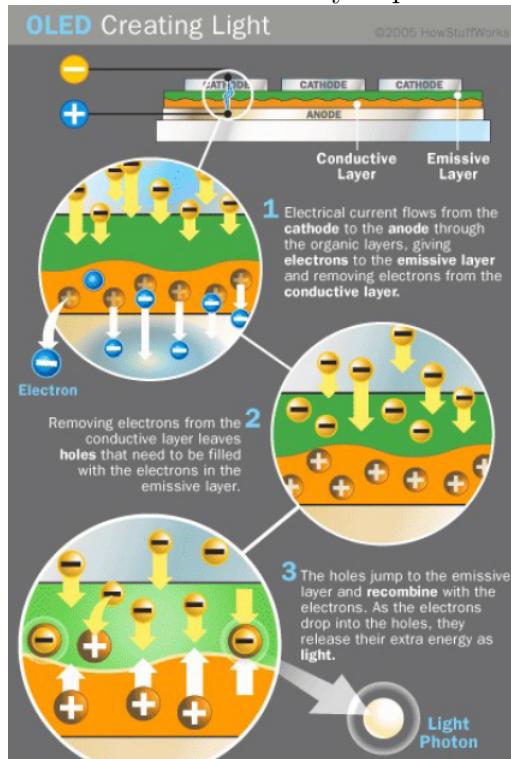


Figura 4.2: Proceso emisión de luz

Una de las ventajas de las pantallas OLED con respecto a pantalla LCD es que no requieren de una luz de fondo ni de filtros. Esto hace que las pantallas OLED sean más eficientes en términos de energía, más fáciles de fabricar y mucho más finas. A parte pueden ser flexibles y transparentes por ser muy delgadas, se comunican por I2C o SPI y producen una imagen más brillante y nítida que una pantalla LCD.

4.1.1. Controlador SSD1306

Las pantallas OLED no son más que eso, pantallas. Por si solas no hacen nada. Para poder mostrar datos e información en ellas se necesita un driver o controlador.

Normalmente viene todo ensamblado en un mismo módulo. Así es como lo encontrarás en las tiendas de electrónica. Sin embargo, debe quedar claro que en ese módulo está la pantalla OLED, el controlador y la electrónica necesaria para que todo funcione correctamente. Estos módulos utilizan un controlador bastante conocido que se llama SSD1306. Pero pueden utilizar otro tipo de controladores. Se trata de un potente controlador OLED CMOS.

Básicamente lo que hace el SSD1306 es comunicar con el microcontrolador para obtener los datos y enviarlos a la pantalla OLED para que dibuje esos datos.

La comunicación entre el SSD1306 y el microcontrolador, ya sea un Arduino o un ESP (en nuestro caso usaremos el ESP32), se realiza mediante SPI o I2C.

Gracias a que el SSD1306 es tan versátil, puedes encontrar pantallas OLED con Arduino con diferentes tamaños y colores. Los tamaños más típicos son los siguientes: De 0,69 y 128 x 32 píxeles y de 0,96 y 128 x 64 píxeles. El voltaje de operación del SSD1306 es de 1,65V a 3,3V. Todo esto es posible porque gracias a los dobladores de tensión y los circuitos de bomba de carga que permiten obtener voltajes superiores al voltaje de alimentación.

A parte, los módulos de pantalla OLED integran un regulador de tensión o LDO. Este regulador permite un voltaje de entrada entre 1,8V y 6V y mantiene un voltaje de salida de 3,3V constante.

Gracias a toda esta electrónica se puede alimentar un módulo de pantalla OLED que utilice un controlador SSD1306 con una alimentación de 3,3V o de 5V. Lo que lo hace viable para trabajar en nuestro proyecto.

4.1.2. Visualización de gráficos en pantalla OLED

Para mostrar los datos en la pantalla, el controlador SSD1306 tiene una memoria RAM gráfica que se llama GDDRAM (viene del inglés Graphic Display Data RAM) que ocupa 1 KB.

Esto equivale a 1.024 bytes o 8.192 bits que se distribuyen en la pantalla en una matriz de filas (páginas) y columnas (segmentos). En total hay 8 páginas (filas) y cada página tiene 128 segmentos (columnas) que, a su vez, cada segmento almacena 1 byte. Cada bit representa un píxel en la pantalla OLED y mediante la programación, se puede encender o apagar para que muestre cualquier información. Teniendo en cuenta que esta distribución es válida para las pantallas de 128 x 64. Las pantallas OLED de 128 x 32 solo muestran 4 páginas que equivalen a medio KB de la memoria RAM del SSD1306.

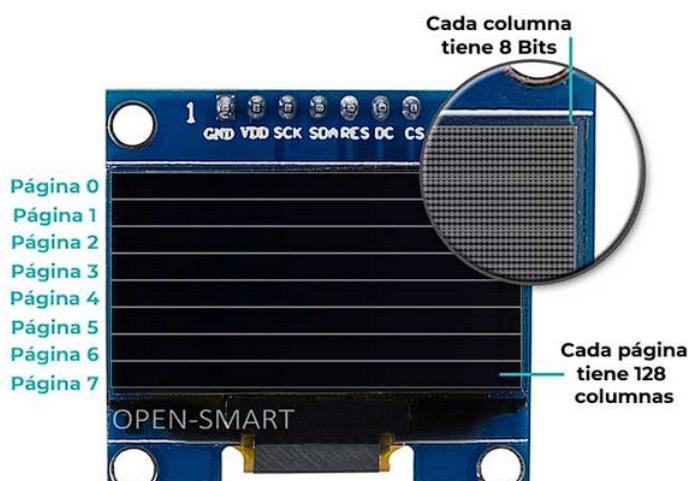


Figura 4.3: Matriz de la pantalla OLED

4.1.3. Pineado módulo pantalla OLED

Como dijimos anteriormente las pantallas OLED utilizan interfaz I2C y cuentan con 4 pines:

GND: pin de tierra.

VCC: es el pin de alimentación. Se puede alimentar la pantalla entre 1,8V y 6V.

SCL: es el pin de la señal de reloj de la interfaz I2C.

SDA: es el pin de la señal de datos de la interfaz I2C



Figura 4.4: Pines pantalla OLED

Para nuestro proyecto usaremos la pantalla OLED de 128 x 32 como se muestra en siguiente imagen:



Figura 4.5: Imagen UTN

4.1.4. Especificaciones Display Oled 128x32 SSD1306

- Voltaje de operación: 3V - 5.5V DC
- Driver: SSD1306
- Interfaz: I2C
- Resolución: 128*32 píxeles
- Monocromo: píxeles blancos (fondo negro)
- Ángulo de visión: 160°
- Área visible (display): 22.4*5.8 mm
- Consumo de energía ultra bajo: 0.04W (cuando están encendidos todos los píxeles)
- Temperatura de trabajo: -30°C - 70°C
- Dimensiones: 38*12*2.6 mm
- Peso: 4 gramos

4.2. Pruebas de pantalla SSD-1306

Para imprimir mensajes y tener el control que deseamos sobre nuestra pantalla OLED SSD-1306 con nuestro microcontrolador utilizamos el esquema de la figura 4.6

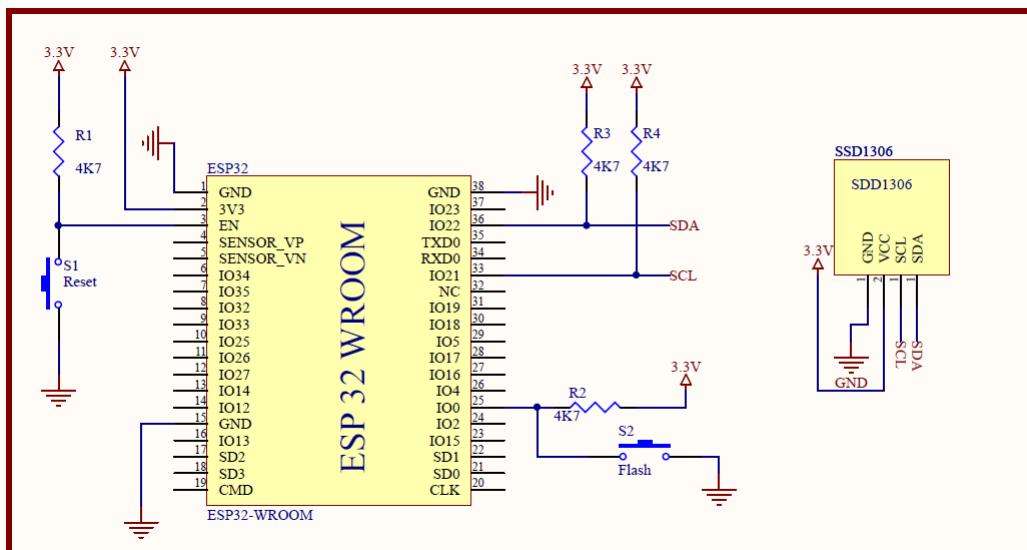


Figura 4.6: Esquematico ESP32 y Pantalla OLED

Para la implementación de la pantalla utilizamos la librería ssd1306, Ver aquí para descargas la librería. Las librerías en nuestro microcontrolador quedarían de la siguiente manera:

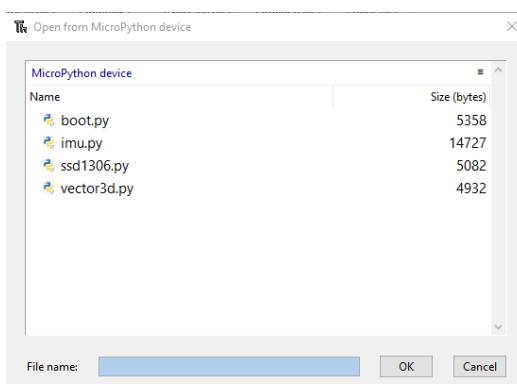


Figura 4.7: Librerías en ESP32

Nuestro programa es el siguiente:

```

1 """
2 Programa: Test de pantalla SSD1306
3
4 Proposito: Verificar funcionamiento
5 de pantalla oled
6
7 Programadores: Facundo Cosentino, Sergio Zelaya
8 Fecha: 21/08/2022
9 """
10 #Importamos Pin e I2C de machine
11 from machine import Pin, I2C
12 #Importamos ssd1306

```

```

13 import ssd1306
14
15 import time
16
17 #Configuramos comunicacion I2C
18 i2c = I2C(0,sda=Pin(21), scl=Pin(22), freq=400000)
19
20 #Configuramos pantalla ssd1306
21 display = ssd1306.SSD1306_I2C(128, 32, i2c)
22
23 #Mostramos mensaje
24 display.fill(0)
25 display.fill_rect(0, 0, 32, 32, 1)
26 display.fill_rect(2, 2, 28, 28, 0)
27 display.vline(9, 8, 22, 1)
28 display.vline(16, 2, 22, 1)
29 display.vline(23, 8, 22, 1)
30 display.fill_rect(26, 24, 2, 4, 1)
31 display.text('MicroPython', 40, 0, 1)
32 display.text('SSD1306', 40, 12, 1)
33 display.text('OLED 128x32', 40, 24, 1)
34 display.show()

```

Algoritmo 4.1: Test-esp32-ssd1306

La salida es la siguiente:



Figura 4.8: Print en pantalla OLED

4.3. Logo en pantalla

Para mostrar imágenes en nuestra pantalla debemos a nuestra imagen .jpg, .png, .jpeg, etc. convertirla a un arreglo de bytes, para ello se puede usar cualquier función en linea para este sencillo proceso de pasar de un formato imagen a un arreglo de bytes. por ejemplo, nuestra imagen a mostrar es la siguiente



Figura 4.9: Imagen a convertir

Y el arreglo de bytes quedaría de la siguiente forma:

```

25 #Cargamos logo de UTN
26 logo = [0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
27           0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

```

```
28      0xff, 0xff, 0x83, 0xf0, 0x7c, 0x0f, 0xc0,
29      0x3f, 0xf8, 0x03, 0xc0, 0x00, 0x00, 0x03,
30      0xc0, 0x1f, 0xfc, 0x03, 0x81, 0xf0, 0x7c,
31      0x0f, 0x80, 0x1f, 0xf8, 0x03, 0xc0, 0x00,
32      0x00, 0x03, 0xc0, 0x0f, 0xfc, 0x01, 0x81,
33      0xf0, 0x7c, 0x1f, 0x80, 0x1f, 0xf8, 0x03,
34      0xc0, 0x00, 0x00, 0x03, 0xc0, 0x0f, 0xfc,
35      0x01, 0xc1, 0xf0, 0x7c, 0x1f, 0x80, 0x1f,
36      0xf8, 0x03, 0xc0, 0x00, 0x00, 0x03, 0xc0,
37      0x07, 0xfc, 0x01, 0xc0, 0xf0, 0x78, 0x1f,
38      0x80, 0x1f, 0xf8, 0x03, 0xc0, 0x00, 0x00,
39      0x03, 0xc0, 0x03, 0xfc, 0x01, 0xc0, 0x70,
40      0x70, 0x3f, 0x80, 0x1f, 0xf8, 0x03, 0xc0,
41      0x00, 0x00, 0x03, 0xc0, 0x03, 0xfc, 0x01,
42      0xe0, 0x30, 0x60, 0x3f, 0x80, 0x1f, 0xf8,
43      0x03, 0xc0, 0x00, 0x00, 0x03, 0xc0, 0x01,
44      0xfc, 0x01, 0xf0, 0x00, 0x00, 0x7f, 0x80,
45      0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x07, 0xff,
46      0xc0, 0x00, 0xfc, 0x01, 0xf8, 0x00, 0x00,
47      0xff, 0x80, 0x1f, 0xf8, 0x03, 0xff, 0xe0,
48      0x0f, 0xff, 0xc0, 0x00, 0x7c, 0x01, 0xfc,
49      0x00, 0x01, 0xff, 0x80, 0x1f, 0xf8, 0x03,
50      0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x00, 0x7c,
51      0x01, 0xfe, 0x00, 0x03, 0xff, 0x80, 0x1f,
52      0xf8, 0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0,
53      0x00, 0x3c, 0x01, 0xff, 0x80, 0x0f, 0xff,
54      0x80, 0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x0f,
55      0xff, 0xc0, 0x00, 0x3c, 0x01, 0x80, 0x00,
56      0x00, 0x1f, 0x80, 0x1f, 0xf8, 0x03, 0xff,
57      0xe0, 0x0f, 0xff, 0xc0, 0x00, 0x1c, 0x01,
58      0x80, 0x00, 0x00, 0x0f, 0x80, 0x1f, 0xf8,
59      0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x00,
60      0x0c, 0x01, 0x80, 0x00, 0x00, 0x0f, 0x80,
61      0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x0f, 0xff,
62      0xc0, 0x00, 0x04, 0x01, 0x80, 0x00, 0x00,
63      0x0f, 0x80, 0x1f, 0xf8, 0x03, 0xff, 0xe0,
64      0x0f, 0xff, 0xc0, 0x10, 0x04, 0x01, 0x80,
65      0x00, 0x00, 0x0f, 0x80, 0x1f, 0xf8, 0x03,
66      0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x18, 0x00,
67      0x01, 0xff, 0x00, 0x0f, 0xff, 0x80, 0x1f,
68      0xf8, 0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0,
69      0x1c, 0x00, 0x01, 0xfe, 0x00, 0x07, 0xff,
70      0xc0, 0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x0f,
71      0xff, 0xc0, 0x1c, 0x00, 0x01, 0xfc, 0x00,
72      0x01, 0xff, 0xc0, 0x1f, 0xf8, 0x03, 0xff,
73      0xe0, 0x0f, 0xff, 0xc0, 0x1e, 0x00, 0x01,
74      0xf8, 0x00, 0x00, 0xff, 0xc0, 0x1f, 0xf0,
75      0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x1f,
76      0x00, 0x01, 0xf0, 0x00, 0x00, 0x7f, 0xc0,
77      0x0f, 0xf0, 0x07, 0xff, 0xe0, 0x0f, 0xff,
```

```

78      0xc0, 0x1f, 0x00, 0x01, 0xe0, 0x00, 0x00,
79      0x7f, 0xc0, 0x07, 0xe0, 0x07, 0xff, 0xe0,
80      0x0f, 0xff, 0xc0, 0x1f, 0x80, 0x01, 0xe0,
81      0x70, 0x70, 0x3f, 0xe0, 0x00, 0x00, 0x07,
82      0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x1f, 0xc0,
83      0x01, 0xc0, 0xf0, 0x78, 0x1f, 0xe0, 0x00,
84      0x00, 0x0f, 0xff, 0xe0, 0x0f, 0xff, 0xc0,
85      0x1f, 0xc0, 0x01, 0xc1, 0xf0, 0x78, 0x1f,
86      0xf0, 0x00, 0x00, 0x0f, 0xff, 0xe0, 0x0f,
87      0xff, 0xc0, 0x1f, 0xe0, 0x01, 0x81, 0xf0,
88      0x7c, 0x1f, 0xf8, 0x00, 0x00, 0x1f, 0xff,
89      0xe0, 0x0f, 0xff, 0xc0, 0x1f, 0xf0, 0x01,
90      0x81, 0xf0, 0x7c, 0x1f, 0xfc, 0x00, 0x00,
91      0x3f, 0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x1f,
92      0xf0, 0x01, 0x83, 0xf0, 0x7c, 0x0f, 0xfe,
93      0x00, 0x00, 0xff, 0xff, 0xe0, 0x0f, 0xff,
94      0xc0, 0x1f, 0xf8, 0x01, 0xc3, 0xf0, 0x7e,
95      0x1f, 0xff, 0xc0, 0x03, 0xff, 0xff, 0xf0,
96      0x0f, 0xff, 0xc0, 0x3f, 0xfc, 0x03, 0xff,
97      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
98      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
99      0xff]

```

y para mostrarlo seria tan sencillo como :

```

16  #Importamos Pin e I2C de machine
17 from machine import Pin, I2C
18 #Importamos ssd1306
19 import ssd1306
20 #Importamos framebuf
21 import framebuf

.

99 #Configuramos comunicacion I2C
100 I2c = I2C(0,sda=Pin(21),scl=Pin(22),freq=400000)

.

103 #Configuramos pantalla ssd1306
104 display = ssd1306.SSD1306_I2C(128, 32, I2c)

.

.

107 #Mostramos logo
108 bf = bytearray(logo)
109 fb = framebuf.FrameBuffer(bf,128,32,framebuf.MONO_HLSB)
110 display.fill(0)
111 display.blit(fb,0,0)
112 display.show()

```

y obtenemos la siguiente salida:



Figura 4.10: Salida de imagen UTN

4.4. Implementación MPU6050 y SSD1306

En el siguiente código veremos todo lo antes mencionado para poder conectar la pantalla y el sensor MPU6050 con el microcontrolador y ver el valor de las variables en los 3 ejes. el circuito que haremos será el siguiente:

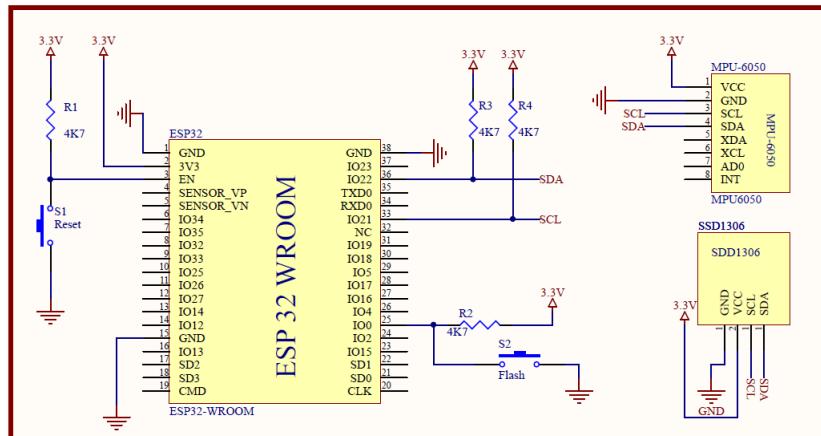


Figura 4.11: Circuito ESP32 MPU6050 SSD1306

El algoritmo usado es el siguiente:

```

1 /**
2 Programa: test ssd mpu
3
4 Proposito: Verificar el correcto complemento
5 de interfaz visual que nos da la pantalla OLED
6 y mostrar los valores obtenidos de MPU6050
7
8 Programadores: Facundo Cosentino, Sergio Zelaya
9 Fecha: 23/08/2022
10 */
11
12 #Importamos MPU_6050 de imu
13 from imu import MPU6050
14 #Importamos time para los delay
15 import time
16 #Importamos Pin e I2C de machine
17 from machine import Pin, I2C
18 #Importamos ssd1306
19 import ssd1306
20 #Importamos framebuffer

```

```

21 import framebuffer
22
23 #Cargamos logo de UTN
24 logo = [0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
25         0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
26         0xff, 0xff, 0x83, 0xf0, 0x7c, 0x0f, 0xc0,
27         0x3f, 0xf8, 0x03, 0xc0, 0x00, 0x00, 0x03,
28         0xc0, 0x1f, 0xfc, 0x03, 0x81, 0xf0, 0x7c,
29         0x0f, 0x80, 0x1f, 0xf8, 0x03, 0xc0, 0x00,
30         0x00, 0x03, 0xc0, 0x0f, 0xfc, 0x01, 0x81,
31         0xf0, 0x7c, 0x1f, 0x80, 0x1f, 0xf8, 0x03,
32         0xc0, 0x00, 0x00, 0x03, 0xc0, 0x0f, 0xfc,
33         0x01, 0xc1, 0xf0, 0x7c, 0x1f, 0x80, 0x1f,
34         0xf8, 0x03, 0xc0, 0x00, 0x00, 0x03, 0xc0,
35         0x07, 0xfc, 0x01, 0xc0, 0xf0, 0x78, 0x1f,
36         0x80, 0x1f, 0xf8, 0x03, 0xc0, 0x00, 0x00,
37         0x03, 0xc0, 0x03, 0xfc, 0x01, 0xc0, 0x70,
38         0x70, 0x3f, 0x80, 0x1f, 0xf8, 0x03, 0xc0,
39         0x00, 0x00, 0x03, 0xc0, 0x03, 0xfc, 0x01,
40         0xe0, 0x30, 0x60, 0x3f, 0x80, 0x1f, 0xf8,
41         0x03, 0xc0, 0x00, 0x00, 0x03, 0xc0, 0x01,
42         0xfc, 0x01, 0xf0, 0x00, 0x00, 0x7f, 0x80,
43         0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x07, 0xff,
44         0xc0, 0x00, 0xfc, 0x01, 0xf8, 0x00, 0x00,
45         0xff, 0x80, 0x1f, 0xf8, 0x03, 0xff, 0xe0,
46         0x0f, 0xff, 0xc0, 0x00, 0x7c, 0x01, 0xfc,
47         0x00, 0x01, 0xff, 0x80, 0x1f, 0xf8, 0x03,
48         0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x00, 0x7c,
49         0x01, 0xfe, 0x00, 0x03, 0xff, 0x80, 0x1f,
50         0xf8, 0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0,
51         0x00, 0x3c, 0x01, 0xff, 0x80, 0x0f, 0xff,
52         0x80, 0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x0f,
53         0xff, 0xc0, 0x00, 0x3c, 0x01, 0x80, 0x00,
54         0x00, 0x1f, 0x80, 0x1f, 0xf8, 0x03, 0xff,
55         0xe0, 0x0f, 0xff, 0xc0, 0x00, 0x1c, 0x01,
56         0x80, 0x00, 0x00, 0x0f, 0x80, 0x1f, 0xf8,
57         0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x00,
58         0x0c, 0x01, 0x80, 0x00, 0x00, 0x0f, 0x80,
59         0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x0f, 0xff,
60         0xc0, 0x00, 0x04, 0x01, 0x80, 0x00, 0x00,
61         0x0f, 0x80, 0x1f, 0xf8, 0x03, 0xff, 0xe0,
62         0x0f, 0xff, 0xc0, 0x10, 0x04, 0x01, 0x80,
63         0x00, 0x00, 0x0f, 0x80, 0x1f, 0xf8, 0x03,
64         0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x18, 0x00,
65         0x01, 0xff, 0x00, 0x0f, 0xff, 0x80, 0x1f,
66         0xf8, 0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0,
67         0x1c, 0x00, 0x01, 0xfe, 0x00, 0x07, 0xff,
68         0xc0, 0x1f, 0xf8, 0x03, 0xff, 0xe0, 0x0f,
69         0xff, 0xc0, 0x1c, 0x00, 0x01, 0xfc, 0x00,
70         0x01, 0xff, 0xc0, 0x1f, 0xf8, 0x03, 0xff,
71         0xe0, 0x0f, 0xff, 0xc0, 0x1e, 0x00, 0x01,
72         0xf8, 0x00, 0x00, 0xff, 0xc0, 0x1f, 0xf0,
73         0x03, 0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x1f,
74         0x00, 0x01, 0xf0, 0x00, 0x00, 0x7f, 0xc0,
75         0x0f, 0xf0, 0x07, 0xff, 0xe0, 0x0f, 0xff,
76         0xc0, 0x1f, 0x00, 0x01, 0xe0, 0x00, 0x00,
77         0x7f, 0xc0, 0x07, 0xe0, 0x07, 0xff, 0xe0,
78         0x0f, 0xff, 0xc0, 0x1f, 0x80, 0x01, 0xe0,
79         0x70, 0x70, 0x3f, 0xe0, 0x00, 0x00, 0x07,
80         0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x1f, 0xc0,

```

```

81      0x01, 0xc0, 0xf0, 0x78, 0x1f, 0xe0, 0x00,
82      0x00, 0x0f, 0xff, 0xe0, 0x0f, 0xff, 0xc0,
83      0x1f, 0xc0, 0x01, 0xc1, 0xf0, 0x78, 0x1f,
84      0xf0, 0x00, 0x00, 0x0f, 0xff, 0xe0, 0x0f,
85      0xff, 0xc0, 0x1f, 0xe0, 0x01, 0x81, 0xf0,
86      0x7c, 0x1f, 0xf8, 0x00, 0x00, 0x1f, 0xff,
87      0xe0, 0x0f, 0xff, 0xc0, 0x1f, 0xf0, 0x01,
88      0x81, 0xf0, 0x7c, 0x1f, 0xfc, 0x00, 0x00,
89      0x3f, 0xff, 0xe0, 0x0f, 0xff, 0xc0, 0x1f,
90      0xf0, 0x01, 0x83, 0xf0, 0x7c, 0x0f, 0xfe,
91      0x00, 0x00, 0xff, 0xe0, 0x0f, 0x0f, 0xff,
92      0xc0, 0x1f, 0xf8, 0x01, 0xc3, 0xf0, 0x7e,
93      0x1f, 0xff, 0xc0, 0x03, 0xff, 0xff, 0xf0,
94      0x0f, 0xff, 0xc0, 0x3f, 0xfc, 0x03, 0xff,
95      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
96      0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
97      0xff]
98
99
100 #Configuramos comunicacion I2C
101 I2c = I2C(0,sda=Pin(21),scl=Pin(22),freq=400000)
102 #Configuramos MPU6050
103 imu = MPU6050(I2c)
104 #Configuramos pantalla ssd1306
105 display = ssd1306.SSD1306_I2C(128, 32, I2c)
106
107
108 #Mostramos logo
109 bf = bytearray(logo)
110 fb = framebuffer.FrameBuffer(bf,128,32,framebuf.MONO_HLSB)
111 display.fill(0)
112 display.blit(fb,0,0)
113 display.show()
114 time.sleep(3)
115
116 #Mostramos Titulo
117 display.fill(0)
118 display.text('VER VALORES', 0, 0, 1)
119 display.text('DE LOS 3 EJES', 0, 10, 1)
120 display.text('X Y Z', 0, 20, 1)
121 display.show()
122 time.sleep(3)
123
124 while True:
125     # Leemos Aceleracion en X
126     Gx = imu.accel.x - 0.06112061
127     # Leemos Aceleracion en Y
128     Gy = imu.accel.y + 0.03150146
129     # Leemos Aceleracion en Z
130     Gz = imu.accel.z + 0.07965576
131
132     #Mostramos mensaje
133     display.fill(0)
134     display.text('X:', 0, 0, 1)
135     display.text('Y:', 50, 0, 1)
136     display.text('Z:', 90, 0, 1)
137     display.text(str(round(Gx,1)), 0, 20, 1)
138     display.text(str(round(Gy,1)), 50, 20, 1)
139     display.text(str(round(Gz,1)), 90, 20, 1)
140     display.show()

```

```
141      time.sleep(0.05)          #Esperamos un 1 segundo  
142
```

Algoritmo 4.2: Implementacion ssd1306 y mpu6050

la salida de todo el algoritmo es el valor de las variables como se ve en la figura.

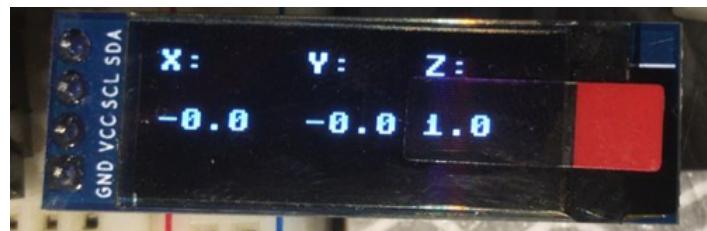


Figura 4.12: Salida del algoritmo 4.2

Capítulo 5

Microcontrolador ESP32-wroom

5.1. Descripción y Características

5.1.1. Wifi y Bluetooth en un solo chip

ESP32 es una serie de SoC (por sus siglas en inglés, System on Chip) y módulos de bajo costo y bajo consumo de energía creado por Espressif Systems.

El ESP-WROOM-32 es un potente módulo que integra WiFi y Bluetooth, ideal para desarrollar productos de IoT. La integración de Bluetooth, Bluetooth LE y Wi-Fi permite una amplia gama de aplicaciones, el uso de WiFi permite una comunicación de mediano alcance y conectarse a una red LAN y a través de un Modem Router conexión a Internet, mientras que el Bluetooth nos permite conectarse directamente a otro dispositivo como un celular. La corriente de reposo del chip ESP32 es inferior a 5 A, por lo que es adecuado para aplicaciones de electrónica portátiles con batería. En el núcleo de este módulo está el SoC ESP32-D0WDQ6. El chip integrado está diseñado para ser escalable y adaptado. Hay dos núcleos de CPU que se pueden controlar individualmente, y la frecuencia del reloj es ajustable de 80 MHz a 240 MHz. El usuario también puede apagar el CPU y utilizar el co-procesador de baja potencia para supervisar constantemente los periféricos para detectar cambios de estado. ESP32 integra un amplio conjunto de periféricos como sensores táctiles capacitivos, sensores Hall, amplificadores de bajo nivel de ruido, interfaz para SD, Ethernet, SPI, UART, I2S e I2C. El módulo ESP-WROOM-32 trabaja a 3.3V en alimentación y GPIO por lo que NO se debe alimentar con 5V. Para ello le colocamos un capacitor de 100uF en paralelo con la fuente de alimentación para filtrar los picos de corriente. Los pines de entradas/salidas (GPIO) trabajan a 3.3V por lo que para la conexión a sistemas de 5V es necesario utilizar conversores de nivel como: Conversor de nivel 3.3-5V 4CH o Conversor de nivel bidireccional 8CH - TXS0108E.

Además de todo eso, logra un consumo de energía muy bajo a través de funciones de ahorro de energía que incluyen sincronización de reloj y múltiples modos de operación. Todo esto lo convierte en la herramienta ideal para tus proyectos energizados con baterías o aplicaciones IoT.

Los ESP32 poseen un alto nivel de integración. En su pequeño encapsulado se incluyen:

- interruptores de antena.
- balun de RF.
- amplificador de potencia.
- amplificador de recepción de bajo ruido.
- filtros y módulos de administración de energía.

5.1.2. Especificaciones

A continuación las especificaciones del microcontrolador elegido:

- Microcontrolador Tensilica LX6 de doble núcleo de 240 MHz con 600 DMIPS
- SRAM integrada de 520 KB
- Transceptor Wi-Fi 802.11b/g/n HT40 integrado, banda base, pila y LWIP
- Bluetooth de modo dual integrado (clásico y BLE)
- Flash de 4 MByte incluido en el módulo WROOM32
- Antena PCB integrada
- Amplificador analógico de ultra bajo ruido
- sensor de pasillo
- Interfaz táctil capacitiva 10x
- oscilador de cristal de 32 kHz
- 3 x UART (solo dos están configurados de manera predeterminada en el soporte de Feather Arduino IDE, un UART se usa para cargar/depurar)
- 3 x SPI (solo uno está configurado por defecto en el soporte Feather Arduino IDE)
- 2 x I2C (solo uno está configurado por defecto en el soporte Feather Arduino IDE)
- 12 canales de entrada ADC
- 2 audios I2S
- 2 DAC
- PWM/entrada/salida de temporizador disponible en cada pin GPIO.
- Interfaz de depuración OpenOCD con búfer TRAX de 32 kB
- SDIO maestro/esclavo 50 MHz
- Soporte de interfaz de tarjeta SD

5.2. Pin Out y conexiones básicas

5.2.1. Pin Out ESP32wroom

Pin Out: descripción de cada pin del ESP32:

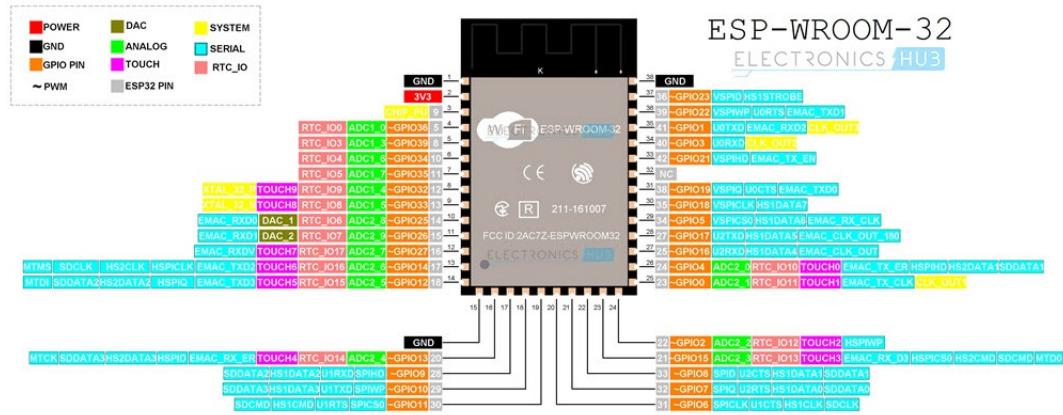


Figura 5.1: Pin Out ESP32 wroom

*Los pines SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 y SCS/CMD, es decir, GPIO6 a GPIO11 están conectados al flash SPI integrado en el módulo y no se recomiendan para otros usos.

5.2.2. Conexionado básico

A continuación veremos el conexionado básico entre cada uno de los pines usados para que el ESP32 funcione correctamente:

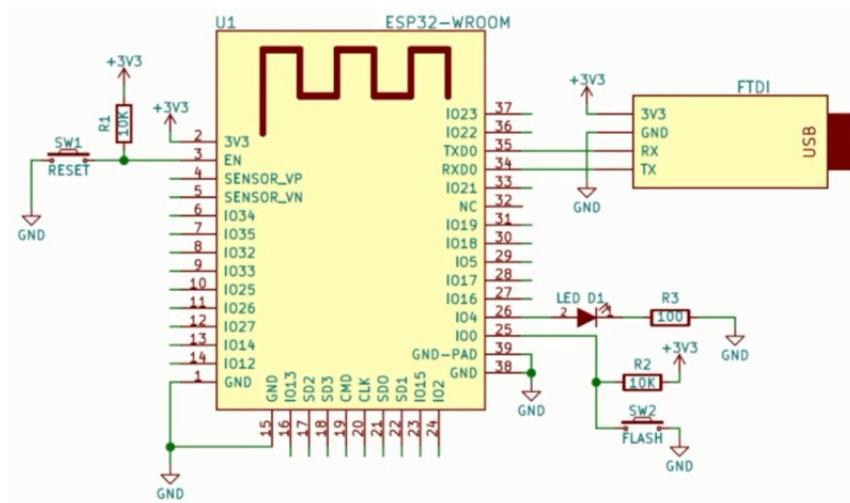


Figura 5.2: Configuración básica del ESP32 wroom

5.3. Diagrama circuitual completo

5.3.1. Partición y secciones del circuito

CONEXIONES DE LA FUENTE DE ALIMENTACIÓN:

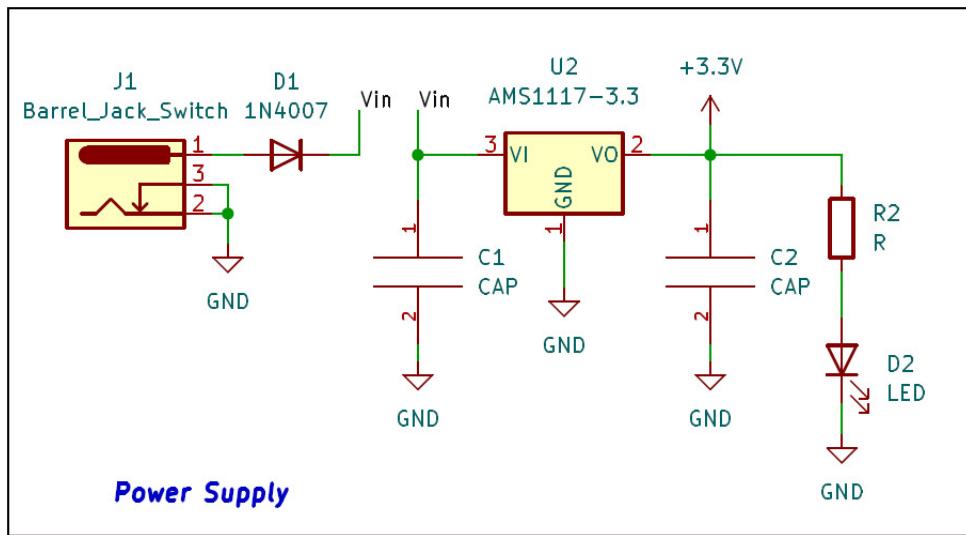


Figura 5.3: Conexiónado de la fuente de alimentación

SECCIÓN DE COMUNICACIONES USB-UART:

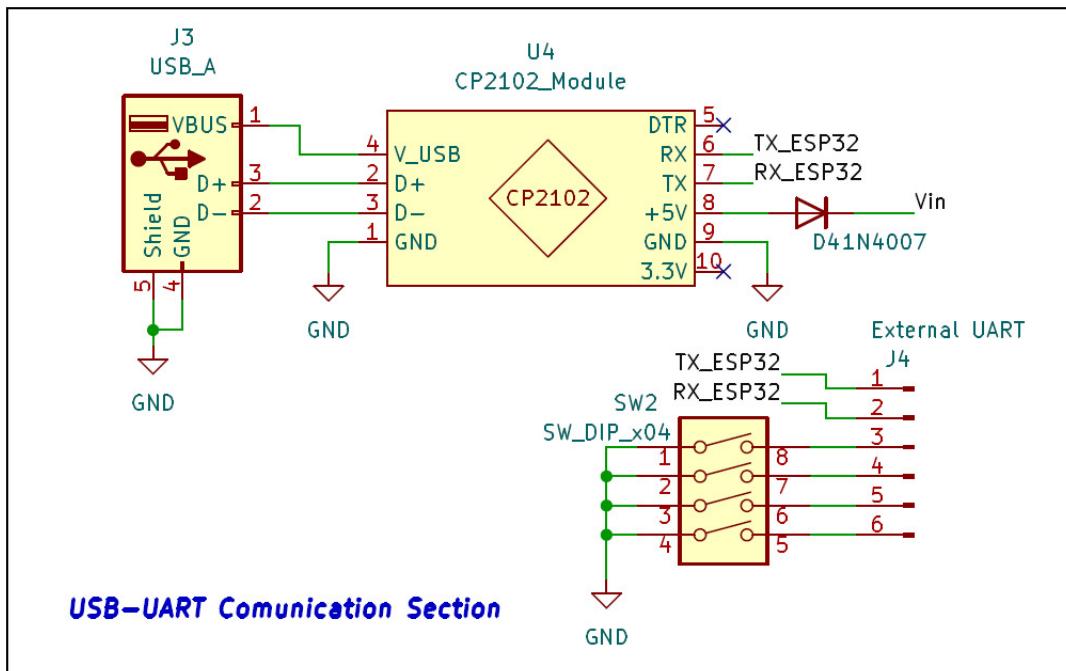


Figura 5.4: Conexiónado para las comunicaciones USB-UART

CONFIGURACIÓN USADA DEL ESP32WROOM:

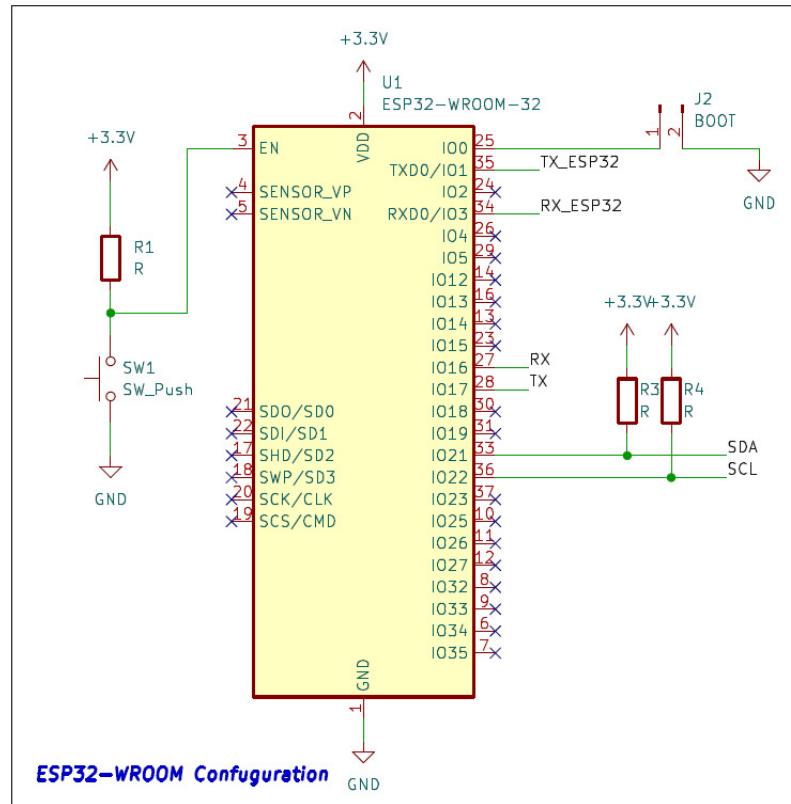


Figura 5.5: Configuración usada para conectar el ESP32wroom

PERIFÉRICOS UTILIZADOS:

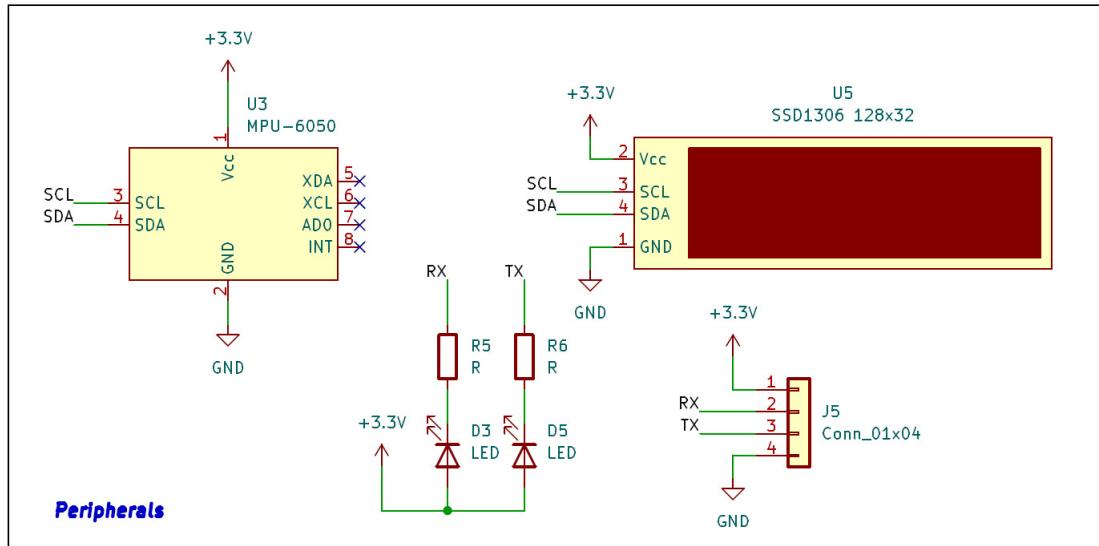


Figura 5.6: Periféricos usados en el circuito final

5.3.2. Diagrama Final

Implementation of neural networks in ESP32

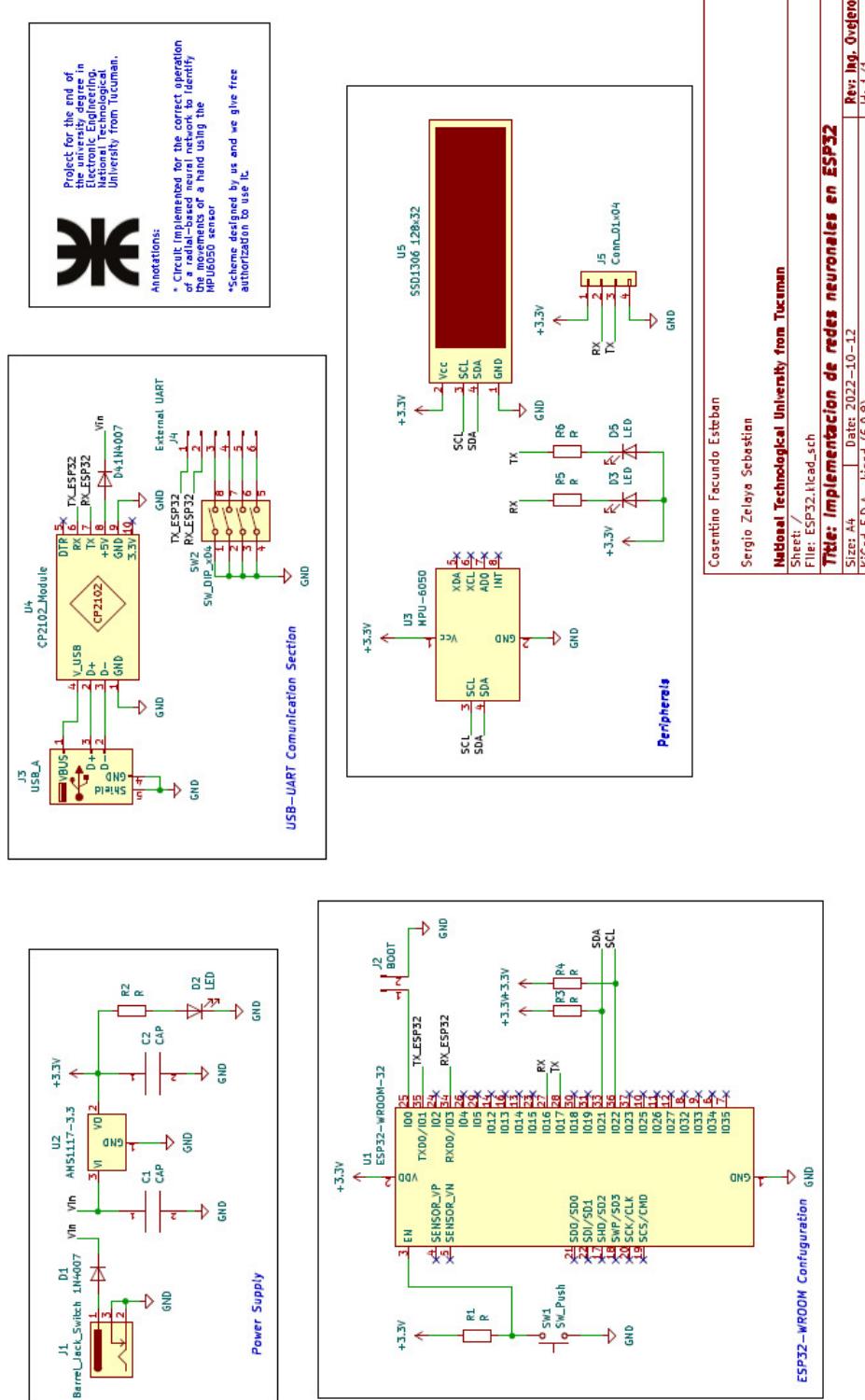


Figura 5.7: Circuito final implementado con ESP32wroom

5.3.3. Diseño de PCB

Para el diseño del circuito impreso utilizamos el software KiCAD EDA ya que nos resultó mucho más fácil el diseño esquemático del circuito electrónico y su conversión a placa de circuito impreso.

Usamos una placa de doble cara para construir el circuito, a continuación veremos ambos lados de la plaqueta con sus respectivas dimensiones y la disposición de los componentes electrónicos.

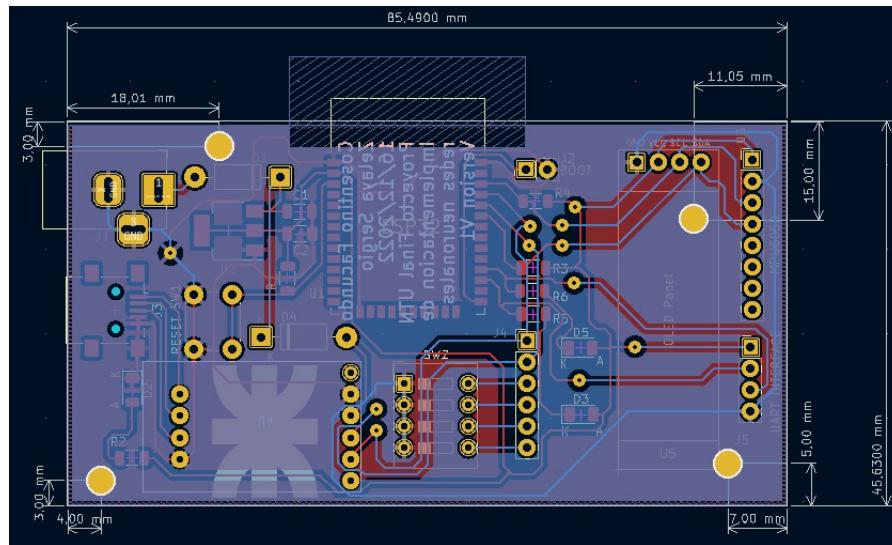


Figura 5.8: PCB vista de cara inferior

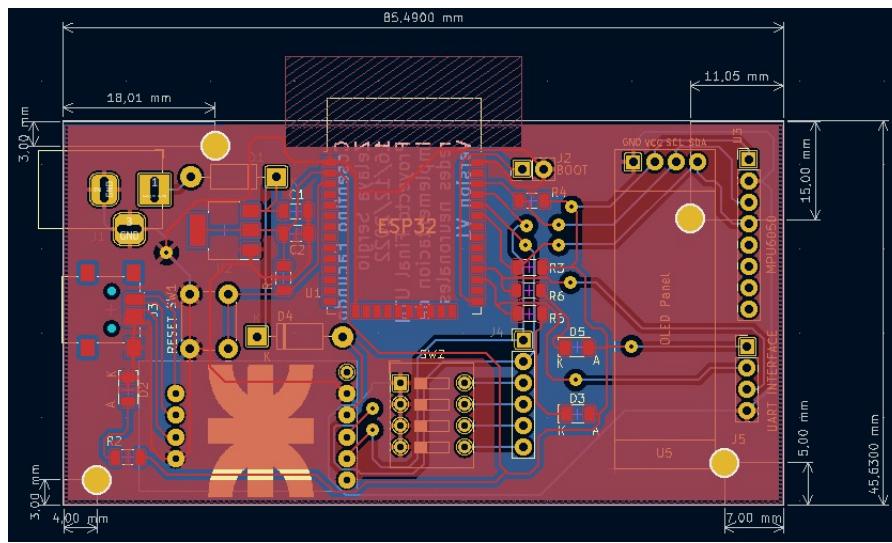


Figura 5.9: PCB vista de cara superior

El software con el que trabajamos nos permite visualizar como quedaría la placa con sus respectivos componentes, la siguiente imagen es una captura de pantalla de la misma:

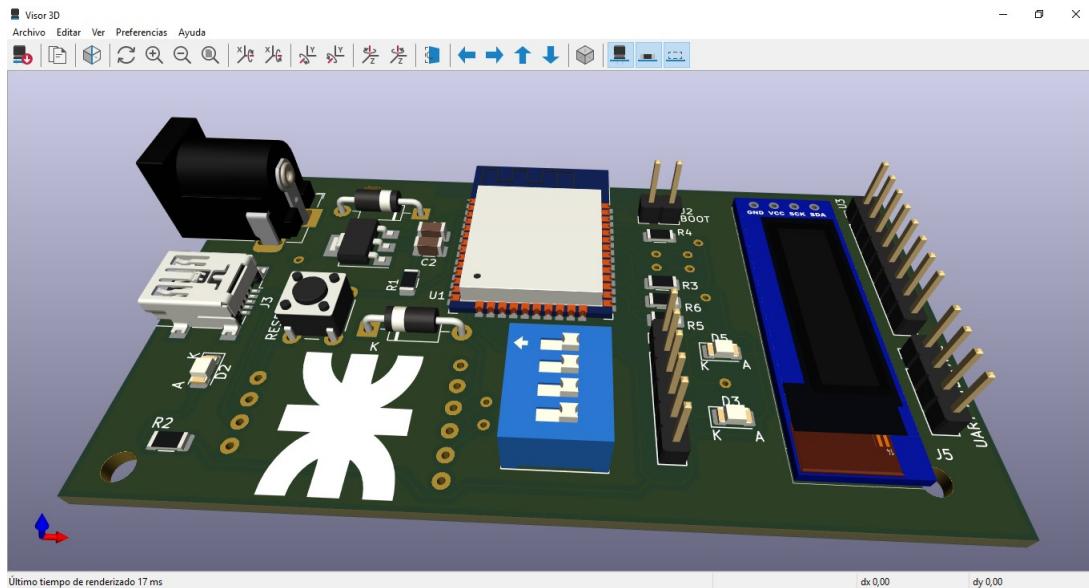


Figura 5.10: Vista 3D del proyecto terminado con los componentes montados

5.3.4. Armado y montaje final

Después de muchas horas de trabajo logramos realizar el armado completo de nuestro proyecto, al cual le adaptamos una base para mayor comodidad al usarlo:

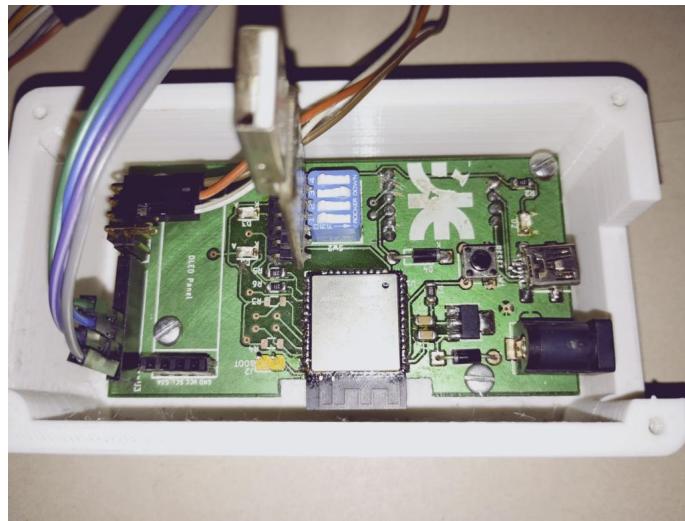


Figura 5.11: Plaqueta física terminada preparada para la toma de datos con conexión a PC



Figura 5.12: Plaqueta lista y entrenada para su funcionamiento



Figura 5.13: Adición de guante para posterior prueba de funcionamiento



Figura 5.14: Trabajo final terminado y funcionando correctamente

Capítulo 6

Adquisición de los datos

6.1. Data Set

6.1.1. Definición

Un dataset, como su nombre indica, es un conjunto de datos, que habitualmente están estructurados, como ejemplo podríamos decir que una tabla de una base de datos de SQL sería un dataset, en el que cada columna de la tabla corresponde a una variable y las filas representan los diferentes registros que almacena cada una de las columnas o variables de la tabla. Estas filas y columnas, junto con los valores, conforman el dataset o conjunto de datos en cuestión, un ejemplo podría ser la tabla de una base de datos de una empresa que vende zapatillas de deporte, que recoge en las columnas variables como Talla, marca, categoría o precio, y en las filas estarían los valores que corresponden a cada par de zapatillas. El conjunto de datos que conforman la tabla que recoge esta información sería un dataset, un conjunto de datos.

6.1.2. Tipos de Datasets

Tipos de Datasets Existen cuatro tipos de Datasets catalogados según su origen y formato, los cuales son usados según las necesidades de los modelos de datos a trabajar.

Archivo: es un fichero independiente en el que se almacena toda la información con la que se va a trabajar. Tiene como ventajas, la seguridad y rapidez para el trabajo con los datos, ya que siempre se explotan y se visualizan de manera local, sin embargo, la escalabilidad y conexión con otros Datasets que no están almacenados en la misma máquina se dificulta.

Folder: es la suma de diferentes Datasets almacenados en una misma carpeta, los cuales están conectados entre ellos. Estos archivos deben compartir un mismo formato como puede ser .csv, .mif o dxf.

Bases de datos: este tipo de Dataset puede llegar a confundir con el archivo, pero se diferencia por su nivel de especialidad, es decir, son bases de datos con formatos específicos diseñadas para programas puntuales. Por ejemplo, las bases de datos de Oracle, las cuales solo funcionan para sus desarrollos.

Web: es la compilación de datos que se almacenan dentro de un

sitio web. El nombre que se le asigna por defecto a este Dataset es el correspondiente a la URL.

6.2. Creación del conjunto de datos

6.2.1. ¿Por qué necesito un conjunto de datos?

En los proyectos de Machine Learning, necesitamos un conjunto de datos de capacitación. Es el conjunto de datos real utilizado para entrenar el modelo para realizar varias acciones. ML depende en gran medida de los datos, sin datos, es imposible que una “AI” aprenda. Es el aspecto más crucial que hace posible el entrenamiento de algoritmos. No importa cuán grande sea el equipo de AI o el tamaño del conjunto de datos, si el conjunto de datos no es lo suficientemente bueno, todo el proyecto de AI fracasará.

Durante un desarrollo de AI, siempre confiamos en los datos. Desde la capacitación, el ajuste, la selección de modelos hasta las pruebas, utilizamos tres conjuntos de datos diferentes: el conjunto de capacitación, el conjunto de validación y el conjunto de pruebas. Teniendo en cuenta que los conjuntos de validación se utilizan para seleccionar y ajustar el modelo ML final.

Se puede llegar a pensar que la recopilación de datos es suficiente, pero es lo contrario. En todos los proyectos de AI, la clasificación y el etiquetado de conjuntos de datos nos lleva la mayor parte del tiempo, especialmente los conjuntos de datos lo suficientemente precisos para reflejar una visión realista del mercado / mundo.

Entonces vamos a necesitar 2 conjuntos de datos: el conjunto de datos de entrenamiento y el conjunto de datos de prueba ya que se usan para diferentes propósitos durante el proyecto y el éxito de un proyecto depende mucho de ellos.

1.-El conjunto de datos de entrenamiento es el que se utiliza para entrenar a un Algoritmo para entender cómo aplicar conceptos como las redes neuronales, para aprender y producir resultados. Incluye tanto los datos de entrada como los resultados esperados.

2.-El conjunto de datos de prueba se utiliza para evaluar qué tan bien se entrenó su algoritmo con el conjunto de datos de entrenamiento . En los proyectos de AI, no podemos usar el conjunto de datos de entrenamiento en la etapa de prueba porque el algoritmo ya sabrá por adelantado el resultado esperado, que no es nuestro objetivo.

Durante el proceso se debe tener en cuenta de no realizar un ajuste excesivo, es un error de modelo que ocurre cuando una función se ajusta demasiado a un conjunto limitado de puntos de datos. Para ello se evita ajustar pasada la fase de prueba.

6.2.2. Procesamiento de Datos

Una vez reunido los datos necesarios, esenciales, diversos y mas representativos para el proyecto continuamos con el procesamiento de los mismos que incluye la selección correcta de datos y la creación de un

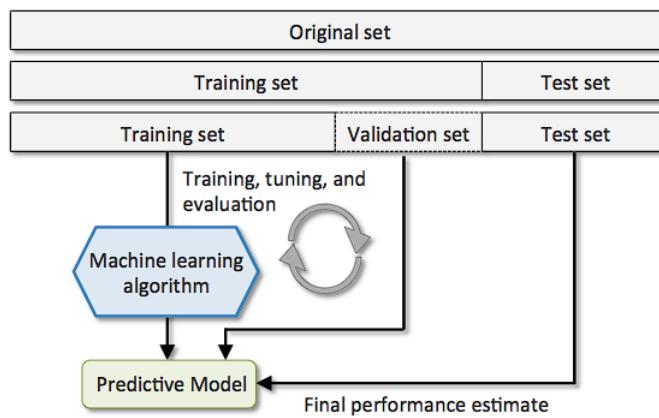


Figura 6.1: Proceso del Data Set

conjunto de capacitación. El proceso de reunir los datos en este formato óptimo se conoce como **transformación de características**.

1. Formato: Los datos se pueden distribuir en diferentes archivos. Por ejemplo, los resultados de ventas de diferentes países con diferentes monedas, idiomas, etc., que deben reunirse para formar un conjunto de datos.

2. Limpieza de datos: En este paso, nuestro objetivo es lidiar con valores perdidos y eliminar caracteres no deseados de los datos.

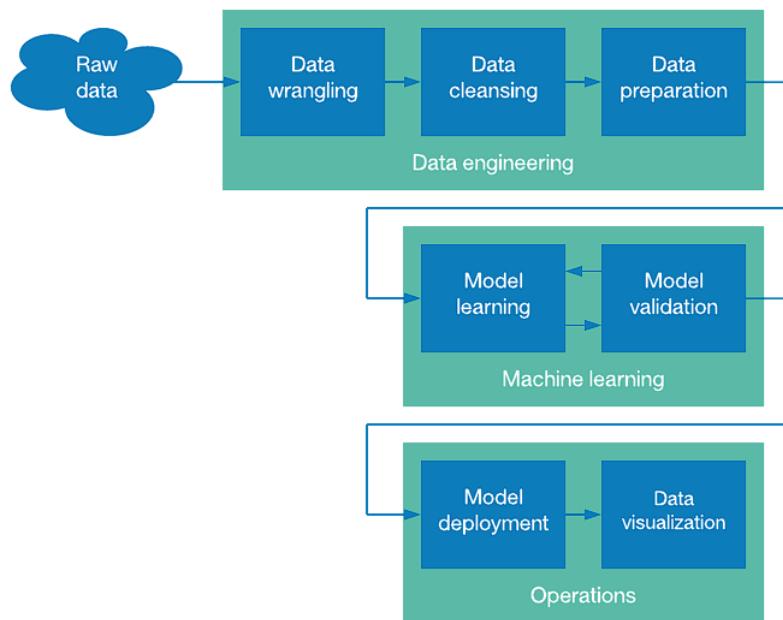


Figura 6.2: Procesamiento de Datos

3. Extracción de características: En este paso, nos centramos en el análisis y la optimización del número de características. Por lo general, un miembro del equipo tiene que averiguar qué características son importantes para la predicción y seleccionarlas para cálculos más rápidos y bajo consumo de memoria. (ver figura 6.2)

6.2.3. Objetivo a largo plazo: La estrategia perfecta

Los proyectos de AI más exitosos son aquellos que Integrar una estrategia de recolección de datos durante el ciclo de vida del producto / servicio. De hecho, la recopilación de datos no puede ser una serie de ejercicios únicos. Debe estar integrado en el producto central en sí. Básicamente, cada vez que un usuario se involucra con su producto / servicio, desea recopilar datos de la interacción. El objetivo es utilizar este flujo de datos nuevo y constante para mejorar su producto / servicio. Cuando alcanza este nivel de uso de datos, cada nuevo cliente que agrega hace que el conjunto de datos sea más grande y, por lo tanto, el producto sea mejor, lo que atrae a más clientes. lo que hace que el conjunto de datos sea mejor, y así sucesivamente. Es una especie de círculo positivo.

Los mejores proyectos de ML orientados a largo plazo son aquellos que aprovechan conjuntos de datos dinámicos y constantemente actualizados. La ventaja de crear dicha estrategia de recopilación de datos es que se vuelve muy difícil para los competidores replicar nuestro conjunto de datos. Con los datos, la AI mejora y, en algunos casos, como el filtrado colaborativo, es muy valioso. El filtrado colaborativo hace sugerencias basadas en la similitud entre los usuarios, mejorará con el acceso a más datos; cuantos más datos de usuario tenga uno, más probable será que el algoritmo pueda encontrar un usuario similar. Esto significa que se necesita una estrategia para la mejora continua del conjunto de datos siempre que haya algún beneficio para el usuario para un mejor modelo.

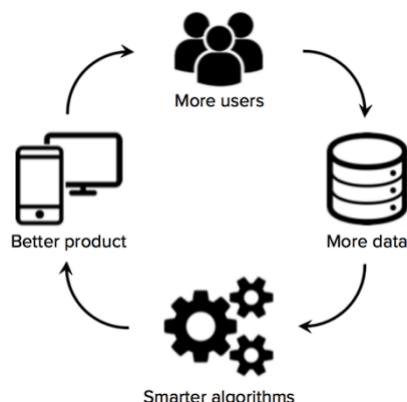


Figura 6.3: El proceso ideal para nuestro proyecto

Todo este contenido se publicó originalmente por Alexandre Gonfalonieri. (2019). “How to Build A Data Set For Your Machine Learning Project”¹.

6.3. Normalizar en Data Science

Ahora explicaremos sobre la normalización de datos como una introducción general al proceso de preparación de datos para el modelado de ML y como una forma de ayudar a los lectores a comprender la importancia de considerar la escala y magnitud de las características en el modelado de ML.

¹Link al documento

Particularmente en nuestro trabajo los datos del sensor MPU6050 tienen la misma escala y magnitud, entonces no es necesario realizar la normalización de datos. Dicha normalización se utiliza principalmente para evitar que las diferencias de escala en las características dominen el modelo de ML y para mejorar la eficacia de los algoritmos de aprendizaje.

Cuando las características tienen la misma escala y magnitud, el modelo de ML puede aprender de manera efectiva las relaciones entre ellas sin la necesidad de normalizar los datos. En este caso, la normalización de datos no tendría un impacto significativo en el rendimiento del modelo de ML. Por lo tanto, en este trabajo, justificamos que no se realizó la normalización de datos debido a que los datos en X y Y tienen la misma escala y magnitud, lo que significa que no hay una diferencia significativa en la escala de las características que pueda afectar la eficacia del modelo de ML.

Para que funcionen mejor muchos algoritmos de ML usados en Data Science, hay que normalizar las variables de entrada al algoritmo. Normalizar significa, en este caso, comprimir o extender los valores de la variable para que estén en un rango definido. Sin embargo, una mala aplicación de la normalización, o una elección descuidada del método de normalización puede arruinar tus datos, y con ello tu análisis. Vamos a ver unos ejemplos de los métodos de normalización más usados actualmente.

6.3.1. Métodos más usados

MÉTODO 1: Escalado de variables, Feature Scaling o Min-Max Scaler.

En este caso, cada entrada se normaliza entre unos límites definidos:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \in [0, 1]$$

El problema de este tipo de normalización, es que comprime los datos de entrada entre unos límites empíricos (el máximo y el mínimo de la variable). Esto quiere decir que si existe ruido, éste va a ser ampliado. Es decir, tomando como ejemplo una señal estable como la de la figura:

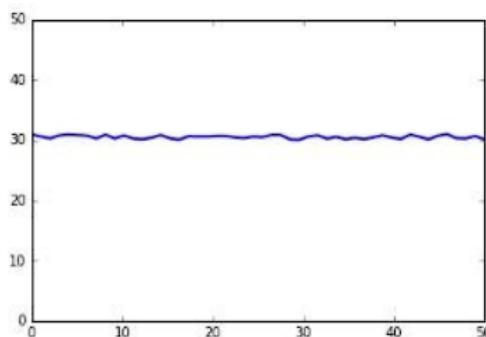


Figura 6.4: A modo de ejemplificar, se toman distintos valores próximos a 30, considerando un período de 50 partes (horas, días, etc).

Vamos a proceder a escalar esta variable en el rango 0-1, usando como máximo y mínimo los máximos y mínimos de la señal.

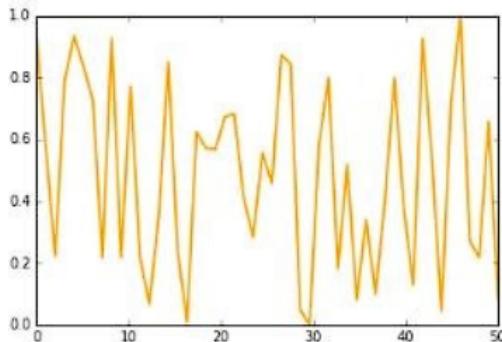


Figura 6.5: Misma gráfica pero tomando como referencia los máximos y los mínimos en el rango de 0 a 1.

Después del escalado, nuestros datos se han distorsionado. Lo que era una gráfica estable, ahora parece tener muchas variaciones. Esto nos dice que este método de normalización no es adecuado para señales estables.

MÉTODO 2: Escalado estándar (Standard Scaler)

Una alternativa al escalado de variables es usar otra técnica conocida como escalado estándar (a cada dato se le resta la media de la variable y se le divide por la desviación estándar)

$$X_{standard} = \frac{X - \mu}{\sigma}$$

Los datos estadísticos que se usan (media y desviación típica) son muy sensibles a valores anómalos (muy grandes o muy pequeños con respecto al resto). Veamos un ejemplo de toma de datos en la que tenemos picos (datos alejados de la media):

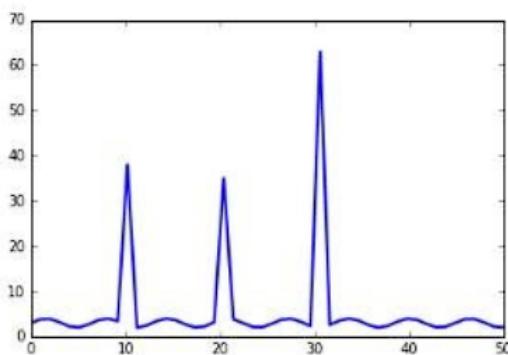


Figura 6.6: Ejemplo de gráfica tomando muestras alejadas a la media.

Antes de normalizar, calculamos la media (5.55) y la desviación típica (10.53). Ya podemos ver que la media está en torno a 5, cuando nuestros datos sin anomalías no pasan de valores en torno al 4 (mala señal). Si aplicamos ahora la normalización estándar, tenemos lo siguiente:

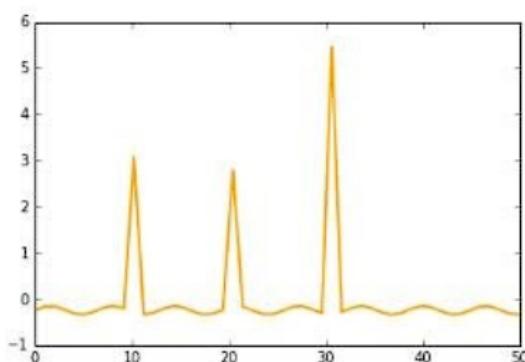


Figura 6.7: La normalización estándar para los datos anteriores no es una buena elección

Como podemos ver con este método no hemos conseguido normalizar entre 0-1. Además ahora tenemos valores negativos, cuando antes no los teníamos. Por si esto fuera poco, nuestros valores pico y valle han quedado muy atenuados por culpa de las anomalías. Una solución a esto sería eliminar las anomalías antes de normalizar.

Por lo tanto, no existe un método ideal de normalización que funcione para todas las formas de variables. Es trabajo del Data Scientist conocer cómo se distribuyen los datos, saber si existen anomalías, comprobar rangos, etc. Con este conocimiento, se puede seleccionar la mejor técnica para no distorsionar los datos.

Contenido escrito por Santiago Morante, PhD, Científico de Datos en LUCA (2018). ²

²Link al documento

6.4. Lectura y Envío de datos

El diagrama en flujo representado en la figura 6.8 se muestra como hacemos la adquisición de los datos dados por el sensor MPU6050, los procesamos para darle un determinado formato y enviarlos por UART a la pc para un procesamiento posterior

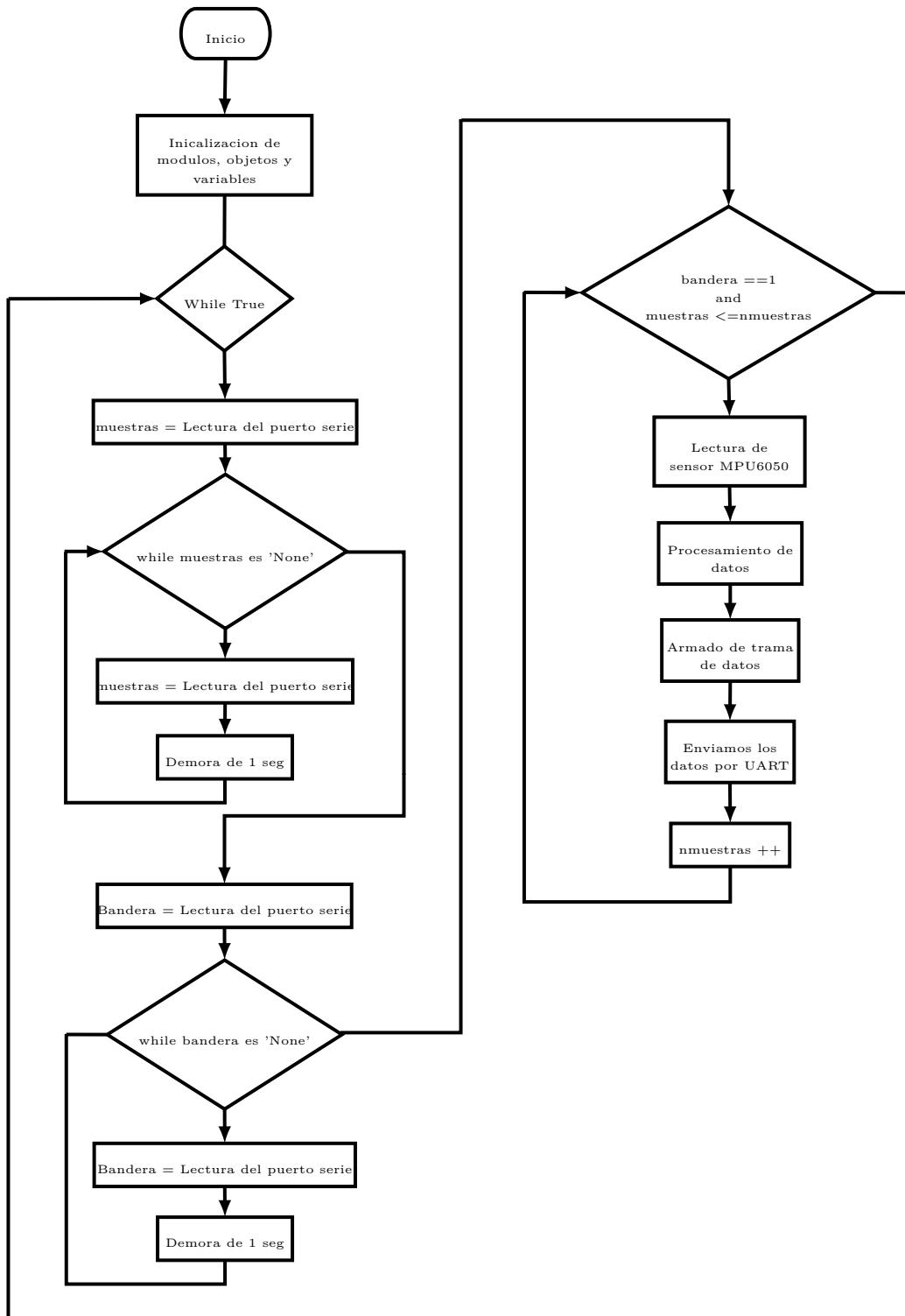


Figura 6.8: Diagrama en flujo de Python-Made-DataSet

```
1 '''
2 Programa: PYTHON-MADE-DATASET
3
4 Programadores: Facundo Cosentino, Sergio Zelaya
5 Fecha: 2/08/2022
6
7 Rectificacion: de "PLOT-PYTHON" para enviar
8 cantidad de muestras por puerto serie.
9
10 Fecha: 11/09/2022
11
12 '''
13
14 # Importamos MPU_6050 de imu
15 from imu import MPU6050
16 # Importamos time para los delay
17 import time
18 # Importamos Pin e I2C de machine
19 from machine import Pin, I2C, UART
20
21
22 # Configuramos comunicacion I2C
23 I2c = I2C(0,sda=Pin(21),scl=Pin(22),freq=400000)
24 # Configuramos MPU6050
25 imu = MPU6050(I2c)
26 # Configuracion UART 9600baudios y el modulo 2
27 uart= UART(2,9600)
28
29 # Variable para guardar el tiempo anterior
30 tiempoant=0
31 # Variable para controlar la cantidad de muestras
32 nmuestras=0
33 # Variable para contar el numero de muestras
34 muestras=0
35 # Variable para guardar la trama
36 trama=' '
37
38 while True:
39
40     # Esperamos una cantidad de muestras deseadas
41     nmuestras = uart.readline()
42     while nmuestras == None:
43         nmuestras = uart.readline()
44         time.sleep(1)
45     # convertimos la nmuestras de byte a str y luego a int
46     nmuestras = int(nmuestras)
47
48     # Esperamos una bandera para empezar a enviar datos
49     flag = uart.readline()
50     while flag == None:
51         flag = uart.readline()
52         time.sleep(1)
53
54     # convertimos la bandera de byte a str y luego a int
55     flag = int(flag)
56
57     # inicializamos las muestras en 0
58     muestras=0
59
60     # leeremos 150 muestras luego de tener
```

```
61 #la bandera en '1'
62 while flag == 1 and muestras<=nmuestras:
63     #Medimos el tiempo en el que estamos
64     tiempoact = time.ticks_ms()
65
66     #Vemos si ya paso 100ms entre la ultima medida
67     if(tiempoact-tiempoant >= 100):
68         #Actualizamos el tiempo
69         tiempoant = time.ticks_ms()
70
71     #Leemos Aceleracion en X
72     Gx = imu.accel.x-0.06112061
73
74     #Leemos Aceleracion en Y
75     Gy = imu.accel.y+0.03150146
76
77     #Leemos Aceleracion en Z
78     Gz = imu.accel.z+0.07965576
79
80     #Damos formato al dato
81     datax = str(int(Gx*100))
82     datay = str(int(Gy*100))
83     dataz = str(int(Gz*100))
84
85     #Armamos la trama de datos
86     trama=datax+','+datay+','+dataz
87     #Enviamos Trama de dato
88     uart.write(trama)
89     #Enviamos salto de linea
90     uart.write('\n')
91
92     #Imprimimos el numero de muestras
93     print(muestras)
94
95     #Incrementamos el numero de
96     #muestras recibidas
97     muestras = muestras + 1
```

Algoritmo 6.1: Python-Made-Dataset

El diagrama en flujo representado en la figura 6.9 y 6.10 se muestra como hacemos para leer, descomponer, guardar y agrupar todos los dataset creados.

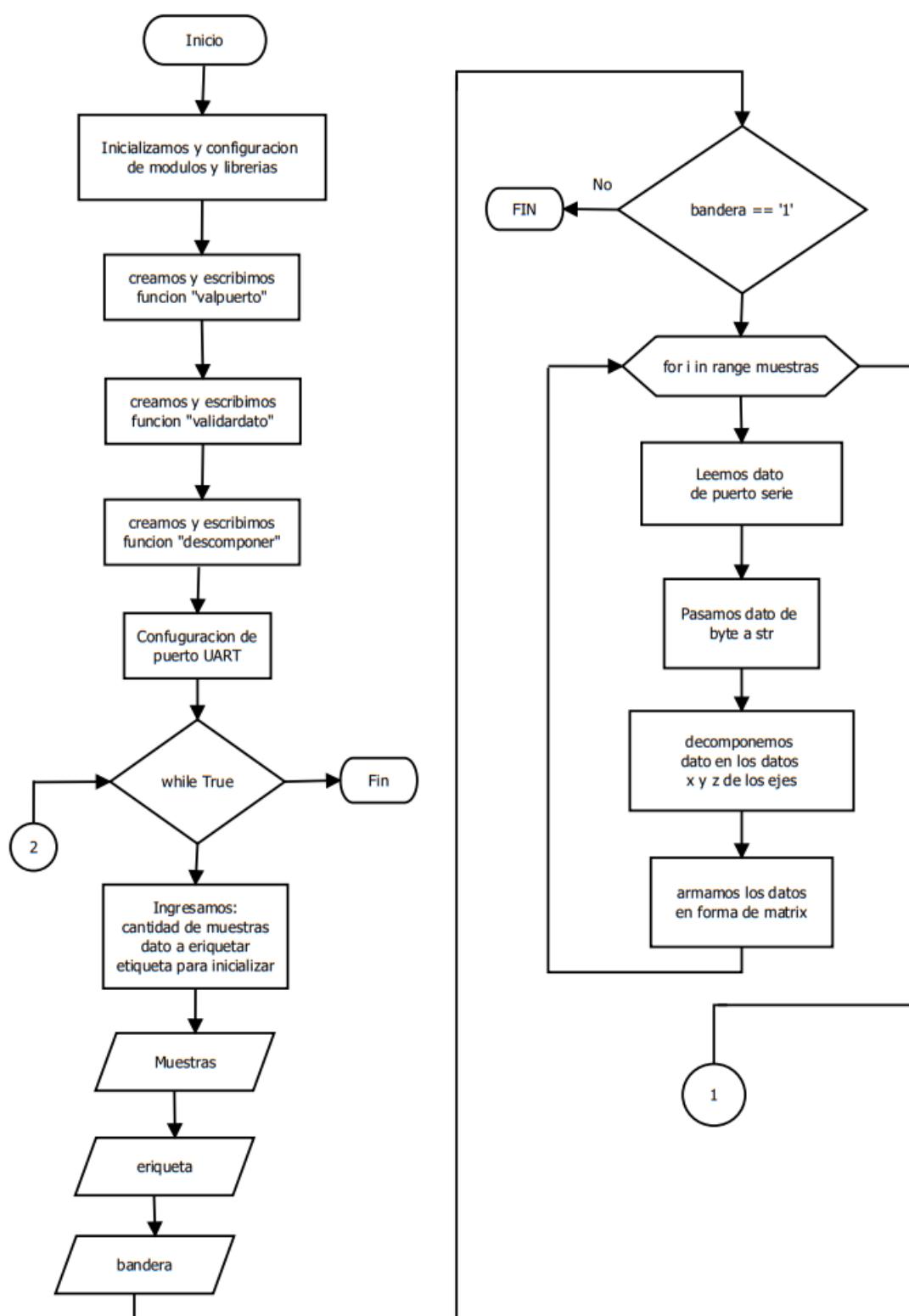


Figura 6.9: 1er parte Diagrama en flujo de Read and Save data

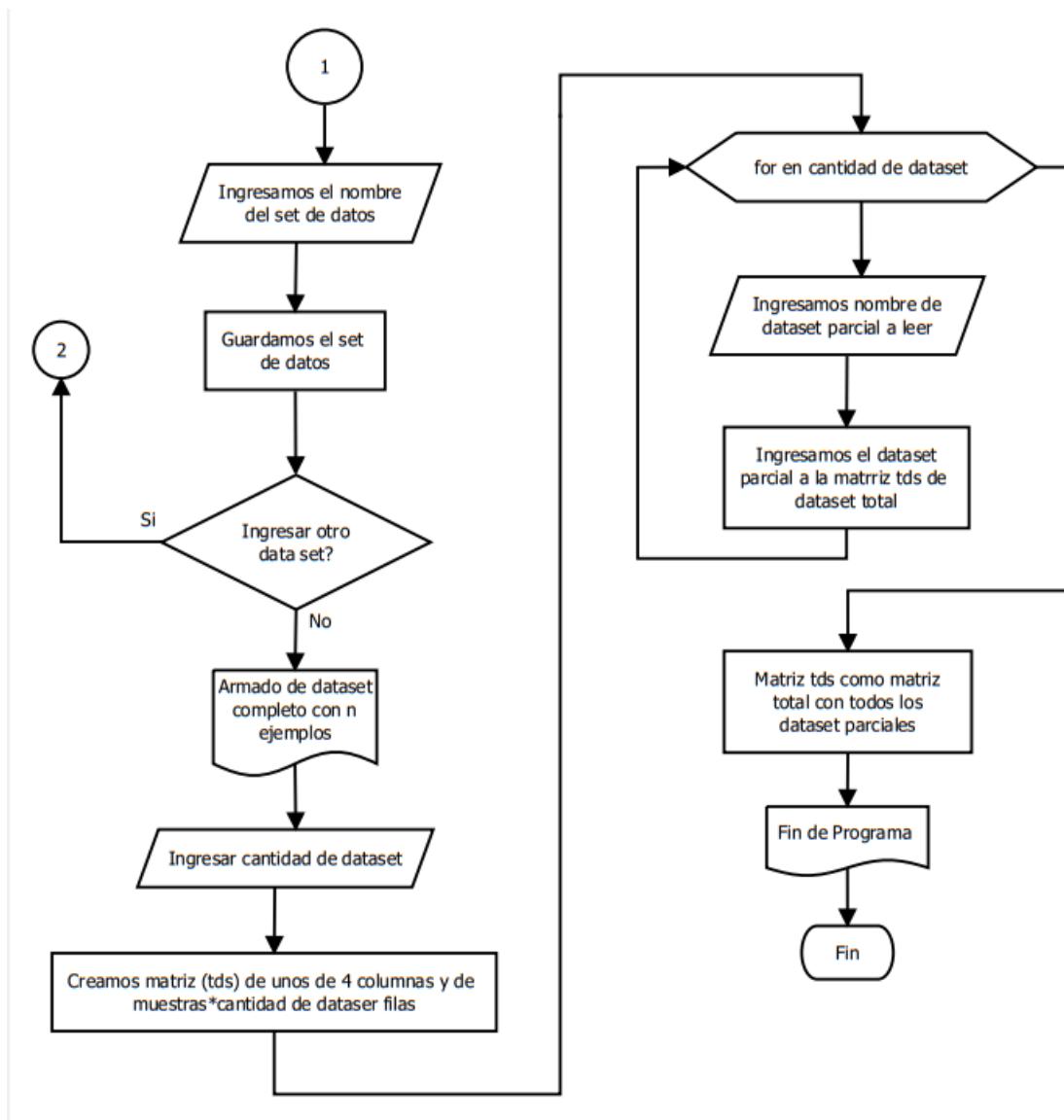


Figura 6.10: 2da parte Diagrama en flujo de Read and Save data

```

1 """
2 Programa: READ AND SAVE DATA SET
3
4 Proposito: Recepcion de datos de sensor MPU-6050
5 para determinados casos de pruebas y guardar
6 el data set
7
8 Programadores: Facundo Cosentino, Sergio Zelaya
9 Fecha: 17/09/2022
10 """
11
12 """
13 -----
14 Librerias a Utilizar
15 -----
16 """
17
18 #Libreria para manejo de Tiempos
19 import time

```

```
20 #Libreria para manejo de UART
21 import serial
22 #Libreria para manejo de arrays
23 import numpy as np
24 #Libreria para hacer graficas
25 import matplotlib.pyplot as plt
26
27 '''
28 -----
29 Declaracion de variable
30 -----
31 '''
32 #Cantidad de muestras
33 #muestras=150
34
35 #Bandera para controlar ciclos
36 #de recoleccion de datos
37 rp=1
38
39 #Bandera para controlar ciclos
40 #de opciones
41 rs=1
42
43 '''
44 -----
45 Configuracion de comunicacion UART
46 -----
47 '''
48
49 def valpuerto():
50     #Elejimos el puerto y lo configuramos
51
52     portx='COM'
53     puertos=[]
54     comsok=[]
55     flagcont=0
56
57     for i in range (100):
58         ports = portx+str(i)
59         puertos.append(ports)
60
61     for port in puertos:
62         try:
63             ser = serial.Serial(port)
64             ser.close()
65             comsok.append(port)
66         except(OSError, FileNotFoundError):
67             pass
68
69     while flagcont == 0:
70         print("-----")
71         print("Seleccione puerto a conectarse")
72         print("-----")
73
74         for i in range (len(comsok)):
75             print("Presionar: "+str(i)+" para "+comsok[i])
76             print("\n")
77             opt=int(input("Seleccione puerto: "))
78
79             if opt < len(comsok):
```

```

80         flagcont = 1
81
82     else:
83         print("Puerto no valido\n")
84         time.sleep(1)
85     comf=comsok[opt]
86     return comf
87
88 """
89 -----
90 Validacion del nombre de dataset
91 -----
92 """
93 def validardato():
94
95     rt=1
96
97     while (rt==1):
98         print("\n")
99         nombre = input("Ingrese Nombre: ")
100
101        try:
102            ndataset = np.load(nombre+'.npy')
103            rt=0
104            print("Exito")
105            return(nombre)
106        except(OSError,FileNotFoundException):
107            print("El datosrt "+nombre+" no existe")
108            print("Intente de nuevo")
109
110 """
111 -----
112 Funcion para descomponer String
113 -----
114 """
115
116 def descomponer(cadena): #ingresa cadena
117
118     #Remplazamos los 'b por ' '
119     cadena = cadena.replace("b'"," ")
120     #Remplazamos los ' por ' '
121     cadena = cadena.replace("'", " ")
122     #Remplazamos los \\n por ' '
123     cadena = cadena.replace("\\\\n", " ")
124     #Remplazamos los espacion en blanco
125     #logrando borrarlos de la cadena
126     cadena = cadena.replace(" ", "")
127
128     #convertimos este string en un entero
129     return int(cadena)
130
131 """
132 -----
133 Programa principal
134 -----
135 """
136
137 puerto = valpuerto()
138 ser = serial.Serial()
139 ser.port = puerto

```

```
140 ser.baudrate = 9600
141 ser.open()
142
143 while True:
144
145     print("\n")
146     print("-----")
147     print("Inicio de programa")
148     print("-----")
149     print("\n")
150
151 muestras=input('Ingrese cantidad de muestras: ')
152 bmuestras=muestras+'\n'
153 bmuestras=bmuestras.encode()
154
155 ser.flushInput() #Vaciamos el buffer serial
156
157 ser.write(bmuestras) #Mandamos cantidad
158                 #de muestra que queremos
159
160 muestras=int(muestras)
161
162 print("\n")
163 out=int(input('dato a etiquetar: '))
164 print("\n")
165 entrada=input('Ingrese 1 para proceder: ')
166
167
168
169 ser.write(b'1\n') #Mandamos una bandera para
170             #saber que debe empezar
171             #a transmitir datos
172
173 '''
174 -----
175 Declaracion de variable en tiempo de
176 ejecucion
177 -----
178 '''
179
180 #Array avacio para eje X, Y, Z
181 x = np.ones(muestras).reshape(muestras,1)
182 y = np.ones(muestras).reshape(muestras,1)
183 z = np.ones(muestras).reshape(muestras,1)
184
185 #Array avacio para etiqueta
186 aout = np.ones(muestras).reshape(muestras,1)
187
188 #Base de tiempo
189 t = np.arange(0,muestras/10, 0.1, dtype=float)
190 t = t.reshape(muestras,1)
191 #Numero de muestras
192 tm=np.arange(0,muestras, 1, dtype=float)
193 tm=tm.reshape(muestras,1)
194
195 if(entrada == '1'):
196     #Iteramos la cantidad de muestras
197     for i in range(muestras):
198         #Leemos el puerto serie
199         dato = ser.readline()
```

```

200         #convertimos el dato tipo byte
201         #en un tipo string
202         cadena = str(dato)
203
204         #separamos los datos X Y Z
205         ex,ey,ez = cadena.split(sep=',')
206
207         #--- Convertimos X Y Z a tipo int ---
208
209         cadena=str(ex)
210         ex = descomponer(cadena)
211         cadena=str(ey)
212         ey = descomponer(cadena)
213         cadena=str(ez)
214         ez = descomponer(cadena)
215
216         #---Armamos los arreglos ---
217         x[i,0]=ex/100      #Armamos el arreglo X
218         y[i,0]=ey/100      #Armamos el arreglo Y
219         z[i,0]=ez/100      #Armamos el arreglo Z
220         aout[i,0]=out       #Armamos la etiquetas
221
222         ...
223         -----
224         Armamos y guardamos un data set
225         -----
226         ...
227         print("\n")
228         print("-----")
229         print("Guardamos el dataset parcial")
230         print("-----")
231         print("\n")
232
233
234         dataset = np.concatenate((x, y, z,aout), axis=1)
235         nombre=input("Ingresar nombre del conjunto de
236         datos: ")
237         np.save(nombre, dataset)
238         print(dataset)
239
240         ...
241         -----
242         Control de flujo del programa
243         -----
244         ...
245         rs=1
246         while rs==1:
247             print("\n")
248             print("-----")
249             print("Seleccionar como proceder")
250             print("-----")
251             print("\n")
252             print("1 para volver a crear un dataset")
253             print("0 para salir")
254
255             a=int(input("ingrese opcion: "))
256
257             if a==0 or a==1:
258                 rs=0
259                 if a == 0:

```

```

259             rp=0
260         else:
261             rp=1
262         else:
263             print("Opcion no valida")
264             rs=1
265         else:
266             break
267
268     if (rp==0):
269         break
270
271 '''
272 -----
273 Armamos y guardamos el data set
274 deseado para la aplicacion
275 -----
276 '''
277
278 if entrada == '1':
279
280     print("\n")
281     print("-----")
282     print("Armamos del dataset total")
283     print("-----")
284     print("\n")
285     cantds=input("Ingrese la cantidad de dataset: ")
286     cantds=int(cantds)
287     tds = np.ones([muestras*cantds ,4])
288     for i in range (cantds):
289         nameds=validardato()
290         ndataset = np.load(nameds+'.npy')
291         for k in range (4):
292             for j in range (muestras):
293                 tds[j+(muestras*i),k]=ndataset[j,k]
294
295     np.save("Data set Total",tds)
296     print(tds)
297
298
299 '''
300 -----
301 Letreros de fin de programa
302 -----
303 '''
304
305
306 print("\n")
307 print("-----")
308 print("Programa Terminado")
309 print("-----")
310 print("\n")

```

Algoritmo 6.2: Read and Save Database

En las siguientes imágenes podemos ver la posición de los ejes y cual predomina según la posición de la mano.

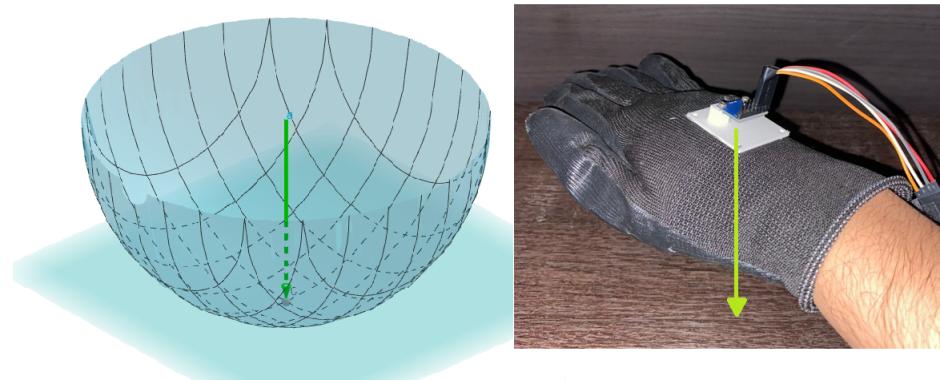


Figura 6.11: Posición 1

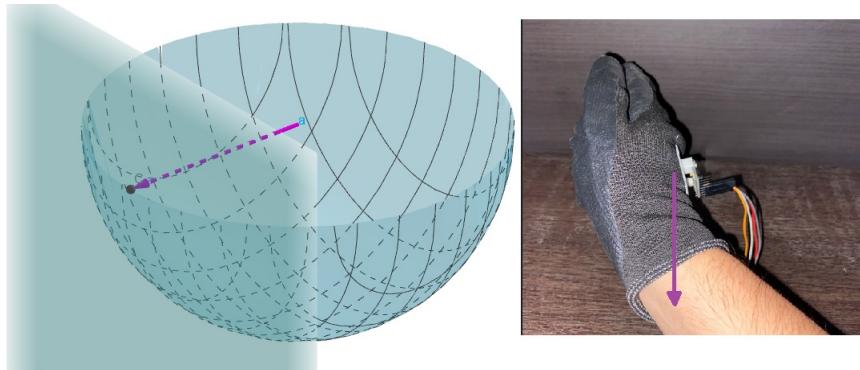


Figura 6.12: Posición 2

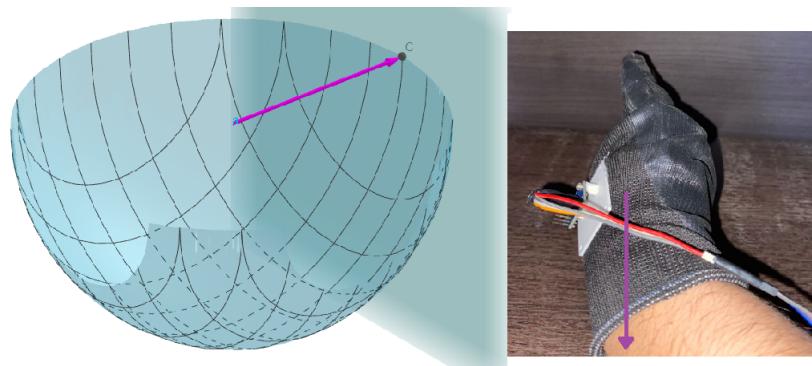


Figura 6.13: Posición 3

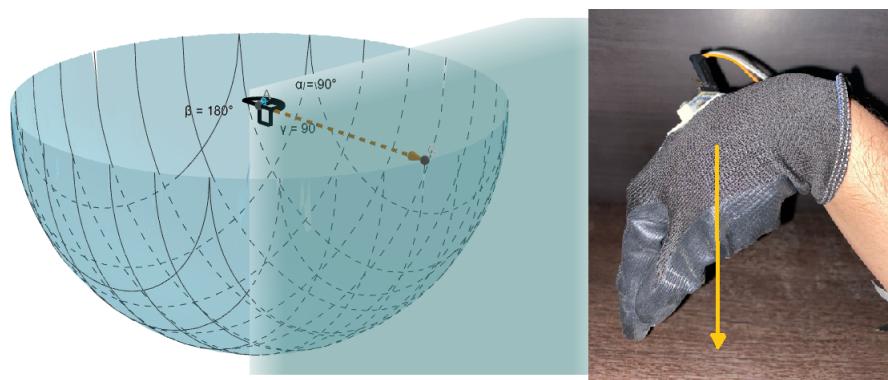


Figura 6.14: Posición 4

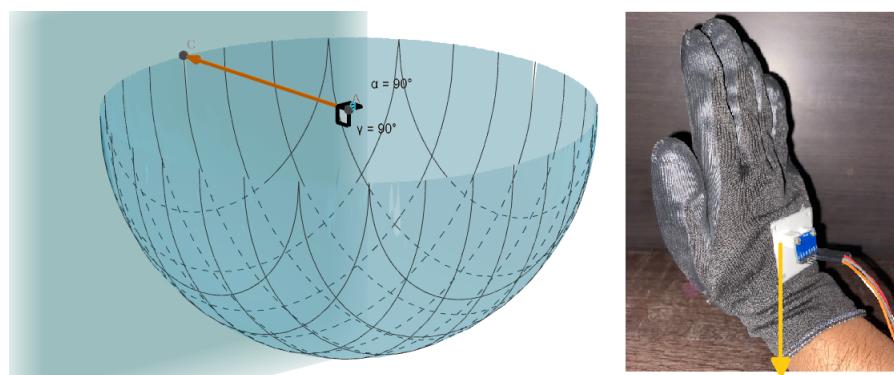


Figura 6.15: Posición 5

Finalmente los datos quedarán distribuidos como se ve en la siguiente gráfica:

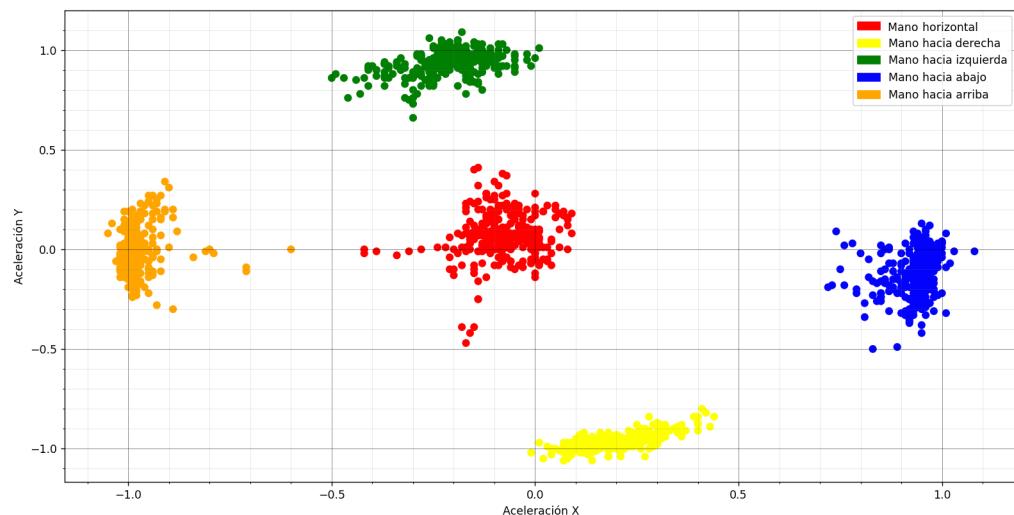


Figura 6.16: Gráfica de los 5 grupos de datos tomados para cada posición de la mano, tener en cuenta las referencias para cada uno

Capítulo 7

Practica de Redes Neuronales

7.1. Muestreo de datos para las distintas posiciones de la mano

Para el entrenamiento de la red neuronal mandaremos 300 datos por cada posición que toma la mano (indicado en página 62 y 63 o en su respectivo gráfico). Para lograr esto trabajamos con el siguiente código comunicándonos hacia la PC mediante el dispositivo UART.

A continuación se mostrará el código con su gráfica correspondiente de cada muestreo y en ella su dominio respectivo (color azul), como así también podremos apreciar un msj en pantalla.

MUESTREO 1:

```

1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 import random
5 from matplotlib.colors import ListedColormap
6
7 np.set_printoptions(suppress=True)
8
9
10 #Cargamos el dataset y la muestras X
11
12 datos = np.load('Data set Total.npy')
13 X = datos[:, :2]
14
15 #Entrenamiento primer muestreo
16
17 Xc=datos[:300,:2]
18 y1_real = datos[:300,3]
19 y2_real = datos[300:,3]
20 y2_real[:] = 0
21 y = np.concatenate((y1_real, y2_real))
22 labelclase='Mano horizontal'
23
24 # Agregar etiquetas a los ejes
25 plt.xlabel('Aceleración en X')
26 plt.ylabel('Aceleración en Y')
27
28 # Agregar grid principal con separación de 0.5
29 plt.grid(which='major', linewidth='0.5', color='black',

```

```

    alpha=0.5)

30
31 # Agregar grid secundario con separación de 0.1
32 plt.grid(which='minor', linewidth='0.5', color='gray',
33           alpha=0.2)
33 plt.minorticks_on()
34 plt.gca().xaxis.set_minor_locator(plt.MultipleLocator(0.1)
35           )
35 plt.gca().yaxis.set_minor_locator(plt.MultipleLocator(0.1)
36           )

37 #Centro
38
39 centro = np.mean(Xc, axis=0)
40 tmuestras = 1500
41 K=1
42
43 G=np.ones((tmuestras,K))
44 epsilon = 0.3
45 umbral = 0.5
46
47 for i in range(K):
48     for j in range(tmuestras):
49         dif = np.linalg.norm(centro-X[j])
50         salida=np.exp((-1/(2*epsilon**2))*dif**2)
51         G[j,0]=salida
52
53 G_inv=np.linalg.pinv(G)
54 y = y.reshape(1500,1)
55 W=np.dot(G_inv,y)
56 pmuestra=np.dot(G,W)

57 def redneuronal(x):
58     for i in range(1):
59         dif = np.linalg.norm(centro-x)
60         salida=np.exp((-1/(2*epsilon**2))*dif**2)
61         y = np.dot(salida, W)
62         if y > umbral:
63             return 1
64         else:
65             return 0
66     return y
67
68 #Representación de la superficie de decisión
69 colors = (('white','blue'))
70 cmap = ListedColormap(colors[:len(np.unique(y))])
72 resolution=0.02
73
74 x1_min, x1_max = -2,2
75 x2_min, x2_max = -2,2
76 xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
77                         resolution),np.arange(x2_min, x2_max, resolution))

78 Z = np.zeros(xx1.shape)
79
80 for i in range(xx1.shape[0]):
81     for j in range(xx1.shape[1]):
82         data = np.array([xx1[i,j],xx2[i,j]])
83         Z[i,j] = redneuronal(data)
84

```

```

85 print('El centro del conjunto es: ')
86 print(centro)
87 print('El valor del peso W es: ')
88 print(W)
89 print('El valor de epsilon es: ')
90 print(epsilon)
91 print('El umbral de escalon es: ')
92 print(umbral)
93
94 plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
95 plt.xlim(xx1.min(), xx1.max())
96 plt.ylim(xx2.min(), xx2.max())
97
98 # Agregar leyenda para las distintas clases de muestras
99 labels = [labelclase, 'Resto de las muestras']
100 colors = ['yellow', 'purple']
101 handles = [plt.scatter([], [], c=colors[i], alpha=0.3) for
102 i in range(len(labels))]
103 plt.legend(handles, labels, loc='upper right',
104 labelspacing=1.2)
105 plt.scatter(X[:,0], X[:,1], marker='o', c=y, alpha=0.3)
106 plt.show()

```

Algoritmo 7.1: Muestreo 1

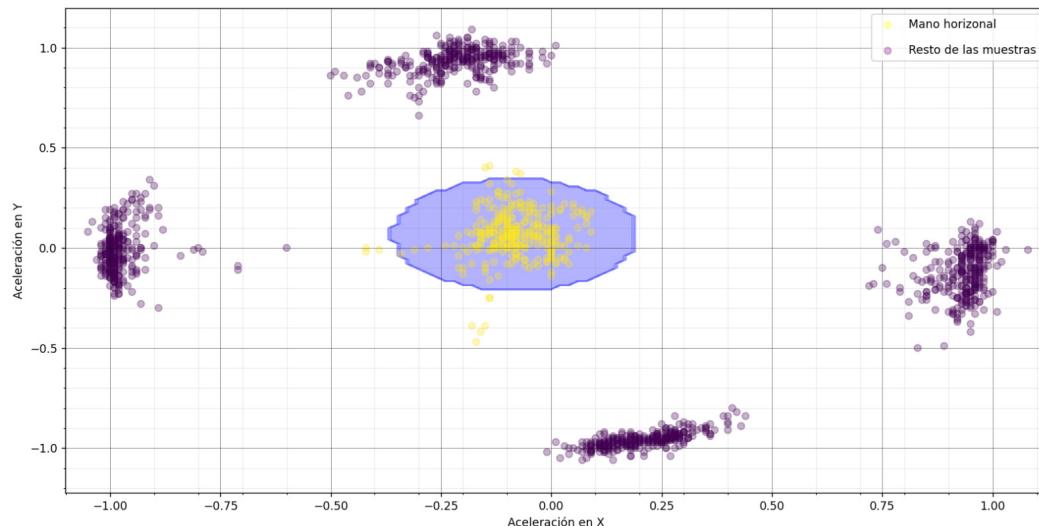


Figura 7.1: Primer muestreo: Mano horizontal

MUESTREO 2:

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 import random
5 from matplotlib.colors import ListedColormap
6
7 np.set_printoptions(suppress=True)
8
9
10 #Cargamos el dataset y la muestras X
11
12 datos = np.load('Data set Total.npy')
13 X = datos[:, :2]
14
15 #Entrenamiento segundo muestreo
16
17 Xc=datos[300:600,:2]
18 y1_real = datos[:300,3]
19 y1_real[:] = int(0)
20 y2_real = datos[300:600,3]
21 y2_real[:] = int(1)
22 y3_real = np.concatenate((y1_real, y2_real))
23 y4_real = datos[600: ,3]
24 y4_real[:] = int(0)
25 y = np.concatenate((y3_real, y4_real))
26 labelclase='Mano hacia derecha'
27
28
29 # Agregar etiquetas a los ejes
30 plt.xlabel('Aceleración en X')
31 plt.ylabel('Aceleración en Y')
32
33 # Agregar grid principal con separación de 0.5
34 plt.grid(which='major', linewidth='0.5', color='black',
           alpha=0.5)
35
36 # Agregar grid secundario con separación de 0.1
37 plt.grid(which='minor', linewidth='0.5', color='gray',
           alpha=0.2)
38 plt.minorticks_on()
39 plt.gca().xaxis.set_minor_locator(plt.MultipleLocator(0.1))
40 plt.gca().yaxis.set_minor_locator(plt.MultipleLocator(0.1))
41
42 #Centro
43
44 centro = np.mean(Xc, axis=0)
45
46 tmuestras = 1500
47 K=1
48
49 G=np.ones((tmuestras,K))
50 epsilon = 0.3
51 umbral = 0.5
52
53 for i in range(K):
54     for j in range(tmuestras):
55         dif = np.linalg.norm(centro-X[j])
```

```

56     salida=np.exp((-1/(2*epsilon**2))*dif**2)
57     G[j,0]=salida
58
59 G_inv=np.linalg.pinv(G)
60 y = y.reshape(1500,1)
61 W=np.dot(G_inv,y)
62 pmuestra=np.dot(G,W)
63
64 def redneuronal(x):
65     for i in range(1):
66         dif = np.linalg.norm(centro-x)
67         salida=np.exp((-1/(2*epsilon**2))*dif**2)
68         y = np.dot(salida, W)
69         if y > umbral:
70             return 1
71         else:
72             return 0
73     return y
74
75 #Representación de la superficie de decisión
76 colors = ('white','blue')
77 cmap = ListedColormap(colors[:len(np.unique(y))])
78 resolution=0.02
79
80 x1_min, x1_max = -2,2
81 x2_min, x2_max = -2,2
82 xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),np.arange(x2_min, x2_max, resolution))
83
84 Z = np.zeros(xx1.shape)
85
86 for i in range(xx1.shape[0]):
87     for j in range(xx1.shape[1]):
88         data = np.array([xx1[i,j],xx2[i,j]])
89         Z[i,j] = redneuronal(data)
90
91 print('El centro del conjunto es:')
92 print(centro)
93 print('El valor del peso W es: ')
94 print(W)
95 print('El valor de epsilon es: ')
96 print(epsilon)
97 print('El umbral de escalon es: ')
98 print(umbral)
99
100 plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
101 plt.xlim(xx1.min(), xx1.max())
102 plt.ylim(xx2.min(), xx2.max())
103
104 # Agregar leyenda para las distintas clases de muestras
105 labels = [labelclase, 'Resto de las muestras']
106 colors = ['yellow', 'purple']
107 handles = [plt.scatter([], [], c=colors[i], alpha=0.3) for
    i in range(len(labels))]
108 plt.legend(handles, labels, loc='upper right',
    labelspacing=1.2)
109
110
111 plt.scatter(X[:,0],X[:,1],marker='o',c=y,alpha=0.3)

```

112 plt.show()

Algoritmo 7.2: Muestreo 2

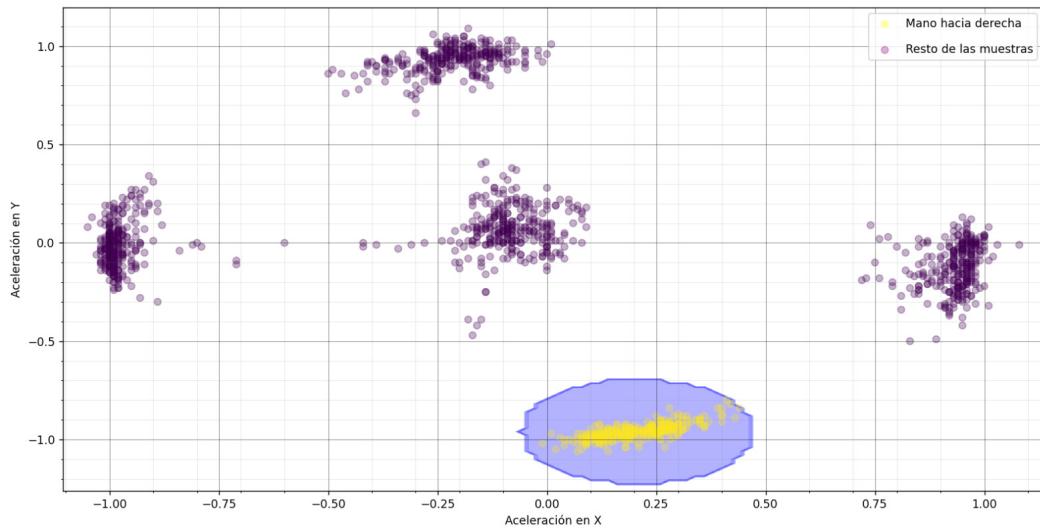


Figura 7.2: Segundo muestreo: Mano hacia derecha

MUESTREO 3:

```

1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 import random
5 from matplotlib.colors import ListedColormap
6
7 np.set_printoptions(suppress=True)
8
9
10 #Cargamos el dataset y la muestras X
11
12 datos = np.load('Data set Total.npy')
13 X = datos[:, :2]
14
15 #Entrenamiento tercer muestreo
16 Xc=datos[600:900,:2]
17 y1_real = datos[:600,3]
18 y1_real[:] = int(0)
19 y2_real = datos[600:900,3]
20 y2_real[:] = int(1)
21 y3_real = np.concatenate((y1_real, y2_real))
22 y4_real = datos[900:,3]
23 y4_real[:] = int(0)
24 y = np.concatenate((y3_real, y4_real))
25 labelclase='Mano hacia izquierda'
26
27 # Agregar etiquetas a los ejes
28 plt.xlabel('Aceleración en X')
29 plt.ylabel('Aceleración en Y')
30
31 # Agregar grid principal con separación de 0.5
32 plt.grid(which='major', linewidth='0.5', color='black',
   alpha=0.5)
33
34 # Agregar grid secundario con separación de 0.1

```

```

35 plt.grid(which='minor', linewidth='0.5', color='gray',
36           alpha=0.2)
37 plt.minorticks_on()
38 plt.gca().xaxis.set_minor_locator(plt.MultipleLocator(0.1)
39                               )
40 plt.gca().yaxis.set_minor_locator(plt.MultipleLocator(0.1)
41                               )
42 #Centro
43
44 centro = np.mean(Xc, axis=0)
45
46 tmuestras = 1500
47 K=1
48 G=np.ones((tmuestras,K))
49 epsilon = 0.3
50 umbral = 0.5
51
52 for i in range(K):
53     for j in range(tmuestras):
54         dif = np.linalg.norm(centro-X[j])
55         salida=np.exp((-1/(2*epsilon**2))*dif**2)
56         G[j,0]=salida
57
58 G_inv=np.linalg.pinv(G)
59 y = y.reshape(1500,1)
60 W=np.dot(G_inv,y)
61 pmuestra=np.dot(G,W)
62
63 def redneuronal(x):
64     for i in range(1):
65         dif = np.linalg.norm(centro-x)
66         salida=np.exp((-1/(2*epsilon**2))*dif**2)
67         y = np.dot(salida, W)
68         if y > umbral:
69             return 1
70         else:
71             return 0
72     return y
73
74 #Representación de la superficie de decisión
75 colors = (('white','blue'))
76 cmap = ListedColormap(colors[:len(np.unique(y))])
77 resolution=0.02
78
79 x1_min, x1_max = -2,2
80 x2_min, x2_max = -2,2
81 xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
82                         resolution),np.arange(x2_min, x2_max, resolution))
83 Z = np.zeros(xx1.shape)
84
85 for i in range(xx1.shape[0]):
86     for j in range(xx1.shape[1]):
87         data = np.array([xx1[i,j],xx2[i,j]])
88         Z[i,j] = redneuronal(data)
89
90 print('El centro del conjunto es:')

```

```

91 print(centro)
92 print('El valor del peso W es: ')
93 print(W)
94 print('El valor de epsilon es: ')
95 print(epsilon)
96 print('El umbral de escalon es: ')
97 print(umbral)
98
99 plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
100 plt.xlim(xx1.min(), xx1.max())
101 plt.ylim(xx2.min(), xx2.max())
102
103 # Agregar leyenda para las distintas clases de muestras
104 labels = [labelclase, 'Resto de las muestras']
105 colors = ['yellow', 'purple']
106 handles = [plt.scatter([], [], c=colors[i], alpha=0.3) for
107     i in range(len(labels))]
108 plt.legend(handles, labels, loc='upper right',
109             labelspacing=1.2)
110 plt.scatter(X[:,0], X[:,1], marker='o', c=y, alpha=0.3)
111 plt.show()

```

Algoritmo 7.3: Muestreo 3

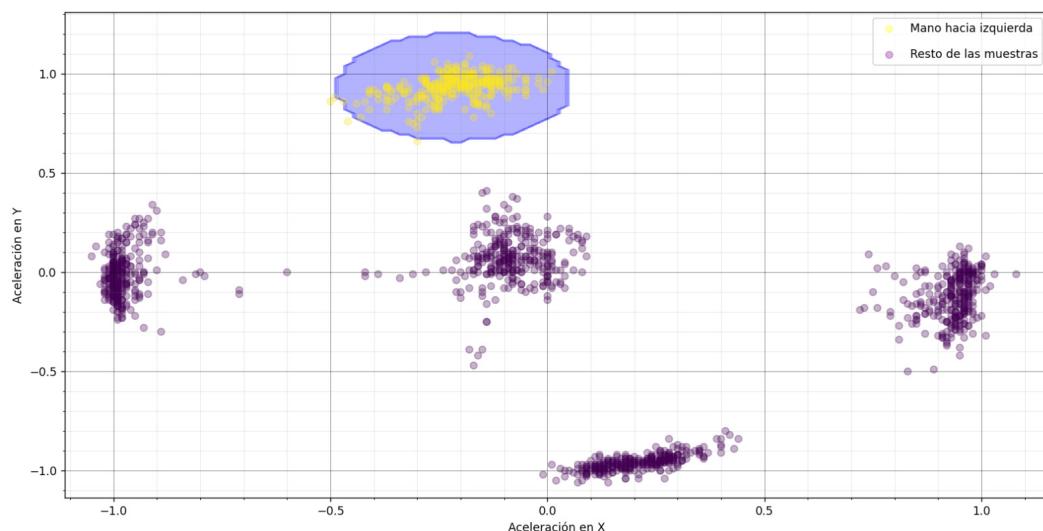


Figura 7.3: Tercer muestreo: Mano hacia izquierda

MUESTREO 4:

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 import random
5 from matplotlib.colors import ListedColormap
6
7 np.set_printoptions(suppress=True)
8
9
10 #Cargamos el dataset y la muestras X
11
12 datos = np.load('Data set Total.npy')
13 X = datos[:, :2]
14
15 #Entrenamiento cuarto muestreo
16
17 Xc=datos[900:1200,:2]
18 y1_real = datos[:900,3]
19 y1_real[:] = int(0)
20 y2_real = datos[900:1200,3]
21 y2_real[:] = int(1)
22 y3_real = np.concatenate((y1_real, y2_real))
23 y4_real = datos[1200:,:3]
24 y4_real[:] = int(0)
25 y = np.concatenate((y3_real, y4_real))
26 labelclase='Mano hacia abajo'
27
28 # Agregar etiquetas a los ejes
29 plt.xlabel('Aceleración en X')
30 plt.ylabel('Aceleración en Y')
31
32 # Agregar grid principal con separación de 0.5
33 plt.grid(which='major', linewidth='0.5', color='black',
           alpha=0.5)
34
35 # Agregar grid secundario con separación de 0.1
36 plt.grid(which='minor', linewidth='0.5', color='gray',
           alpha=0.2)
37 plt.minorticks_on()
38 plt.gca().xaxis.set_minor_locator(plt.MultipleLocator(0.1))
39 plt.gca().yaxis.set_minor_locator(plt.MultipleLocator(0.1))
40
41 #Centro
42
43 centro = np.mean(Xc, axis=0)
44
45 tmuestras = 1500
46 K=1
47
48 G=np.ones((tmuestras,K))
49 epsilon = 0.3
50 umbral = 0.5
51
52 for i in range(K):
53     for j in range(tmuestras):
54         dif = np.linalg.norm(centro-X[j])
55         salida=np.exp((-1/(2*epsilon**2))*dif**2)
```

```

56     G[j,0]=salida
57
58 G_inv=np.linalg.pinv(G)
59 y = y.reshape(1500,1)
60 W=np.dot(G_inv,y)
61 pmuestra=np.dot(G,W)
62
63 def redneuronal(x):
64     for i in range(1):
65         dif = np.linalg.norm(centro-x)
66         salida=np.exp((-1/(2*epsilon**2))*dif**2)
67         y = np.dot(salida, W)
68         if y > umbral:
69             return 1
70         else:
71             return 0
72     return y
73
74 #Representación de la superficie de decisión
75 colors = (('white','blue'))
76 cmap = ListedColormap(colors[:len(np.unique(y))])
77 resolution=0.02
78
79 x1_min, x1_max = -2,2
80 x2_min, x2_max = -2,2
81 xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),np.arange(x2_min, x2_max, resolution))
82
83 Z = np.zeros(xx1.shape)
84
85 for i in range(xx1.shape[0]):
86     for j in range(xx1.shape[1]):
87         data = np.array([xx1[i,j],xx2[i,j]])
88         Z[i,j] = redneuronal(data)
89
90 print('El centro del conjunto es:')
91 print(centro)
92 print('El valor del peso W es: ')
93 print(W)
94 print('El valor de epsilon es: ')
95 print(epsilon)
96 print('El umbral de escalon es: ')
97 print(umbral)
98
99 plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
100 plt.xlim(xx1.min(), xx1.max())
101 plt.ylim(xx2.min(), xx2.max())
102
103 # Agregar leyenda para las distintas clases de muestras
104 labels = [labelclase, 'Resto de las muestras']
105 colors = ['yellow', 'purple']
106 handles = [plt.scatter([], [], c=colors[i], alpha=0.3) for
    i in range(len(labels))]
107 plt.legend(handles, labels, loc='upper right',
    labelspacing=1.2)
108
109
110 plt.scatter(X[:,0],X[:,1],marker='o',c=y,alpha=0.3)
111 plt.show()

```

Algoritmo 7.4: Muestreo 4

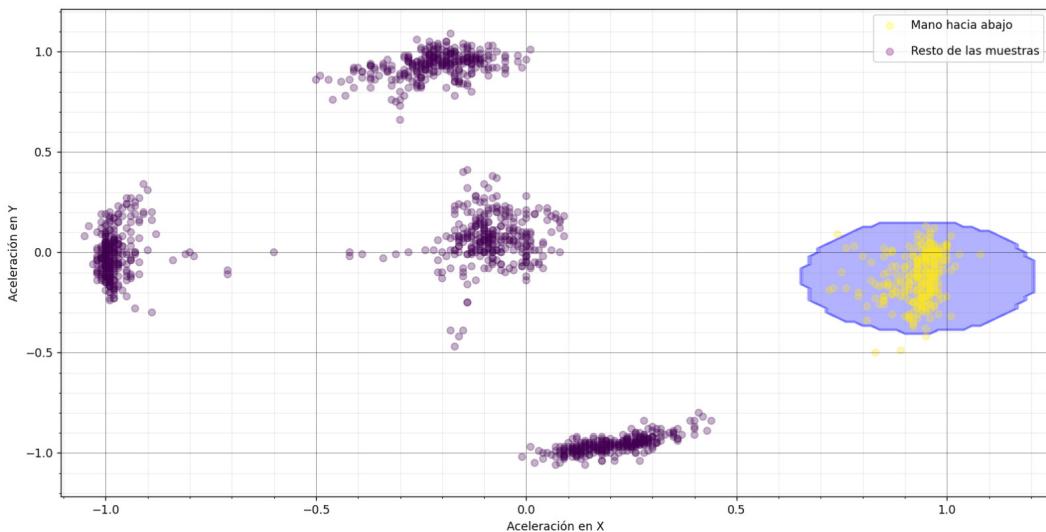


Figura 7.4: Cuarto muestreo: Mano hacia abajo

MUESTREO 5:

```

1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 import random
5 from matplotlib.colors import ListedColormap
6
7 np.set_printoptions(suppress=True)
8
9
10 #Cargamos el dataset y la muestras X
11
12 datos = np.load('Data set Total.npy')
13 X = datos[:, :2]
14
15 #Entrenamiento quinto muestreo
16
17 Xc=datos[1200:,:2]
18 y1_real = datos[:1200,3]
19 y1_real[:] = int(0)
20 y2_real = datos[1200:,:3]
21 y2_real[:] = int(1)
22 y = np.concatenate((y1_real, y2_real))
23 labelclase='Mano hacia arriba'
24
25 # Agregar etiquetas a los ejes
26 plt.xlabel('Aceleración en X')
27 plt.ylabel('Aceleración en Y')
28
29 # Agregar grid principal con separación de 0.5
30 plt.grid(which='major', linewidth='0.5', color='black',
           alpha=0.5)
31
32 # Agregar grid secundario con separación de 0.1
33 plt.grid(which='minor', linewidth='0.5', color='gray',
           alpha=0.2)
34 plt.minorticks_on()
35 plt.gca().xaxis.set_minor_locator(plt.MultipleLocator(0.1)
           )
36 plt.gca().yaxis.set_minor_locator(plt.MultipleLocator(0.1))

```

```

    )
37
38 #Centro
39
40 centro = np.mean(Xc, axis=0)
41
42 tmuestras = 1500
43 K=1
44
45 G=np.ones((tmuestras ,K))
46 epsilon = 0.3
47 umbral = 0.5
48
49 for i in range(K):
50     for j in range(tmuestras):
51         dif = np.linalg.norm(centro-X[j])
52         salida=np.exp((-1/(2*epsilon**2))*dif**2)
53         G[j ,0]=salida
54
55 G_inv=np.linalg.pinv(G)
56 y = y.reshape(1500,1)
57 W=np.dot(G_inv,y)
58 pmuestra=np.dot(G,W)
59
60 def redneuronal(x):
61     for i in range(1):
62         dif = np.linalg.norm(centro-x)
63         salida=np.exp((-1/(2*epsilon**2))*dif**2)
64         y = np.dot(salida, W)
65         if y > umbral:
66             return 1
67         else:
68             return 0
69     return y
70
71 #Representación de la superficie de decisión
72 colors = (('white','blue'))
73 cmap = ListedColormap(colors[:len(np.unique(y))])
74 resolution=0.02
75
76 x1_min, x1_max = -2,2
77 x2_min, x2_max = -2,2
78 xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),np.arange(x2_min, x2_max, resolution))
79
80 Z = np.zeros(xx1.shape)
81
82 for i in range(xx1.shape[0]):
83     for j in range(xx1.shape[1]):
84         data = np.array([xx1[i,j],xx2[i,j]])
85         Z[i,j] = redneuronal(data)
86
87 print('El centro del conjunto es:')
88 print(centro)
89 print('El valor del peso W es: ')
90 print(W)
91 print('El valor de epsilon es: ')
92 print(epsilon)
93 print('El umbral de escalon es: ')
94 print(umbral)

```

```

95 plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
96 plt.xlim(xx1.min(), xx1.max())
97 plt.ylim(xx2.min(), xx2.max())
98
99 # Agregar leyenda para las distintas clases de muestras
100 labels = [labelclase, 'Resto de las muestras']
101 colors = ['yellow', 'purple']
102 handles = [plt.scatter([], [], c=colors[i], alpha=0.3) for
103             i in range(len(labels))]
104 plt.legend(handles, labels, loc='upper right',
105             labelspacing=1.2)
106
107 plt.scatter(X[:,0], X[:,1], marker='o', c=y, alpha=0.3)
108 plt.show()

```

Algoritmo 7.5: Muestreo 5

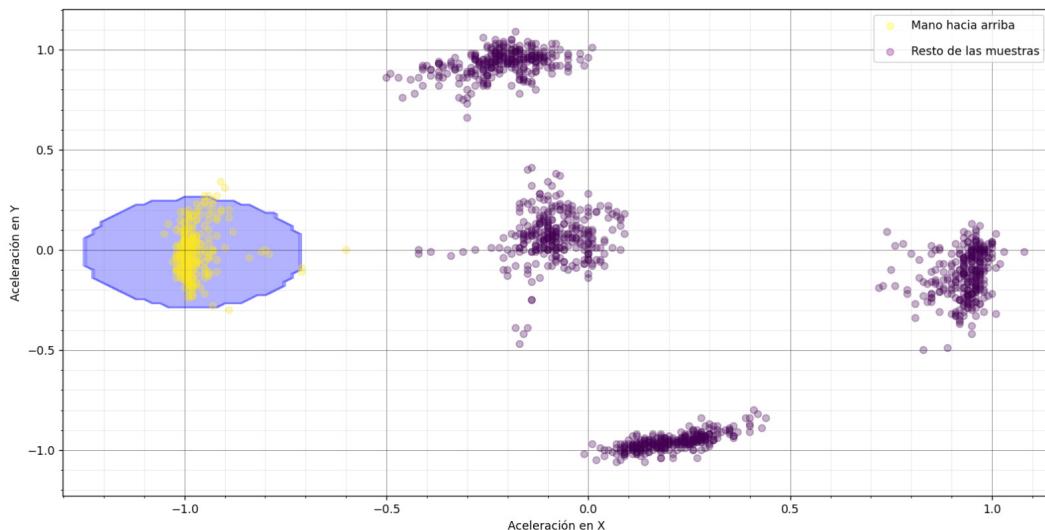


Figura 7.5: Quinto muestreo: Mano hacia abajo

7.2. Implementación en el ESP32

A continuación se muestra cómo queda el código final del microcontrolador:

```

1 # Importamos math
2 from math import sqrt
3 # Importamos MPU_6050 de imu
4 from imu import MPU6050
5 # Importamos time para los delay
6 import time
7 # Importamos Pin e I2C de machine
8 from machine import Pin, I2C, UART
9 # Importamos ssd1306
10 import ssd1306
11
12 # Configuramos comunicación I2C
13 I2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=400000)
14 # Configuramos MPU6050

```

```
15 imu = MPU6050(I2c)
16 #Configuracion UART 9600baudios y el modulo 2
17 uart= UART(2,9600)
18
19 #Definimos los centros
20 c1 = [-0.08246667 ,0.06743333]
21 c2 = [ 0.208 ,-0.9589]
22 c3 =[ -0.2146 , 0.93196667]
23 c4 =[ 0.93346667, -0.12263333]
24 c5 =[-0.9771,-0.01273333]
25
26
27 #definimos las varianza
28 sig = 0.3
29
30 #definimos los pesos
31 w1=1.08297439
32 w2=1.04464011
33 w3=1.05831915
34 w4=1.07999194
35 w5=1.06825504
36
37 #definimos el vector de salida
38 OUT = [0 , 0, 0, 0, 0]
39
40 #Configuramos pantalla ssd1306
41 display = ssd1306.SSD1306_I2C(128, 32, I2c)
42
43 #Mostramos mensaje
44 display.fill(0)
45 display.fill_rect(0, 0, 32, 32, 1)
46 display.fill_rect(2, 2, 28, 28, 0)
47 display.vline(9, 8, 22, 1)
48 display.vline(16, 2, 22, 1)
49 display.vline(23, 8, 22, 1)
50 display.fill_rect(26, 24, 2, 4, 1)
51 display.text('UTN - FRT', 40, 0, 1)
52 display.text('RBFNN', 40, 12, 1)
53 display.text('Ing. Electro', 40, 24, 1)
54 display.show()
55 time.sleep(3)
56 display.fill(0)
57 display.text('Red Neuronal RBF', 0, 0, 1)
58 display.text('Cosentino F.', 0, 10, 1)
59 display.text('Zelaya S.', 0, 20, 1)
60 display.show()
61 time.sleep(3)
62
63 while True:
64     #Leemos Aceleracion en X
65     Gx = imu.accel.x-0.06112061
66
67     #Leemos Aceleracion en Y
68     Gy = imu.accel.y+0.03150146
69
70     #Leemos Aceleracion en Z
71     Gz = imu.accel.z+0.07965576
72
73     if(Gz <= 0): #Para control del domino
74         #Dato de entrada
```

```

75     data = [Gx , Gy]
76
77     #Creamos la red neuronal RBF
78     def redNeuronalRBF(muestra,centro,sigma,w,escalon)
79     :
80         dif=sqrt((centro[0]-muestra[0])**2+(centro[1]-
81         muestra[1])**2)
82         salida1=(2.718281**((-1/(2*sigma**2))*dif**2))
83         *w
84         if salida1 > escalon:
85             return 1
86         else:
87             return 0
88     #Salida de caparadil por W y escalon
89     OUT[0]=redNeuronalRBF(data,c1,sig,w1,0.5)
90     OUT[1]=redNeuronalRBF(data,c2,sig,w2,0.5)
91     OUT[2]=redNeuronalRBF(data,c3,sig,w3,0.5)
92     OUT[3]=redNeuronalRBF(data,c4,sig,w4,0.4)
93     OUT[4]=redNeuronalRBF(data,c5,sig,w5,0.5)
94
95     #Salida de funcion de salida
96     if all(elem == 0 for elem in OUT):
97         salida = 0
98     else:
99         max_valor = max(OUT)
100        salida = OUT.index(max_valor) + 1
101        uart.write('Vector salida: [ ')
102        uart.write(str(OUT[0]))
103        uart.write(' ')
104        uart.write(str(OUT[1]))
105        uart.write(' ')
106        uart.write(str(OUT[2]))
107        uart.write(' ')
108        uart.write(str(OUT[3]))
109        uart.write(' ')
110        uart.write(str(OUT[4]))
111        uart.write(' ]\r\n')
112        display.fill(0)
113        display.text('[', 0, 0, 1)
114        for i in range(5):
115            display.text(str(OUT[i]), (15*i)+10, 0, 1)
116        display.text(']', 80, 0, 1)
117        display.show()
118    else:
119        salida = 0
120        uart.write('Vectores fuera del dominio\r\n')
121        display.text('Vectores fuera', 0, 10, 1)
122        display.text('del dominio', 0, 20, 1)
123        display.show()
124    print(OUT)
125    time.sleep(1)

```

Algoritmo 7.6: Código final del micro

7.3. Medición de tasa de acierto

Creamos un algoritmo iterativo para testear la red neuronal a diferentes cantidades de muestras aleatorias, el código que implementamos es el que se muestra a continuación:

```

1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 import random
5 from matplotlib.colors import ListedColormap
6
7 np.set_printoptions(suppress=True)
8
9
10 '''Cargamos el dataset y la muestras X'''
11
12 datos = np.load('Data set Total.npy')
13 X = datos[:, :2]
14 y = datos[:, 3]
15
16
17 centros = np.array([[-0.08246667, 0.06743333],
18                     [0.208, -0.9589],
19                     [-0.2146, 0.93196667],
20                     [0.93346667, -0.12263333],
21                     [-0.9771, -0.01273333]])
22
23 pesos = np.array([[1.08297439],
24                   [1.04464011],
25                   [1.05831915],
26                   [1.07999194],
27                   [1.06825504]])
28
29 OUT = np.zeros([1, 5])
30 errores = []
31
32 for variables in range(1, 1001):
33     #variables = 100
34     # Establecer la semilla en 123
35     random.seed(123)
36     # Generar un array de 300 posiciones con valores
37     # aleatorios entre 1 y 1500
38     arr = [random.randint(0, 1499) for _ in range(variables)]
39     y_wait = np.zeros([variables, 1])
40     y_real = np.zeros([variables, 1])
41     for i in range(len(arr)):
42         y_wait[i, 0] = y[arr[i]]
43
44     epsilon = 0.3
45
46     def redneuronal(x, centro, W):
47         for i in range(1):
48             dif = np.linalg.norm(centro - x)
49             salida = np.exp((-1 / (2 * epsilon ** 2)) * dif ** 2)
50             y = np.dot(salida, W)
51             if y > 0.5:
52                 return 1
53             else:
54                 return 0
55             return y
56
57     for j in range(variables):
58         test = X[arr[j], :2]

```

```

59     #print(redneuronal(test,centros[0],pesos[0]))
60
61     for i in range(5):
62         OUT[0,i] = redneuronal(test,centros[i],pesos[i])
63
64     max_valor = np.argmax(OUT)
65     salida=int(max_valor)+1
66     y_real[j,0] = salida
67
68     #print(y_wait.shape)
69     #print("-----")
70     #print(y_real.shape)
71     dife = y_wait-y_real
72     error= np.count_nonzero(dife)
73     #print(error)
74     errores.append(((variables-error)/variables)*100.00)
75
76 plt.ylim(95, 102)
77 # Agregar etiquetas a los ejes
78 plt.xlabel('Numero de muestras')
79 plt.ylabel('Tasa de acierto')
80
81 plt.plot(errores)
82 plt.grid(True)
83
84 plt.show()

```

Algoritmo 7.7: Tasa de acierto

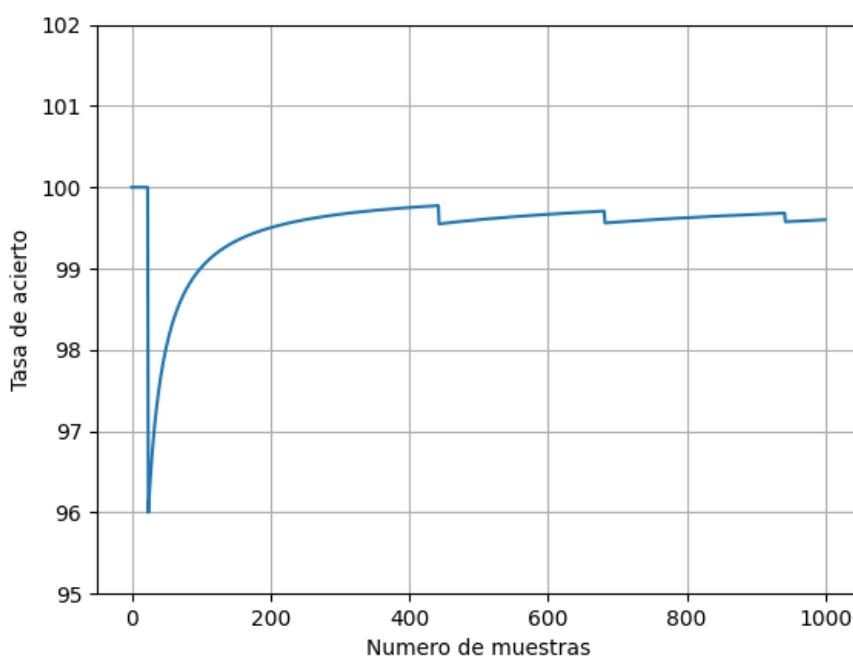


Figura 7.6: Gráfico correspondiente a la tasa de acierto

Capítulo 8

Metodología y planificación

Metodología

La metodología empleada para este proyecto de desarrollo aplicado es ir desde lo general a lo particular, con desarrollos sectoriales e individuales y en forma de cascada y consecutiva. La metodología de trabajo consta de:

Planificación:

- Identificación de ámbito de aplicación.
- Necesidades.
- Definición de objetivos.
- Análisis de situación.
- Tecnologías vinculantes en componentes electrónicos: comunicación, industrial, servicios, militar, bioquímicas, etc.
- Intervención de especialistas interdisciplinarios Aportes de otras ciencias.
- Definición preliminar del producto a desarrollar.

Desarrollo conceptual:

- Analizar el diseño y construcción de esquemas electrónicos funcionales con tecnología de última generación.
- Estudio y caracterización de las señas obtenidas por los dispositivos sensores seleccionados
- Documentación

Diseño Preliminar:

- Diseños de dispositivos electrónicos.
- Esquemas Electrónicos con tecnología actual.
- Simulaciones. PCB.

- Documentación

Diseño Definitivo:

- Prototipo funcional con tecnología actual.
- Ensayos y pruebas finales de laboratorio
- Documentación Final.
- Análisis Económico
- Conclusiones.

Cronograma de actividades

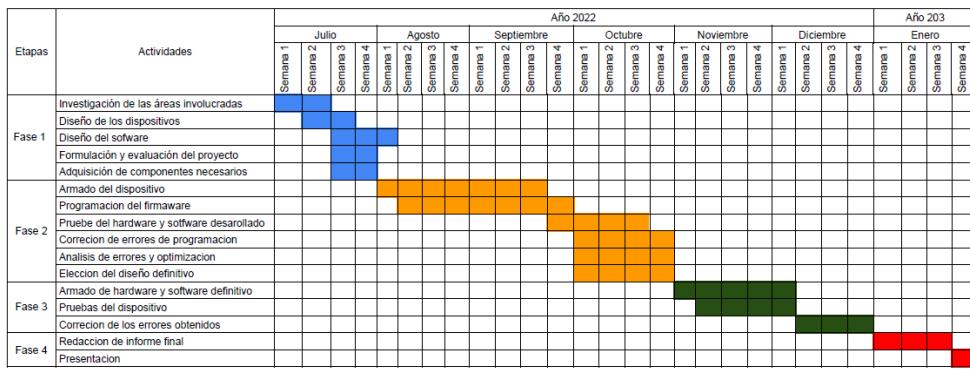


Figura 8.1: Cronograma

Apendice

Datasheets

Hojas de datos de los dispositivos utilizados en este trabajo.

1. ESP32 Generico
2. ESP32 WROOM
3. Acelerometro MPU 6050
4. Pantalla OLDE 1306
5. CP2102 USB a UART
6. AS1117-3-3 Regulador de voltage

Bibliografía

- [1] Matich D.J. (2001) *Redes Neuronales: Conceptos Básicos y Aplicaciones*. Click aqui para ir al documento
- [2] Gonfalonieri A. (2019) *How to Build A Data Set For Your Machine Learning Project*. Click aqui para ir al documento
- [3] Basogain Olabe X. (2008) *Redes neuronales artificiales y sus aplicaciones*. Click aqui para ir al documento
- [4] Morante S. (2018) *Precauciones a la hora de normalizar datos en Data Science*. Click aqui para ir al documento
- [5] Morales Iturrealde, A. L. (2021). *Diseño de un guante con sensores de flexibilidad que traducen letras del abecedario del lenguaje sordo mudo utilizando Micropython* (Bachelor's thesis).
- [6] Izaurieta, F., Saavedra, C. (2000). *Redes neuronales artificiales*. Departamento de Física, Universidad de Concepción Chile.
- [7] Dhanalakshmi, P., Palanivel, S., Ramalingam, V. (2009). *Classification of audio signals using SVM and RBFNN*. Expert systems with applications, 36(3), 6069-6075.
- [8] Pérez Ramírez, F. O., Fernández Castaño, H. (2007). *Las redes neuronales y la evaluación del riesgo de crédito*. Revista Ingenierías Universidad de Medellín, 6(10), 77-91.
- [9] Montazer, G. A., Giveki, D., Karami, M., Rastegar, H. (2018). *Radial basis function neural networks: A review*. Comput. Rev. J, 1(1), 52-74.
- [10] Plauska, I., Liutkeviius, A., Janaviit, A. (2022). *Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller*. Electronics, 12(1), 143.
- [11] Tollervey, N. H. (2017). *Programming with MicroPython: embedded programming with microcontrollers and Python*. O'Reilly Media, Inc.