TRABAJO DE FIN DE GRADO

GRADO EN FÍSICA

Área de electrónica

Inteligencia artificial sobre procesadores *edge computing*

Autor

Jorge Cuartero Galindo

Directores

Belén Calvo López

Nicolás Medrano Márques

Junio de 2024

Universidad Zaragoza

Lista de Figuras

Lista de Tablas

Listado de Acrónimos

ADC Analog-to-Digital Converter. I, 5, 7, 13, 27

BLE Bluetooth Low Energy. 3, 5, 18, 24

dps Degrees per second. 6

IA Inteligencia Artificial. 3

IMU Inertial Meausurement Unit. 5, 21

LSB Least Significant Bit. 6

ReLU Rectified Linear Unit. 15

TCP Transmision Control Protocol. 19

UDP User Datagram Protocol. 19

Índice

1. Motivación y objetivos

Desde la invención del primer circuito integrado, la búsqueda de su miniaturización es uno de los principales retos tecnológicos actuales. Dispositivos cada vez más pequeños y menos costosos aparecen en nuestras vidas, pudiendo tener a día de hoy en un reloj más poder de computación del que tenían ordenadores enteros en el pasado.

Bajo este paradigma nace el *edge computing*, que como contraparte al *cloud computing*, busca descentralizar el procesado de los datos, encargándose un mismo dispositivo de tomarlos y analizarlos. Esto se traduce en que, mientras que en el *cloud computing* se envían los datos en bruto, aquí enviamos los datos procesados localmente, lo que aumenta la seguridad en términos de acceso a la información, que se condensa antes del envío, lo que además supone envíos más rápidos; a cambio, son menores los recursos disponibles, limitados a los propios del dispositivo *edge*.

A esto se suma el creciente desarrollo de la Inteligencia Artificial (IA) en los últimos años. Esta tecnología, en un intento por imitar el funcionamiento del cerebro, se basa en redes neuronales que, mediante procesos de aprendizaje, son capaces de resolver tareas difíciles de abordar mediante algoritmos convencionales, pudiendo aplicarse desde la generación de imágenes hasta diagnósticos en medicina.

Una de las aplicaciones del uso de procesadores *edge* para desarrollar algoritmos basados en IA es el del desarrollo de interfaces naturales hombre-computador, como llevamos viendo años con los asistentes personales de voz (Alexa, Google nest) o las más recientes gafas de realidad mixta (Apple visión pro).

En este contexto, el trabajo propuesto busca desarrollar un sistema de adquisición y procesado de datos en un microcontrolador, ayudándonos de redes neuronales en el proceso, para implementar un sistema capaz de reconocer los gestos y posiciones de la mano y de este modo poder controlar de forma inalámbrica un ordenador.

El trabajo nace como continuación al Trabajo de Fin de Grado de Rubén Muñoz [1], en el cual se utilizó el microcontrolador Arduino 33 BLE (Bluetooth Low Energy). En este caso, se usará el microcontrolador Raspberry RP2040 [2], además de usar componentes y protocolos distintos. Sin embargo, muchas de las decisiones tomadas en el presente trabajo vienen guiadas por la experiencia adquirida en esta primera aproximación.

En cuanto a cómo se va a abordar el trabajo, se pueden diferenciar 4 grandes bloques, que son: sistema de adquisición de medida, procesado de datos, envío de la información, e interpretación de esta.

El primer bloque se centra principalmente en el hardware y los datos que adquiere. Dentro del hardware, se verá el esquema general del montaje, y se pondrá el foco en cada uno de los sistemas encargados de la adquisición de los distintos datos, referentes tanto a la orientación de la mano como a la posición (flexión) de los dedos.

Esto deriva en el segundo bloque, donde se tratará el procesado de los dos tipos de datos por separado: Los datos correspondientes a la orientación de la mano serán procesados para obtener su posición respecto al sistema de referencia conveniente, mientras que para la posición de los dedos desarrollaremos un modelo de inteligencia artificial, que nos permita una mayor adaptabilidad durante el desarrollo. Todos estos procesos serán implementados sobre el microcontrolador que hemos escogido como procesador *edge*, el Raspberry RP2040.

Los bloques restantes se centrarán en mostrar cómo se ha configurado el sistema de transmisión inalámbrica de los datos procesados a un ordenador, y su aplicación a una tarea previamente elegida.

Con todo esto, la memoria queda organizada del siguiente modo: en la Sección 2 se verá el esquema completo de la interfaz hardware desarrollada con los distintos sistemas de adquisición de datos; la Sección 3 presenta el tratamiento de los datos referentes a la orientación, mientras en la Sección 4 presenta el proceso de selección y entrenamiento en alto nivel de la red neuronal responsable de procesar los datos de flexión de los dedos y convertirlos a códigos de identificación, así como su adaptación para su ejecución en un procesador de bajos recursos; en la Sección 5 se tratará lo correspondiente la transmisión de los datos a un PC, y por último en la Sección 6 se verá cómo se usa la información recibida en el ordenador. Finalmente, la Sección 7 presenta los resultados obtenidos, y en la Sección 8 se ofrecerán las conclusiones del proyecto, así como opciones de ampliación.

2. Interfaz *hardware*

El sistema completo, montado en su totalidad sobre un guante debido a la aplicación escogida, se muestra en la Fig. 2.1. Podemos distinguir los distintos componentes: en cada dedo, del anular al meñique, sujetas por unas pequeñas piezas, fijamos bandas piezorresistivas que nos permitirán determinar su flexión. La información proporcionada por estas bandas se introduce en un Raspberry RP2040 Connect, alojado en una caja central fijada a la muñeca. Por último, una batería se encarga de alimentar todo el sistema. Todas las piezas plásticas del montaje han sido obtenidas mediante un proceso de fabricación aditiva 3D.

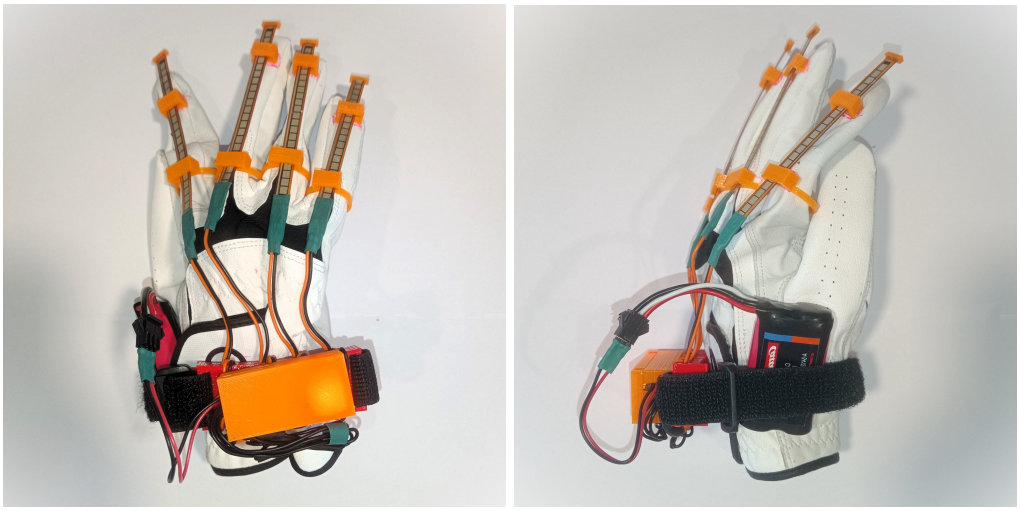


Figura 2.1: *Fotos del montaje final del guante con todas sus componentes.*

2.1. Arduino RP2040 Connect

Como ya hemos mencionado, el microcontrolador será un módulo Arduino RP2040 Connect, que incorpora un microcontrolador Raspberry Pi Pico [2]. La elección de este microcontrolador y no el Arduino 33 BLE usado anteriormente es porque proporciona mayor potencia de cálculo, un factor relevante si queremos evitar errores de integración (Apartado 2.1.1); y además permite emplear un protocolo WiFi para la transmisión de los datos.

En cuanto a las especificaciones generales, el RP2040 Connect opera a 3.3 V. Consta de un microcontrolador Raspberry Pi Pico (133 MHz de reloj, 256 kB de memoria SRAM) con 8 entradas analógicas (configuradas como convertidores analógico-digitales o ADCs), de las que 4 estarían destinadas a la adquisición de las posiciones de los dedos. Cuenta además con una *Inertial Meausurement Unit (IMU)* de 6 ejes (LSM6DSOXTR [3]).

2.1.1. Módulo LSM6DSOXTR

Este módulo [3] consta de un acelerómetro y un giróscopo, proporcionando así información de 6 ejes (3 de aceleración y 3 de rotación espacial). Aunque ambos pueden operar a distintos fondos de escala y resoluciones, la librería utilizada (Arduino_LSM6DSOX [4]) viene configurada con una sensibilidad de 0.122 mg/LSB (siendo g la aceleración de la gravedad y LSB el bit menos significativo o *Least Significant Bit*) para el acelerómetro y una sensibilidad de 0.07 dps/LSB para el giróscopo (donde dps son *Degrees per second*, grados por segundo), valores adecuados para nuestra aplicación (menores que el propio ruido). Esta librería dispone de dos funciones, una encargada de devolver 3 valores en representación de punto flotante para los tres ejes de la aceleración, y la otra de devolver los 3 valores referentes a la velocidad de rotación en cada eje en el mismo formato. Es importante tener en cuenta que los ejes son intrínsecos a la placa Arduino, aspecto que veremos en la Sección 3.

La frecuencia de medida está fijada en 104 Hz. Debido a que el giróscopo nos devuelve velocidades angulares que tenemos que integrar, va a existir un error de integración que aumentará cuanto mayor sea el tiempo entre medidas. Por ello se buscará minimizar el tiempo entre medidas, que con la limitación de 104 muestras por segundo, se traduce en que el tiempo de integración entre medidas queda fijado en unos 10 ms. Esto significa que un ciclo de medida y procesado de nuestro Arduino no puede durar más de ese tiempo, forzando a optimizar los tiempos de cada proceso.

2.2. Sistema de adquisición de posiciones de la mano

El sistema de determinación de la flexión de los dedos se basa en las bandas piezorresistivas antes mencionadas. Una piezorresistencia es un material o componente que, bajo un esfuerzo mecánico, sufre un cambio en su resistividad. En nuestro caso se usarán los *Flex Sensor* de *Spectra Symbol* [5], caracterizados por valores nominales entre 7 kΩ y 13 kΩ en reposo, de forma que al ser dobladas, como mínimo duplican su valor resistivo. Como el Arduino no puede medir directamente resistencias, implementamos un circuito auxiliar que nos permita traducir estos cambios en tensión.

2.2.1. El circuito eléctrico

Las variaciones de resistencia asociadas a las diferentes posiciones de cada dedo son convertidas a variaciones de voltaje mediante un divisor resistivo (Fig. 2.2), formado por una resistencia fija (R_{fija}) en serie con el elemento piezorresistivo (R_{piezo}) alimentado a 3.3 V, proporcionados por el microcontrolador.

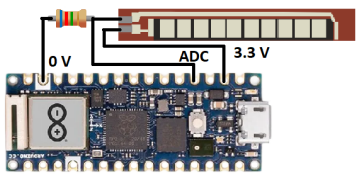


Figura 2.2: *Esquema del divisor resistivo.*

La elección de R_{fija} se realizará tal que maximice el rango de variación de voltaje, optimizando así la precisión en la medida (Anexo 1). De este modo obtenemos:

$$V_{divisor}(R_{piezo}) = 3.3\,\frac{R_{fija}}{R_{piezo}+R_{fija}} \Longrightarrow R_{fija} = \sqrt{R_{max}\cdot R_{min}} \tag{2.1}$$

donde R_{max} y R_{min} son los valores máximo y mínimo que puede tomar R_{piezo} respectivamente.

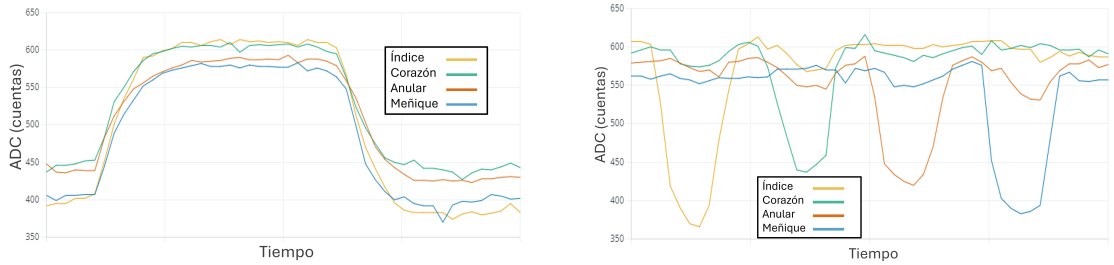
A partir de la medida de los valores máximos y mínimos de cada piezorresistencia, y en base a (2.1), obtendremos el valor óptimo de R_{fija} (Tabla 1). Podemos ver que no se han tomado exactamente los mismos valores que los teóricos, sino los valores nominales más próximos posibles.

	Resistencia en flexión (kΩ)		**Resistencia en extensión (kΩ)**	**Resistencia fija según ec. 2.1 (kΩ)**	**Resistencia fija montada (kΩ)**
	Pico	Relajado			
Meñique (dedo 1)	21,5	19	8,5	23	22
Anular (dedo 2)	25	23	12	17	18
Corazón (dedo 3)	25	23	12	17	18
Índice (dedo 4)	38	35	14	13	12

Tabla 1: *Valores resistivos de los piezoeléctricos y sus correspondientes resistencias fijas.*

La salida del divisor se conecta entonces a los pines analógicos del microcontrolador. Estos se configuran como entradas del ADC del microcontrolador, con una resolución de 10 bits (1024 valores) y un rango entre 0 (valor de lectura 0) y 3.3 V (valor de lectura 1023), con lo que se tiene una resolución de 3.2 mV. Además, estos tienen tiempos de medida de menos de 30 μs, compatible con los requisitos de velocidad del sistema mencionados anteriormente (Apartado 2.1.1).

La Fig. 2.3 muestra las lecturas del ADC en el microcontrolador con distintas flexiones. Se observa cómo la medida del ADC alcanza valores más bajos cuando se realiza la flexión y permanece en valores en torno a 600 mientras los dedos están estirados.



(a) *Apertura y cierre de toda la mano.*

(b) *Flexión de los dedos uno tras otro.*

Figura 2.3: *Distintas medidas obtenidas directamente del ADC al cambiar la posición de la mano.*

2.3. Sistema de alimentación

Para dotar al sistema de autonomía de movimiento, la placa electrónica es alimentada mediante una batería de 7.4 V con una capacidad de almacenamiento de energía de 1000 mAh (Fig. 2.1 derecha) conectada al pin V_{in}, que a través de un regulador incluido en la propia placa de Arduino proporciona los 3.3 V necesarios para todo el sistema.

3. Procesamiento de datos: orientación espacial

En esta sección veremos cómo se procesan los datos obtenidos por el módulo LSM6DSOXTR (Apartado 2.1.1) para obtener información útil y simplificada de la orientación espacial de la mano.

3.1. Sistema de referencia

Es preciso definir un sistema de referencia, pues mientras que como usuario siempre vamos a estar frente al mismo sistema de coordenadas, los ejes del Arduino rotan junto a este. Tomando entonces un sistema fijo, definimos una superficie esférica y sobre esta definimos qué ángulos nos interesan.

Como muestra la Fig. 3.1, el sistema de referencia viene determinado por la gravedad, pues el acelerómetro estará detectando permanentemente este valor (1 g). Conocer cómo se reparte este valor entre los ejes del acelerómetro permite conocer la inclinación del sistema al que está fijado el sensor respecto al eje horizontal (α). Concretamente, se precisa la componente y, pues es la que coincide con la dirección de apuntado.

En cuanto al ángulo θ, este no aporta información sobre la dirección a la que estamos apuntando, pero es necesario a la hora de integrar las velocidades del giróscopo. Este ángulo también se puede determinar únicamente con la información del acelerómetro.

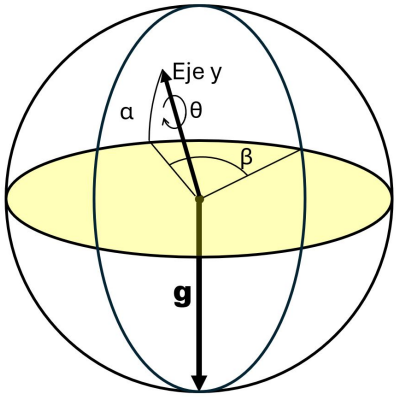


Figura 3.1: *Esquema del sistema de referencia, siendo g la dirección de la gravedad, el plano amarillo el plano horizontal, y representando la flecha la dirección en la que apunta la mano, que se corresponde con el eje y de la placa de Arduino.*

Por otro lado, hay que conocer el ángulo en el plano horizontal (perpendicular a g), β. Para esto usaremos la información del giróscopo, definiendo un ángulo 0, que haremos que se restablezca cada vez que el giróscopo reinicie su operación (ya que no siempre opera, como se verá en la Sección 6).

Aplicando las relaciones trigonométricas correspondientes (Anexo 2) se llega a las siguientes ecuaciones para determinar los ángulos:

$$\begin{cases} \alpha = arcsin(\frac{a_y}{a}) \\ \theta = -arctg(\frac{a_z}{a_x}) \\ \dfrac{\mathrm{d}\beta}{\mathrm{d}t} = \dfrac{(cos(\theta)\cdot\omega_x - sen(\theta)\cdot\omega_z)}{cos(\alpha)} \end{cases} \tag{3.1}$$

con a_i las aceleraciones y ω_i las velocidades angulares medidas en cada eje.

Como de β no conocemos el ángulo, sino su velocidad angular, tendremos que integrarla a lo largo del tiempo. Lo haremos del modo más simple posible, usando:

$$\beta_{t+\Delta t} = \beta_t + \frac{\mathrm{d}\beta}{\mathrm{d}t}\cdot\Delta t \tag{3.2}$$

donde Δt, que es el tiempo transcurrido entre cada actualización de β. Esto regresa a lo mencionado en el Apartado 2.1.1 del tiempo de integración, pues será necesario que $\Delta t < 10\ ms$ para tener el menor error de integración posible. Además, como ya se ha mencionado, $\beta = 0^{\mathrm{o}}$ se redefine cada vez que se usa el giróscopo, borrando así todo posible error de integración hasta el momento.

3.2. Calibrado del sistema de adquisición

Antes de realizar los cálculos del apartado anterior, es necesario eliminar diversos errores intrínsecos a la medida, como el ruido, el *offset*, la escala, o el error de cuadratura.

3.2.1. Acelerómetro

El principal problema del acelerómetro es el *offset* en las medidas. Orientando cada uno de los ejes del Arduino en sentido de la gravedad, deberíamos obtener 1 g en el eje correspondiente y 0 en el resto. Sin embargo, se obtienen los valores recogidos en la Tabla 2.

	Eje x		Eje y		Eje z	
a_x (g)	-1.02	0.97	-0.03	-0.03	-0.04	-0.01
a_y (g)	-0.01	0.00	-1.02	0.99	-0.01	-0.01
a_z (g)	0	0.01	-0.01	-0.03	-1.01	0.99

Tabla 2: *Valores de aceleración promediados para las distintas orientaciones del Arduino. Las unidades g se refieren a la aceleración de la gravedad terrestre. En amarillo aparece resaltado el eje que se está orientando con la gravedad ($a_i = \pm g$).*

Se observa cómo la aceleración no es simétrica al invertir la posición del Arduino. Sin embargo, esto puede corregirse muy fácilmente agregando un valor de *offset*, que no es otra cosa que sumar una constante a las medidas. Como ejemplo, en el caso del eje z sumar 0.01 g devolvería $|a_z| = 1$ g en ambos sentidos.

Además de esto, se mide aceleración no nula en ejes perpendiculares a la gravedad. Algunos de estos errores se solucionan al sumar el *offset*, como es el caso de a_y, que se corrige sumando 0.01 g, y esto hace que pase a ser 0 en ejes perpendiculares. Sin embargo, esto no siempre es suficiente, por lo que será necesario añadir términos cruzados para corregir el *offset* de otro eje en función de su valor. Por ejemplo, una corrección a a_z que dependerá del valor de a_y.

Partiendo de los resultados mostrados en la Tabla 2, se obtienen los siguientes valores calibrados de la aceleración:

$$\begin{cases} a_x = a_x(old) + 0.025 \\ a_y = a_y(old) + 0.01 \\ a_z = a_z(old) + 0.01 + 0.01\,|a_y| \end{cases} \tag{3.3}$$

El otro problema remarcable que se observa es el ruido. Este no está realmente debido a un problema del acelerómetro, pues en estático vemos niveles de ruido muy bajos. Sin embargo, a la hora de usar el guante, la mano tiembla, y el acelerómetro es muy sensible a estos cambios, por lo que se vuelve imposible señalar con precisión. Para ello, añadimos un umbral, de modo que solo se recalcule α (en θ este ruido no tiene consecuencias) cuando el cambio en este supere este umbral. Este valor está fijado en 0.2^{o}, aunque al ser un parámetro subjetivo está sujeto a cambios según el usuario y el receptor donde pretenda utilizarse.

3.2.2. Giróscopo

El giróscopo cuenta con los mismos problemas que el acelerómetro, a los que se suman la corrección de escala y la dependencia temporal del *offset*. Es importante tener mucho más controlados los errores en este sensor, pues al medir velocidades y no posiciones absolutas, terminaríamos con un desplazamiento del origen permanente (*drift*) que crece con el tiempo.

Respecto al error de escala, se observa que al integrar el ángulo para giros completos (360^{o}) a distintas velocidades en ninguna ocasión el giróscopo indica 360^{o}, sino que devuelve valores en torno a los 300^{o}. Para resolverlo, se promedian los ángulos obtenidos para varios giros y se obtiene un factor de proporcionalidad que utilizaremos para corregir las velocidades de cada eje (Tabla 3).

	Eje x		**Eje z**	
Angulo medido (º)	297.8	-295.3	303.3	-303.3
Factor de corrección	1.209	1.219	1.187	1.187

Tabla 3: *Ángulos medidos en giros completos en distintos ejes y sus respectivos factores de corrección.*

Remarcar que sólo necesitamos los ejes x y z, pues el eje y nos da información sobre θ, que ya viene determinada por el acelerómetro. Además, durante el proceso se observa que no en todos los puntos del giro la velocidad se comporta igual, por lo que realmente sería necesario un factor de corrección dependiente de los ángulos, pero esto requeriría demasiado tiempo y no merece la pena en nuestro caso, por lo que nos limitaremos a la aproximación simplificada anterior.

En cuanto al *offset*, a diferencia del acelerómetro, no es constante a lo largo del tiempo. Para caracterizar su comportamiento, durante 17 días se tomó una medida del *offset* promediando medidas durante 5 y 30 minutos, para además poder ver qué tan aleatoria o rápida es su variación. La Fig. 3.2 muestra los resultados obtenidos.

Se puede apreciar que no hay correlación entre las medidas de 5 y de 30 minutos, por lo que no parece que haya una dependencia temporal. Además, los valores medios son prácticamente iguales (varían en la 3ª cifra significativa). Estos dos factores indican que, aunque efectivamente el *offset* cambia con el tiempo, lo hace muy rápido y de forma aleatoria, de manera que utilizaremos el promedio (de las medidas largas) como factor de corrección.

Por último, también a diferencia del acelerómetro, el ruido sí es notorio aún en posición estática. Es por ello que, para reducir su efecto, se añade también un umbral, de forma que β solo se actualizará si la velocidad supera este umbral. Este ruido es muy pequeño en comparación a los movimientos de la mano, por lo que se espera que únicamente filtre los casos en los que la mano está quieta.

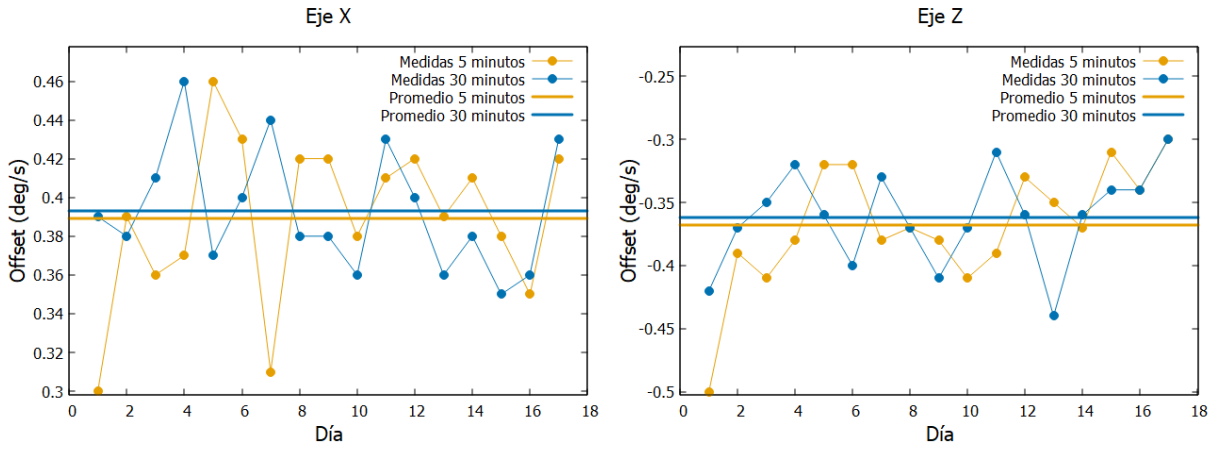


Figura 3.2: *Medidas promediadas del offset para ambos ejes en días distintos.*

4. Procesamiento de datos: flexión de dedos

Mientras que el bloque anterior se encargaba de tratar los datos referentes a la orientación, en este se mostrará el tratamiento de la información recibida de los divisores piezorresistivos concerniente a la posición de los dedos. Como ya se ha mencionado en la introducción, esta labor será llevada a cabo por un pequeño módulo de inteligencia artificial, pues aunque sería posible ir ajustando cada posible posición mediante alguna técnica clásica, las nuevas herramientas de implementación de modelos neuronales sobre microcontroladores proporcionan soluciones muy flexibles y computacionalmente ligeras.

Como sabemos, una red neuronal es un conjunto de nodos (o neuronas) interconectados entre sí, de forma que al introducir la información (estímulos) a los nodos de entrada, se ejecutan determinadas operaciones en las sucesivas capas de procesamiento hasta llegar a la capa de salida, donde se nos devuelve un conjunto de valores, las salidas. El interés entonces de estas redes reside en ajustar su comportamiento hasta producir el resultado deseado. Esto se logra mediante un proceso de entrenamiento, donde le proporcionamos a la red datos de entrada junto al resultado esperado. De este modo, la red va modificando los valores de sus conexiones (pesos) buscando mejorar su precisión en la respuesta. Dependiendo de la arquitectura de la red y de la tarea que se pretende resolver, los algoritmos, dimensiones y codificación de los datos variará.

En nuestro caso concreto trabajaremos con una red *fully connected*, donde todos los nodos de una capa conectan con todos los nodos de la siguiente (Fig. 4.1).

Este proceso de ajuste se realiza con librerías especializadas que simplifican las tareas anteriores. En nuestro caso, emplearemos TensorFlow [6]. TensorFlow es un paquete de funciones de Python de código abierto lanzada por Google en el año 2015 que proporciona herramientas para desarrollar una red neuronal de forma mucho más simple.

Sin embargo, antes de comenzar a definir las características de la red, es necesario conseguir un conjunto de datos, el *dataset*, para llevar a cabo el proceso de ajuste del sistema mediante entrenamiento.

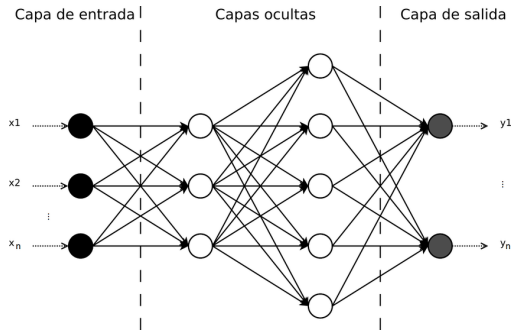


Figura 4.1: *Esquema de una red neuronal fully connected.*

4.1. *Dataset*

El *dataset*, como ya hemos dicho, es el conjunto de datos que se emplea para ajustar el comportamiento de un modelo neuronal a la tarea que se espera que resuelva. En este caso, consistirá en los vectores de entrada que representan los diversos tipos de ejemplos que se quieren aprender, junto con sus correspondientes etiquetas identificativas, que harán de salidas esperadas. Los datos de entrada serán los 4 valores del ADC correspondientes a las lecturas de flexión de cada uno de los dedos, y la salida será una identificación de la posición correspondiente codificada como un número entre 0 y 9. Las posiciones que vamos a ser capaces de distinguir en este trabajo serán las 10 posiciones representadas en la Fig. 4.2.

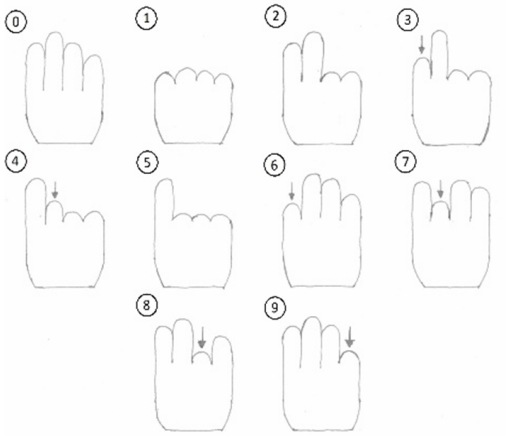


Figura 4.2: *Representación de las posiciones definidas para el data set. Las flechas representan flexión parcial del dedo como ocurre al clicar en el ratón.*

Algunas de estas posiciones se toman por ser estados comunes de la mano, como las posiciones 0 o 1, mientras que otras, como las posiciones 2 a 4, representan la mano usando un ratón y clicando.

Tras seleccionar los tipos de ejemplos que queremos emplear, lo que hacemos es un programa que cada 4 segundos solicite una posición de las indicadas en la Fig. 4.2, y lea las posiciones de los dedos y almacene las lecturas en un fichero de texto. Para dar la mayor flexibilidad a la red, el *dataset* se tomó en 5 días distintos, de modo que al colocar cada día el guante algo distinto o variar la posición del brazo esperamos cierta dispersión.

En total, se obtuvieron 1953 datos, lo que son unos 200 datos para cada posición. Podemos ver la distribución de los datos en la Fig. 4.3. En cuanto al reparto de los datos, el 80 % irán destinados al entrenamiento, el 10 % a la validación, y el 10 % restante al test, seleccionados aleatoriamente.

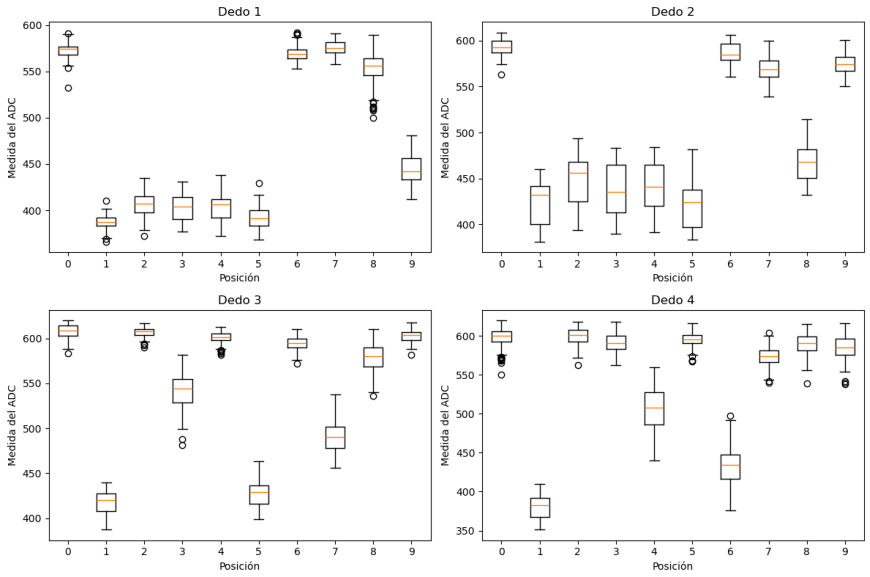


Figura 4.3: *Gráficos de bigotes y cajas que muestran la distribución de valores obtenidos para cada dedo en cada posición. Se puede observar, por ejemplo, como la posición 0 tiene todos los valores cerca de 600 (dedos estirados, mano abierta en la Fig. 4.2), mientras que la 1 muestra todos los valores bajos (dedos flexionados, mano cerrada).*

Además, es importante asegurarse que no haya un número mucho mayor de alguna de las posiciones que del resto, pues esto podría producir sesgos en la red. Aunque a lo largo de los días se observó distribuciones desiguales, la distribución final (Fig. 4.4) quedó bastante uniforme.

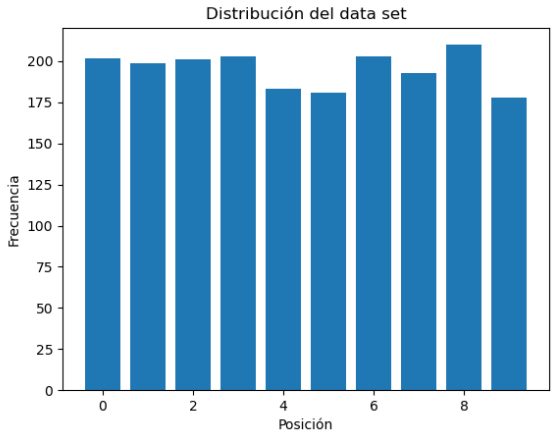


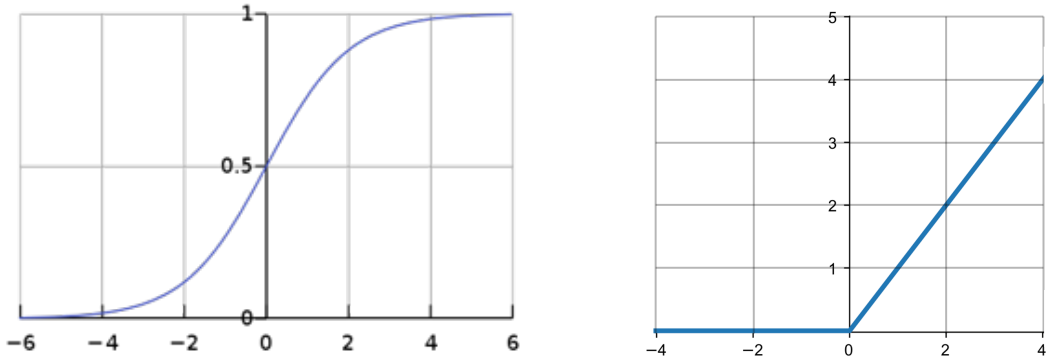
Figura 4.4: *Histograma del número de datos en el dataset.*

Por conveniencia, lo siguiente que haremos con el conjunto de datos es escalarlo acotando todos los valores entre 0 y 1. La ventaja de esto es que cada vez que utilicemos el guante, podremos igualmente escalar los datos respecto a un calibrado previo, y de este modo todo funcionará correctamente, aunque haya ligeros cambios en el funcionamiento de los sensores u operación del usuario.

4.1.1. *One-hot encoding*

Hasta ahora, cada conjunto de datos de entrada se ha guardado junto a su salida esperada, un valor entre 0 y 9. Sin embargo, al ser un problema de clasificación de datos, se ha escogido una codificación de la salida denominada *one-hot encoding* [7]. En esta, la red cuenta con tantas salidas como tipos de datos queremos clasificar (en nuestro caso 10). Una vez entrenada, la red proporciona como salida un vector de 10 componentes donde cada una de ellas representa la probabilidad de que el vector de entrada presentado corresponda a cada una de las posiciones. Por ello, las salidas del *dataset* (valor numérico entre 0 y 9) son sustituidas por un *array* de 10 valores, con un 0 en todas las posiciones menos en la que se corresponda con la posición del dato, que tendrá un 1, pues la probabilidad de que esa sea la posición es del 100 %. Por ejemplo, un dato etiquetado como posición 0 (mano abierta, Fig. 4.2), le asociaremos el *array* "1000000000", mientras que la etiqueta 4 (corazón extendido e índice flexionado) se asociaría con "0000100000".

El *one-hot encoding*, emplea en los procesadores de la capa de salida la función de activación *softmax* (Fig. 4.5a). Esta función, como podemos ver, devuelve un valor entre 0 y 1, y, por tanto, es la encargada de que a la salida tengamos las probabilidades de cada estado. Sin embargo, esta función no se encuentra implementada en la librería empleada en la simplificación de modelos neuronales para su exportación a microcontroladores. Por simplicidad, la sustituiremos por la función *ReLU* (Fig. 4.5b), presente en el resto de procesadores de la red, dado que es muy simple y por ende muy versátil y rápida. Aunque con ello perdemos la información de probabilidad que da *softmax*, los resultados experimentales que se obtienen en este trabajo siguen siendo satisfactorios.



(a) *Softmax.* (b) *ReLU.*

Figura 4.5: *Funciones de activación.*

4.2. Entrenamiento y reducción de la red

Buscamos conseguir una red lo más pequeña posible manteniendo una precisión cercana al 100 %, pues a más pequeña sea la red, menos memoria ocupará en el microcontrolador y más rápida será.

Concluimos que es suficiente con redes de 4 capas, correspondiéndose 2 de ellas a las capas de entrada y salida, que ya tienen fijado el número de neuronas en 4 y 10 respectivamente. Se probarán entonces distintas configuraciones de las 2 capas ocultas. En la Fig. 4.6, vemos cómo evoluciona la precisión de la red con el tamaño de la primera capa oculta, habiendo fijado la segunda en 15 neuronas.

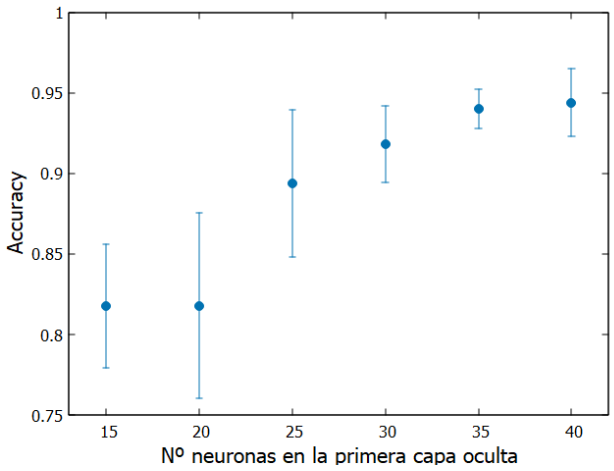


Figura 4.6: *Evolución de la precisión de la red con el tamaño de la primera capa oculta, fijada la segunda en 15 neuronas.*

A partir de 35 neuronas en la primera capa la precisión no crece, por lo que se entrenan varias redes con distintos tamaños en torno a estos valores (35-15, siendo los números de neuronas en 1ª y 2ª capa respectivamente), cambiando el tamaño de ambas capas, pero guardando solo aquellos con al menos un 95 % de precisión. Obtenemos redes que van de tamaño 30-15 a 40-20.

Por otro lado, durante el entrenamiento podemos ajustar los parámetros: número de épocas y *dropout*. El número de épocas es el número de iteraciones que se realizan en el entrenamiento, mejorando la precisión al aumentar este parámetro. Sin embargo, a partir de un punto la red deja de mejorar, ya que alcanzará una situación en la que cualquier cambio que intente hacer le lleva a dar peores resultados (un óptimo local), que no significa que la red esté en su configuración óptima.

Para eso introducimos el *dropout*, que reinicia una parte de los pesos entre épocas, forzando a la red a reentrenarse en cada paso, evitando así los atascos, pues puede encontrar una nueva solución que no lo lleve a un óptimo local y, por tanto, a un atasco. Esto permite alcanzar precisiones del 99 %, aunque supone tiempos de entrenamientos más largos.

4.3. TensorFlow Lite

El siguiente paso es introducir la red en el Arduino. Para ello, primero se usará la librería TensorFlow Lite de Python para comprimir la red [8], y una adaptación de esta para Arduino (*Eloquent* [9]) que se encargará de interpretarla.

El proceso de compresión consiste en eliminar todo aquello que ya no sea necesario en la red por un lado, y por otro reducir la precisión de los parámetros de la red. Respecto lo primero, se convertirán todos los valores de la red de variables a constantes y se eliminará toda la arquitectura necesaria solo en el entrenamiento de la red, siendo esta la forma definitiva de la red. En cuanto a la reducción de precisión, estaremos pasando de punto flotante a punto fijo en 8 bits, reduciendo bastante el espacio en memoria.

La compresión nos devuelve un fichero ".h" que cargaremos en nuestro programa de Arduino con la librería correspondiente. La precisión de las redes puede verse alterada por el proceso de compresión, por lo que realizamos pruebas para ver qué redes han perdido precisión y cuáles no. Además, es importante medir los tiempos de respuesta de cada una de ellas, pues como hemos mencionado antes, necesitamos que el tiempo de ejecución del programa sea lo más rápido posible.

Observamos que muchas de las redes son incapaces de distinguir bien las posiciones 2, 3 y 4 (posiciones con índice y corazón estirados o flexionando un poco uno de ellos, Fig. 4.2). Durante el entrenamiento de la red, inicialmente estas posiciones eran las responsables de la mayor parte del error, pues son bastante similares, pero vemos que todavía es mayor el error al comprimir.

En cuanto a los tiempos, la red más pequeña tarda 0.46 ms, mientras que la más grande tarda 0.52 ms. Una diferencia de 0.06 ms respecto a los 10 ms que aspiramos a alcanzar no debería suponer un problema, por lo que usaremos la red de mayor tamaño, pues además es la que menos problemas da discerniendo entre las posiciones 2, 3 y 4. Podemos ver su arquitectura, excluyendo la capa de entrada de 4 neuronas, en la Fig. 4.7.

Complementando a la red, se implementó un sistema que solo considerara un cambio en la posición tras medir la nueva posición durante un tiempo determinado. Esto es porque el sistema mide posiciones de la mano cada 10 ms, que es un tiempo muy inferior a la capacidad del usuario de cambiar la posición de la mano, lo que da lugar a que en transiciones entre posiciones o por error de la red, midamos posiciones incorrectas puntualmente.

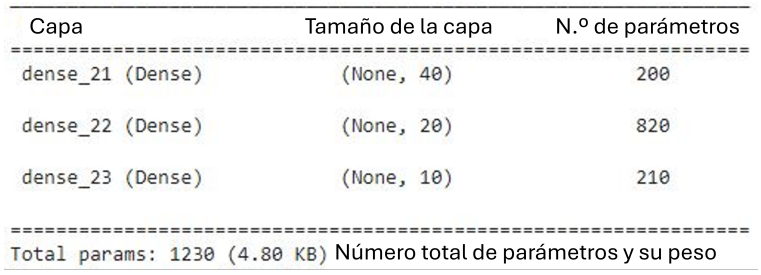


Figura 4.7: *Tamaño de las distintas capas de la red, número de parámetros entrenables, y tamaño antes de la compresión. Falta la capa de entrada que es dada por implícita en TensorFlow, y es de tamaño 4. Al ser una red fully conected, el número de parámetros se obtiene como el número de neuronas de la capa actual por el número de neuronas de la anterior (pesos de las conexiones) más el número de neuronas de la capa actual (bias).*

5. Conectividad

Habiendo procesado ya los datos, lo siguiente es hacerlos llegar desde el microcontrolador al ordenador que queremos controlar. Estos datos son los ángulos α, β y la posición de la mano, y su envío se realizará mediante WiFi usando el módulo NINA-W102 [10].

5.1. NINA-W102

Este módulo de la empresa u-blox puede establecer conexión mediante bluetooth (normal y BLE), y mediante WiFi 802.11b/g/n. Como ya hemos comentado, en nuestro caso se probará la conexión mediante WiFi, pues a pesar de su mayor costo energético, basándonos en la experiencia del trabajo previo, puede ser un método de conexión más simple.

El WiFi 802.11b/g/n es el que comúnmente encontramos como 2.4 G. Actualmente, lo normal al conectarse a una red WiFi es encontrar redes de 2.4 G y de 5 G. Como el módulo solo opera con redes de 2.4 G, habrá que usar una red WiFi que emita en esta banda.

5.2. Estructura de la red

El siguiente paso es conseguir una red WiFi a través de la cual podamos conectar el Arduino con el ordenador. Esta red tendrá que cumplir algunos requisitos:

- **Nombre y clave:** Dentro del código del Arduino, debe estar incluido el nombre y la clave de la red a la que va a conectarse, y aunque sería posible cambiarla cada vez en el Arduino, es mucho más cómodo que la red siempre tenga el mismo nombre y clave.
- **Red 2.4 G:** Como ya hemos dicho, la red tendrá que tener banda de 2.4 G si queremos que pueda conectarse con el Arduino.
- **Portátil:** Sería conveniente, para poder usar el guante en distintos dispositivos, poder transportar la red allá donde fuéramos.

Con estos puntos en mente, la mejor solución posible sería disponer de un dispositivo que pueda hacer de *hotspot* (punto de acceso), y debido a su masificación, el teléfono móvil parecía la mejor opción, pues siempre habrá uno disponible. Por ende, la idea final será conectar tanto el Arduino como el ordenador al móvil, en el cual crearemos un punto de acceso con nombre y contraseña conocidos (los proporcionados al Arduino).

El único problema de este método es que el móvil es incapaz de estar conectado a una red WiFi de una banda y operar como *hotspot* en esta misma. Por lo que si el móvil llegara a conectarse a una red 2.4 G, el sistema deja de funcionar. Para evitar esto es conveniente que el móvil tenga siempre desactivada la conexión a redes WiFi durante el tiempo que se quiera usar el guante.

5.3. Protocolo de comunicación

En cuanto al protocolo utilizado, valoramos como opciones usar TCP y UDP. El primero, *Transmision Control Protocol (TCP)*, se caracteriza por un sistema de verificación de paquetes. Es decir, que es necesario crear un canal de comunicación entre el Arduino y el ordenador, y que cada paquete de información que queramos enviar deberá ser confirmado por el receptor. Este protocolo tiene la ventaja de asegurarnos que no perdemos ningún paquete, pero ralentiza el proceso, hasta el punto de poder perder unos segundos en las pruebas realizadas [11].

Además, el *User Datagram Protocol (UDP)* elimina la verificación de paquetes, de modo que el emisor continúa enviando paquetes lleguen o no a su destino. Esto asegura continuidad en el envío de datos y por ende una velocidad mayor, pero sobre todo más constante.

A excepción de los datos en la fase de calibrado, el Arduino mandará continuamente datos cada 10 ms. Esto significa que, si queremos que el ratón responda con una latencia pequeña, no podemos permitir atascos en la recepción de datos. Por otro lado, perder un paquete no supondría ninguna molestia, pues simplemente nos quedaríamos sin información del ratón durante los 10 ms que tarde en llegar otro, que es mucho mejor que algunos segundos de retraso. Con todo esto, elegimos el protocolo UDP para el envío de datos.

Por otro lado, usando UDP no necesitamos crear un canal de comunicación, por lo que no es necesario que el ordenador se comunique con el Arduino para transmitirle su IP, sino que el Arduino transmitirá por *broadcast*, lo que quiere decir que enviara los datos a todos los miembros de la red, no teniendo que preocuparnos por la IP del ordenador. Esto además no consumirá tiempo extra al Arduino, pues la distribución del mensaje a todos los dispositivos la realiza el móvil encargado de hacer de punto de acceso.

5.4. *Firewall*

Un problema que encontramos es que en la mayoría de ordenadores, al conectarse a la red WiFi del móvil, considera esta como una red pública, lo que se traduce en que el firewall bloquea los datos procedentes del Arduino.

Entonces, aunque podríamos apagar el *firewall*, no es lo recomendado, por lo que vamos a crear una excepción que permita la llegada de datos desde el guante. Durante el proceso de desarrollo, tenemos que crear una excepción para Python, que es donde estamos realizando el código del ordenador. En algunos dispositivos esta excepción viene preestablecida.

Por otro lado, la intención es poder usar el guante en cualquier ordenador, disponga o no de Python, por lo que será necesario crear una aplicación (convertir el código en un fichero .exe [12]). Con esto en mente, añadimos en el código una sección que cree una excepción en el firewall para la propia aplicación al principio del código, asegurando así la llegada de datos al ordenador. El desarrollo de la aplicación queda fuera de este trabajo, pero se creó un prototipo de esta que verifica el correcto funcionamiento de la excepción.

5.5. Codificado de los paquetes

Asegurada la conexión entre el ordenador y el Arduino, buscaremos codificar los datos de tal modo que requieran el menor número de bytes posibles, o equivalentemente *chars*, pues de este modo reduciremos el tiempo de envío.

Respecto a los ángulos, según la precisión que queramos, usaremos más o menos bits para cada uno. Como en principio el guante no va a trabajar en todas las direcciones, sino que solo operará al frente, de modo que solo consideraremos ángulos de -128^{o} a $+128^{o}$ tanto para α como para β. Aunque 128^{o} es mucho más de lo que esperamos necesitar, es más conveniente durante la codificación al ser una potencia de 2.

En cuanto a la resolución, parece que con 0.25^{o} será suficiente. Esto significa que necesitaremos 10 bits para cada ángulo, 8 para la parte entera y 2 para la parte decimal, trabajando así los decimales en punto fijo (Fig. 5.1). Para esto necesitamos únicamente 3 *bytes* (24 bits), sobrándonos 4 bits que podrían ser utilizados para aumentar la resolución o enviar la posición de la mano.

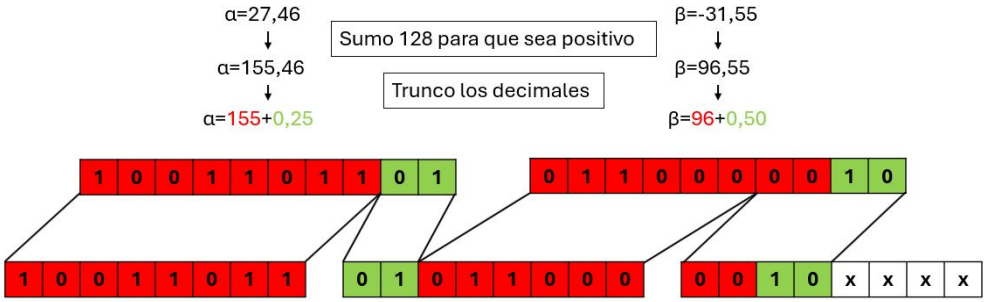


Figura 5.1: *Esquema del proceso de codificación de los ángulos en 3 bytes.*

Sin embargo, la posición de la mano la enviaremos en un *byte* aparte. Esto es porque, para acelerar el código, solo enviaremos el ángulo cuando sea necesario, mientras que la posición de la mano se mandará siempre. Por ende, preferimos separar la información de ambos para facilitar la decodificación en el ordenador.

Un detalle relevante es que por como almacena Arduino los datos a enviar (en *chars*, 8 bits), no podemos tener un *char* que todos sus bits sean 0, pues lo interpreta como nulo y deja de leer. Por ello, a la hora de codificar la posición de la mano sumamos un 1 que luego eliminaremos en la codificación. Para los ángulos, en el caso de que alguno de los *bytes* sea todo 0's, sumaremos a ambos ángulos (para no perder tiempo en identificar de donde viene el problema) 0.25^{o} (1 bit). Esto se traduce en que hay algún ángulo que es inaccesible, aunque no debería ser un problema.

6. Interfaz para control remoto

Establecida la comunicación entre el Arduino y el ordenador, solo queda dar un uso a los datos recibidos. Dentro de los usos se incluyen el control del ratón y otras funciones del ordenador, así como la representación en pantalla de desplegables para acceder distintas funcionalidades. Por ello, requeriremos de la librería win32api para la parte del ratón y de la biblioteca PyQt5 para la parte gráfica.

La librería win32api sirve para controlar muchas de las funciones del propio sistema operativo Windows. En nuestro caso, nos interesaremos únicamente por aquellas que tienen que ver con el control de dispositivos externos como teclado o ratón, lo que nos permitirá mandar órdenes que emulen un ratón desde Python.

En cuanto a PyQt5, es una librería para diseño gráfico, por lo que nos servirá para desarrollar menús que faciliten la comunicación entre el usuario y el ordenador, haciéndose innecesario emplear la terminal de Python para interactuar con el ordenador.

Con esto, y teniendo en cuenta todos los procesos explicados a lo largo de la memoria, se puede observar en la Fig. 6.1 un esquema de cuál será el proceso desde la adquisición de datos hasta que dan lugar a una acción en el ordenador.

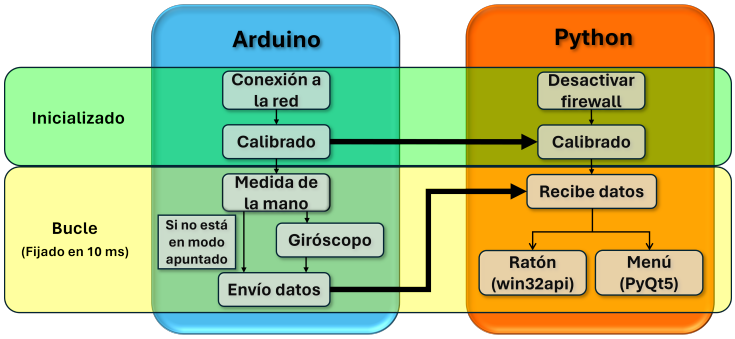


Figura 6.1: *Diagrama de flujo del funcionamiento del sistema ordenador/Arduino.*

Vemos que dentro del bucle pueden ocurrir dos cosas distintas. Esto se debe a que, aunque inicialmente se planteó el uso como ratón, al haber entrenado la red con todas las posiciones indicadas en la Fig. 4.2, se quiso aprovechar todas ellas. Por ello, se dividió en dos bloques, las posiciones para el ratón, que son las posiciones 2, 3, 4 y 5; y las posiciones de opción, que servirán para otras funcionalidades, así como manejar un menú de opciones.

Es por eso que en la Fig. 6.1, vemos en el Arduino que si no estamos en modo apuntado (posiciones 2, 3, 4 o 5), no hacemos uso de la IMU. A su vez, en Python usaremos o bien el ratón o bien el menú, según las posiciones que pongamos. Realmente muchas veces las posiciones se intercalarán, pero esa es la idea general.

Destacar también la etapa de calibrado. En esta establecemos una relación entre ángulo y posición de la pantalla, por un lado, y por otro obtenemos los máximos y mínimos de extensión de los dedos para poder escalar los valores entre 0 y 1 como se había nombrado en la Subsección 4.1.

7. Resultados

7.1. Tiempos

Durante todo el desarrollo, uno de los objetivos ha sido conseguir que el código en el Arduino sea lo suficientemente rápido como para asegurar que se aprovechaban todas las medidas del giróscopo y de este modo se tuviera el menor error de integración posible.

Teniendo ya la versión final del código, lo que vamos a hacer es medir el tiempo de cada uno de los procesos que ocurren durante la parte del bucle (Fig. 6.1). Se obtienen entonces los siguientes tiempos:

	ADC	Red neuronal	Orientación	Envío de datos	Otros	Bucle completo
Tiempo (ms)	0,168 ± 0,002	0,571 ± 0,005	2,93 ± 0,14	1,33 ± 0,03	0,3 ± 0,2	5,26 ± 0,14

Tabla 4: *Tiempos de ejecución para los principales procesos mencionados a lo largo del trabajo.*

Podemos ver que casi todo el peso recae sobre la parte de orientación y el envío de datos mediante WiFi. Aun así nos sobra mucho tiempo respecto a la marca de 10 ms propuesta inicialmente, lo que es una muy buena señal si se quisiera transportar nuestro código a un microcontrolador menos potente en un futuro.

7.2. Funcionamiento final

Como ya hemos mencionado en la Sección 6, vamos a definir funcionalidades para nuestro dispositivo, distinguiendo hasta cierto punto entre los modos de ratón o apuntado, y los modos de menú. Dentro de este último, vamos a distinguir entre las posiciones 0 y 1 (Fig. 4.2), cuya función es abrir y cerrar el menú principalmente, y los botones, que se corresponden con las posiciones 6, 7, 8 y 9 de la Fig. 4.2.

7.2.1. Ratón

Este se activa al tener la mano una de las posiciones 2, 3, 4 o 5. A excepción del modo pintar que veremos más adelante, este siempre tiene el mismo funcionamiento. En la posición 2 apuntamos con el ratón, en las posiciones 3 y 4 hacemos clic derecho e izquierdo respectivamente, y para el doble clic se usará la posición 0 (solo tiene esta función si estamos en modo ratón). En cuanto a la posición 5, esta sirve para usar el guante a modo de puntero.

Por otro lado, los denominados botones también nos sirven aquí, pudiendo además elegir su funcionalidad según distintos modos que se pueden acceder y seleccionar en el menú (Subsección 7.2.2). Estos modos son:

- **Ratón:** Este modo asocia a los botones las acciones que normalmente haría la rueda del ratón.
- **Teclado:** En este modo, asociamos a los botones las flechas de adelante y atrás, así como la tecla de espacio. Esto es porque a la hora de reproducir videos o de hacer una presentación, estas teclas nos servirán para pausar (en el caso de los videos), avanzar y retroceder.
- **Off:** Este modo no solo desactiva los botones, sino que desactiva también el poder usar el guante en modo ratón. Sirve para evitar mandar órdenes indeseadas mientras se realiza otra acción como podría ser usar el teclado del ordenador.

7.2.2. Menú

La posición con la mano abierta (posición 0) la hemos considerado la posición de reposo. Si cerramos (posición 1) y abrimos la mano, por un lado se inhabilitará el modo ratón, y por otro se abrirá un menú donde encontraremos nuevas funcionalidades (Fig. 7.1).

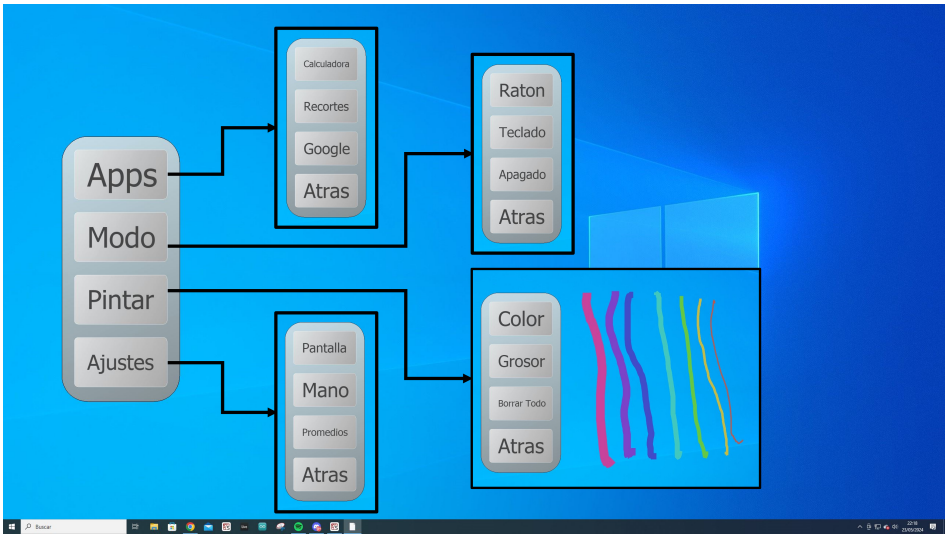


Figura 7.1: *Menú desplegable para acceder al resto de funciones, pudiendo ver las opciones dentro de cada submenú y una muestra del funcionamiento del modo pintar.*

En todo momento el menú constará de 4 opciones, de modo que los botones (posiciones 6, 7, 8 y 9) nos permitirán navegar por este. El menú se cierra del mismo modo que se abre, cerrando la mano. Dentro del menú, distinguimos las siguientes opciones:

- **Apps:** Permite el acceso directo a distintas aplicaciones.
- **Modo:** Permite cambiar entre los modos de ratón explicados en la sección anterior.
- **Pintar:** Nos permite pintar sobre la pantalla como se muestra en la Fig. 7.1.
- **Ajustes:** Permite repetir las calibraciones iniciales o ajustar el número de promedios para suavizar si es necesario el movimiento del ratón.

8. Conclusiones y trabajo futuro

En este trabajo se ha abordado la implementación de un pequeño sistema de inteligencia artificial sobre un microcontrolador de bajos recursos como modelo de *edge computing*. Para ello, se ha propuesto como ejemplo el desarrollo de un sistema de identificación de la posición de la mano, tanto de su posición espacial como de la flexión de los dedos, siendo esto último para lo que se ha desarrollado el modelo neuronal.

Para ello, en primer lugar, se ha empleado una herramienta de desarrollo de redes neuronales del alto nivel (TensorFlow), obteniendo una arquitectura neuronal lo suficientemente pequeña en cuanto al número de parámetros para poder ser incluida en el microcontrolador, y con tiempos de ejecución que no provoquen sensación de retraso en la interacción. Posteriormente, se ha empleado una versión compatible con el microcontrolador escogido de la librería TensorFlow Lite para la eliminación de hiperparámetros, reducción numérica y su inclusión en el código de control del micro.

Complementariamente, sé desarrollo un protocolo sencillo de transmisión de la información a través del estándar WiFi que permite su recepción en el dispositivo de destino. En este caso, se ha escogido un PC en el que los datos recibidos desde el micro son interpretados como diversas acciones ejecutables por el ratón, entre otras cosas.

Aunque el conjunto funciona sin problemas, el trabajo ha dejado fuera aspectos como:

- **Revisión del protocolo de comunicación:** Aunque se descartó directamente en nuestro caso usar algo que no fuera WiFi, es posible que usar bluetooth (BLE) sea una buena opción debido a su bajo consumo. Así mismo, sería conveniente eliminar la necesidad de un nodo central haciendo de *hotspot*.
- **Consumo energético:** En ningún momento del proceso hemos estudiado cuánto consume realmente el dispositivo. En medidas rápidas se vieron consumos pico de 100 mA (durante el envío por WiFi), que es un consumo bastante alto para un microcontrolador. El cambio a BLE, el uso de otros microcontroladores alcancen las prestaciones con menor consumo, u optimizar fases del proceso, podrían suponer una disminución considerable del consumo.
- **Diseño del guante:** Aunque actualmente el diseño es funcional, la pieza central que sujeta al Arduino es algo aparatosa, además de que la banda elástica puede llegar a hacer difícil ponerte el guante. Por otro lado, la batería no tiene un sitio en el que alojarse, por lo que sería conveniente conseguir una batería más pequeña e incluirla en la pieza central.
- **Añadir funcionalidades:** Además de mejorar las actuales o incluso dar una mayor profundidad al menú. También se podría añadir una red que se encargara de medir movimientos de mano, de forma que tuviéramos otro modo de interactuar con el ordenador.

En definitiva, con los desarrollos realizados se ha conseguido un sistema *edge computing* que con ayuda de una pequeña red neuronal nos permite interaccionar de forma inalámbrica con el ordenador, y aunque todo el proceso funciona sin problemas, sería conveniente un mayor estudio para conseguir un resultado sobre todo más eficiente.

Bibliografía

[1] Rubén Muñoz. *Estudio sobre la integración de machine learning en dispositivos de IoT. Trabajo de fin de grado en Física*. 2021.

[2] *Arduino RP2040*. URL: https://docs.arduino.cc/hardware/nano-rp2040-connect/ (visitado 24-05-2024).

[3] *LSM6DSOX*. URL: https://www.st.com/en/mems-and-sensors/lsm6dsox.html (visitado 20-05-2024).

[4] *Librería LSM6DSOX*. URL: https://github.com/arduino-libraries/Arduino_LSM6DSOX (visitado 20-05-2024).

[5] *Flex-Sensor*. URL: https://www.spectrasymbol.com/resistive-flex-sensors/spectraflex-flex-sensors (visitado 20-05-2024).

[6] *TensorFlow*. URL: https://www.tensorflow.org/?hl=es (visitado 21-05-2024).

[7] *One-hot encoding*. URL: https://medium.com/thedeephub/one-hot-encoding-with-tensorflow-and-keras-569cfa91b162 (visitado 21-05-2024).

[8] Gian Marco Iodice. *TinyMl Cookbook*. Packt Publishing Ltd, 2022.

[9] *Librería eloquentarduino*. URL: https://github.com/eloquentarduino (visitado 20-05-2024).

[10] *NINA-W102*. URL: https://www.u-blox.com/en/product/nina-w10-series-open-cpu (visitado 22-05-2024).

[11] *Wireshark*. URL: https://www.wireshark.org/ (visitado 11-06-2024).

[12] *Auto-py-to-exe*. URL: https://pypi.org/project/auto-py-to-exe/ (visitado 24-05-2024).

Anexos

A. Cálculo de la resistencia óptima

Para obtener el valor óptimo de la resistencia fija (R_{fija}), lo que tenemos que hacer es obtener cuánto vale el voltaje para R_{min} y R_{max} (valores máximo y mínimo de R_{piezo} respectivamente) en función de R_{fija}. Para esto, partimos de que en un divisor resistivo como el propuesto (Fig. 2.2), el voltaje es:

$$V_{divisor}(R_{piezo}) = 3,3\,\frac{R_{fija}}{R_{piezo}+R_{fija}} \Longrightarrow \begin{cases} V_{max}(R_{piezo}) = 3,3\,\frac{R_{fija}}{R_{min}+R_{fija}} \\ V_{min}(R_{piezo}) = 3,3\,\frac{R_{fija}}{R_{max}+R_{fija}} \end{cases} \tag{A.1}$$

Para conocer la variación, únicamente tenemos que hacer la diferencia, obteniendo:

$$\Delta V = 3,3\left(\frac{R_{fija}}{R_{min}+R_{fija}} - \frac{R_{fija}}{R_{max}+R_{fija}}\right) = 3,3\,\frac{R_{fija}(R_{max}-R_{min})}{(R_{min}+R_{fija})(R_{max}+R_{fija})} \tag{A.2}$$

Ahora hay que derivar la función ΔV en función de R_{fija}, de modo que si luego igualamos a 0, localizaremos los óptimos de la función. Obtenemos:

$$\frac{\mathrm{d}(\Delta V)}{\mathrm{d}(R_{fija})} = 3,3\,(R_{max}-R_{min})\frac{(R_{min}+R_{fija})(R_{max}+R_{fija}) - R_{fija}(2\,R_{fija}+R_{max}+R_{min})}{((R_{min}+R_{fija})(R_{max}+R_{fija}))^2} \tag{A.3}$$

Que podemos simplificar e igualar a 0, llegando a:

$$\frac{\mathrm{d}(\Delta V)}{\mathrm{d}(R_{fija})} = 3,3\,(R_{max}-R_{min})\frac{-R_{fija}^2 + R_{min}R_{max}}{((R_{min}+R_{fija})(R_{max}+R_{fija}))^2} = 0 \tag{A.4}$$

De ahí, solo el numerador puede ser 0, por lo que obtenemos:

$$-R_{fija}^2 + R_{min}R_{max} = 0 \Longrightarrow R_{fija} = \sqrt{R_{min}R_{max}} \tag{A.5}$$

Que será la solución óptima para la resistencia. Formalmente, el siguiente paso sería verificar que $\left(\frac{\mathrm{d}^2(\Delta V)}{\mathrm{d}(R_{fija})^2}\right) < 0$ para que nuestra solución sea un máximo. Sin embargo, no será necesario, pues si volvemos a la ecuación (A.4), vemos que a excepción del numerador, todos los términos son siempre positivos, el denominador porque está al cuadrado y $(R_{max}-R_{min})$ por la definición de estos valores. Con esto en mente, vemos que sí $R_{fija} < \sqrt{R_{min}R_{max}}$, la derivada será positiva, y sí $R_{fija} > \sqrt{R_{min}R_{max}}$, la derivada será negativa, lo que significa que estamos en una situación de máximo.

Podemos ver esto en la Fig. A.1, donde en el eje X vemos la relación $R_{fija}/R_{min} = x$, y en el eje y veremos tanto la derivada (que debería cumplir lo que acabamos de mencionar), como ΔV. Además, siguiendo con las especificaciones de las piezorresistencias [5], hemos considerado $R_{max} = 2R_{min}$, de modo que esperamos el máximo en $R_{fija} = \sqrt{2}R_{min}$, o equivalentemente $x = \sqrt{2}$.

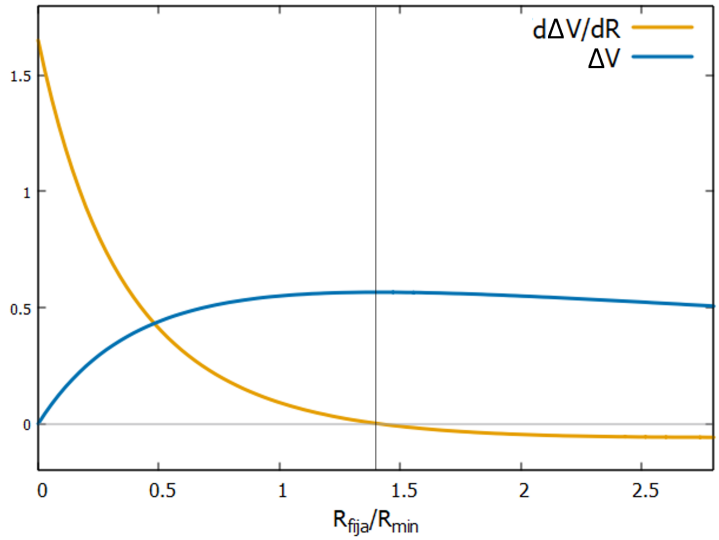


Figura A.1: *Gráficas en las que se observan ΔV y su derivada, observando cuando la derivada se iguala a 0 y como en ese punto ΔV es máximo. Aunque se representen bajo el mismo eje y, las unidades de ΔV son V, y las de su derivada son V/Ω. El eje x si que es adimensional.*

Además de ver que es máximo, vemos que la variación de voltaje esperada es en torno a 0,5 V, y que cambios en R_{fija} son muy poco notables en torno al óptimo. Por ello, no disponer justamente del valor resistivo calculado teóricamente no será muy relevante. Por otro lado, cabe esperar una variación de unas 176 unidades del ADC, como comprobaremos experimentalmente (Fig. 2.3).

B. Obtención de las ecuaciones para el sistema de coordenadas

Partiendo del esquema representado en la Fig. 3.1, situamos en el centro de la esfera nuestro microcontrolador, y asociamos el vector dirección al eje y del microcontrolador (Fig. B.1). Durante todo el desarrollo, se ignorará el sentido de los ejes por simplicidad, pues se puede corregir al final.

Remarcar que el uso de seno o coseno, así como los signos en lo respectivo a θ es una decisión arbitraria para simplificar el código, ya que, por ejemplo, a_z/a_x no es capaz de distinguir entre cuadrantes (dividir dos positivos o dos negativos da el mismo resultado), devolviendo únicamente valores de -90^{o} a $+90^{\text{o}}$. Esto se soluciona añadiendo una condición que sí que distinga cuadrantes, y para hacer θ continua bajo esa condición en el rango que se espera que trabaje, se deben usar senos y cosenos como se muestra en (3.3) y obtendremos aquí.

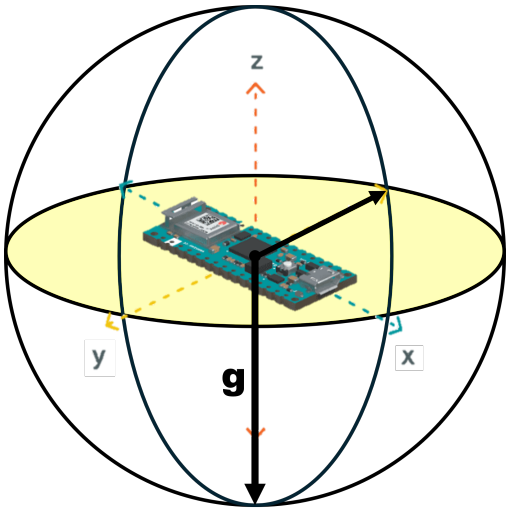


Figura B.1: *Esquema del sistema de referencia centrado en el microcontrolador, siendo g la dirección de la gravedad y representando la flecha la dirección en la que apunta la mano, que vemos que coincide con el eje y del microcontrolador.*

Lo primero que hacemos es, suponiendo $\theta = 0^{\text{o}}$, ver la relación de α con las aceleraciones (Fig. B.2a). En este caso, al rotar en α, cambian los valores de a_y y a_z. Sin embargo, si consideramos rotación en el eje y ($\theta \neq 0^{\text{o}}$), vemos en la Fig. B.2b que realmente lo que cambia es a_y y la composición de a_x y a_z (a_{xz}). Aunque podríamos usar cualquiera de ambos, lo más rápido será usar a_y. Se ve también que sí $\alpha = 0^{\text{o}}$, g no proyecta nada sobre el eje y, mientras que sí $\alpha = 90^{\text{o}}$, g estaría completamente sobre el eje y. Con esto, llegamos a:

$$sin(\alpha) = \frac{a_y}{g} \tag{B.1}$$

(a) *Ejes del microcontrolador al rotar α con $\theta = 0^o$.*

(b) *Ejes del microcontrolador al rotar α con $\theta = 0^o$.*

Figura B.2: *Representación de los ejes del microcontrolador partiendo de la Fig. B.1 en función de α.*

La composición de todos los ejes debería estar midiendo continuamente g, por lo que sustituimos en la ecuación anterior g por $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$ y despejamos α:

$$\alpha = arcsin\left(\frac{a_y}{a}\right) \tag{B.2}$$

Obteniendo así la expresión para α. La ventaja de usar a en vez de g, es que la mayor parte del tiempo, el movimiento de la mano provocara que a>g, o incluso a_y>g. Si ocurriera lo segundo, en el caso de usar a_y/g, este valor sería mayor que 1 y la función *arcsin* nos daría error. No ocurre así si usamos a, pues a siempre será mayor que a_y. Es cierto que esto puede dar lugar a que ciertos movimientos de la mano (que no sean giros) alteren α, pero tras varias pruebas, solo llega a ser notorio para movimientos muy rápidos y antinaturales.

Continuamos con θ. Este ángulo depende de la rotación en el eje y, por lo que no dependerá de a_y. Si miramos perpendicular al eje y (Fig. B.3), vemos que no podemos hablar de la relación de a_x o a_z respecto a g, pues esto dependerá de α. Sin embargo, la composición a_{xz} será una componente fijada por α, de forma que puedo conocer θ si conozco cuanto contribuye a a_{xz} cada una de las componentes, lo que se traduce en:

$$tan(\theta) = \frac{a_z}{a_x} \Longrightarrow \theta = arctan\left(\frac{a_z}{a_x}\right) \tag{B.3}$$

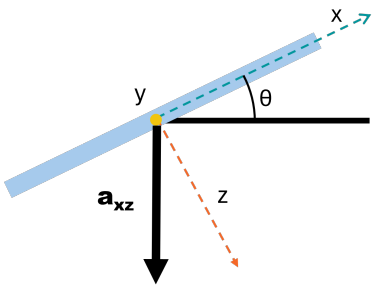


Figura B.3: *Esquema de los ejes del microcontrolador para un α cualquiera y en función de θ.*

Se podría haber usado seno o coseno respecto a a_{xz}, pero eso nos haría perder tiempo calculándolo, y no parece necesario.

Por último, el más difícil de obtener de todos los ángulos es β, pues necesitamos usar el giróscopo. En primer lugar, vemos que ω_y no influye, pues la rotación en y va asociada con θ. De hecho podríamos obtener θ integrando ω_y, pero no tendría mucho sentido.

Sí que necesitaremos ω_x y ω_z, pero tenemos que ver cuánto influye cada una. Podemos ver los casos extremos en la Fig. B.4. De ahí podemos deducir que la aportación a ω_β dependerá de que tanto se proyecten los ejes x y z en la componente vertical, o equivalentemente cuanto aporten a_x y a_z a a_{xz}. Sabemos que $a_x \propto cos(\theta)$ y $a_z \propto sin(\theta)$, por lo que ω_x irá mediada $cos(\theta)$ y ω_z irá mediada por $sin(\theta)$. Tras aplicar los signos correctos obtenemos:

$$\frac{\mathrm{d}\beta}{\mathrm{d}t} \propto (cos(\theta) \cdot \omega_x - sin(\theta) \cdot \omega_z) \tag{B.4}$$

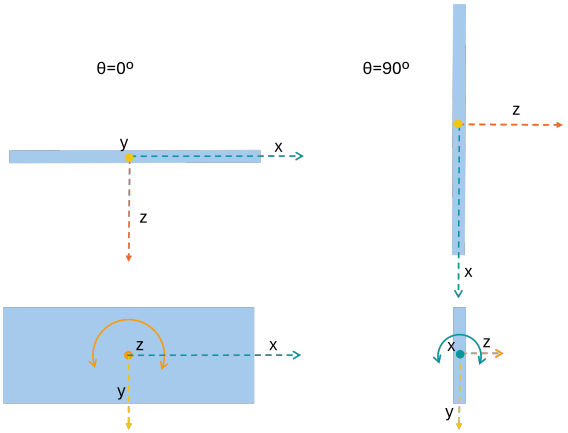


Figura B.4: *Vista de los ejes de un microcontrolador para dos valores distintos de θ y suponiendo $\alpha = 0$. Para cada ángulo, se tienen dos vistas de la figura rotadas 90º, viendo en la inferior la rotación en el plano horizontal.*

Para terminar de darle forma, hay que ver cómo influye α. Para ello, nos fijamos en la Fig. B.5. En esta, vemos que al realizar un desplazamiento en β con un determinado ángulo α, nosotros lo que estamos midiendo de forma directa es β', que será siempre más pequeño que β, que se mide en el plano horizontal.

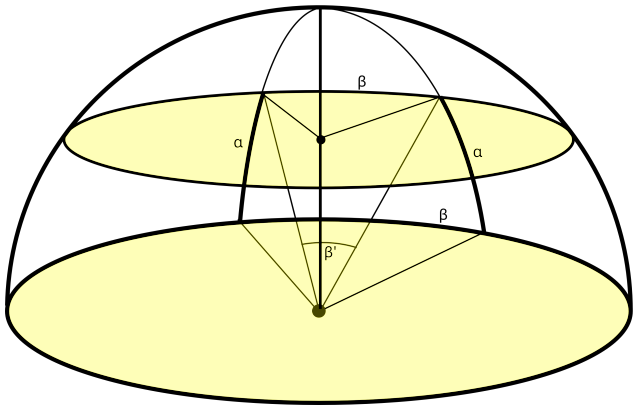


Figura B.5: *Esquema que muestra la relación entre el ángulo medido y el ángulo respecto al plano horizontal en 2 alturas distintas.*

Sin embargo, vemos que el arco de circunferencia trazado por β' es el mismo que el trazado por β en el plano superior. Por ello, lo que hacemos es obtener cuál es el radio en ese nuevo plano (R'), valor que dependerá de α (Fig. B.6). De ahí obtendremos que la relación entre radios es:

$$cos(\alpha) = \frac{R'}{R} \tag{B.5}$$

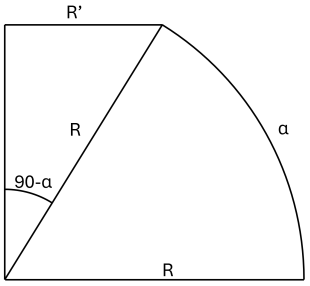


Figura B.6: *Esquema de la relación entre radios para los distintos arcos de la Fig. B.5.*

Con esto, podemos establecer una relación entre arcos de circunferencia, pues $arco = \beta \cdot R$. Igualamos los arcos y obtenemos:

$$\beta' \cdot R = \beta \cdot R' \Longrightarrow \beta = \beta' \frac{R}{R'} = \frac{\beta'}{cos(\alpha)} \tag{B.6}$$

Obteniendo que es necesario un factor de corrección $1/cos(\alpha)$ para que el ángulo que obtenemos sea siempre el ángulo en el plano horizontal.

Con todo esto, llegamos a:

$$\frac{\mathrm{d}\beta}{\mathrm{d}t} = \frac{(cos(\theta) \cdot \omega_x - sin(\theta) \cdot \omega_z)}{cos(\alpha)} \tag{B.7}$$

Pudiendo así juntar todo y obtener la ecuación (3.3).

C. Códigos usados durante el desarrollo (GitHub)

Para esto, se ha creado un repositorio en GitHub donde podemos encontrar los dos programas principales (uno en Python para el ordenador y otro en Arduino para el microcontrolador), además de algunos de los programas auxiliares de los que se hace mención durante la memoria.

Repositorio en GitHub

Se deja espacio además a futuras modificaciones, correspondiéndose la versión 1.0 con la versión en el momento de la entrega de la memoria, y en la que se basan los resultados obtenidos.