_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Interface mediante gestos de los dedos de un sistema incorporable al equipamiento de bomberos

Marcos Fernández Carbonell

Tutor: José Manuel Mossi García

Cotutor: Manuel Palacios Hurtado

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 15 de abril de 2018

Resumen

El presente trabajo final de grado tiene como finalidad el diseño y desarrollo de un sistema de lectura basado en un microcontrolador y un conjunto de sensores ubicados en un guante, que permita la navegación a través de las diferentes opciones de información presentadas sobre un *Head-Up Display*. En primer lugar, se desarrolla el sistema capaz de gestionar los sensores para la lectura de datos de los acelerómetros y de los giróscopos. En segundo lugar, se diseña e implementa una serie de aplicaciones que permitan la captura y preparación de los datos para poder generar un conjunto de muestras de entrenamiento. Por último, como puesta a prueba del trabajo realizado, a partir del set de datos generado, se entrena una red neuronal artificial simple para hacerla capaz de identificar entre un grupo de gestos previamente definidos. Los resultados obtenidos serán utilizados en futuros proyectos con el propósito de mejorar y completar el sistema global que facilitará las labores desempeñadas por el cuerpo de bomberos.

Resum

El present treball de fi de grau té com a finalitat el disseny i el desenvolupament d'un sistema de lectura basat en un microcontrolador i un conjunt de sensors ubicats en un guant, que permeta la navegació a través de les diferents opcions d'informació presentades sobre un *Head-Up Display*. En primer lloc, es desenvolupa el sistema capaç de gestionar els sensors per a la lectura de les dades dels acceleròmetres i dels giroscopis. En segon lloc, es dissenya i implementa una sèrie d'aplicacions que permeten la captura i preparació de les dades per a poder generar un conjunt de mostres d'entrenament. Per últim, com a posada a prova del treball realitzat, a partir del set de dades generat, s'entrena una xarxa neuronal artificial simple per a fer-la capaç d'identificar entre un grup de gestos prèviament definits. Els resultats obtinguts seran utilitzats en futurs projectes amb el propòsit de millorar i completar el sistema global que facilitarà les tasques exercides pel cos de bombers.

Abstract

This final year project aims to design and develop a reading system constituted by a microcontroller and a combination of sensors placed in a glove, allowing navigation through the different information options presented in a Head-Up Display. Firstly, the system capable of managing the sensors for reading data from accelerometers and gyroscopes is developed. Secondly, several applications are designed and implemented in order to capture and prepare data to generate a training set. Finally, as a proof of concept of this study, a simple artificial neural network is trained from the generated data set to classify the different gestures from the group previously defined. The results obtained will be used in future projects with the purpose of improving and completing the global system, so that the work performed by the fire brigade will be facilitated.

Índice

Capítulo 1. Introducción 5
Capítulo 2. Objetivos 6
Capítulo 3. Metodología 7
3.1 Gestión del proyecto 7
3.2 Distribución en tareas 7
3.3 Diagrama temporal 7
Capítulo 4. Desarrollo y resultados 8
4.1 Introducción 8
4.1.1 Arduino 8
4.1.2 MATLAB 8
4.2 Lectura de sensores 8
4.2.1 ESP8266 8
4.2.2 Posibles soluciones para el sistema de control 10
4.2.3 MPU-6050 12
4.2.4 Comunicación mediante bus I^2C 12
4.2.5 Propuesta y desarrollo del sistema de lectura 13
4.2.6 Resultados 18
4.3 Interfaz de usuario para la creación de sets de entrenamiento 20
4.3.1 Controlador Contour ShuttleXpress 20
4.3.2 Aplicación Captura 21
4.3.3 Aplicación Sincronización 23
4.3.4 Aplicación Etiquetado y Exportación 27
4.3.5 Resultados 30
4.4 Red neuronal 30
4.4.1 Introducción 30
4.4.2 Entrenamiento de la red neuronal con MATLAB 32
4.4.3 Resultados 32
Capítulo 5. Presupuesto 36
5.1 Costes de personal 36
5.2 Costes de *hardware* 36
5.3 Costes de *software* 37
5.4 Presupuesto total del proyecto 38
Capítulo 6. Conclusiones y propuesta de trabajo futuro 39
Capítulo 7. Bibliografía 40

Capítulo 8. Anexos 41
8.1 Código fuente lectura MPU-6050..... 41

Índice de figuras

Figura 1. Prototipo de interfaz creada por Joseph Juhnke. 5
Figura 2. Planificación temporal mediante diagrama de Gantt. 7
Figura 3. Módulo ESP-01 y programador. 9
Figura 4. Diagrama de bloques de la tarjeta de desarrollo RobotDyn WiFi NodeM. 9
Figura 5. CaptoGlove. *Wereable* controlador para videojuegos y dispositivos inteligentes. 11
Figura 6. Guante con marcadores para el seguimiento y posicionamiento de los mismos [7]. 11
Figura 7. Módulo GY-521. 12
Figura 8. Cronograma simplificado comunicación I^2C. 13
Figura 9. Conexiones sistema propuesto. 13
Figura 10. Montaje de cuatro MPU-6050 sobre la placa de pruebas. 16
Figura 11. Salida monitor serie durante el proceso de lectura de cuatro módulos. 17
Figura 12. Respuesta al impulso sobre el eje x. 18
Figura 13. Prototipo guante de control. 18
Figura 14. Gesto toque simple dedo índice con pulgar. 19
Figura 15. Efecto provocado por cada uno de los gestos sobre el eje x. 19
Figura 16. Proceso general y función de cada uno de las aplicaciones. 20
Figura 17. Controlador multimedia Contour ShuttleXpress con números de referencia. 20
Figura 18. Flujograma del funcionamiento básico aplicación “Captura”. 21
Figura 19. Diferencia de tiempos entre la captura de un fotograma y la siguiente. 22
Figura 20. Interfaz gráfica aplicación “Captura”. 23
Figura 21. Instantes de referencia para el proceso de sincronización. 24
Figura 22. Interfaz gráfica aplicación “Sincronización”. 25
Figura 23. Flujograma del funcionamiento básico aplicación “Sincronización”. 26
Figura 24. Interfaz gráfica aplicación “Etiquetado y Exportación”. 27
Figura 25. Flujograma del funcionamiento básico aplicación “Etiquetado y Exportación”. 28
Figura 26. Ejemplo desplazamiento ventana 60 muestras (exportación). 29
Figura 27. Sistema completo de creación del set de datos. 30
Figura 28. Estructura perceptrón de una capa. 31
Figura 29. Estructura red *feedforward* con dos capas ocultas. 31
Figura 30. Esquema de la red neuronal empleada. 32
Figura 31. Matrices de confusión de la red neuronal entrenada. 33
Figura 32. Fragmento ejemplo de salida de la red neuronal (toque simple dedo índice). 34
Figura 33. Fragmento ejemplo de salida de la red neuronal con postprocesado. 35
Figura 34. Interfaz gráfica para la clasificación de los gestos en tiempo real. 35

Índice de tablas

Tabla 1. Comparativa de módulos que integran el ESP8266. 10
Tabla 2. Comparativa de tarjetas de desarrollo que integran el ESP8266. 10
Tabla 3. Ejemplo de lectura valores *raw* en posición de reposo. 14
Tabla 4. Sensibilidad acelerómetro. 14
Tabla 5. Sensibilidad giróscopo. 14
Tabla 6. Valores obtenidos aplicando factor de escala. 15
Tabla 7. Valores *raw* con corrección de *offset*. 15
Tabla 8. Ejemplo de lectura eliminando el efecto de la gravedad mediante DMP. 16
Tabla 9. Ejemplo de lectura cuatro IMU en reposo. 17
Tabla 10. Costes de personal. 36
Tabla 11. Costes de *hardware* guante 36
Tabla 12. Costes resto de *hardware*. 37
Tabla 13. Costes de *software*. 37
Tabla 14. Presupuesto total del proyecto siendo alumno de la UPV. 38
Tabla 15. Presupuesto total del proyecto sin ser alumno de la UPV. 38

Capítulo 1. Introducción

Un gran número de víctimas pierde su vida a causa de la inhalación de gases tóxicos en incidentes relacionados con el fuego cada año. Pese a que España sea uno de los países que presente una menor tasa de fallecimiento por incendios, 175 es el número de personas que fallecieron en el año 2016 por esta causa. Si bien es cierto que se produjeron menos incendios que en 2015, lamentablemente la cifra de víctimas mortales aumentó un 22%, contribuyendo con la tendencia ascendente de los últimos años [1].

Los bomberos juegan un papel crucial en la sociedad, velan y se exponen a numerosos riesgos durante el desempeño de su trabajo. Uno de los principales obstáculos que tienen a la hora de extinguir incendios es la escasa visibilidad debida al humo, sobre todo en sitios cerrados. Es por ello que dentro de su equipamiento es posible encontrar cámaras térmicas que, a pesar del humo, mediante el uso de radiación infrarroja, son capaces de formar imágenes visibles por el ojo humano. Sin embargo, estas cámaras suelen ser de mano y voluminosas. Esta fue la razón por la que se inició este proyecto, con el fin de diseñar y desarrollar un sistema manos libres completo basado en realidad aumentada capaz de dotar al cuerpo de bomberos de visión térmica, comunicación entre compañeros y otro tipo de información valiosa como el ritmo cardiaco, el nivel de oxígeno disponible en las botellas o la presencia de gases peligrosos en el recinto. El sistema se controlará mediante gestos con los dedos de la mano, gracias a la utilización de sensores.

Se debe destacar que en 2010 se presentó un proyecto[1] muy similar por parte de Joseph Juhnke, Timothy Mills, Dan Delaney y Tim Chapel, en el que se planteaba crear un *Head-Up Display* integrado en la propia máscara de oxígeno donde se superpondría información relevante para el cuerpo de bomberos. No obstante, este concepto no se llegó a implementar de manera formal.

Figura 1. Prototipo de interfaz creada por Joseph Juhnke.

Por último, cabe señalar que este trabajo solo es una parte de un amplio proyecto en proceso de desarrollo, en el que más alumnos de la escuela están trabajando de manera paralela con el fin de obtener un futuro producto funcional, competitivo y de bajo coste.

[1] ABC7, The Future of Firefighting: https://youtu.be/EmCWh4gFWIA [Último acceso: abril de 2018]
CNN, The Future of Firefighting: https://youtu.be/yHIynl9W3GM [Último acceso: abril de 2018]

Capítulo 2. Objetivos

Para abordar con éxito un proyecto de esta índole será necesario dividir y paralelizar el trabajo. Por esta razón el presente documento se centrará exclusivamente en la parte de control de la interfaz.

Es cierto que existen numerosas maneras de navegar por una interfaz, mediante el uso de la voz, botones, movimiento ocular, etc. No obstante, una de las formas más naturales y que mejor se adapta a la situación, es mediante la realización de gestos con los dedos de las manos. Por ello, se emplearán sensores que sean capaces de proporcionar información relativa al movimiento de los dedos. Estas señales serán capturadas y etiquetadas por una interfaz gráfica para generar un set de entrenamiento, el cual será utilizado para reconocer los gestos mediante el uso de redes neuronales.

Por todo ello, los objetivos principales de este trabajo se resumen en los siguientes:

1. Gestión de varios sensores para leer datos de aceleración y de los giróscopos.
2. Diseño y desarrollo de una interfaz gráfica de usuario capaz de capturar simultáneamente señales de los sensores y el vídeo de los gestos, para poder generar un conjunto de muestras de entrenamiento de manera sencilla y eficiente.

Además, se realizará una prueba entrenando una red neuronal simple con los datos recogidos, para así comprobar si el sistema responde según lo esperado.

Capítulo 3. Metodología

3.1 Gestión del proyecto

La gestión del proyecto se podría dividir en cinco fases:

1. Planificación: en esta etapa se analiza el problema para a partir de ahí poder establecer unos objetivos. Después, se definen las tareas a realizar y el tiempo estimado.
2. Programación: se reparten las tareas a lo largo del tiempo, creando así un calendario previo a la ejecución del proyecto.
3. Seguimiento y control: para el seguimiento del proyecto se ha utilizado un documento llamado "Informe de progreso". En él se van reflejando las líneas generales del trabajo realizado cada quince días, ofreciendo de esta manera una visión de la evolución del mismo. Este documento es compartido con el tutor vía Dropbox.
4. Análisis y evolución: esta última etapa se basa en hacer una evaluación posterior a la ejecución del proyecto. Esto será útil para obtener conclusiones e ideas para futuras líneas de trabajo.

3.2 Distribución en tareas

Durante la etapa de planificación el trabajo se dividió en las siguientes tareas:

- Tarea 1. Investigación sobre el microcontrolador ESP8266 y los distintos módulos que lo integran.
- Tarea 2. Investigación sobre los sensores MPU-6050.
- Tarea 3. Investigación sobre el protocolo de comunicación a utilizar para la comunicación entre el ESP8266 y el MPU-6050.
- Tarea 4. Puesta a punto del entorno de trabajo Arduino IDE[2].
- Tarea 5. Lectura de un sensor.
- Tarea 6. Análisis de los valores capturados y ajustes.
- Tarea 7. Lectura de múltiples sensores.
- Tarea 8. Diseño y desarrollo de la interfaz gráfica de usuario en MATLAB[3].
- Tarea 9. Recogida de muestras de entrenamiento.
- Tarea 10. Investigación redes neuronales artificiales.
- Tarea 11. Diseño y entrenamiento de la red neuronal en MATLAB.
- Tarea 12. Prueba del comportamiento global del sistema con la red neuronal en MATLAB.

3.3 Diagrama temporal

En la siguiente figura se utiliza un diagrama de Gantt para mostrar la dedicación prevista para cada tarea a lo largo del tiempo.

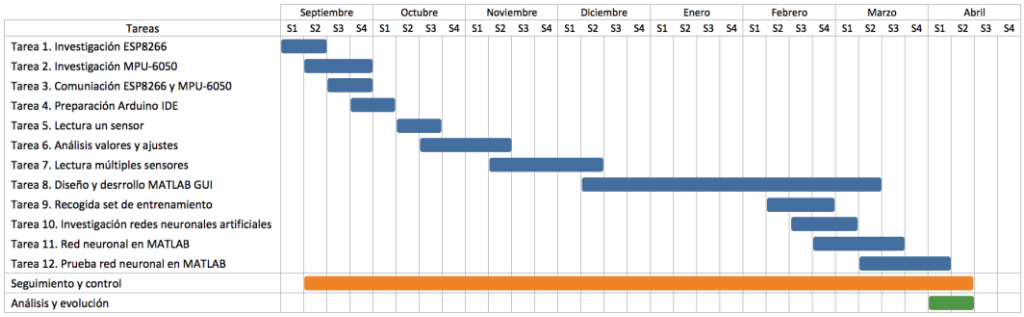


Figura 2. Planificación temporal mediante diagrama de Gantt.

[2] Arduino IDE: https://www.arduino.cc/en/Main/Software [Último acceso: abril de 2018]
[3] MathWorks, MATLAB: https://es.mathworks.com/products/matlab.html [Último acceso: abril de 2018]

Capítulo 4. Desarrollo y resultados

4.1 Introducción

Este capítulo se ha estructurado de manera que permita al lector seguir el proceso llevado a cabo para el desarrollo del proyecto. En primer lugar, en el punto 4.2 se detallará todo el proceso relacionado con la adquisición de datos proporcionados por los sensores. Seguidamente, en el punto 4.3 se expondrá el diseño y desarrollo de la interfaz de usuario para la creación de sets de entrenamiento. Por último, el punto 4.4 contendrá todo lo relativo al entrenamiento y prueba de una red neuronal simple para el reconocimiento de los gestos. Pero antes, se hará una breve introducción de las dos principales plataformas sobre las que se ha trabajado.

4.1.1 Arduino

Arduino es una plataforma electrónica de código abierto cuyo principio fundamental se basa en conseguir un uso sencillo tanto del *software* como del *hardware*. Para alcanzar esto se utiliza el lenguaje de programación Arduino, basado en Wiring[4]; y el *software* Arduino IDE, basado en Processing[5]. Arduino es utilizado en miles de proyectos debido al bajo coste de sus placas, *software* multiplataforma, entorno de programación claro y sencillo, pero principalmente porque su *software* es de código abierto y extensible mediante el uso de librerías en C++[6], abriendo de esta manera un gran abanico de posibilidades [2].

En este proyecto se utilizará Arduino IDE 1.8.3 para compilar y transferir el código fuente a un módulo que integra el microcontrolador ESP8266. Para su configuración, será necesario ir al gestor de tarjetas e instalarla desde el repositorio GitHub de la compañía Espressif Systems[7] y ajustar algunos parámetros dentro del entorno de desarrollo, ya que no es una tarjeta con soporte oficial por parte de Arduino.

4.1.2 MATLAB

MATLAB es una herramienta de *software* matemático que combina un entorno de desarrollo perfeccionado para el análisis iterativo y los procesos de diseño, con un lenguaje de programación propio orientado a objetos que permite realizar operaciones de vectores y matrices directamente [3]. Además, destaca por la fácil implementación de algoritmos, funciones, creación de interfaces de usuario, por la gran cantidad de *Toolboxes* desarrolladas profesionalmente, probadas rigurosamente y totalmente documentadas. Siendo estos los principales motivos que han impulsado el uso de esta herramienta frente a otras. La versión que se empleará para el desarrollo del trabajo será MATLAB R2017b.

4.2 Lectura de sensores

4.2.1 ESP8266

El ESP8266 es un circuito integrado WiFi capaz de cumplir con las principales demandas de los usuarios, consumo eficiente, diseño compacto y un rendimiento adecuado para la industria de los IoT (*Internet of Things*). Este integra un procesador Tensilica de 32 bits, 17 pines GPIO (*General Purpose Input/Output*), conmutador de antena, RF balun, amplificador de potencia, amplificador de bajo ruido, filtros, gestor de energía y alrededor de 50 kB de SRAM (*Static Random Access Memory*). Hay que tener en cuenta que el circuito integrado no incluye memoria ROM (*Read-Only Memory*), por lo que los programas se tendrán que almacenar en una memoria *Flash* externa [4].

Para este proyecto se plantean dos posibilidades a la hora de trabajar con el ESP8266, trabajar con un módulo más un programador o trabajar con un kit de desarrollo. Por lo que, en primer

[4] Wiring: http://wiring.org.co/about.html [Último acceso: abril de 2018]
[5] Processing: https://processing.org/ [Último acceso: abril de 2018]
[6] C++: http://www.cplusplus.com/info/ [Último acceso: abril de 2018]
[7] GitHub de Espressif Systems: https://github.com/espressif [Último acceso: abril de 2018]

lugar, será necesario conocer qué es un módulo y qué es un kit de desarrollo; para más tarde realizar un análisis de los distintos módulos y placas de desarrollo existentes, y así poder seleccionar con qué se trabajará.

Un módulo ESP es una placa que incluye los componentes esenciales para que el microcontrolador ESP8266 sea totalmente funcional. Sin embargo, para ser programado es necesario utilizar un programador, convirtiendo esta opción menos atractiva para la etapa de desarrollo de este proyecto, debido a que se necesitaría estar realizando pruebas con estos dos elementos (véase Figura 3).

Figura 3. Módulo ESP-01 y programador.

Por otro lado, un kit de desarrollo es una tarjeta compuesta por un módulo y un conjunto de componentes que, entre otras funcionalidades, permiten prescindir del uso de un programador externo. Este motivo, junto con otros que se verán a continuación, convertirán el uso de una tarjeta de desarrollo en la mejor opción para la realización del trabajo.

Como se puede observar en la Figura 4, el kit de desarrollo podría dividirse en tres bloques siguiendo una jerarquía lineal de tres niveles, donde el kit de desarrollo es padre del módulo y el módulo es padre del microcontrolador. Tal y como se ha dicho anteriormente, el ESP8266 no integra una memoria ROM, por lo que para almacenar los programas será necesario añadir una memoria *Flash*, siendo esta, junto con la antena, uno de los nuevos componentes que se encuentran dentro del módulo. Por último, dentro del kit de desarrollo se incluyen componentes como el regulador de tensión, el convertidor serie y el puerto Micro-USB, que facilitan la realización de prototipos y la programación.

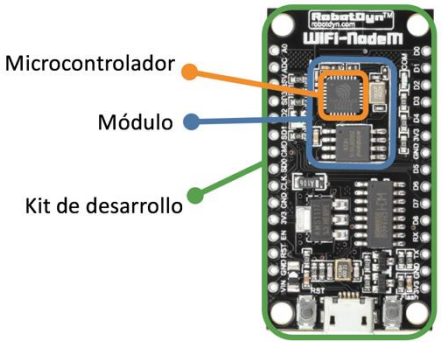


Figura 4. Diagrama de bloques de la tarjeta de desarrollo RobotDyn WiFi NodeM.

Una vez explicados los conceptos de módulo y kit de desarrollo, se procede a realizar un pequeño análisis de algunas de las principales características de los más relevantes del mercado actual.

A pesar de que los módulos ESP necesitan de un programador para ser programados, su reducido tamaño y su bajo coste los convierte en una buena elección para un futuro trabajo de rediseño del sistema, siendo el ESP-12 la mejor opción hasta el momento debido al gran número de GPIO y a que la antena se encuentra incluida en la propia placa (véase Tabla 1). No obstante, para este trabajo se utilizará un kit de desarrollo, pues estos facilitan la programación y ofrecen funcionalidades adicionales.

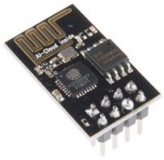
	ESP-01	ESP-05	ESP-12	ESP-201
				
Pins GPIO	2	-	11	11
ADC	-	-	1	1
Antena	PCB	Externa	PCB	PCB/Externa
Tamaño	27,7 mm x 14,5 mm	14,2 mm x 14,2 mm	24 mm x 16 mm	35 mm x 25 mm
Breadboard friendly	Medio	Alto	Bajo	Bajo
Precio	≈ 2,50 €	≈ 2,50 €	≈ 2,50 €	≈ 4 €

Tabla 1. Comparativa de módulos que integran el ESP8266.

Tal y como se puede observar en la Tabla 2, la mayoría de las características de las tarjetas son muy similares, por lo que se propone adquirir tres WiFi-NodeM de la marca RobotDyn para realizar pruebas y un primer diseño, ya que son relativamente económicas; y dos Feather HUZZAH de la marca Adafruit para incorporarlas al sistema una vez desarrollado, ya que su coste es más elevado y pueden ser alimentadas mediante el uso de una batería.

	NodeMCU v0.9	WiFi-NodeM	Feather HUZZAH ESP8266 WiFi	WiFi D1 MINI
				
Marca	NodemMCU	RobotDyn	Adafruit	RobotDyn
Pins GPIO	11	11	11	11
ADC	1	1	1	1
Antena	PCB	PCB	PCB	PCB
Tamaño	27,7 mm x 14,5 mm	8 cm x 4 cm	5,1cm x 2,3 cm	10 cm x 6 cm
Alimentación	USB	USB	USB/Batería LiPo	USB
Breadboard friendly	Bajo	Alto	Alto	Alto
Precio	≈ 5 €	≈ 3 €	≈ 14 €	≈ 4 €

Tabla 2. Comparativa de tarjetas de desarrollo que integran el ESP8266.

4.2.2 Posibles soluciones para el sistema de control

El control de sistemas mediante el uso de gestos es cada vez más empleado en dispositivos de uso habitual, como son los *smartphones* y los ordenadores portátiles (por medio del *trackpad*). Esto se debe, principalmente, a que son capaces aportar funcionalidades adicionales a la hora de interactuar con la interfaz de una manera natural e intuitiva. Por otro lado, todos los avances en el desarrollo de dispositivos inteligentes compatibles con VR (*Virtual Reality*) y AR (*Augmented Reality*), han provocado que los dispositivos de control tiendan a ser *wearables* [5] (véase Figura 5).

Figura 5. CaptoGlove[8]. *Wereable* controlador para videojuegos y dispositivos inteligentes.

Por todo ello, se pretende encontrar una solución basada en el reconocimiento de gestos de los dedos de la mano que permita la navegación a través del sistema de realidad aumentada.

A continuación, se plantearán distintas posibles soluciones que permitirían aportar información relativa al movimiento de los dedos, desarrollando la opción más viable.

Una posible solución sería utilizar galgas extensiométricas, sensores resistivos cuya resistencia varía si se producen cambios en su longitud, sección o resistividad [6]. Estos sensores son utilizados en campos como la arquitectura civil, la medicina y la robótica; para medir esfuerzos mecánicos que provocan deformaciones. Su alto uso es debido a que la mayoría de ellos tienen un precio asequible y reducido tamaño. Sin embargo, uno de los principales problemas que presentan es que no tienen un comportamiento lineal. Además, actualmente existen pocos módulos en el mercado que faciliten el desarrollo de esta solución.

Otra alternativa podría ser utilizar guantes con marcadores, de manera que, por medio del uso de una o varias cámaras, se consiguiera realizar un seguimiento y posicionamiento de los puntos de referencia (véase Figura 6). No obstante, esta opción se descarta debido a los problemas de visibilidad ya mencionados en el entorno de trabajo de los bomberos.

Figura 6. Guante con marcadores para el seguimiento y posicionamiento de los mismos [7].

Como última opción se propone la utilización de IMU (*Inertial Measurement Unit*), dispositivos electrónicos que permiten estimar la orientación de un cuerpo a partir de las fuerzas inerciales que el cuerpo experimenta. El principio de funcionamiento de estos está basado en la medición de la aceleración y la velocidad angular ejercida sobre pequeñas masas localizadas en su interior. A diferencia de las galgas extensiométricas, existen numerosos módulos que incluyen IMU y que, de esta manera, permiten una implementación más ágil del sistema. Además, la mayoría de estos módulos suelen ser de bajo coste, reducido tamaño e incorporan protocolos de comunicación como el I^2C o el SPI, ambos compatibles con el microcontrolador que se utilizará.

Teniendo en cuenta todo lo comentado, la solución basada en el uso de IMU será la utilizada para obtener información relativa al movimiento de los dedos.

[8] Vídeo Captoglove Kickstarter: https://youtu.be/PcKzD8ohlGc [Último acceso: abril de 2018]

4.2.3 MPU-6050

Para el desarrollo del sistema de captura de movimiento de los dedos se utilizará la IMU MPU-6050 de la compañía InvenSense. Este forma parte de la familia del MPU-60XX, los primeros dispositivos *MotionTracking* de 6 ejes en el mundo que combinan un giróscopo de tres ejes, un acelerómetro de tres ejes y un DMP (*Digital Motion Processor*) en un circuito integrado de 4x4x0,9mm [8].

Este tipo de circuitos integrados son incluidos en módulos como el GY-521 (véase Figura 7). Una placa de bajo coste que, además, incorpora la electrónica necesaria para conectarlo de forma sencilla al kit de desarrollo que se empleará.

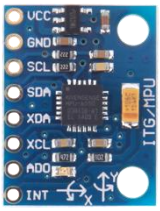


Figura 7. Módulo GY-521.

A continuación, en base a la Figura 7 se describirán brevemente cada uno de los pines del módulo:

- VCC: tensión de alimentación. Como el módulo incorpora un regulador de tensión la entrada puede ser de 3,3 V o 5 V.
- GND: tensión de referencia.
- SCL: su nombre hace referencia a *Serial Clock Line*. Se trata de la línea de la señal de reloj del protocolo de comunicación I^2C.
- SDA: su nombre hace referencia a *Serial Data Line*. Se trata de la línea de datos del protocolo de comunicación I^2C.
- XDA: su nombre hace referencia a *Auxiliary Data*. Se utiliza como línea maestra de datos del protocolo I^2C para conectar el modulo a un sensor externo.
- XCL: su nombre hace referencia a *Auxiliary Clock*. Se utiliza como línea maestra de señal de reloj del protocolo I^2C para conectar el modulo a un sensor externo.
- AD0: se utiliza para cambiar la dirección I^2C del MPU-6050 (si el pin se encuentra a nivel bajo la dirección será 0x68, mientras que si está a nivel alto será 0x69).
- INT: salida digital de interrupción.

4.2.4 Comunicación mediante bus I^2C

Una vez definidos los pines del módulo y antes de proceder al diseño del sistema se realizará una breve introducción al protocolo de comunicación I^2C.

El protocolo I^2C está basado en la utilización de un *bus multi-master*. Esto significa que la comunicación puede ser entre varios *masters* y varios *slaves*. Los *masters* son los encargados de iniciar la transmisión de datos, generar las señales de reloj y terminar las transmisiones de datos. Mientras que los *slaves* son aquellos dispositivos dirigidos por los *masters*.

Algunas de las características principales del *bus* I^2C son [9]:

- Solo son necesarias dos líneas de *bus*: una línea de datos serial (SDA) y una línea para la señal de reloj (SCL).
- Es un *bus multi-master* real, incluye detección de colisiones y arbitraje para prevenir que dos o *masters* inicien una transmisión de datos simultáneamente.
- Comunicación serial, orientada a 8 bits, con una tasa de transferencia de datos bidireccional de hasta 100 kbit/s en modo estándar o hasta 400 kbit/s en modo rápido.
- El número de dispositivos que se pueden conectar al *bus* serie solo es limitado por la capacitancia máxima del *bus*, que es 400 pF.

A continuación, con ayuda del cronograma simplificado de la Figura 8, se resumirá el protocolo de comunicación.

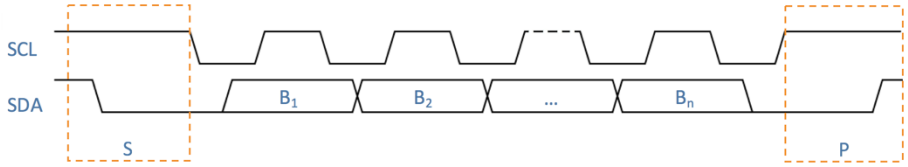


Figura 8. Cronograma simplificado comunicación I²C.

Cuando el *bus* está libre ambas líneas se encuentran a nivel alto. La comunicación es iniciada por el *master* generando una condición de inicio (S) seguido por la dirección del *slave* (B_1). Esta dirección de 7 bits viene acompañada de otro bit que indica si la operación es de escritura o de lectura. Si este bit es 0, el dispositivo *master* realizará un proceso de escritura hacia el *slave* (B_2). De lo contrario, el siguiente *byte* será leído del *slave*. Una vez han sido escritos o leídos todos los *bytes* (B_n), con su respectivo ACK (*Acknowledgement*), el *master* genera una condición de parada (P), indicando al resto de dispositivos que la comunicación ha terminado y que el *bus* puede ser utilizado por otro dispositivo [9] [10].

Este tipo de comunicación se contempla en la librería *Wire* de Arduino. Sin embargo, se utilizará la librería de código abierto I²Cdevlib[9] desarrollada por Jeff Rowberg[10]. Esta librería está formada por una amplia colección de clases uniformes y bien documentadas que proporcionan interfaces simples e intuitivas para una colección cada vez mayor de dispositivos I²C, entre los cuales se encuentra el MPU-6050. Asimismo, dentro de ella se incluyen códigos de ejemplo que son de gran utilidad a la hora de programar.

4.2.5 Propuesta y desarrollo del sistema de lectura

Tras esta introducción teórica y después de haber realizado un análisis de las distintas soluciones, se propone desarrollar un sistema de lectura formado por un kit de desarrollo RobotDyn WiFi-NodeM/Adafruit Feather HUZZAH ESP8266 WiFi y cuatro módulos GY-521 (integran el circuito integrado MPU-6050) (véase Figura 9). Todos estos elementos serán incorporados en un guante, pero antes se comprobará el funcionamiento del sistema en una placa de pruebas.

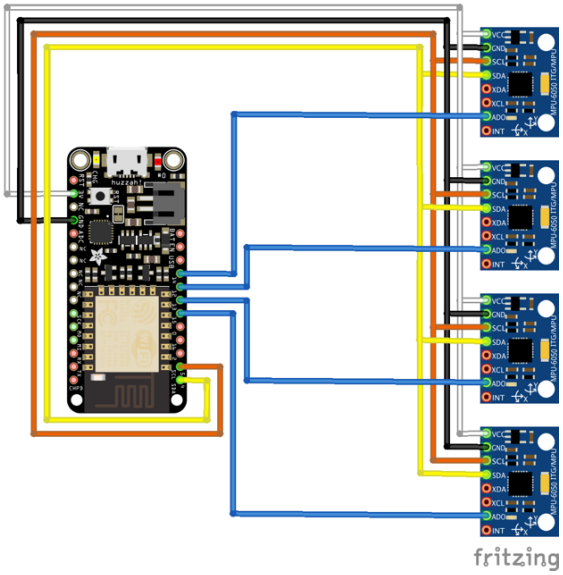


Figura 9. Conexiones sistema propuesto.

[9] I²Cdevlib: https://www.i2cdevlib.com/ [Último acceso: abril de 2018]
[10] GitHub librería I²Cdevlib de Jeff Rowberg: https://github.com/jrowberg/i2cdevlib [Último acceso: abril de 2018]

El kit de desarrollo será el encargado de leer los datos adquiridos por las IMU mediante el uso del protocolo de comunicación I^2C, y transferirlos a un ordenador a través del puerto serie por USB. Como se verá en capítulos siguientes, estos datos serán capturados y etiquetados para crear un set de entrenamiento y así poder utilizarlo en el proceso de aprendizaje de una red neuronal que sea capaz de reconocer gestos.

A continuación, se realizará un recorrido sobre los aspectos más relevantes tenidos en cuenta para la implementación final del sistema de lectura.

El primer paso es interpretar y analizar los datos *raw* que proporciona el módulo GY-521. Así pues, se montará un sistema simple que permita la lectura de una única IMU. Como se ha comentado anteriormente, la librería I^2Cdevlib proporciona algunos códigos de ejemplo para el MPU-6050. Por lo que, adaptando un poco el código de ejemplo, se consigue una correcta lectura de datos.

En la Tabla 3, se pueden observar cuatro ejemplos de los valores *raw* devueltos por el MPU-6050 en posición de reposo sobre una mesa.

n	Acc_x	Acc_y	Acc_z	G_x	G_y	G_z
1	-388	124	16604	-195	448	-121
2	-260	40	16384	-183	426	-105
3	-48	-28	16400	-225	370	-119
4	-128	-36	16268	-241	343	-141

Tabla 3. Ejemplo de lectura valores *raw* en posición de reposo.

Lo primero que uno se pregunta es qué significan estos valores y cómo se podría llegar a una unidad fácilmente interpretable. Para ello, es necesario tener en cuenta que se trata de valores digitales adimensionales y que conocer el valor de la sensibilidad a la que se está trabajando es un requisito primordial. La sensibilidad del acelerómetro y del giroscopio es uno de los parámetros ajustables del MPU-6050. En la Tabla 4 y la Tabla 5, se recogen los posibles valores de sensibilidad del acelerómetro y el giróscopo respectivamente, extraídos del *datasheet* [8].

Rango de escala	Factor de escala (sensibilidad)
±2 *g*	16.384 LSB/*g*
±4 *g*	8.192 LSB/*g*
±8 *g*	4.096 LSB/*g*
±16 *g*	2.048 LSB/*g*

Tabla 4. Sensibilidad acelerómetro.

Rango de escala	Factor de escala (sensibilidad)
±250 º/s	131 LSB/(º/s)
±500 º/s	65,5 LSB/(º/s)
±1000 º/s	32,8 LSB/(º/s)
±2000 º/s	16,4 LSB/(º/s)

Tabla 5. Sensibilidad giróscopo.

Dividiendo el valor proporcionado por las IMU entre el factor de escala correspondiente, es posible conseguir un valor de aceleración en fuerza *g* (podría por lo tanto convertirse en $m/s^{2)}$; y un valor de velocidad angular en grados por segundo. Teniendo en cuenta que la sensibilidad del acelerómetro y el giróscopo por defecto en la librería I²Cdevlib es de 16.384 LSB/*g* y 131 LSB/(º/s) respectivamente, los valores obtenidos son los reflejados en la Tabla 6.

n	Acc_x (*g*)	Acc_y (*g*)	Acc_z (*g*)	G_x (º/s)	G_y (º/s)	G_z (º/s)
1	-0,02368	0,00757	1,01342	-1,48855	3,41985	-0,92366
2	-0,01587	0,00244	1	-1,39694	3,25190	-0,80153
3	-0,00293	-0,00170	1,00098	-1,71756	2,82443	-0,90840
4	-0,00781	-0,00219	0,99291	-1,83969	2,61832	-1,07633

Tabla 6. Valores obtenidos aplicando factor de escala.

Como se puede observar, a pesar de que los valores de la Tabla 6 proporcionen un valor físico que a priori ayuda a entender el significado de los valores de la Tabla 3, se considera que la utilización de estos llevaría a errores de redondeo. Por esto, de ahora en adelante se seguirá trabajando sin aplicar el factor de escala.

Otro de los aspectos que llaman la atención tras observar los datos de la Tabla 3, es que todos los valores sean tan distintos de cero. Tanto en el caso de la aceleración en x e y, como en la lectura obtenida por el giróscopo para los tres ejes, es debido a que las IMU tienen un error de *offset* causado por distintos factores, como el proceso de ensamblaje mecánico y fluctuaciones en la temperatura. Sin embargo, a pesar de que este problema también existe para la aceleración en el eje z, la gran diferencia numérica en este caso es debido a la fuerza de la gravedad, siendo el valor esperado 16.384 para un rango de escala de ±2 *g*.

Para solucionar el primer problema, será necesario hallar el *offset* de cada uno de los módulos a utilizar. Para ello se utilizará el código "MPU6050_calibration.ino" implementado por Luis Ródenas[11]. Una vez obtenidos los valores en cuestión y ajustados los *offsets*, se obtienen unos resultados que se asemejan más a los esperados (véase Tabla 7).

n	Acc_x	Acc_y	Acc_z	G_x	G_y	G_z
1	4	-4	16392	-3	39	7
2	32	52	16412	14	2	-4
3	-92	20	16392	34	61	4
4	56	20	16368	11	-28	32

Tabla 7. Valores *raw* con corrección de *offset*.

Sabiendo que la fuerza de la gravedad se ve reflejada en los datos obtenidos por el acelerómetro, los valores de la aceleración serán muy distintos en función de la posición en la que el módulo GY-521 se encuentre. Por esta razón, será necesario eliminar de alguna manera el efecto de la gravedad. Para la realización, se utilizarán algunas de las funciones proporcionadas en la librería I²Cdevlib y el uso del DMP, un bloque integrado en el MPU-6050 que reduce la carga de cálculo relativa a los algoritmos de procesamiento que se suele realizar en el procesador anfitrión [8]. Los resultados obtenidos tras el procesamiento para el módulo en reposo, se recogen en la Tabla 8. Tal y como se puede observar, el efecto de la gravedad ha sido eliminado correctamente.

[11] Enlace al hilo donde Luis Ródenas comparte el código: https://www.i2cdevlib.com/forums/topic/96-arduino-sketch-to-automatically-calculate-mpu6050-offsets/ [Último acceso: abril de 2018]

n	Acc_x	Acc_y	Acc_z
1	-4	-5	26
2	17	0	36
3	4	3	23
4	0	-7	19

Tabla 8. Ejemplo de lectura eliminando el efecto de la gravedad mediante DMP.

El siguiente paso es trasladar la lectura de un módulo a cuatro (véase Figura 10). El principal problema de este paso radica en que el MPU-6050 solo dispone de dos direcciones I^2C, la 0x68 y la 0x69. Seguidamente, se explicará la lógica aplicada para solventar el problema.

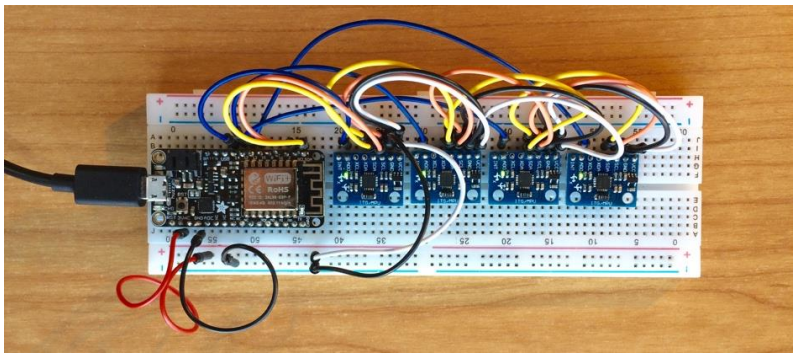


Figura 10. Montaje de cuatro MPU-6050 sobre la placa de pruebas.

Como ya se introdujo en el punto 4.2.3, la dirección puede ser seleccionada mediante el uso del pin AD0. Si esta línea se encuentra a nivel bajo será 0x68 y si está a nivel alto será 0x69. Para que no haya problemas en los procesos de lectura y escritura, es necesario que no exista más de un *slave*, sobre el cual se realiza alguna de estas acciones, con la misma dirección.

Teniendo en cuenta todo esto, se podría utilizar la línea AD0 como si de un *chip select* se tratara, definiendo en el código fuente que los procesos de lectura y de escritura siempre serán sobre la dirección 0x68 y que solo un módulo podrá tener esta dirección mientras que los demás tienen la 0x69. En resumen, se irá seleccionando (poniendo la línea a nivel bajo) el módulo del cual se quiere obtener datos, y desactivando el resto (poniendo la línea a nivel alto) de los que no se quiere obtener datos. El código fuente final se encuentra en el punto 8.1 del capítulo Anexos.

La lectura de las cuatro IMU se hará de manera consecutiva, por lo que, para una mejor recogida e interpretación de los datos, estos siempre seguirán el siguiente patrón:

1. L (indicador inicio de nuevos datos).
2. Número de la IMU.
3. Tiempo (ms).
4. Aceleración en x.
5. Aceleración en y.
6. Aceleración en z.

En la Figura 11 se puede observar cómo los valores enviados por el puerto serie siguen el patrón indicado.

Figura 11. Salida monitor serie durante el proceso de lectura de cuatro módulos.

A continuación, se comprobará que se cumplen los requisitos temporales necesarios.

La librería con la que se está trabajando permite ajustar la frecuencia de muestreo de la IMU desde el archivo de cabecera "MPU6050_6Axis_MotionApps20.h". La frecuencia de muestreo a la que se trabajará es de 50 Hz, por lo que cada módulo refrescará sus datos cada 20 ms. Para que no existan problemas de pérdidas de datos los 4 módulos habrán de ser leídos en menos de dicho tiempo.

Tal y como se puede observar en la Tabla 9, el periodo existente para una misma IMU entre un ciclo (n) y el siguiente (n+1), es de 20 ms; y el tiempo de lectura de un ciclo completo es de 18 ms. Por todo esto, se podría decir que el sistema diseñado cumple los requisitos temporales necesarios.

n	Cabecera	IMU	Tiempo (ms)	Acc_x	Acc_y	Acc_z
1	L	1	70178	-4	-3	66
1	L	2	70187	4	-5	-29
1	L	3	70189	-2	0	-38
1	L	4	70196	5	3	-11
2	L	1	70198	-7	-1	63
2	L	2	70207	7	-2	-26
2	L	3	70209	2	7	-34
2	L	4	70216	2	-2	-15

Tabla 9. Ejemplo de lectura cuatro IMU en reposo.

El último paso antes de integrar todos los elementos dentro del guante, es comprobar el sincronismo de las señales. Para ello, la placa de pruebas será sometida a un desplazamiento sobre uno de los ejes de referencia de las IMU, el eje x; y se representará el resultado mediante el uso de MATLAB. En la Figura 12 se puede comprobar que la máxima diferencia de tiempos es de 21 ms (39469 ms – 39448 ms). Esto es aproximadamente un solo ciclo de la frecuencia de muestreo de las IMU, por lo que se podría decir que las cuatro señales están sincronizadas.

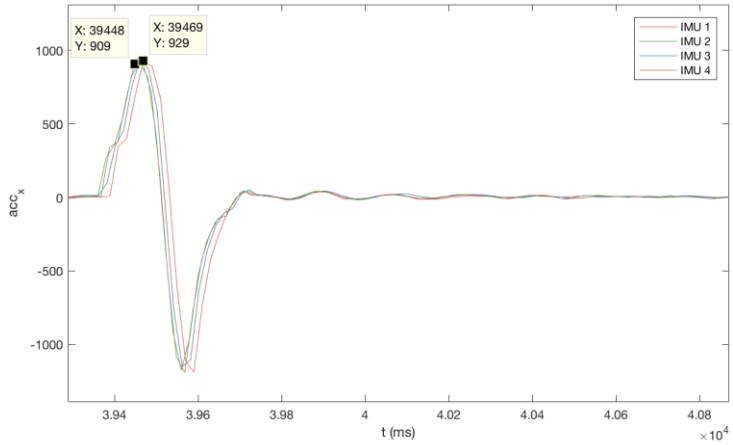


Figura 12. Respuesta al impulso sobre el eje x.

4.2.6 Resultados

Una vez han sido realizadas las pruebas necesarias para comprobar el correcto funcionamiento del sistema de lectura en la placa de pruebas, se integran todos los elementos en el guante y se comprueba si las señales son leídas correctamente.

El prototipo consta de cuatro módulos GY-521 situados en las puntas de los dedos índice, corazón, anular y meñique; mediante el uso de velcro. Los cables son reconducidos por la parte superior de la mano utilizando bridas e hilo hasta la muñeca, donde cada una de las conexiones acaba en un conector hembra. De esta manera, el guante queda independiente del kit de desarrollo, permitiendo intercambiarlo sin dificultades. Desde cada uno de los pines del kit sale un cable que acaba en conector macho y que se conecta a su respectivo conector hembra. Tanto el microcontrolador como las conexiones quedan ocultas dentro de un bolsillo muñequera, que ha sido cosido al guante. El prototipo incluye un led que indica el estado actual del sistema (véase Figura 13).

Figura 13. Prototipo guante de control.

Debido a la posición de los MPU-6050 en el guante, la información relativa a los gestos realizados con los dedos se encuentra mayoritariamente en el eje x. Esto se puede contemplar en la Figura 14, en la cual se presenta el efecto que tiene un gesto de un toque simple (dedo índice con dedo pulgar), mediante tres gráficas que representan la aceleración existente en los tres ejes, para cada uno de los dedos.

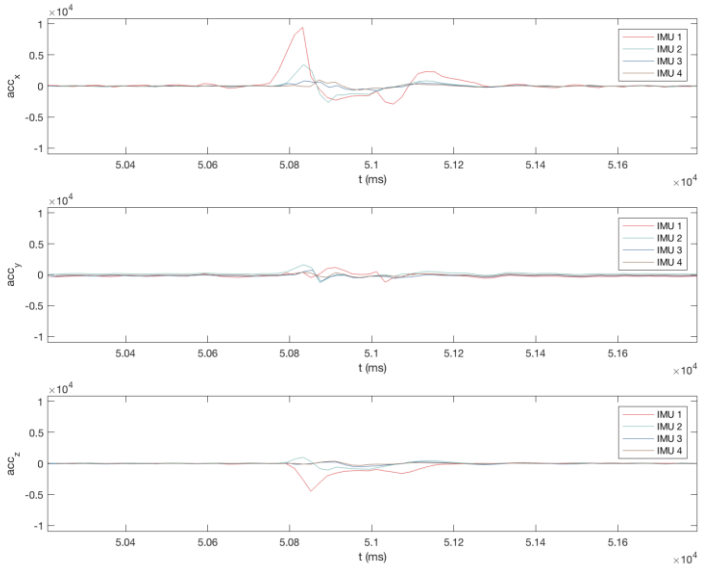


Figura 14. Gesto toque simple dedo índice con pulgar.

Por último, como muestra del proceso de verificación, se exponen y comentan los resultados obtenidos de la lectura de la aceleración en el eje x, para cada uno de los gestos definidos.

Como se puede apreciar en la Figura 15, los gestos simples y dobles se diferencian esencialmente por la presencia de uno o dos picos principales positivos, en la señal correspondiente al dedo movido. Sin embargo, dependiendo del sujeto y el gesto, existe la posibilidad de que estos picos no solo aparezcan sobre la señal correspondiente al dedo con el cual se realiza el toque, sino también en otras señales. Esto se cumple para los casos "Toque simple – Dedo corazón con pulgar", "Toque doble – Dedo corazón con pulgar", "Toque simple – Dedo meñique con pulgar" y "Toque doble – Dedo meñique con pulgar". A pesar de que el de ausencia de toque, se tendrá en cuenta para el entrenamiento de la red, no ha sido recogido en la figura debido a su amplia variabilidad.

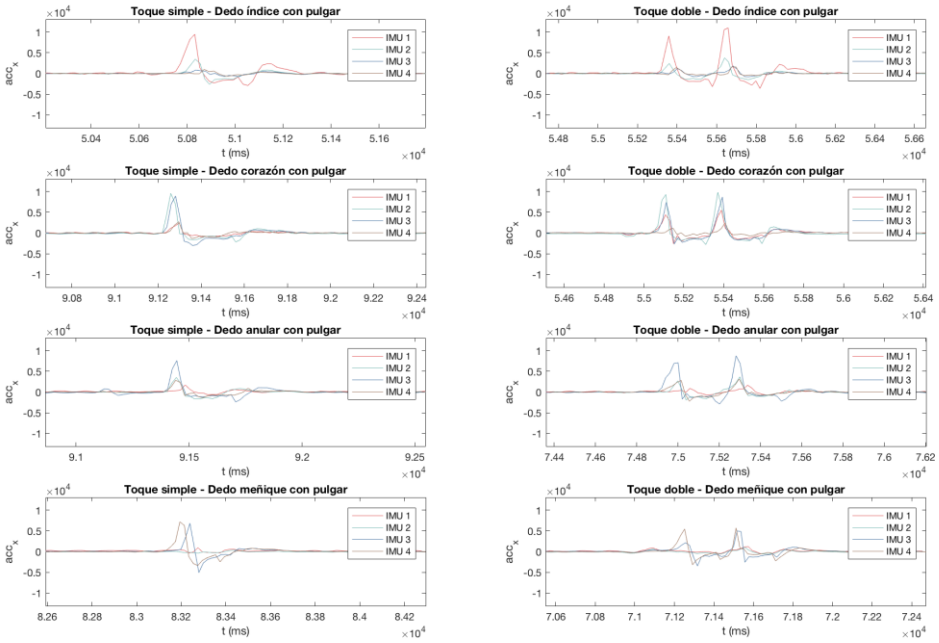


Figura 15. Efecto provocado por cada uno de los gestos sobre el eje x.

4.3 Interfaz de usuario para la creación de sets de entrenamiento

Las interfaces de usuario, también conocidas como interfaces gráficas de usuario o GUI (*Graphical User Interface*), permiten un control sencillo de las aplicaciones *software*, eludiendo la necesidad de aprender un lenguaje de programación y utilizar comandos para manejar una aplicación.

Como se introdujo al principio del presente documento, uno de los objetivos de este trabajo es diseñar y desarrollar una interfaz gráfica de usuario que permita capturar simultáneamente las señales de los sensores y el vídeo de los gestos para, más tarde, poder generar un conjunto de muestras de entrenamiento de manera sencilla y eficiente. Por consiguiente, se utilizará GUIDE, el entorno de desarrollo de GUI de MATLAB. Esta herramienta permite diseñar de manera gráfica el *front-end* de la aplicación y más tarde autogenerar el código de MATLAB necesario para construir la interfaz, el cual se ha de modificar para definir el comportamiento de la aplicación.

Para cumplir dicho objetivo, es necesario analizar qué pasos habría que seguir para transformar las muestras que llegan del sistema de lectura en un set de entrenamiento.

En primer lugar, habría que capturar los valores leídos por el sistema de lectura y registrar, mediante el uso de una cámara, a qué tipo de gesto corresponden las medidas. A continuación, la señal de vídeo tendría que ser sincronizada con los datos recogidos. Por último, se procedería al etiquetado y exportación de las muestras.

Teniendo en cuenta el proceso anterior, se propone diseñar y desarrollar un conjunto de tres aplicaciones para realizar estas tres funciones (véase Figura 16).

Figura 16. Proceso general y función de cada uno de las aplicaciones.

Antes de entrar en detalle en cada uno de estos bloques, se introducirá un elemento que ha sido utilizado para agilizar el proceso de sincronización y etiquetado.

4.3.1 Controlador Contour ShuttleXpress

Un controlador es un periférico que permite realizar acciones sobre un *software*. La principal ventaja que presenta, es que la mayoría de ellos son configurables, es posible personalizar las acciones de cada uno de los botones, diales, *joysticks*…; aumentando la productividad y la eficiencia del usuario. Por esta razón se ha optado por el uso del controlador multimedia Contour ShuttleXpress (véase Figura 17).

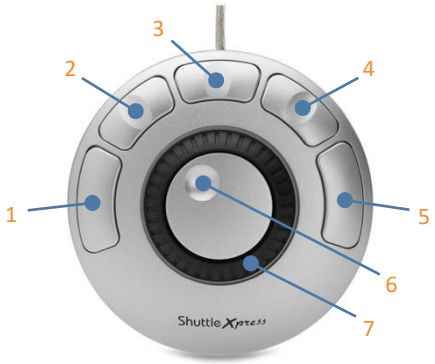


Figura 17. Controlador multimedia Contour ShuttleXpress con números de referencia.

El controlador cuenta con siete elementos de control totalmente personalizables mediante el *software* disponible en la página web del fabricante. En base a los números de referencia de la Figura 17, se indican las acciones que realiza cada elemento y su efecto dentro de la interfaz de usuario:

1. Escribe la letra "h" treinta veces por segundo (retrocede treinta fotogramas por segundo).
2. Escribe la letra "n" (realiza *zoom in* sobre el eje y).
3. Escribe la letra "j" (tecla de selección)
4. Escribe la letra "m" (realiza *zoom out* sobre el eje y).
5. Escribe la letra "k" treinta veces por segundo (avanza treinta fotogramas por segundo)
6. Girando hacia la izquierda escribe la letra "h" (retrocede un fotograma). Girando hacia la derecha escribe la letra "k" (avanza un fotograma).
7. Sin asignar.

El uso de este dispositivo agilizará la navegación a través de los vídeos capturados, así como el proceso de sincronización y etiquetado.

4.3.2 Aplicación Captura

Esta aplicación se encarga de capturar las muestras procedentes del sistema de lectura y mediante el uso de una cámara, dejar constancia de a qué gesto corresponden las señales. El hecho de tener una grabación de los gestos será de gran ayuda a la hora de interpretar las señales y etiquetar los gestos; contribuyendo de esta manera a la creación de un set de entrenamiento más fiable.

Una vez diseñada la interfaz gráfica desde el editor de diseño GUIDE, MATLAB autogenera un código para construir la interfaz. Cabe decir que este código solo es un esqueleto, en él todavía no se encuentra definido el comportamiento de la aplicación, por lo que será necesario programar cada una de las acciones deseadas.

A continuación, con ayuda del diagrama de flujo de la Figura 18, se explicará el funcionamiento simplificado de la aplicación, y se entrará en detalle en aquellos puntos que se consideren de mayor interés.

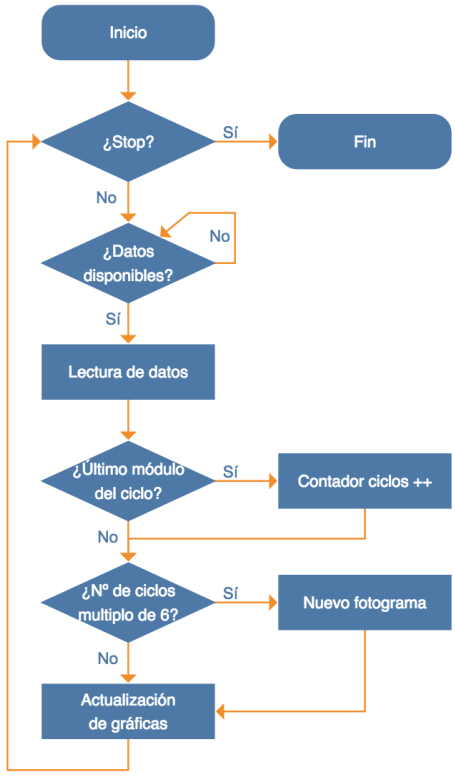


Figura 18. Flujograma del funcionamiento básico aplicación "Captura".

Tras haber iniciado la comunicación serie con el ESP8266, se valida si existen datos disponibles para leer en el *buffer* de entrada. Si la respuesta es negativa, se sigue esperando hasta que el número de bytes disponibles sea distinto de cero; si por lo contrario es afirmativa, se leerán los datos y se comprobará si estos corresponden al último módulo (el IMU 4, el del dedo meñique). Si es el caso, se incrementará el contador de ciclos de lectura (un ciclo de lectura corresponde con la lectura de datos de los cuatro módulos). Este contador, junto con la condición reflejada en el flujograma anterior (número de ciclos múltiplo de seis), se utiliza para controlar la frecuencia de muestreo de la cámara. Controlándola de este modo se consigue asociar de manera uniforme la relación existente entre fotograma y muestras. Antes de continuar, se comprobará que el método empleado funciona según lo esperado.

Como se dijo en el punto 4.2.5, el tiempo de lectura de un ciclo completo es de 18 ms. Teniendo en cuenta que la cámara realiza una captura cada seis ciclos, se podría deducir que realiza una captura cada 108 ms (sin contemplar los posibles retardos que la interfaz pueda provocar), por lo que la frecuencia de muestreo teórica será de 9 fotogramas por segundo. Para comprobar que esto se cumple se emplearán las funciones “tic”, inicia un cronometro al ejecutarse [11]; y “toc”, muestra por el monitor serie el tiempo transcurrido desde el inicio del cronómetro [12], de MATLAB, para conocer la frecuencia de muestreo de la cámara.

En la Figura 19 se pude observar que el tiempo medio entre una captura y la siguiente es de 120 ms, teniendo en cuenta los retardos ocasionados por la interfaz, dando una frecuencia de muestreo de 8 fotogramas por segundo.

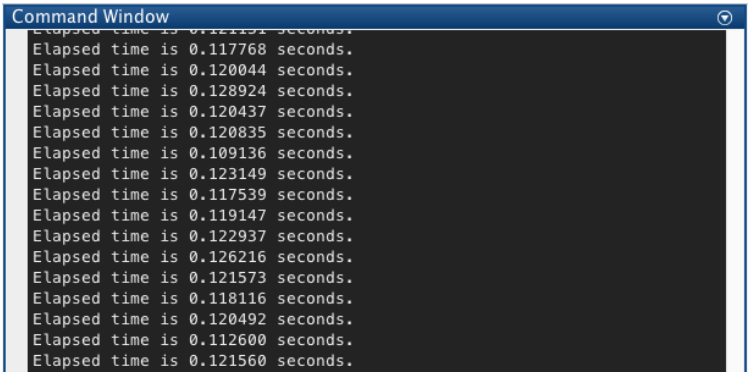


Figura 19. Diferencia de tiempos entre la captura de un fotograma y la siguiente.

Tras la acción anterior se representan los nuevos valores de aceleración capturados y se vuelve a empezar. A pesar de que en la Figura 18 no se especifique, la actualización de las gráficas no es un proceso que se realice cada vez que se obtienen valores nuevos, sino cada veinticinco ciclos. Esto reducirá el alto coste computacional que supondría actualizarlas cada vez que se capturan datos.

Una vez explicado el funcionamiento básico de la aplicación, se presentan y comentan los principales elementos de la interfaz (véase Figura 20).

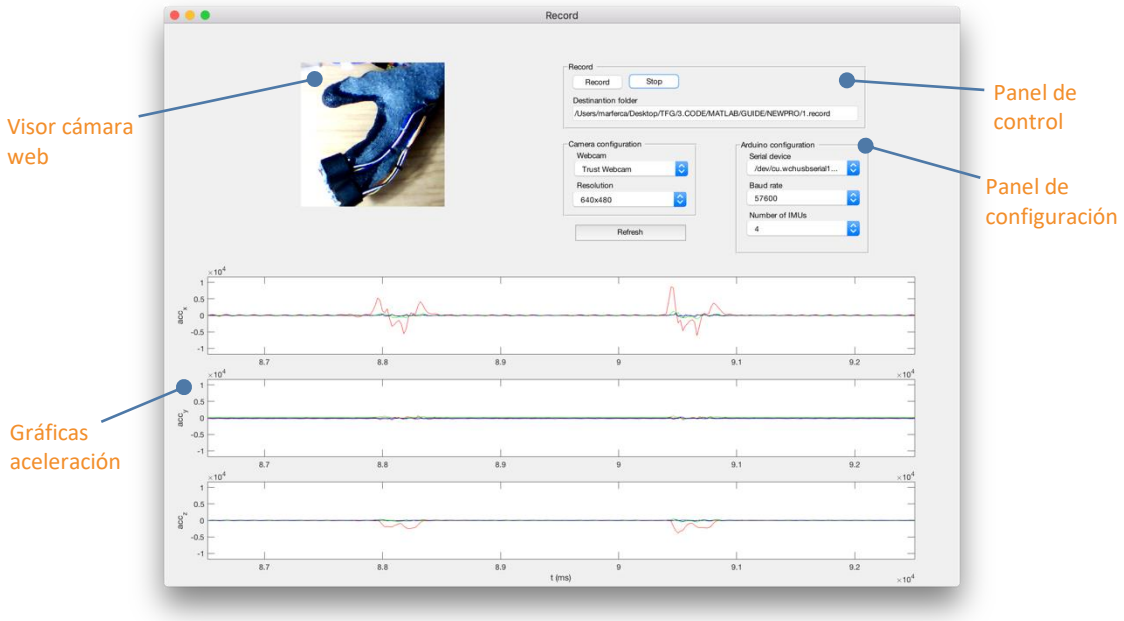


Figura 20. Interfaz gráfica aplicación "Captura".

Como se puede ver en la Figura 20, la interfaz está compuesta principalmente por cuatro elementos. El visor de la cámara web proporciona una previsualización de lo que la cámara está capturando. Las tres gráficas recogen la aceleración a lo largo del tiempo de cada uno de los módulos para los ejes x, y, z respectivamente. El panel de control ofrece al usuario un manejo sencillo sobre las acciones básicas (grabar y parar) además de indicar la ruta donde serán almacenados los datos. El panel de configuración permite, de manera accesible, configurar todos los parámetros relativos a la captura tanto de los datos (seleccionar el dispositivo serie, el *baud rate*, el número de IMU utilizados), como del vídeo (escoger la cámara web y su resolución). Este último bloque incluye también un botón que permite actualizar el listado de dispositivos serie y las cámaras web disponibles.

Por último, comentar que, tras cada grabación, la aplicación generará dos archivos: un vídeo y un archivo ".mat". En el archivo ".mat" se almacenarán los siguientes datos en forma de matriz:

1. acx: aceleración en el eje x para cada uno de los sensores.
2. acy: aceleración en el eje y para cada uno de los sensores.
3. acz: aceleración en el eje z para cada uno de los sensores.
4. t: tiempo correspondiente en milisegundos.
5. syncVideoArduino: vector utilizado para relacionar muestras y fotogramas.

Estos dos archivos serán autonombrados utilizando una marca temporal que reflejará la hora y la fecha en la que han sido generados, de esta manera se evitarán problemas de reemplazado de ficheros y permitirá una mayor organización de los mismos.

Los dos archivos serán utilizados como parámetros de entrada de la siguiente aplicación.

4.3.3 Aplicación Sincronización

La aplicación se emplea para compensar la falta de sincronismo existente entre las muestras recogidas por el sistema de lectura y los fotogramas capturados por la cámara web.

Este proceso de sincronizado es muy común en el mundo audiovisual, siendo el uso de la claqueta uno de los métodos más sencillos y populares. Por esto motivo, a continuación, se analizará este método y se intentarán aplicar los principios del mismo para acompasar las señales de aceleración con el vídeo.

El principio de este método se basa en realizar una acción claramente identificable que se refleje tanto en la pista de sonido como en la de vídeo. Esto se consigue cerrando la barra articulada de la claqueta delante de la cámara, generando un pico de sonido que se corresponderá con el fotograma del vídeo en el que la claqueta pase de estar abierta a cerrada.

El principal problema que existe al intentar trasladar este principio para sincronizar las señales de aceleración con el vídeo es, que a diferencia del micrófono que capta el *clak* de la claqueta, el acelerómetro capta todo el movimiento que, por ejemplo, el dedo índice realiza hasta llegar al dedo pulgar para simular la claqueta. Es decir, la acción no se corresponde con un único instante fácilmente identificable, por lo que para sincronizar la señal se opta por utilizar varios instantes de referencia.

En la Figura 21 se observa el resultado de una buena sincronización mediante el uso de instantes de referencia. Para conseguir este resultado, se analizan las imágenes y se relacionan con la señal de aceleración teniendo en cuenta los siguientes aspectos:

1. En ausencia de movimiento la aceleración en x es prácticamente nula.
2. El pico positivo de aceleración corresponde justo con el instante anterior al contacto del dedo en cuestión con el pulgar.
3. El paso de aceleración máxima positiva a negativa corresponde con el instante en el que el dedo en cuestión choca con el pulgar.
4. El pico negativo de aceleración corresponde con el movimiento del dedo para regresar a su posición de reposo.
5. La aceleración positiva que aparece tras este pico negativo corresponde con la llegada del dedo a su posición de reposo.
6. Tras el gesto, la aceleración en x vuelve a ser cercana a cero.

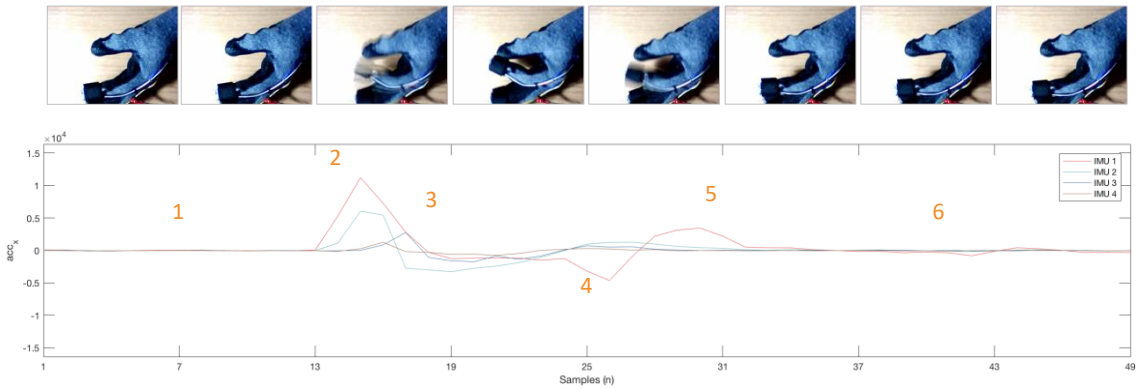


Figura 21. Instantes de referencia para el proceso de sincronización.

Una vez son conocidas las directrices a seguir para completar el proceso con éxito, se presentan y comentan los principales elementos de la interfaz para, de esta manera, facilitar la explicación del funcionamiento básico de la aplicación.

Como se muestra claramente en la Figura 22, la interfaz está compuesta principalmente por cinco elementos. La barra de herramientas permite tanto cargar los archivos necesarios para iniciar el proceso de sincronizado, como guardar el resultado del mismo. En la parte superior derecha se encuentran dos indicadores que indican si los archivos han sido cargados correctamente. Justo debajo de estos hay cinco visores en los que se proyectan cinco fotogramas de la secuencia. A los pies de los fotogramas se encuentran cinco casillas que indican su estado. En último lugar se encuentran las gráficas de aceleración que representan las muestras de aceleración correspondientes con los fotogramas.

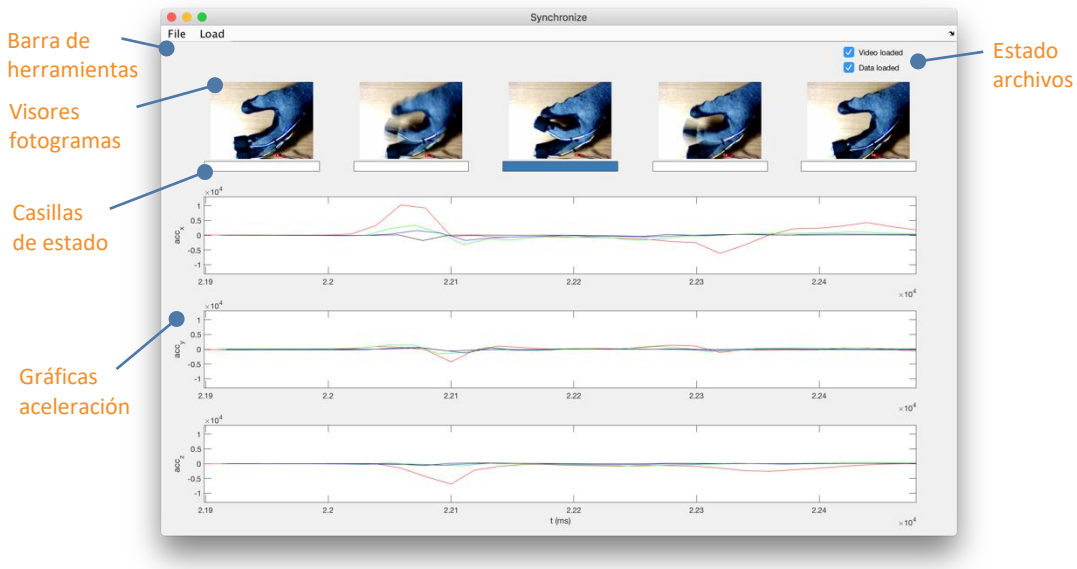


Figura 22. Interfaz gráfica aplicación "Sincronización".

Antes de continuar, cabe indicar que la aplicación pasa por tres estados a lo largo del proceso:

1. Selección de fotograma: al navegar tanto los fotogramas como los valores de aceleración se mueven en conjunto. Recuadros de estado en blanco.
2. Selección de muestra: al navegar los fotogramas permanecen estáticos mientras que las señales de aceleración se mueven. Recuadro de estado del fotograma seleccionado en verde.
3. Vídeo y señales sincronizadas: es posible navegar para comprobar que el ajuste realizado es correcto. Por si no lo fuera, será posible deshacer la sincronización. Recuadro de estado del fotograma seleccionado en azul.

Dicho esto, se procede a explicar el funcionamiento simplificado de la aplicación con ayuda del diagrama de flujo de la Figura 23.

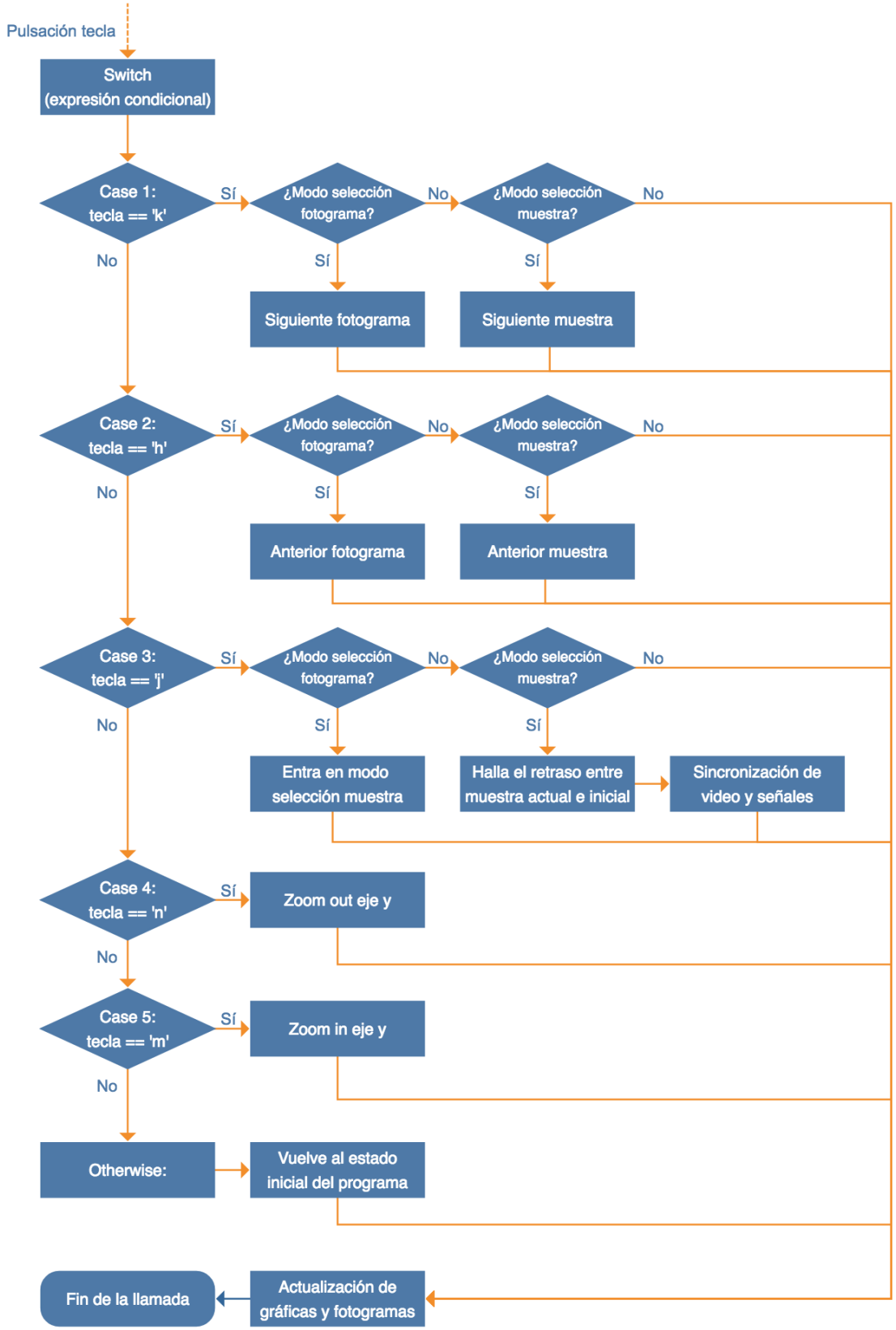


Figura 23. Flujograma del funcionamiento básico aplicación "Sincronización".

Tras cargar los dos ficheros de salida (datos y vídeo) de la aplicación "Captura", en la ventana se muestran los cinco primeros fotogramas del vídeo, acompañados de unos recuadros que indicarán el modo en el que se encuentra el programa, y sus valores de aceleración asociados (no

sincronizados). Después de este proceso inicial, el programa no realiza ninguna acción hasta que una tecla sea pulsada, bien sea a través del teclado o mediante el uso del controlador introducido en el punto 4.3.1. En el momento en el que esto ocurra, se llevará a cabo la acción correspondiente a la tecla pulsada y al estado en el que se encuentre (véase Figura 23).

Para seguir con el proceso de sincronizado, se navega por el vídeo en busca de una parte en la que existan los mayores instantes característicos posibles, y se pulsa la letra "j" para entrar en el modo selección de muestra, donde los fotogramas pasan a ser estáticos y las señales de aceleración dinámicas. Por último, se desplazan las señales de aceleración hasta que se encuentran en sincronía con el vídeo y se guarda el resultado.

Tras terminar con el proceso de sincronización, la aplicación genera un nuevo archivo en el que las matrices de aceleración y de tiempo han sido ajustadas para compensar el retraso existente entre el vídeo y las muestras.

4.3.4 Aplicación Etiquetado y Exportación

El uso de esta aplicación permite al usuario seleccionar aquellos instantes donde identifique un gesto, para más tarde indicar su tipo y poder exportar el set de gestos con su etiqueta correspondiente.

Antes de entrar a explicar el funcionamiento básico de la aplicación se comentarán los principales elementos de la interfaz. No obstante, como puede observarse en la Figura 24, la interfaz es muy similar a la de la aplicación de sincronizado por lo que únicamente se describirán aquellos elementos nuevos.

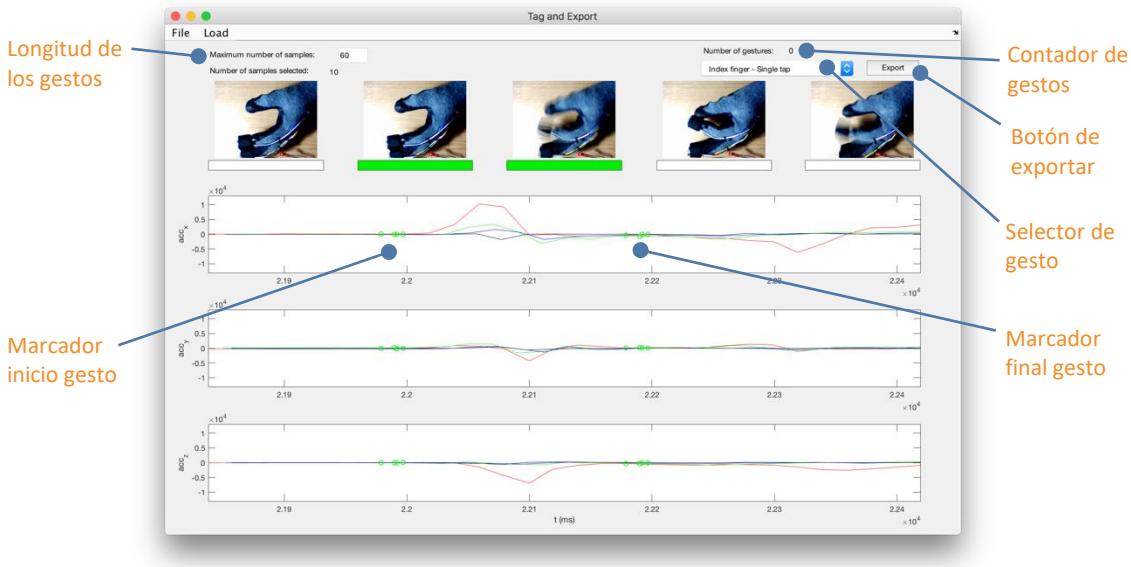


Figura 24. Interfaz gráfica aplicación "Etiquetado y Exportación".

La aplicación incorpora un recuadro editable de texto desde el que se puede especificar la longitud máxima que tendrán los gestos. Justo debajo hay un indicador que refleja el número de muestras seleccionadas. Este número indica las muestras existentes entre el marcador de inicio del gesto y marcador de su final. Mediante estas dos marcas, se delimitan los gestos que más tarde son etiquetados por medio del selector desplegable que hay en la parte superior derecha de la ventana. Junto a este listado se ha incorporado un contador que indica el número de gestos identificados. Por último, pulsando el botón de exportar, el usuario podrá obtener todas las señales etiquetadas en un fichero.

El funcionamiento básico de la aplicación será explicado con la ayuda del diagrama de flujo de la Figura 25.

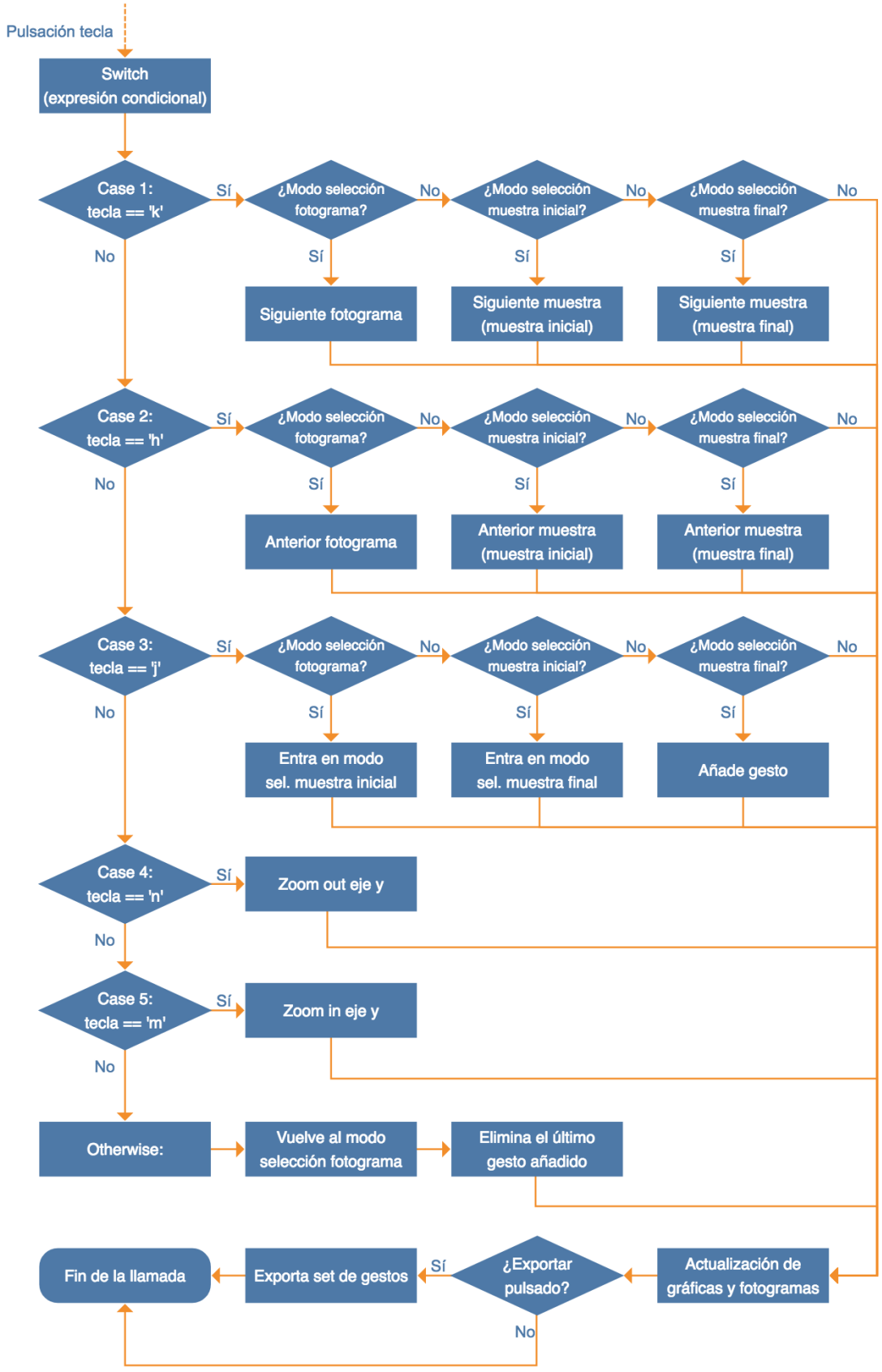


Figura 25. Flujograma del funcionamiento básico aplicación "Etiquetado y Exportación".

De igual manera que en la aplicación "Sincronizado", una vez cargados los ficheros, en la ventana se muestran los cinco primeros fotogramas del vídeo acompañados de unas casillas que indican

el modo en el que se encuentra el programa y sus valores de aceleración asociados, esta vez ya sincronizados con el vídeo. Tras el proceso inicial, el programa no realiza ninguna acción hasta que una tecla sea pulsada. En el momento que esto ocurra, se llevará a cabo la acción correspondiente a la tecla pulsada y al estado en el que se encuentre el programa (véase Figura 25).

Una de las diferencias principales con respecto al programa anterior son los estados. Por ello, a continuación, se detallará cada uno de los estados por los que pasa el programa:

1. Selección de fotograma: al navegar tanto los fotogramas como los valores de aceleración se mueven en conjunto. Recuadro de estado en blanco si no existe gesto registrado. Recuadro en azul en todos los fotogramas que contengan un gesto registrado.
2. Selección muestra inicial: al navegar tanto los fotogramas como los valores de aceleración se mueven en conjunto. Aparece un marcador en verde para indicar el inicio del gesto. Recuadro del fotograma en el que se encuentre la muestra en verde.
3. Selección muestra final: al navegar tanto los fotogramas como los valores de aceleración se mueven de manera conjunta. Aparece un segundo marcador en verde para indicar el final de gesto. Todos los fotogramas que estén comprendidos entre la muestra inicial y la final estarán en verde.

Para seguir con el proceso de etiquetado se navega por el vídeo en busca de un gesto. En el momento en el que se encuentra un gesto se pulsa la letra "j", para acceder al modo selección de muestra inicial y así marcar el inicio del gesto. Una vez este ha sido marcado, se debe marcar el final del mismo pulsando nuevamente la letra "j". El proceso es repetido tantas veces como gestos haya.

Tras haber registrado todos los gestos se selecciona el tipo de gesto y se exportan los datos. A continuación, se explica el proceso de exportación para más tarde poder entender el set de datos creado.

Como ya se ha comentado previamente, dentro de la aplicación existe un parámetro variable que es el número de muestras máximas, al definirlo la aplicación solo permitirá identificar gestos que contengan un número menor o igual al especificado. Por lo que hasta ahora, se podría decir que la duración de los gestos no es homogénea, pues cada uno está formado por un determinado número de muestras. Este problema será solucionado dentro del proceso de exportación.

Si se analiza uno de los gestos que más tiempo dura (gesto de doble toque) se obtendrán unos valores similares a los de la Figura 26, donde se puede observar que este dura unas 40 muestras. Como el tamaño máximo definido por defecto es de 60 muestras, se irá desplazando una ventana de dicha longitud siempre conteniendo el gesto, creando para cada desplazamiento un nuevo registro de datos. Por ejemplo, del gesto de la figura saldrían 21 gestos, pues la ventana de 60 muestras se puede desplazar 21 veces (los límites son incluidos).

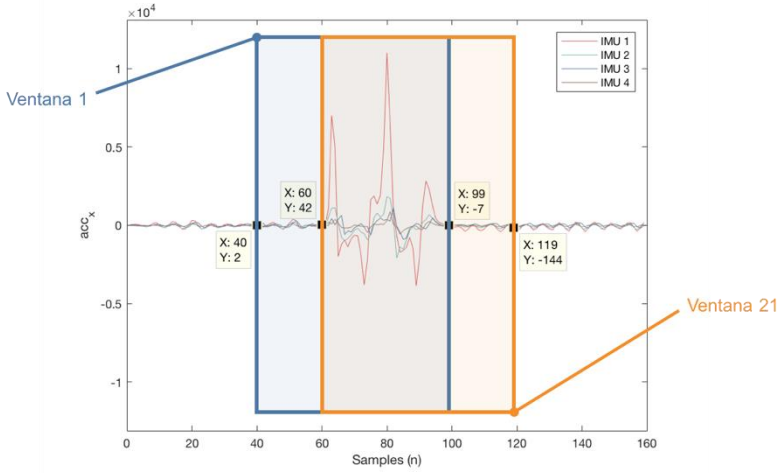


Figura 26. Ejemplo desplazamiento ventana 60 muestras (exportación).

Este fichero final contendrá la siguiente información:

1. Data: matriz de datos de aceleración en el eje x adecuada para el entrenamiento de la red neuronal.
2. Target: matriz donde se encuentran los *targets* correspondientes a cada gesto.

Además de poder exportar los datos, se da la posibilidad de guardar el progreso de selección de los gestos y reanudarlo en otro momento a partir de la carga de este, ya que se trata de un proceso largo. Esta funcionalidad también permitiría en un futuro cambiar la manera de exportar los datos sin perder todas las selecciones realizadas si fuese necesario.

4.3.5 Resultados

Siguiendo el recorrido por las tres aplicaciones desarrolladas, se van obteniendo subconjuntos de datos que, mediante el uso de un *script*, son concatenados obteniendo el set de datos final (véase Figura 27).

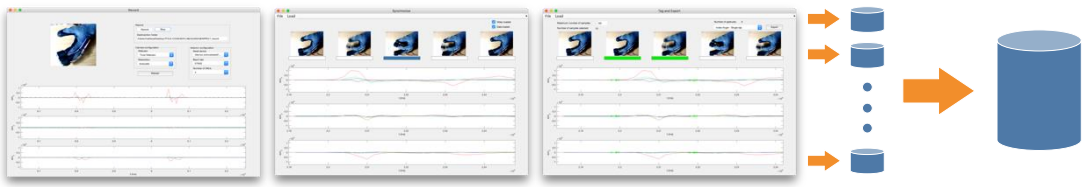


Figura 27. Sistema completo de creación del set de datos.

El set de datos está formado por 250 ejemplos aproximadamente para cada gesto, que al pasar por el proceso de exportación suman un total de 53.041 gestos (los datos fueron recogidos en tres sujetos distintos para una misma posición).

Recapitulando, como se ha podido ver en los puntos anteriores, se han diseñado y desarrollado tres interfaces gráficas capaces de controlar tres aplicaciones completamente funcionales que facilitan la generación de un set de muestras de entrenamiento de manera sencilla y eficiente, cumpliendo de esta manera con el segundo objetivo del trabajo.

4.4 Red neuronal

Pese a no ser este uno de los objetivos principales del trabajo, se entrenará una red neuronal simple como puesta a prueba del sistema completo de lectura y del set de datos generado. Seguidamente, se realizará una breve introducción a las redes neuronales artificiales y en particular a las de tipo *feedforward*, para más tarde proceder al entrenamiento de la misma y finalmente comprobar su funcionamiento en tiempo real.

4.4.1 Introducción

Una red neuronal es un enorme procesador paralelo y distribuido que tiene la tendencia natural de almacenar conocimientos derivados de la experiencia y hacer que estén disponibles para su uso. Esta se asemeja al cerebro en dos aspectos. En primer lugar, el conocimiento es adquirido por la red a través de un proceso de aprendizaje. En segundo lugar, las fuerzas de conexión entre neuronas, conocidas como pesos sinápticos, son utilizadas para almacenar el conocimiento adquirido [13].

El perceptrón es la red neuronal más simple capaz de clasificar patrones linealmente separables. Como puede observarse en la Figura 28, esta consta de una simple neurona con pesos sinápticos ajustables, *bias* y una función de transferencia que genera la salida de la red.

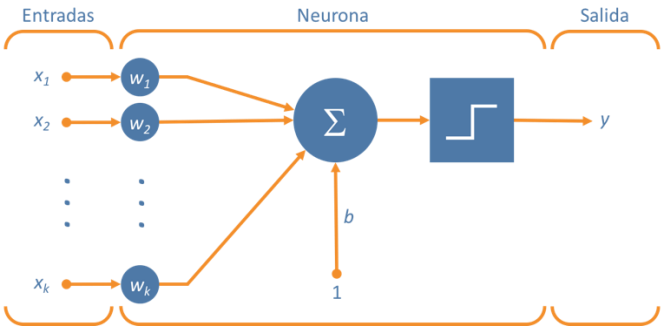


Figura 28. Estructura perceptrón de una capa.

La limitación principal que estos presentan es que son clasificadores lineales binarios, es decir, que solo son capaces de clasificar patrones linealmente separables y al tener solo una neurona, únicamente son capaces de distinguir entre dos clases. Sin embargo, estos dos problemas se resuelven aumentando el número de neuronas y capas respectivamente; entrando de esta manera, a los perceptrones multicapa también llamados redes *feedforward* multicapa.

Las redes *feedforward* multicapa están compuestas por una capa de entrada, una o varias ocultas y una de salida (véase Figura 29).

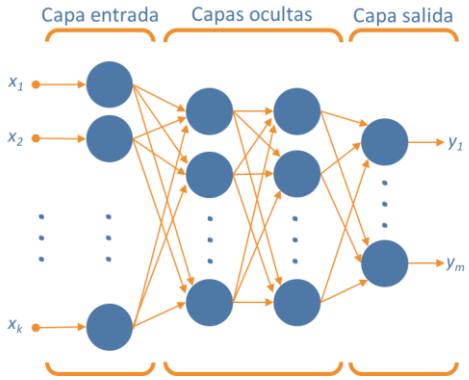


Figura 29. Estructura red *feedforward* con dos capas ocultas.

Algunas de las características principales de los perceptrones multicapa son [13]:

1. El modelo de cada neurona de la red incluye una función de transferencia no lineal. La no linealidad de la función es importante pues, de lo contrario, la relación entrada salida de la red podría reducirse a la de un perceptrón de una sola capa. Una de las funciones más utilizadas es la *sigmoid*.
2. Las capas ocultas, que no forman parte ni de la entrada ni de la salida, permiten a la red extraer progresivamente características significativas de los valores de entrada.
3. La red tiene un alto grado de conectividad, determinado por las conexiones entre neuronas. Cualquier cambio realizado en la conectividad de la red requiere un cambio sobre los pesos sinápticos.

Como se ha dicho anteriormente, los conocimientos adquiridos durante el proceso de aprendizaje de la red quedan reflejados en los pesos sinápticos. La estimación de estos valores se realiza mediante el uso de algoritmos, siendo el de *backpropagation* uno de los algoritmos supervisados más utilizado para el entrenamiento de este tipo de redes. Este se puede resumir en las siguientes fases:

1. *Forward pass*: a partir de un valor de entrada de ejemplo, se calculan los valores de salida para los pesos y *bias* actuales (los valores iniciales son aleatorios). A partir de este resultado se halla el error existente entre el valor obtenido y el valor esperado (también llamado *target* o *ground truth*).

2. *Backward pass*: haciendo uso de los errores calculados, se actualizan los pesos y *bias* de las neuronas para que el valor de salida sea más cercano a los *targets*.

Estas dos fases son repetidas un determinado número de veces con el fin de minimizar cada vez más el error. Sin embargo, uno de los inconvenientes del algoritmo es que no converge, por lo que no existe un criterio bien definido que diga cuándo hay que parar. A pesar de esto, cabe decir que la fácil implementación y el alto número de soluciones efectivas para problemas complejos que este proporciona, lo convierte en uno de los métodos más utilizados para los problemas de clasificación de patrones [13].

4.4.2 Entrenamiento de la red neuronal con MATLAB

En este punto, se explicará el proceso seguido para llevar a cabo el entrenamiento de la red neuronal *feedforward* multicapa capaz de clasificar las señales entrantes del sistema de lectura en nueve clases. Para llevar a cabo este proceso se hará uso de la *ToolBox* "Neural Network" de MATLAB.

Antes de empezar a entrenar la red, se define el número de neuronas a utilizar en cada una de las capas.

El número de neuronas de la capa de entrada para este caso viene dado por la longitud de las ventanas temporales tomadas y el número de sensores utilizados. Debido a que se utilizan cuatro MPU-6050 y las ventanas temporales son de 60 muestras, el número total de neuronas a la entrada será de 240, esto introducirá un retardo de 120 ms pues, como se dijo en capítulos anteriores, la frecuencia de muestreo de las IMU es de 50 Hz. Con respecto al número de neuronas en la capa de salida, este es definido por la cantidad de clases a identificar, o lo que es lo mismo, el número de gestos existentes, con lo que habrá 9 neuronas a la capa de salida. Por otro lado, no existe una manera exacta de determinar las neuronas necesarias para la capa oculta, por lo que, tiene que ser elegido mediante prueba y error. Tras realizar varias pruebas, el número que mejor resultados da es 112. En la Figura 30, puede observarse un esquema de la red neuronal diseñada.

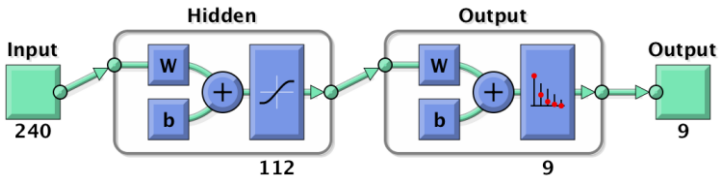


Figura 30. Esquema de la red neuronal empleada.

Para llevar a cabo el entrenamiento de la red es necesario dividir el set de datos en tres subsets (entrenamiento, validación y test). Normalmente, este reparto suele hacerse de manera proporcional sobre el set global. Sin embargo, se ha comprobado durante la realización de pruebas que, debido al proceso de exportación, en el cual, a partir de la señal delimitada de tamaño "a" se crean tantas señales como diferencia exista entre el tamaño de la ventana establecido y "a", no tiene sentido aplicarlo en este caso, pues al set de test llegarían entradas muy parecidas a las ya utilizadas para el entrenamiento, proporcionando de esta manera información sin validez.

Por esta razón, se utilizarán tres sets (de tres sujetos distintos) agrupados y divididos proporcionalmente para el proceso de entrenamiento (70%) y validación (30%). Mientras que para la realización del test se emplearán dos sets distintos a los anteriores. Para evitar que existan registros de datos muy parecidos dentro del grupo de entrenamiento se añadirá un ruido blanco Gaussiano sobre este.

Por último, se normalizan los valores a la entrada de la red (para mejorar el rendimiento del algoritmo de *backpropagation* [13]) y se inicia el entrenamiento.

4.4.3 Resultados

Una vez finalizado dicho proceso se analizan los resultados obtenidos mediante el uso de matrices de confusión (véase Figura 31).

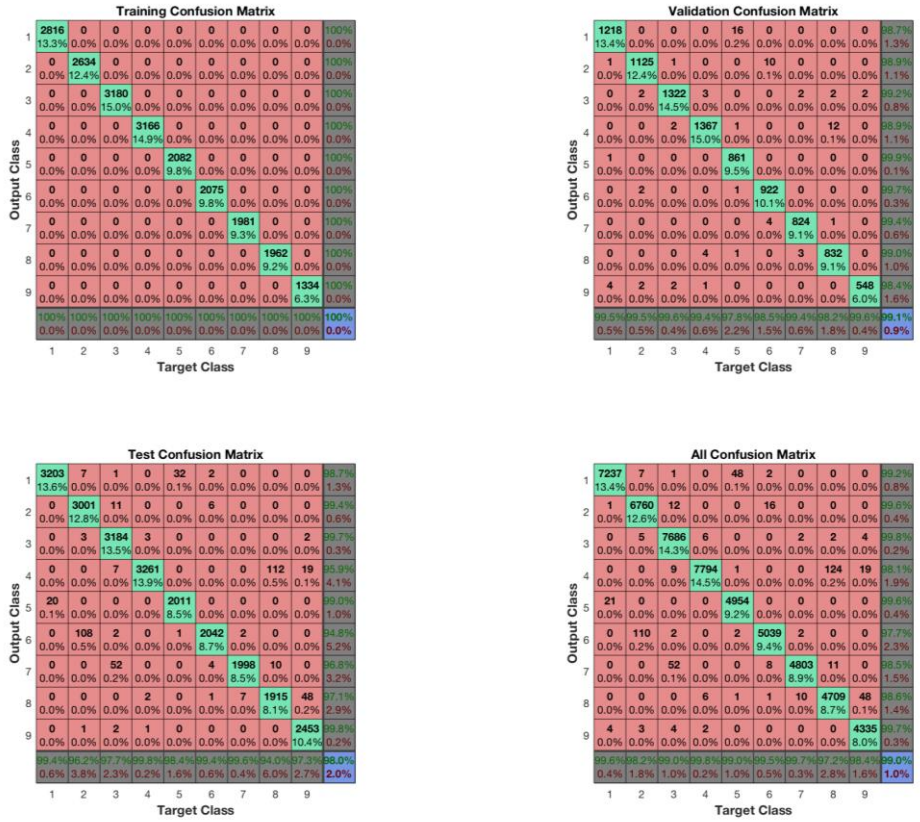


Figura 31. Matrices de confusión de la red neuronal entrenada.

Estas matrices recogen el resultado de salida obtenido por la red para cada set de muestras. El eje de las abscisas corresponde con los valores esperados (*ground truth* o *targets*), mientras que el de las ordenadas con el valor de salida. La columna de la derecha representa la precisión de la red, también llamada valor predictivo positivo (en verde), y el valor predictivo negativo (en rojo). Por otro lado, la fila inferior representa la sensibilidad, también llamada ratio de verdaderos negativos (en verde), y el ratio de falsos positivos (en rojo). De esta manera es posible comprobar de manera visual y rápida si la red neuronal es capaz de distinguir entre los distintos gestos. En este caso, tal y como puede observarse en la Figura 31, en la parte inferior derecha de la matriz de confusión de test, la *accuracy* global de la red es del 98%.

Antes de utilizar la red neuronal para reconocer los gestos en tiempo real, se simulará la salida de la misma ante un pequeño set de prueba. Se representarán las aceleraciones en el eje x de cada uno de los dedos acompañadas de una barra superior que, mediante el uso de un punto de color, indicará si ha detectado toque simple (color azul) o doble (color verde), mientras que la ausencia de toque será identificada con una "x" marrón. (véase Figura 32).

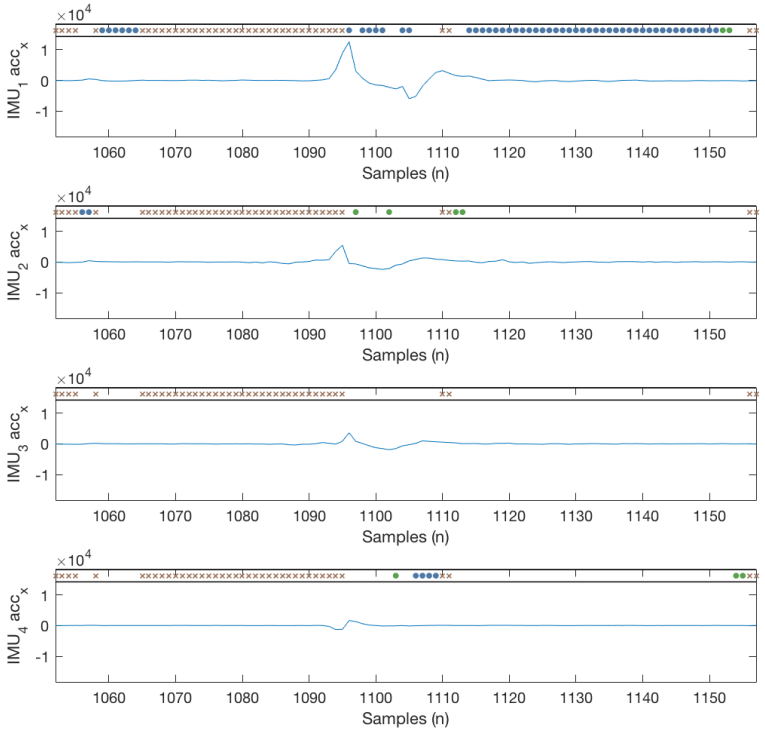


Figura 32. Fragmento ejemplo de salida de la red neuronal (toque simple dedo índice).

A continuación, mediante el ejemplo de la Figura 32 se analizan los valores de salida obtenidos. En primer lugar, para poder interpretar esta figura es necesario tener en cuenta que la entrada de la red está formada por 4 ventanas de 60 muestras (una para cada sensor), que se desplazan conjuntamente muestra a muestra a lo largo eje x, dando un valor de salida para cada uno de los desplazamientos. Además, al tratarse de un sistema causal (el cual no predice futuros valores de entrada) los marcadores que indican la detección de un gesto aparecerán tras el mismo.

Tal y como se puede observar, al tratarse de un toque simple realizado con el dedo índice, la mayor parte de los puntos azules se encuentran en la primera gráfica, por lo que está siendo bien clasificado. No obstante, debido al desplazamiento de la ventana que provoca la múltiple evaluación de la red, es posible que no todos los valores de salida coincidan con el esperado. Por esta razón, se aplicará un postprocesado simple sobre las señales de salida. Este consistirá en almacenar las últimas quince clasificaciones y únicamente se tendrá en cuenta si existe una mayoría de 12 aciertos. A pesar de que de esta manera se introduce un retardo mínimo de 12 muestras (240 ms), se consigue una salida que se asemeja más a la esperada (véase Figura 33).

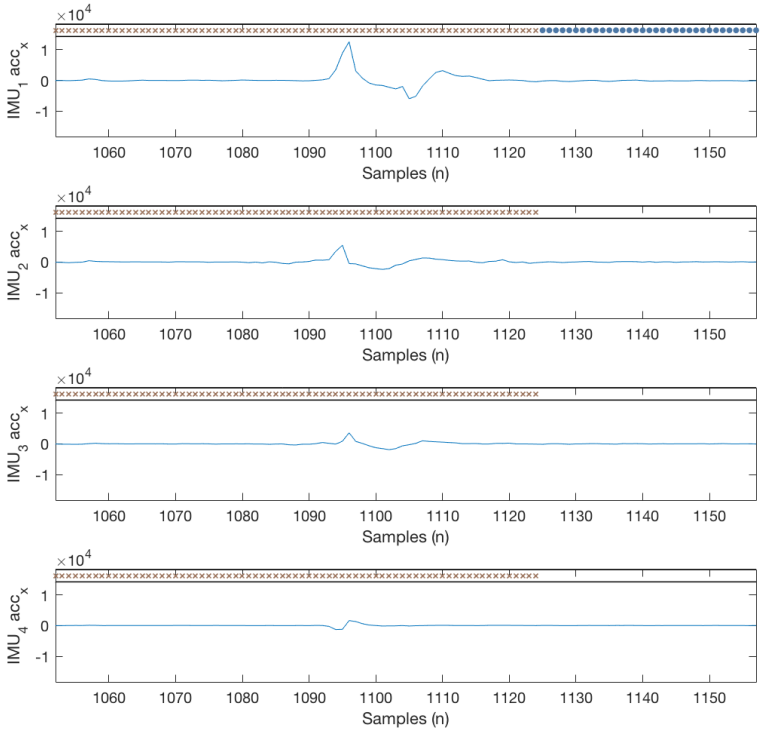


Figura 33. Fragmento ejemplo de salida de la red neuronal con postprocesado.

Por último, siguiendo la misma lógica, se diseña y desarrolla una nueva aplicación en MATLAB capaz de mostrar de manera gráfica los gestos realizados con los dedos de la mano (véase Figura 34). En esta aplicación se identificarán los gestos mediante la superposición de una huella dactilar de color en el dedo correspondiente.

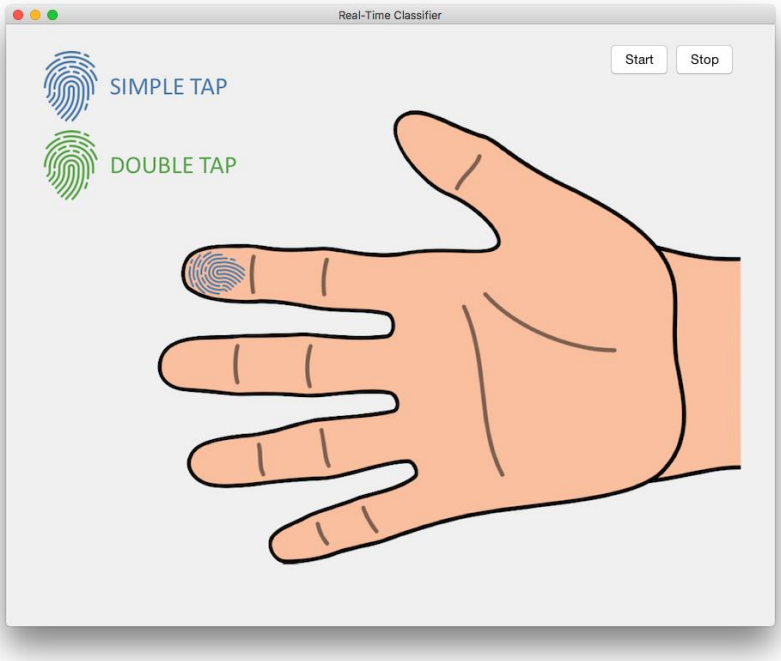


Figura 34. Interfaz gráfica para la clasificación de los gestos en tiempo real.

Capítulo 5. Presupuesto

En este capítulo se detallará el presupuesto necesario para el desarrollo del proyecto. Este se desglosará en tres bloques. En primer lugar, se recogerán los costes de personal (trabajo realizado por las personas implicadas). En segundo lugar, los costes relacionados con el *hardware* empleado. Por último, los costes asociados al *software* utilizado.

5.1 Costes de personal

En este punto se detalla la remuneración bruta de cada uno de los participantes en el proyecto, considerándose la contribución de José Manuel Mossi García (como tutor), Manuel Palacios Hurtado (como cotutor) y Marcos Fernández Carbonell (como alumno).

Concepto	Unidades	Nº de unidades	Coste unitario (€)	Importe (€)
Alumno	h	350	33,00	11.550,00
Cotutor	h	30	38,00	1.140,00
Tutor	h	20	50,00	1.000,00
				13.690,00

Tabla 10. Costes de personal.

5.2 Costes de *hardware*

En este punto se recoge el coste asociado al *hardware*. Dado que algunos de estos elementos no han sido obtenidos para la realización exclusiva del trabajo, el coste se dividirá en dos bloques, permitiendo de esta manera tener en cuenta el periodo amortizado cuando sea necesario.

Concepto	Unidades	Nº de unidades	Coste unitario (€)	Importe (€)
RobotDyn WiFi-NodeM	u	3	2,40	7,20
Feather HUZZAH ESP8266 WiFi	u	2	11,20	22,40
Módulo GY-521	u	4	2,20	8,80
Guante	u	1	1,20	1,20
Pack cables y conectores	u	1	10,50	10,50
				50,10

Tabla 11. Costes de *hardware* guante

Concepto	Unidades	Nº de unidades	Coste unitario (€)	Amortización (meses)	Periodo amortizado (meses)	Importe (€)
Placa de pruebas	u	1	7,59	24	8	2,53
Contour ShuttleXpress	u	1	37,13	48	8	6,19
Trust Exis WebCam	u	1	8,65	36	8	1,92
MacBook Pro (2,7 GHz Intel Core i5)	u	1	1.184,21	60	8	157,89
						168,53

Tabla 12. Costes resto de *hardware*.

5.3 Costes de *software*

En este punto se detalla el coste ligado al uso de *software*. Para ello, además de tener en cuenta el periodo de amortización, se reflejan los costes para dos casos (siendo alumno de la Universitat Politècnica de València y sin serlo).

Concepto	Unidades	Nº de unidades	Coste unitario (€)	Amortización (meses)	Periodo amortizado (meses)	Importe (€)
MATLAB R2017b (Academic license)	u	1	0,00 800,00*	12	8	0,00 533,34*
MATLAB Image Acquisition Toolbox	u	1	0,00 400,00*	12	8	0,00 266,67*
MATLAB Neural Network Toolbox	u	1	0,00 400,00*	12	8	0,00 266,67*
MATLAB Signal Processing Toolbox	u	1	0,00 400,00*	12	8	0,00 266,67*
Office 365 Universitarios	u	1	0,00 79,00*	48	8	0,00 13,17*
						0,00 1.346,52*

Tabla 13. Costes de *software*.

* Coste sin ser alumno de la Universitat Politècnica de València.

5.4 Presupuesto total del proyecto

Para finalizar se detalla de manera individual el coste total del proyecto para los dos casos planteados.

Concepto	Importe (€)
Costes de personal	13.690,00
Costes de *hardware* guante	50,10
Costes resto de *hardware*	168,53
Coste de *software*	0,00
Total del presupuesto de ejecución material	13.908,63
Gatos generales (13%)	1.808,12
Beneficio industrial (6%)	834,52
Total del presupuesto (sin IVA)	16.551,27
IVA (21%)	3.475,77
Total del presupuesto (IVA incluido)	**20.027,04**

Tabla 14. Presupuesto total del proyecto siendo alumno de la UPV.

El coste global del proyecto asciende a una cantidad de **VEINTE MIL VEINTISIETE EUROS CON CUATRO CENTIMOS** (IVA incluido).

Concepto	Importe (€)
Costes de personal	13.690,00
Costes de *hardware* guante	50,10
Costes resto de *hardware*	168,53
Coste de *software*	1.346,52
Total del presupuesto de ejecución material	15.255,15
Gatos generales (13%)	1.983,17
Beneficio industrial (6%)	915,31
Total del presupuesto (sin IVA)	18.153,63
IVA (21%)	3.812,26
Total del presupuesto (IVA incluido)	**21.965,89**

Tabla 15. Presupuesto total del proyecto sin ser alumno de la UPV.

El coste global del proyecto asciende a una cantidad de **VEINTIÚN MIL NOVECIENTOS SESENTA Y CINCO EUROS CON OCHENTA Y NUEVE CENTIMOS** (IVA incluido).

Capítulo 6. Conclusiones y propuesta de trabajo futuro

En este trabajo se ha diseñado y desarrollado un sistema de lectura completo capaz gestionar cuatro sensores para la obtención de información relativa al movimiento de los dedos de la mano, utilizando para ello datos de los acelerómetros y los giróscopos. Además, analizando las señales obtenidas, han sido definidos nueve gestos que podrían llegar a controlar de manera intuitiva el futuro sistema de realidad aumentada de los bomberos.

Por otro lado, se han diseñado e implementado tres aplicaciones totalmente funcionales que, en conjunto, permiten capturar las señales provenientes del sistema de lectura a la vez que los gestos son grabados mediante el uso de una cámara, para su posterior clasificación y exportación. De esta manera se agiliza la creación de un set de datos que será utilizado para un futuro entrenamiento de una red neuronal. Además, mediante el uso de las mismas, se ha creado un set de datos formado por 53.041 gestos, que no solo ha servido para iniciar el proceso de creación del set, sino también para testear y perfeccionar el funcionamiento de estas.

En último lugar, a pesar de no ser este uno de los objetivos del trabajo, ha sido entrenada y puesta a prueba una red neuronal *feedforward* simple, capaz de clasificar las señales de salida del sistema de lectura en los nueve gestos definidos. Asimismo, se ha creado una aplicación con el fin de mostrar de manera gráfica la identificación de los gestos en tiempo real. Cabe aclarar que esta última parte ha sido elaborada como puesta a prueba del trabajo realizado, tanto del sistema de lectura como el set de entrenamiento generado a partir de las aplicaciones desarrolladas, y que en futuros proyectos será íntegramente replanteada.

Todo el trabajo realizado, ha supuesto un avance significativo sobre el proyecto global y ha abierto potenciales líneas de trabajo futuro, como las que se plantean a continuación:

- Comunicar de manera inalámbrica el sistema de lectura con el sistema central donde se encuentra la interfaz.
- Añadir batería al sistema de lectura, ya que hasta el momento ha sido alimentado vía USB.
- Ampliar el set de datos recogiendo ejemplos de gestos para más sujetos.
- Replantear desde cero todo lo relativo con la red neuronal, analizando el posible uso de redes neuronales recurrentes (RNN) y poniendo especial atención en las de tipo LSTM (Long Short Term Memory) con las que se puede guardar información del pasado por largos periodos de tiempo y utilizarla para la obtención de la salida.
- Integración de la red neuronal entrenada dentro del microcontrolador.

Capítulo 7. Bibliografía

[1] Fundación MAPFRE y APTB, «Fundación MAPFRE,» 2017. [En línea]. Disponible en: https://www.fundacionmapfre.org/documentacion/publico/i18n/catalogo_imagenes/grupo.cmd?path=1094554. [Último acceso: abril 2018].

[2] Arduino, «Arduino Introduction,» [En línea]. Disponible en: https://www.arduino.cc/en/Guide/Introduction. [Último acceso: abril 2018].

[3] MathWorks, MATLAB, «Descripción general,» [En línea]. Disponible en: https://es.mathworks.com/products/matlab.html. [Último acceso: abril 2018].

[4] Espressif Systems, «ESP8266EX Datasheet,» 2018. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Último acceso: abril 2018].

[5] J. Kim, Y. H. Kwak, W. Kim, K. Park, J. J. Pak y K. Kim, «Flexible Force Sensor Based Input Device for Gesture Recognition Applicable to Augmented and Virtual Realities,» de *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Jeju, 2017.

[6] F. J. G. Casado, «Galgas Extensiómetricas: Funcionamiento y Circuitos de Medida,» Universitat Politècnica de València. Escuela Técnica Superior de Ingenieros Industriales, mayo 2010. [En línea]. Disponible en: http://hdl.handle.net/10251/7747. [Último acceso: abril 2018].

[7] J. Martínez, A. García, M. Oliver, J. P. Molina y P. González, Artists, *Position of the LED markers used by the tracking system.* [Art]. IEEE Computer Graphics and Applications, 2016.

[8] InvenSense, «MPU-6000 and MPU-6050 Product Specification Revision 3.4,» 2013. [En línea]. Disponible en: https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/. [Último acceso: abril 2018].

[9] Philips Semiconductors, «The I2C-bus and how to use it (including specifications),» 1995. [En línea]. Disponible en: http://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf. [Último acceso: abril 2018].

[10] Texas Instruments, «Understanding the I2C Bus,» 2015. [En línea]. Disponible en: http://www.ti.com/lit/an/slva704/slva704.pdf. [Último acceso: abril 2018].

[11] MATLAB, «MATLAB Documentation tic function,» 2006. [En línea]. Disponible en: https://es.mathworks.com/help/matlab/ref/tic.html. [Último acceso: abril 2018].

[12] MATLAB, «MATLAB Documentation toc function,» 2006. [En línea]. Disponible en: https://es.mathworks.com/help/matlab/ref/toc.html. [Último acceso: abril 2018].

[13] S. Haykin, Neural Networks: A Comprehensive Foundation, Hamilton, Ontario: Macmillan, 1994.

Capítulo 8. Anexos

8.1 Código fuente lectura MPU-6050

```
/**
 *  @file    MPU6050_DMP6_4_ADAFRUIT.ino
 *  @author  Marcos Fernández Carbonell <marferca@teleco.upv.es>
 *  @date    12/29/2017
 *  @version 1.0
 *
 *  @section DESCRIPTION
 *
 *  This is a little program that reads DMP data from four
 *  MPU-6050's and then writes it to the serial port.
 *
 */

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

// LED PIN
#define LED_PIN 16

// ARDUINO PINS TO IMU AD0 PINS
#define IMU_1 14
#define IMU_2 12
#define IMU_3 13
#define IMU_4 15


// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t devStatus;      // return status after each device operation (0 = success, !0 =
error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
```

```
// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float off[4];           // [Gx_off, Gy_off, Gz_off, Az_off] offset calibration


// ===============================================================
// ===                      INITIAL SETUP                      ===
// ===============================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock.
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    Serial.begin(57600);
    while (!Serial);

    // configure pin IMU selector
    pinMode(IMU_1,OUTPUT);  // IMU1
    pinMode(IMU_2,OUTPUT);  // IMU2
    pinMode(IMU_3,OUTPUT);  // IMU3
    pinMode(IMU_4,OUTPUT);  // IMU4

    for (int i = 1; i <= 4; i++){
      switch(i){
        case 1:
          digitalWrite(IMU_1,LOW);  // IMU1
          digitalWrite(IMU_2,HIGH); // IMU2
          digitalWrite(IMU_3,HIGH); // IMU3
          digitalWrite(IMU_4,HIGH); // IMU4
          off[1] = -58;             // XGyroOffset value IMU1
          off[2] = -25;             // YGyroOffset value IMU1
          off[3] = -19;             // ZGyroOffset value IMU1
          off[4] = 1454;            // ZAccelOffset value IMU1
          break;
        case 2:
          digitalWrite(IMU_1,HIGH); // IMU1
          digitalWrite(IMU_2,LOW);  // IMU2
          digitalWrite(IMU_3,HIGH); // IMU3
          digitalWrite(IMU_4,HIGH); // IMU4
          off[1] = 90;              // XGyroOffset value IMU2
          off[2] = 58;              // YGyroOffset value IMU2
          off[3] = -17;             // ZGyroOffset value IMU2
          off[4] = 991;             // ZAccelOffset value IMU2
          break;
        case 3:
          digitalWrite(IMU_1,HIGH); // IMU1
          digitalWrite(IMU_2,HIGH); // IMU2
          digitalWrite(IMU_3,LOW);  // IMU3
          digitalWrite(IMU_4,HIGH); // IMU4
          off[1] = -4;              // XGyroOffset value IMU3
          off[2] = 97;              // YGyroOffset value IMU3
          off[3] = -32;             // ZGyroOffset value IMU3
          off[4] = 1333;            // ZAccelOffset value IMU3
          break;
        case 4:
          digitalWrite(IMU_1,HIGH); // IMU1
          digitalWrite(IMU_2,HIGH); // IMU2
          digitalWrite(IMU_3,HIGH); // IMU3
          digitalWrite(IMU_4,LOW);  // IMU4
          off[1] = 237;             // XGyroOffset value IMU4
          off[2] = -56;             // YGyroOffset value IMU4
          off[3] = -18;             // ZGyroOffset value IMU4
          off[4] = 1081;            // ZAccelOffset value IMU4
          break;
        default:
          break;
      }

      // initialize device
      //Serial.println(F("Initializing I2C devices..."));
```

```
      mpu.initialize();

      // verify connection
      //Serial.println(F("Testing device connections..."));
      //Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));

      // load and configure the DMP
      //Serial.println(F("Initializing DMP..."));
      devStatus = mpu.dmpInitialize();

      // supply your own gyro offsets here, scaled for min sensitivity
      mpu.setXGyroOffset(off[1]);
      mpu.setYGyroOffset(off[2]);
      mpu.setZGyroOffset(off[3]);
      mpu.setZAccelOffset(off[4]);

      // make sure it worked (returns 0 if so)
      if (devStatus == 0) {

        // turn on the DMP, now that it's ready
        //Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        //Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
        Serial.println(packetSize);
        fifoCount = mpu.getFIFOCount();
        Serial.println(fifoCount);

      } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
      }
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);

    Serial.print("\n");
}



// ================================================================
// ===                      MAIN PROGRAM LOOP                   ===
// ================================================================
int imuNumber = 1;
unsigned long time = 0;

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;


    // multiple sensors control
    if (imuNumber > 4) imuNumber = 1;
    switch(imuNumber){
      case 1:
        digitalWrite(IMU_1,LOW);  // IMU1
        digitalWrite(IMU_2,HIGH); // IMU2
        digitalWrite(IMU_3,HIGH); // IMU3
        digitalWrite(IMU_4,HIGH); // IMU4
        break;
      case 2:
        digitalWrite(IMU_1,HIGH); // IMU1
        digitalWrite(IMU_2,LOW);  // IMU2
        digitalWrite(IMU_3,HIGH); // IMU3
        digitalWrite(IMU_4,HIGH); // IMU4
        break;
      case 3:
        digitalWrite(IMU_1,HIGH); // IMU1
        digitalWrite(IMU_2,HIGH); // IMU2
```

```
        digitalWrite(IMU_3,LOW);  // IMU3
        digitalWrite(IMU_4,HIGH); // IMU4
        break;
      case 4:
        digitalWrite(IMU_1,HIGH); // IMU1
        digitalWrite(IMU_2,HIGH); // IMU2
        digitalWrite(IMU_3,HIGH); // IMU3
        digitalWrite(IMU_4,LOW);  // IMU4
        break;
      default:
        break;
    }

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
    if (fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        //Serial.println(F("FIFO overflow!"));
        Serial.print('O'); Serial.print("\n");

    // otherwise, check for DMP data ready (this should happen frequently)
    } else {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize){
          //Serial.println("nix");
          fifoCount = mpu.getFIFOCount();
        }

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // if there is more than a packet reset FIFO
        if (fifoCount > packetSize){
          mpu.resetFIFO();
        }

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        // display real acceleration, adjusted to remove gravity
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetAccel(&aa, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);


        // Led
        Serial.print('L'); Serial.print("\n");
        digitalWrite(LED_PIN, HIGH);

        // FIFO Count
        //Serial.print(fifoCount); Serial.print("\n");

        // IMU number
        Serial.print(imuNumber); Serial.print("\n");

        // Print time value
        time = millis();
        Serial.print(time); Serial.print("\n");

        // Acceleration
        Serial.print(aaReal.x); Serial.print("\n");
        Serial.print(aaReal.y); Serial.print("\n");
        Serial.print(aaReal.z); Serial.print("\n");

        imuNumber++;
    }
}
```