Trabajo de Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

Aplicación de visión por computador basada en Sony Spresense

MEMORIA

Autor: Álvaro García Martín
Director: Manuel Moreno Eguílaz

Convocatoria: Abril 2023

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resumen

Esta memoria describe cómo se ha desarrollado una aplicación de visión por computador basada en la placa de desarrollo Sony Spresense. Toma como punto de referencia el estudio previo realizado por el estudiante Yeray Navarro en su Trabajo de Fin de Grado, *"Placa Programable Spresense de Sony. Desarrollo de una aplicación de visión por computador"* y los modelos de visión por computador definidos en él.

A lo largo del trabajo se definen diversas etapas que engloban el desarrollo de la aplicación, desde las etapas iniciales donde se concretan los objetivos, hasta el uso práctico de la aplicación. El desarrollo de este trabajo se pretende aplicar a la mejora de la prótesis de brazo diseñada y fabricada por el equipo de ARM2u, proyecto universitario impulsado por la Escuela Técnica Superior de Ingeniería Industrial de Barcelona, ETSEIB, de la Universidad Politécnica de Cataluña, UPC.

El proceso de creación de la aplicación se realiza de forma segmentada, empezando por un código base embebido en la placa de desarrollo de captura y clasificación de imágenes, pasando por una primera versión de la aplicación unida al código de movimiento de la prótesis y finalizando con una optimización de la misma.

Al concluir este trabajo se presenta una aplicación embebida en la placa de desarrollo Sony Spresense, cuyas funcionalidades son procesadas completamente por el procesador incorporado en la placa.

Los resultados de diferentes pruebas realizadas sobre la aplicación permiten demostrar su funcionalidad y validar el modelo de aplicación creado. Se configura como una aplicación que permite el movimiento natural de la prótesis pero, a voluntad del usuario, permite el cierre de la pinza al tamaño del objeto resultante de la clasificación de imágenes.

Resum

Aquesta memòria descriu com s' ha desenvolupat una aplicació de visió per computador basada en la placa de desenvolupament Sony Spresense. Pren com a punt de referència l'estudi previ realitzat per l'estudiant Yeray Navarro en el seu Treball de Fi de Grau, *"Placa Programable Spresense de Sony. Desarrollo de una aplicación de visión por computador"* i els models de visió per computador definits en ell.

Al llarg del treball es defineixen diverses etapes que engloben el desenvolupament de l'aplicació, des de les etapes inicials on es fixen els objectius, fins a l'ús pràctic de l'aplicació. El desenvolupament d' aquest treball va dirigit a la millora de la pròtesi de braç dissenyada i fabricada per l'equip ARM2u, projecte universitari impulsat per l'Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, ETSEIB, de la Universitat Politècnica de Catalunya, UPC.

El procés de creació de l'aplicació es realitza de forma segmentada, començant per un codi base embegut a la placa de desenvolupament de captura i classificació d'imatges, passant per una primera versió de l'aplicació unida al codi de moviment de la pròtesi i finalitzant amb una optimització de la mateixa.

En concloure aquest treball es presenta una aplicació embeguda a la placa de desenvolupament Sony Spresense, les funcionalitats de la qual són processades completament pel processador incorporat a la placa.

Els resultats de diferents proves realitzades sobre l'aplicació permeten demostrar la seva funcionalitat i validar el model d' aplicació creat. Es configura com una aplicació que permet el moviment natural de la pròtesi però que, a voluntat de l'usuari, permet el tancament de la pinça a la mida de l' objecte resultant de la classificació d'imatges.

Abstract

This report describes how a computer vision application based on the Sony Spresense development board has been developed. It takes as a point of reference the previous study carried out by the student Yeray Navarro in his Final Degree Project, *"Placa Programable Spresense de Sony. Desarrollo de una aplicación de visión por computador"* and the computer vision models defined in it.

Throughout the work, various stages are defined that encompass the development of the application, from the initial stages where the objectives to be met are set to the practical use of the application. The development of this work is aimed at improving the arm prosthesis designed and manufactured by the ARM2u team, pushed forward by the "*Escola Tècnica Superior d'Enignyeria Industrial de Barcelona*", ETSEIB, of the "*Universitat Politècnica de Catalunya*", UPC.

The process of creating the application is carried out in a segmented way, starting with a base code embedded in the image capture and classification development board, passing through a first version of the application together with the movement code of the prosthesis and ending with an optimization of it.

At the conclusion of this work, an application embedded in the Sony Spresense development board is presented, whose functionalities are completely processed by the processor incorporated in the board.

The results of different tests carried out on the application allow to demonstrate its functionality and validate the application model created. It is configured as an application that allows the natural movement of the prosthesis that, at the user's will, allows the closure of the clamp to the size of the object resulting from the classification of images.

Sumario

1. Prefacio

1.1. Origen del proyecto

La motivación principal para realizar este trabajo viene dada por el interés que tengo alrededor de la visión por computador y el *machine learning*, y cómo estos pueden servir en la mejora de productos y procesos industriales. Otro punto fundamental para el inicio de este trabajo es la voluntad de continuar con el estudio sobre visión por computador y la placa Sony Spresense realizado por el estudiante Yeray Navarro Soler en su Trabajo de Fin de Grado titulado "*Placa programable Spresense de Sony. Desarrollo de una aplicación de visión por computador*" [1] ,cuyo contenido se resume brevemente en el apartado 2.2.1.

Añadido a las motivaciones descritas anteriormente, actualmente dirijo el proyecto de ARM2u, de la Escuela Técnica Superior de Ingeniería Industrial Barcelona (ETSEIB) y participo de su departamento de Electrónica. El objetivo de dicho proyecto es el desarrollo y fabricación de una prótesis transradial con control mioeléctrico. ARM2u une a diversos estudiantes de los Grados y Masters de la ETSEIB que se configuran como integrantes del equipo y están divididos en los departamentos de Mecánica, Electrónica y Gestión, posibilitando la consecución del objetivo descrito anteriormente. En el apartado 2.2.3 se encuentra una explicación más detallada de ARM2u, y en el apartado 2.1.2, la vinculación de este trabajo con la mejora de la propuesta actual del equipo.

Reuniendo todas estas motivaciones, me puse en contacto con el profesor y tutor de este trabajo, Manuel Moreno Eguílaz, para poder asistir a la defensa del TFG de Yeray Navarro y así conocer los antecedentes y el punto de partida del que es mi Trabajo de Fin de Grado.

1.2. Antecedentes

Como se ha descrito anteriormente el presente trabajo toma como antecedentes tres puntos clave, descritos en los siguientes apartados, y que parten directamente del estudio previo en el campo de la visión por computador y la placa de desarrollo Sony Spresense realizado por Yeray Navarro.

1.2.1. Fundamentos del TFG de Yeray Navarro Soler

Partiendo de un análisis histórico y evolutivo de la visión por computador, el TFG "*Placa programable Spresense de Sony. Desarrollo de una aplicación de visión por computador*" describe las características esenciales de la misma. A lo largo del trabajo describe diferentes métodos de *machine* aplicables a la visión por computador, generando unos modelos de

clasificación de imágenes y detección de objetos. Todos los modelos son estudiados y probados con el objetivo de ser aplicados en la placa de desarrollo Sony Spresense.

1.2.2. Punto de partida del presente trabajo

En el último punto de su trabajo, Yeray Navarro propone diversas propuestas para trabajos futuros. Atendiendo a las necesidades futuras del equipo de ARM2u y el uso extendido que se hace en este sobre Arduino IDE, se toma como punto de partida para el presente trabajo la primera propuesta, titulada "*Implementación de la aplicación desarrollada*".

La base para la realización del presente trabajo se sustenta directamente en los resultados prácticos obtenidos en el trabajo realizado por Yeray Navarro. El primer resultado del trabajo fue el desarrollo y creación de una base de datos que posibilita el desarrollo del flujo completo que da pie a la visión por computador, desde el preprocesado de los datos hasta el entrenamiento y mejora del modelo. Dicha base de datos consta de 495 imágenes las cuales se reparten de forma que queda un 80% para entrenamiento y el otro 20% como test, como sugiere Edge Impulse [2]. El segundo resultado parte justamente del uso de Edge Impulse, una herramienta que permite la creación de redes neuronales, en este caso mediante *Transfer Learning*, que, recuperando la definición del anterior trabajo, consiste en utilizar un modelo de red neuronal preentrenado como base del algoritmo. El haber usado esta herramienta y haber explorado las diversas opciones para perfeccionar su uso, ha permitido tener una base sólida para la creación y mejora de los modelos aplicados en el presente trabajo. El tercer y último resultado utilizado como punto de partida del presente trabajo, es la diferenciación entre clasificación de imágenes y detección de objetos. Los resultados obtenidos a partir del estudio de estos dos tipos de algoritmo han permitido una toma de decisiones acorde con el uso que se dará a la aplicación desarrollada en el presente trabajo.

El punto final del desarrollo práctico del anterior trabajo es la aplicación de visión por computador, tanto de clasificación de imágenes como detección de objetos. Mediante el módulo de cámara de la placa Sony Spresense, se capturaban las imágenes y a través del ordenador se realizaba su procesado. El actual trabajo toma como punto de partida la necesidad de integrar todo el procesado en la propia placa de desarrollo para permitir un uso independiente de la conexión a un ordenador.

En conclusión, el extenso estudio realizado por Yeray en su TFG ha permitido sentar unas bases sólidas para el inicio del actual trabajo y el desarrollo posterior del mismo.

1.2.3. Proyecto ARM2u

El equipo de ARM2u es un equipo universitario de la ETSEIB cuyo objetivo principal es el diseño y desarrollo de una prótesis de brazo transradial controlada mediante señales

mioeléctricas manteniendo el coste presente el producirla al menor coste posible sin que vaya en detrimento de la calidad final. Dicho objetivo permite a los integrantes del proyecto aplicar de forma práctica y tangible diferentes conceptos teóricos enseñados durante el Grado (GETI) y Máster (MUEI) de la ETSEIB, desde el modelaje 3D hasta la definición e implementación del esquema electrónico de la prótesis, incluyendo también tareas de la organización y gestión de proyectos.

1.2.3.1. Equipo ARM2u

Los integrantes del equipo ARM2u se dividen en tres departamentos:

- Electrónica: La función principal de este departamento es diseñar y optimizar el control de la prótesis mediante sensores y actuadores. El control se centraliza actualmente en una placa de desarrollo Arduino UNO, a la que se conectan sensores y actuadores, como servomotores. Dicha elección de placa de desarrollo se basa fundamentalmente en la facilidad de aprendizaje del entorno Arduino IDE, a través del cual se diseña el código de control. El software de la prótesis consiste en 3 pilares básicos: la lectura y procesamiento de las señales mioeléctricas, el control de los servomotores, que permiten el cierre y la pronosupinación, y las lecturas de diversos sensores de control de estado, como temperatura, voltaje suministrado por la batería y otros. Por lo que respecta al trabajo habitual de hardware se basa en el diseño y creación de circuitos que permitan la conexión entre los sensores y servomotores, mencionados anteriormente, a la placa de desarrollo, normalmente, o a otros puntos de la prótesis.

- Mecánica: Es el departamento encargado del modelaje 3D de la prótesis usando *SolidWorks* y su posterior fabricación usando fabricación aditiva mediante la impresión 3D de PLA y PETG. Se modela la prótesis para que permita diversos movimientos al usuario y sea adaptable a este, manteniendo, en la medida de lo posible, una línea estética. Se realizan estudios sobre cómo mejorar la adaptabilidad de la prótesis, mejorando la pieza que tiene contacto directo con el usuario, así como diversos estudios de tensiones para determinar los puntos críticos de la prótesis y evitar fallos durante el uso.

- Gestión: Este departamento centra sus esfuerzos en la organización del equipo y los contactos tanto internos, con la ETSEIB, como externos, con sponsors, asociaciones y otros. Es un elemento clave en el equipo debido a que da visibilidad a este, tanto dentro como fuera de la escuela, y a su vez mantiene una organización sobre las tareas a realizar por los departamentos. Dentro de sus responsabilidades se encuentra atender a las necesidades de la prótesis y organizar las tareas para el correcto desarrollo de esta, organizando diversos eventos donde se participe como equipo para ganar visibilidad y el contacto con sponsors y diversos colaboradores.

1.2.3.2. Andrómeda

Andrómeda es la última prótesis fabricada por el equipo ARM2u. Es una prótesis completamente funcional adaptada al piloto actual del equipo, Kyle, que sufre una amputación transradial en el brazo izquierdo. La fabricación de la prótesis Andrómeda finalizó a finales de abril de 2022, pero no adopta un control mioeléctrico debido a la poca fiabilidad que ofrecían las señales mioeléctricas. Uno de los mayores impulsos dados en el último cuatrimestre ha sido dirigido a la investigación de la captura y el procesado de señales mioeléctricas, que ha empezado a dar buenos resultados a finales de 2022, posibilitando de esta manera que la futura prótesis pueda ser controlada mioeléctricamente.

Por lo que respecta a la parte Mecánica, Andrómeda se compone de tres partes:

- Socket: Es la parte más proximal de la prótesis. Tiene contacto directo con la extremidad del piloto y, por ende, es el elemento encargado de sujetar la prótesis al piloto. Debe tener una mínima movilidad para poder realizar movimientos precisos con la prótesis y a su vez, tiene que garantizar la comodidad del usuario para que la pueda usar largos periodos de tiempo.

- Prono: Es el elemento que debe contener todo el circuito electrónico, desde la batería y placa de desarrollo hasta la multitud de cables que conectan con los transductores que forman parte de la electrónica de la prótesis. Es primordial que este componente tenga una estructura rígida pero ligera para evitar fallos estructurales y a su vez garantizar que la prótesis sea cómoda en el uso habitual para el usuario. Esta parte se puede equiparar a un antebrazo.

- Gripper: Es la parte distal de la prótesis, a través de la cual se puede actuar sobre diferentes elementos del entorno. Permite una rotación, llamada pronosupinación, y la apertura y cierre. Actualmente el sistema usado es de gripper paralelo, compuesto por dos barras móviles con movimiento contrapuesto cuyo recorrido finaliza en un contacto paralelo de sus puntas. Ambos movimientos son realizados por dos servomotores independientes. Esta parte se puede equiparar a una mano.

La parte electrónica se compone de los siguientes elementos:

- Placa de desarrollo: La placa de desarrollo es una placa comercial modelo Arduino UNO. Como se ha expuesto anteriormente, la facilidad de aprendizaje y uso del entorno Arduino IDE para la programación frente a otros entornos, es uno de los factores determinantes para la elección de dicho microcontrolador. Además, facilita las conexiones con diferentes elementos, sobre todo en el entorno de pruebas y dispone de pines con asignaciones variadas (GPIO, I2C, etc.) que permite prescindir

de controladores externos y, por lo tanto, centralizar el control.

- Batería: La alimentación de la placa de desarrollo se realiza a través de una batería de litio que suministra un máximo de 8,4 V.

- Pulsadores: Es el sustituto provisional a las señales EMG. Gracias a que nuestro piloto actual puede realizar cierto movimiento con el miembro remanente. En el socket se han dispuesto dos pulsadores con los que se controlan los dos movimientos de la prótesis, el de pronosupinación y el de apertura y cierre.

- Servomotores: Andrómeda dispone de dos servomotores de posición con una rotación máxima de 270º, uno dedicado al movimiento de pronosupinación y otro dedicado a la apertura y cierre del gripper. Ambos se controlan mediante señales PWM y son alimentados por una tensión de 5 V.

- Sensores: Los sensores incorporados actualmente, a falta de incluir los sensores electromiográficos, permiten tomar medidas sobre el entorno de la prótesis como es la temperatura del socket y la batería y el voltaje suministrado por esta. Los datos recibidos por dichos sensores determinan el estado de la prótesis que puede encontrarse en estado de trabajo, de alerta y crítico. El paso de un estado a otro se controla mediante una máquina de estados. Requieren de pines analógicos y digitales.

- Interruptores: La prótesis incluye dos interruptores, que controlan el encendido y apagado de la prótesis y la activación de un estado de seguridad, mediante el cual la prótesis abre el gripper y se sitúa en la posición de inicio.

- Pantalla LCD: Situada en la parte superior de la prono, muestra información sobre el estado de la batería y el estado en el que se encuentra la prótesis (de los tres estados posibles descritos anteriormente).

1.2.3.3. Cybathlon

A pesar de que el objetivo del equipo es avanzar independientemente, la participación en competiciones, concursos y otros eventos favorece la visibilidad y el test del funcionamiento de la prótesis. Uno de estos eventos es la competición denominada Cybathlon [3], organizada por la universidad suiza ETH, donde se ponen a prueba diversos dispositivos biomédicos mediante pruebas basadas en tareas cotidianas, como subir y bajar escaleras, tender la ropa o recoger objetos. Dicha competición se realiza de forma cuatrienal, pero cada año se realizan competiciones de menor participación llamadas *Challenges*. La variabilidad de dispositivos conforma las diferentes pruebas que configuran la competición de Cybathlon, como, entre otras, la *Leg Prosthesis Race*, donde se ponen a prueba prótesis para pierna, la *Wheelchair Race*, donde se ponen a prueba sillas de ruedas o la *Arm Prosthesis Race*, categoría donde

participa el equipo de ARM2u y donde se ponen a prueba las prótesis de brazo.

La *Arm Prosthesis Race* está compuesta por varias pruebas en la edición cuatrienal, pero en las competiciones *Challenges*, se compone de una o dos pruebas pertenecientes al total de pruebas. La próxima competición es entre el 28 y 30 de marzo de 2023 y los esfuerzos del equipo están destinados al completo a poder participar. Esta próxima competición consta de dos pruebas

- *Clean Sweep*: Es una tarea que ya se realizó en mayo de 2022. El desempeño en esta tarea consta de trasladar varios objetos cotidianos (por ejemplo, una tarjeta de crédito, un USB o un bolígrafo) a unas cajas de madera con diferentes ranuras donde depositar los objetos, cada uno en la caja correspondiente. Pone a prueba la habilidad del piloto y la precisión de la prótesis para objetos pequeños. En 2023 los objetos son una llave, un lego, una tarjeta de crédito y una canica.

- *Bottles*: Consiste en colocar botellas con diferente peso en una cesta para, posteriormente, trasladarla a una mesa donde se deben descargar las botellas. Pone a prueba la resistencia de la prótesis a tensiones de tracción y la fuerza del agarre.

1.3. Requerimientos previos

1.3.1. Sony Spresense

El estudio y desarrollo de la aplicación de visión por computador que se describe en este trabajo, se basa en el uso de la placa de desarrollo Sony Spresense [4]. En el anterior trabajo fue usada esta placa de desarrollo y la mayor parte de la investigación realizada depende de la potencia de procesamiento y memoria de dicha placa. A continuación, se exponen las principales características de la Sony Spresense, las cuales fueron comparadas con diferentes placas de desarrollo en el trabajo de Yeray Navarro. El uso de la placa Sony Spresense a lo largo de este trabajo ha sido realizado añadiendo la placa de extensión "*Spresense Extension Board*" y la cámara *"Spresense Camera Board"*. A pesar de que todas las funcionalidades descritas en este trabajo pueden realizarse sin la placa de extensión, esta facilita las conexiones y la alimentación de actuadores, como los servomotores, y sensores. Durante el desarrollo de la aplicación que engloba este trabajo se ha usado la placa de extensión, siempre teniendo en cuenta que todas las funcionalidades son replicables usando únicamente la placa principal Sony Spresense y el módulo de la cámara, con la finalidad de poderla instalarla sin dificultades en la prótesis actual de ARM2u.

1.3.1.1. Dimensiones

Sony Spresense tiene unas dimensiones de 50.0 mm de largo por 20.6 mm de ancho, muy parecidas a la placa de desarrollo Arduino Uno que se encuentra instalada en la prótesis de ARM2u.

1.3.1.2. Pines

La placa de desarrollo dispone de 26 pines digitales E/S, divididos entre GPIO, SPI, I2C UART y I2S. En el apartado 3.2.3 se especifican los pines de uso habitual para la prótesis de ARM2u

1.3.1.3. Memoria

Dispone de 8 MB de memoria Flash y 1,5 MB de memoria SRAM

1.3.1.4. Voltaje de salida

La placa principal Sony Spresense ofrece 1,8 V de salida, mientras que la placa de extensión *"Spresense Extension Board"* ofrece 5 V. La mayor parte de componentes de la prótesis operan a 5V, como se describe en el apartado 3.2.3, ocasionando que el voltaje de salida de la placa principal sea insuficiente para el funcionamiento básico de la prótesis. A pesar de ello existen diversas alternativas, desde alimentar los componentes externamente, por ejemplo, con una batería y un regulador de tensión.

1.3.1.5. Núcleos

Dispone de 6 núcleos ARM Cortex-M4F

A continuación, se incluyen tres tablas resumen, *Tabla 1, Tabla 2 y Tabla 3*, detallando diversas características de la placa principal Sony Spresense, de la placa de extensión *"Spresense Extension Board"* y del módulo de la cámara *"Spresense Camera Board"* [5]:

Nombre del modelo	CXD5602PWBMAIN1
Dimensiones	50,0 mm x 20,6 mm
Procesador	ARM® Cortex®-M4F x 6 núcleos
Frecuencia de reloj máxima	156 MHz

SRAM	1.5 MB
Memoria Flash	8 MB
Entrada/Salida digital	GPIO, SPI, I2C, UART, I2S
Entrada analógica	2 ch (rango 0.7 V)
GNSS	GPS(L1 C/A), GLONASS(L1OF), BeiDou(B1), Galileo(E1 CBOC), QZSS(L1 C/A, L1 S), SBAS(L1 C/A)
Entrada de cámara	Interfaz paralela dedicada

Tabla 1: *Tabla resumen de la placa de desarrollo Sony Spresense. Fuente: Modificada de Sony Spresense*

Nombre del modelo	CXD5602PWBEXT1
Dimensiones	68,6 mm x 53,3 mm
Entrada/Salida de audio	4 canales de entrada de micrófono analógicos o 8 canales de entrada de micrófono digitales, salida a auriculares
Entrada/Salida digital	3,3 V o 5 V E/S digital
Entrada analógica	6 ch (rango 5.0 V)
Interfaz de memoria externa	Ranura tarjeta microSD

Tabla 2:*Tabla resumen de la placa de extensión "Spresense Extension Board". Fuente: Modificada de Sony Spresense*

Nombre del modelo	CXD5602PWBCAM1
Dimensiones	24,0 mm x 25,0 mm
Número de píxeles efectivos	2608 (H) x 1960 (V) = aprox. 5,11 M

	píxeles
Voltaje operacional	DC 3,7 V
Voltaje E/S	DC 1,8 V
Interfaz de cámara	CMOS 8 bit paralelo
Formato de salida	Y/C, RGB y JPEG
Interfaz de control	I2C
Filtro incorporado	IR cut filter
FOV	78° ± 3°
Profundidad de campo	77,5 cm ∼ ∞
Valor F	2,0 ± 5%
Foco	Foco fijo

Tabla 3: *Tabla resumen del módulo de la cámara "Spresense Camera Board". Fuente: Modificada de Sony Spresense.*

2. Introducción

2.1. Objetivos del proyecto

2.1.1. Desarrollo de una aplicación basada en Sony Spresense

Este proyecto nace, como se ha descrito anteriormente, de un trabajo de investigación y desarrollo previo en el campo de la visión por computador y usando como soporte de desarrollo la placa Sony Spresense.

Tal y como se ha expuesto en el apartado 3.2.2 el desarrollo anterior de la aplicación basa todo su procesamiento en el ordenador, al cual se conectaba la placa de desarrollo, marcando el punto final del anterior trabajo y dando paso al inicio del presente. Dado el punto de partida del actual trabajo, el principal objetivo de este es el desarrollo de una aplicación de visión por computador integrada en la placa Sony Spresense. Dicha aplicación debe permitir, como mínimo, la toma de imágenes, su procesado y el cálculo a través de los algoritmos diseñados de forma independiente, sin necesidad de realizar el procesado a través de un ordenador.

Habiendo definido el objetivo principal del proyecto y las condiciones para su cumplimiento, cabe remarcar que, como en todos los desarrollos de código, hay un objetivo subyacente al principal, y es la optimización del propio código. Tal optimización se realiza en diversos estados del desarrollo del código, pero fundamentalmente se realiza una vez el código es funcional y está en fase de prueba. De esta manera, el desarrollo de la aplicación, que configura el objetivo principal del proyecto, comporta también la optimización del código que la define.

2.1.2. Apoyo al desarrollo de ARM2u

Junto al objetivo principal del proyecto nace otro que transcurre en paralelo, y es el estudio de la integración de la placa de desarrollo en la prótesis desarrollada por ARM2u. Este objetivo depende directamente del primero y, a pesar de discurrir paralelamente, si el primer objetivo no se consigue, este segundo tampoco lo hará.

Los beneficios que puede aportar la consecución de ambos objetivos para el equipo de ARM2u son:

- Facilitar el control de la prótesis. El control de apertura y cierre de la prótesis, para ciertas pruebas, ha de ser muy preciso debido a que se requiere alcanzar, transportar y depositar objetos de pequeño tamaño. A pesar de poder incluir diferentes sensores

(p.ej. sensores de presión) para poder crear un control de lazo cerrado junto con la propia habilidad del piloto para ajustar el grado de apertura al objeto, una apertura automatizada para objetos de tamaño conocido permite mayor velocidad y precisión.

- Prescindir de tecnología háptica. De entre las pruebas que actualmente están consideradas para la edición de Cybathlon 2024, existe una prueba que consiste en que el piloto introduzca la prótesis en una caja opaca y con una sola apertura, a través de la cual no se puede ver nada, y clasificar el objeto según su forma y textura. Uno de los métodos más fiables para realizar dicha prueba con acierto, es la incorporación de tecnología háptica, que permite recrear el sentido del tacto mediante la aplicación de fuerzas, vibraciones y movimientos. Esta tecnología es de difícil aplicación para el desarrollo de un equipo como ARM2u, el cual no está dedicado exclusivamente a la investigación. Gracias a la aplicación de visión por computador que integraría la placa Sony Spresense, sería posible clasificar objetos por su forma y textura, a través de un entrenamiento del algoritmo específico.

3. Propuesta de solución

3.1. Elección del entorno de desarrollo: Arduino IDE

Como se ha mencionado en el apartado 2.2.2, el desarrollo de la aplicación de visión por computador que se describe en el presente trabajo, se basa en el entorno de Arduino IDE. Desde la página oficial de Sony Spresense, se presentan tres posibilidades para el desarrollo de aplicaciones [6]:

- Spresense SDK (*Software Development Kit*): Es el entorno de desarrollo original de la placa Sony Spresense y se basa en NuttX[1].

- Spresense CircuitPython: CircuitPython es un lenguaje de programación basado en Python con un acercamiento para el desarrollador más simple y fácil de aprender que el lenguaje original. Contiene librerías y drivers para el hardware y los sensores de la placa Sony Spresense.

- Spresense Arduino: Permite el desarrollo de aplicaciones en el entorno Arduino IDE. Este entorno facilita la programación y desarrollo de aplicaciones mediante diversas librerías predefinidas

Para la selección del entorno donde se desarrolla la aplicación del presente trabajo se han tenido en cuenta tres factores:

- Tiempo de aprendizaje: Es el factor más crítico en el desarrollo de cualquier Trabajo de Fin de Grado, el cual se debe completar en un periodo de tiempo de 6 meses aproximadamente. Algunos trabajos, como el presente, incluyen un componente de desarrollo, el cual debe completarse en un tiempo de 4 meses, para poder disponer del tiempo suficiente para el redactado de la memoria. Dado este estrecho intervalo de tiempo, se priorizan aquellos entornos en los que la curva de aprendizaje es exponencial, es decir, se consigue aprender mucho en poco tiempo. Es un factor subjetivo y depende directamente del trabajo previo en el entorno.

- Uso de librerías comunes: En el desarrollo de una aplicación, el uso de librerías predefinidas es habitual, debido a que facilitan tareas y simplifican el código. Para el desarrollo de la aplicación definida en este proyecto es de utilidad que se disponga de

[1] NuttX es un sistema operativo en tiempo real (RTOS) con énfasis en el cumplimiento de los estándares y el tamaño reducido.[8]

librerías que permitan captura de imágenes y control de servomotores.

- Dificultad en el despliegue y test: Para verificar que el proceso de desarrollo de la aplicación es acorde a lo esperado, es necesario desplegarla en la placa de desarrollo. Al desplegarla es fácil hacer test mediante actuadores, como servos, o dispositivos de señales visuales, como LEDs, e identificar diferentes errores en el código. El método de compilación de código y despliegue varía según el entorno elegido, pudiendo ser desde un botón con ambas funcionalidades hasta tener que diseñar un código específico (*script*) para poder desplegarla.

Teniendo en cuenta estos tres factores, se ha elaborado una tabla, Tabla 4, para agrupar las diferentes posibilidades de desarrollo junto a sus características.

	Tiempo de aprendizaje[2] (semanas)	Librerías comunes[3]	Dificultad de despliegue
Spresense Arduino	2	Camera, Servo, MultiCore (librería multinúcleo)	Muy baja
Spresense SDK	6-8	Clases varias sin agrupación en librerías comunes	Elevada
Spresense CircuitPython	4	Camera, Servo	Muy baja

Tabla 4: *Características de las posibilidades de desarrollo. Fuente: Propia*

Empezando por el análisis del tiempo de aprendizaje, el aprendizaje del SDK de Spresense es demasiado extenso para la duración global del proyecto. Si bien es cierto que un SDK ofrece mejores posibilidades y una solución más adaptada a los objetivos de un proyecto, el tiempo de aprendizaje va en detrimento directo del tiempo de desarrollo de la aplicación funcional. Dado que el tiempo es el factor más crítico en la elaboración de este trabajo, esta opción queda descartada, a la vista que las otras dos opciones ofrecen un tiempo de

[2] El cálculo del tiempo de aprendizaje se basa en una estimación en base a la experiencia previa con el lenguaje de programación y el uso del entorno.

[3] Se especifican exclusivamente aquellas que son imprescindibles para el desarrollo de la aplicación

aprendizaje mucho menor. El desarrollo en CircuitPython es relativamente sencillo, debido a que el lenguaje en sí ofrece muchas facilidades al desarrollador. Aun así, el entorno Arduino ofrece una simplicidad de programación equiparable a CircuitPython y, a parte, se suma la experiencia en diversos proyectos que he realizado basados en este entorno, entre ellos los que se engloban en el proyecto de ARM2u. En conclusión, la diferencia de tiempo de aprendizaje en ambos entornos viene dada por la experiencia previa y no por la dificultad de los entornos en sí. Dicho esto, no es factible descartar la opción de CircuitPython debido a que el tiempo de aprendizaje no es tan extenso como el del SDK y no perjudica tanto el desarrollo posterior de la aplicación.

Teniendo en cuenta que las dos opciones restantes ofrecen un despliegue, que también incluye la compilación del código, realmente sencilla (Arduino IDE dispone de un botón que permite compilar y descargar el código en la placa y CircuitPython se descarga a medida que se cambia el código [7]), el único factor determinante es el uso de librerías comunes. Estas librerías, como se ha mencionado anteriormente, permiten agrupar diferentes funciones y clases, facilitando el escrito del código. Arduino IDE dispone de una gran variedad de librerías propias, como *Servo*, que permite el control de servomotores, y de librerías adaptadas entre las cuales están aquellas adaptadas a la placa de Sony Spresense, como *Camera* o *MultiCore MP*, que permiten la captura de imágenes y el uso de diferentes núcleos del procesador, respectivamente. CircuitPython también ofrece módulos para el uso de la cámara o de servomotores, pero es más limitante en cuanto al uso del multinúcleo. Tomando estas consideraciones, es factible decir que Arduino tiene cierta ventaja en su uso frente a CircuitPython. Añadido a los factores estudiados, la comunidad entorno a Arduino es mucho mayor a la que rodea el entorno Python para este tipo de desarrollos, pudiendo encontrar más referencias o soluciones a diferentes problemáticas que surjan durante el desarrollo de la aplicación.

Cabe destacar que no se ha tenido en cuenta la posibilidad de despliegue de las librerías generadas por Edge Impulse debido a que ofrece la generación de librerías para los tres entornos, igualando, en este aspecto, a las tres opciones de desarrollo.

En conclusión, y teniendo todos los factores en cuenta como se ha descrito anteriormente, se ha elegido el entorno Arduino IDE debido al poco tiempo de aprendizaje y la facilidad que ofrece el uso de sus bibliotecas durante la elaboración del código, frente al resto de opciones.

3.1.1. Edge Impulse y Arduino IDE

Como se ha mencionado anteriormente, Edge Impulse ofrece la posibilidad de desplegar sus algoritmos entrenados. El despliegue de dichos algoritmos se realiza a través de librerías para entornos específicos, como Arduino IDE, así como librerías genéricas en un lenguaje específico como C++ o Python, entre otras. También incluye opciones de despliegue para

ciertas placas como la Sony Spresense en forma de binarios que se pueden ejecutar directamente.

La estructura de la biblioteca desplegada para Arduino IDE es la siguiente:

- **Nombre de la librería**
    - **examples:** Se encuentran ejemplos específicos para algunas placas de desarrollo, entre las cuales no se encuentra la Sony Spresense, para facilitar la ejecución y testeo de la librería. También se encuentra un ejemplo genérico aplicable a cualquier placa.
    - **src**
        - edge-impulse-sdk: Contiene los archivos cabecera con extensión .h con todas las funciones de la librería.
        - model-parameters: Contiene los archivos cabecera con extensión .h con la información básica del modelo, por ejemplo, el tamaño de la imagen a procesar, el tipo de procesado o bien los nombres de los objetos definidos en el modelo.
        - tflite-model: Contiene los archivos cabecera con extensión .h que definen el modelo y llaman a funciones incluidas dentro de la carpeta edge-impulse-sdk.

Dentro de la organización por carpetas fuente *src*, los archivos de una carpeta llaman a las funciones situadas en otra y estas a otras, creando así una ejecución en cascada de funciones que permiten el uso de esta librería. Dado que el seguimiento de las llamadas entre funciones y, consecuentemente, entre carpetas, es complejo y difícil de monitorizar, la edición de los parámetros de forma manual para lograr cambios en los resultados en la ejecución de la librería, resulta en una tarea laboriosa y poco recompensante. Sirva esta conclusión como antecedente a los errores de la librería encontrados durante el desarrollo de la aplicación.

3.2. Clasificación de imágenes frente a detección de objetos

En el apartado 7.1 del trabajo realizado por Yeray Navarro [1], se exponen las principales características de la clasificación de imágenes y la detección de objetos, algoritmos englobados dentro del tratamiento de imágenes en visión por computador. A continuación, se resumen dichas características:

- **Detección de objetos:** Identifica objetos y su posición en una imagen, los cuales se pueden marcar encuadrándolos con un rectángulo (*bounding box*). Dependiendo del software y la capacidad de procesado del microcontrolador, es posible calcular su posición respecto al entorno (en un plano 2D o también tomando en cuenta la profundidad). Las aplicaciones más comunes para este tipo de algoritmo se basan en imágenes continuas (vídeo) y Edge Impulse compila este tipo de algoritmos para que puedan ser ejecutados solamente a través de una secuencia de imágenes continua.

- **Clasificación de imágenes:** Cataloga las imágenes de entrada en diferentes clases, definidas previamente por el desarrollador. El resultado final que facilita el algoritmo es una asignación de probabilidades (que en total suman el cien por cien) a las clases definidas en función de su parecido con la imagen a clasificar. Esta clasificación generalmente se da a través de imágenes no secuenciales, con frecuencias de entrada variables, no definidas. Puede utilizarse en ambientes con alta frecuencia de cambio, como en el análisis de imperfecciones de productos en una cinta transportadora, o bien en ambientes con muy baja frecuencia de cambio, como la detección del nivel de agua en un depósito para alertar de una posible fuga.

Durante la primera parte del desarrollo de la aplicación que se define en este trabajo, fue imprescindible decidir qué tipo de algoritmo se ajustaba más a las necesidades descritas en el apartado 3.1. Dadas las circunstancias que engloban a la competición de Cybathlon y sus requerimientos, es factible pensar que la clasificación de imágenes es el mejor algoritmo para aplicar a la prótesis. Por ejemplo, para la prueba de *Clean Sweep* lo que se ajusta mejor es el algoritmo de clasificación de imágenes para poder definir cuál es el objeto se va a recoger y, en base a unos parámetros establecidos anteriormente, modificar el grado de apertura de la pinza para ajustarlo al tamaño del objeto, logrando rapidez y precisión. Para otras pruebas, por ejemplo, aquella descrita en el apartado 3.1.2, la clasificación de imágenes se adecúa más debido a que no es necesario distinguir entre varios objetos, solamente se debe clasificar un único objeto.

La detección de objetos no permite una selección voluntaria del objeto una vez desplegada la aplicación en la placa de desarrollo, solamente se podría priorizar un objeto ante los demás, mediante un código que fuerce la decisión. Esta problemática causaría que, en cuanto el algoritmo detectara diversos objetos, la aplicación no sería capaz de modular el grado de apertura de la pinza al tamaño de uno en específico, quitando funcionalidades y autonomía al usuario de la prótesis.

Por los motivos expuestos anteriormente, añadido a la intención de ayudar al desarrollo de la prótesis de ARM2u, como se ha descrito en el apartado 3.1, se ha tomado la decisión de

basar la aplicación de visión por computador, desarrollada en este trabajo, en un algoritmo de clasificación de imágenes.

3.3. Desarrollo Multicore

Atendiendo al objetivo subyacente al principal, definido en el tercer párrafo del apartado 3.1.1, la propuesta para la optimización de la aplicación es usar diferentes núcleos del procesador para realizar tareas en paralelo. Como se ha mencionado en el punto 2.3.1.5, Sony Spresense dispone de seis núcleos en su procesador, distribuidos entre un núcleo principal, llamado *MainCore,* y cinco núcleo secundarios, llamados *SubCores*. La realización de tareas de forma simultánea, aprovechando diferentes núcleos, conlleva una optimización de la aplicación ya que permite que esta tenga diversas funcionalidades sin que se solapen, por ejemplo, el uso habitual de la prótesis y la captura y procesado de imágenes de objetos reconocidos por el algoritmo. A pesar de las ventajas que dicha configuración ofrece, existen diversas limitaciones en su aplicación práctica. Las dos limitaciones que tienen más impacto en el desarrollo del código son: limitación de la memoria en los *SubCores* y limitación en el uso de bibliotecas predefinidas en los *SubCores*.

3.3.1. Limitación en la memoria

Tal y como se describe en la documentación técnica de la librería *Multicore MP [9]*, la memoria SRAM de Sony Spresense está distribuida en 12 celdas de 128 kB cada una, y queda repartida entre los diferentes núcleos. Los primeros 768 kB (6 celdas) son asignados al *MainCore*, los siguientes 256 kB (2 celdas) son asignados al *Subcore1* y los restantes 532 kB son asignados en celdas unitarias (4 celdas en total) de 128 kB a los núcleos *Subcore2*, *Subcore3*, *Subcore4* y *Subcore5*. Esta distribución queda ejemplificada en la Fig. 1.

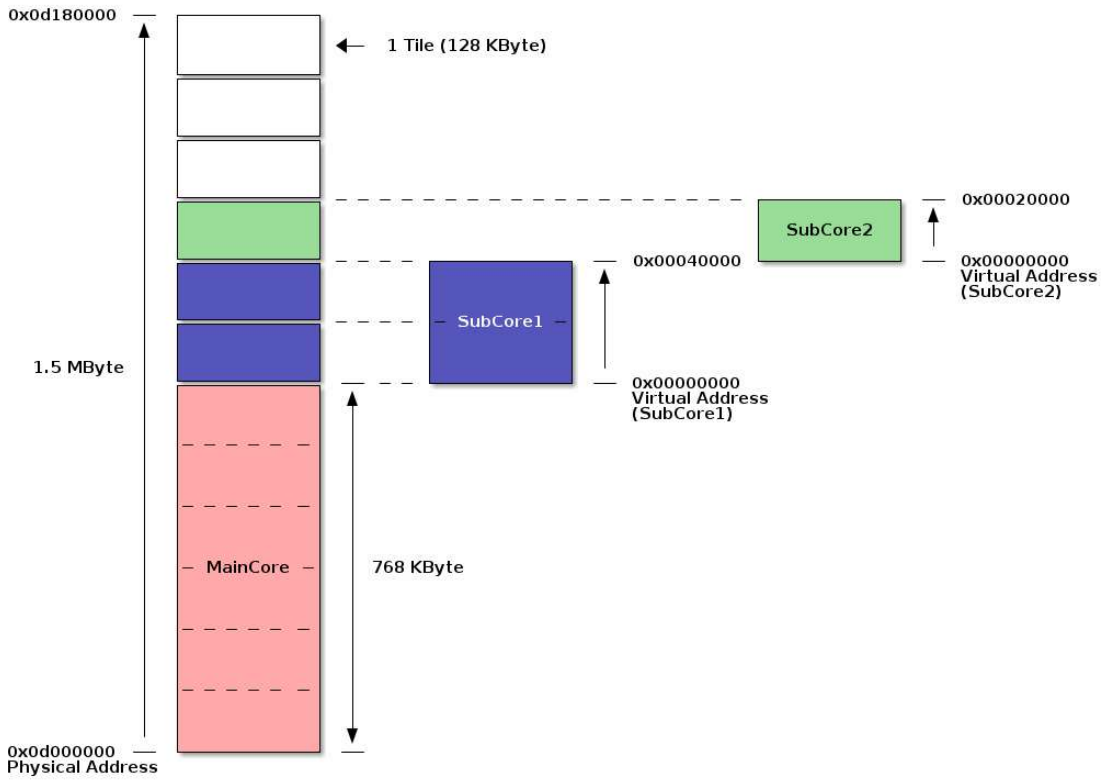

Figura 1: *Esquema de la distribución de la memoria según los núcleos. Fuente: Sony Spresense*

La memoria asignada a los diferentes núcleos puede variar, por ejemplo, si el código en su totalidad se aloja en el *MainCore*, y ocupa 1280 kB, automáticamente se hace una redistribución de la memoria entre los demás núcleos para cubrir esta necesidad. La contrapartida de esta redistribución es que, al haber reducido memoria de los demás núcleos para ajustar la memoria del *MainCore*, habrá núcleos que no se podrán utilizar por falta de memoria.

Dadas las características citadas anteriormente, la distribución de funciones que configuran el código de la aplicación, a lo largo de los diferentes núcleos debe ser estudiada cautelosamente. A fin de evitar errores derivados de la falta de memoria en los núcleos, se debe valorar el peso de las librerías importadas en cada núcleo, dado que son los componentes más pesados en cuanto a memoria se refiere.

3.3.2. Limitación en el uso de bibliotecas

En la misma documentación técnica donde se describen las características de memoria expuestas en el anterior apartado, se delimitan qué librerías son soportadas por el *MainCore* y cuáles lo son por los *SubCore*. En la Tabla 5 se muestra el resumen.

Librería	MainCore	SubCore	Descripción
MP	○	○	
RTC	○	○	La función de Alarma no está disponible en el SubCore
SPI	○	○	
Servo	○	○	
SoftwareSerial	○	○	
Watchdog	○	○	
Wire	○	○	
Audio	○	-	
Camera	○	-	
DNNRT	○	-	
GNSS	○	-	
LowPower	○	-	
LTE	○	-	
Sensing	○	-	
Storage (EEPROM, eMMC, File, Flash, SDHCI)	○	-	

Tabla 5: *Resumen del soporte de librerías en el núcleo principal y secundarios. Fuente: Modificada de Sony Spresense*

Como se puede observar, más de la mitad de librerías no pueden ser usadas por los *SubCore*. De entre estas librerías, la que más relevancia tiene para el actual proyecto, es la librería *Camera*. Teniendo en cuenta esta limitación, la captura de imágenes y demás funciones que incorpore dicha librería, solo se podrán llevar a cabo en el núcleo principal de la placa de desarrollo. Por consiguiente, el procesado de imágenes, es decir el algoritmo diseñado en Edge Impulse, debe realizarse en el mismo núcleo principal, o bien, en un núcleo secundario, habilitando la comunicación entre núcleos. Hasta tener más información sobre los recursos de memoria que precisa el algoritmo para su correcto funcionamiento, no se puede tomar una decisión sobre cómo quedarán distribuidas las librerías y funciones en diferentes núcleos.

4. Desarrollo de la propuesta

En el presente apartado se describe el desarrollo de la aplicación de visión por computador basada en Sony Spresense, lo cual completa el objetivo principal de este trabajo. Recogiendo las conclusiones extraídas de los apartados anteriores, el objetivo es desplegar la aplicación en la placa de desarrollo, de forma que el procesado de imágenes, su clasificación y las acciones derivadas de ella, se hagan, en su totalidad, en la placa Sony Spresense. Los algoritmos serán creados y compilados en librerías a través de Edge Impulse, aprovechando el desarrollo hecho por Yeray Navarro en su Trabajo de Fin de Grado. Dichas librerías serán añadidas al entorno Arduino IDE, posibilitando la creación de la aplicación en su totalidad en el mismo entorno.

4.1. Desarrollo del código de clasificación de imágenes

Edge Impulse ofrece un amplio espectro de posibilidades en cuanto a modelos de *Machine Learning* (ML) se refiere. En el trabajo realizado anteriormente por Yeray Navarro, se definió que el modelo más ajustado a las necesidades del proyecto se consigue a través de *Transfer Learning*. Tomando este estudio como referencia, el estadio posterior a la creación del modelo es el punto de partida para la creación de la aplicación. Dicho punto de partida permite destinar el tiempo al desarrollo básico de la aplicación para, una vez desarrollada y testeada, cambiar parámetros del modelo con la certeza que esta es funcional.

Para el desarrollo del código de clasificación de imágenes son necesarios dos códigos principalmente, el de toma de imágenes y el de procesado y clasificación de las mismas. En este apartado se definen ambos códigos, por separado, y como se deben unir para obtener el resultado deseado.

4.1.1. Características teóricas del modelo preliminar

El modelo mencionado anteriormente es capaz de clasificar imágenes en 9 parámetros de salida distintos, asociados a los 8 objetos de Cybathlon y la mesa sin objetos. Para que el algoritmo sea capaz de determinar si la imagen clasificada corresponde a un objeto o no, siempre se debe añadir un parámetro de salida neutro, en este caso la mesa. De otro modo, en cuanto el algoritmo tratara de clasificar una imagen con un parámetro no estudiado, por ejemplo, un vaso, lo asignaría por similitud a uno de los parámetros establecidos como respuesta, por ejemplo, asignar el vaso a un boli. Es por esto que el parámetro neutro siempre ha de estar presente, como alternativa a la clasificación de una imagen desconocida.

Para crear este modelo se ha usado *Transfer Learning*, como se ha mencionado anteriormente, el modelo MobileNetV2 96x96 0.35. Todos los modelos oficiales de Edge

Impulse de *Transfer Learning* para visión por computador son modelos MobileNet. Este tipo de modelos están desarrollados para ser soportados por aplicaciones embebidas, ya que se basan en la creación de redes neuronales profundas de poco peso [10].

El resumen de características del modelo es el siguiente:

- Parámetros de entrada:
  - Imágenes de 96 píxeles de ancho por 96 píxeles de alto
- Parámetros de salida:
  - 8 objetos de la prueba de Cybathlon: *moneda*, *tarjeta de crédito, llave, llavero, pieza de lego, canica, bolígrafo* y *usb*.
  - 1 objeto neutro, *mesa*, que se asocia a un clasificación de imagen sin objeto asociado.
- Modelo de transfer learning:
  - MobileNetV2 0.35
- Recursos de memoria mínimos requeridos:
  - 350,9 kB de RAM
  - 702,7 kB de memoria Flash
- Tiempo estimado de clasificación (latencia):
  - 1328 ms

4.1.2. Creación de librerías en Edge Impulse

Una vez definido y entrenado el modelo, Edge Impulse permite desplegar dicho modelo en diferentes formatos, siendo uno de ellos una librería para Arduino IDE, como se ha descrito en el apartado 4.1.1. En el apartado 6.2 del TFG de Yeray Navarro, se define la naturaleza de las redes neuronales, indicando que están constituidas por nodos y conexiones entre ellos. Dichas conexiones tienen asociados unos pesos que representan la importancia de las conexiones en el modelo y sus valores son variables continuas que pueden encontrarse en diferentes rangos en función del modelo, incluyendo valores negativos. En general, en los modelos basados en redes neuronales (desde ML a *Deep Learning*), los pesos son definidos por el formato numérico FP32 o float32, es decir, en una representación de coma flotante (*floating point*) de 32 bits según el estándar IEEE-754 [11]. Este formato permite abarcar un gran rango de números pudiendo, de esta manera, lograr una precisión excepcional. La contrapartida de este modelo es que, al componerse de 32 bits y abarcar un rango tan grande de valores, requiere mucha memoria, la cual es un recurso extremadamente importante en aplicaciones embebidas en placas de desarrollo. Por ello, se han desarrollado diferentes modelos de representación de valores y Edge Impulse ofrece uno de ellos, el modelo *quantized* en int8. Dicho modelo permite la equiparación de valores continuos a valores enteros discretos en un rango definido por el propio int8, de -128 a 127, aunque, dependiendo

del modelo, puede haber ligeras modificaciones. La ventaja principal de dicho modelo, junto con otros modelos de optimización, es, principalmente, la drástica reducción de uso de la memoria, en el caso de int8 de hasta un 75%[12]. La aproximación matemática de asociar valores continuos a un rango de valores discretos deriva, en mayor o menor medida según el modelo, en una pérdida de precisión. Para minimizar dicha pérdida, se reajustan los modelos int8 derivados de un FP32 mediante un factor de escala calculado por capa y por tensor[13]. Dicho modelo permite convertir el rango de valores continuos en discretos mediante una formula genérica que iguala el valor real a un valor entero (el cual se almacena en una palabra de 8 bits) multiplicado por un factor de escala. Edge Impulse usa TensorFlow para la creación de los diferentes modelos. En la página web de TensorFlow [14][13], se muestra la especificación de su cuantificación de 8 bits, resultando en la fórmula siguiente:

$$valor\ real = (valor\ entero\ (int8) - punto\ cero) \cdot factor\ de\ escala$$

En el caso particular de TensorFlow, se puede observar que modifican la fórmula incluyendo un punto cero. El rango de valores enteros es de -128 a 127, ambos incluidos.

A modo de resumen, Edge Impulse ofrece las siguientes dos opciones:

- *Unoptimized* (float32): Es la opción que usa más recursos en cuanto a memoria se refiere.
- *Quantized* (int8): Reduce el uso de memoria un 75% respecto la opción anterior. Dependiendo del modelo, la precisión se ve reducida notablemente.

A parte de los dos modelos de optimización, Edge Impulse ofrece un compilador llamado *EON Compiler* que permite optimizar los modelos, reduciendo la memoria hasta un 50% y, a priori, sin perder precisión. A fin de poder decidir qué modelo de optimización es el más adecuado para el proyecto, Edge Impulse ofrece una opción que permite visualizar la matriz de confusión teórica para ambos modelos.

En este punto inicial del desarrollo de la aplicación no se tuvo en cuenta el desarrollo en *MultiCore* como se describe en el punto 4.3, ya que esta parte de desarrollo forma parte de los estadios finales del desarrollo de la aplicación, una vez se ha comprobado que el objetivo principal se ha cumplido. Por lo que refiere a los recursos de memoria usados, solo se tuvo en cuenta que el uso de la librería dejara la suficiente memoria para poder usar el módulo de la cámara y el código de movimiento de la prótesis. No se valoró el dejar recursos de memoria para los núcleos secundarios ya que el procesado completo, de momento, se llevaría a cabo en un solo núcleo. Habiendo valorado esta consideración, se decidió usar aquel modelo que mejor precisión ofrecía, obteniendo la matriz de confusión expuesta en la Fig.2

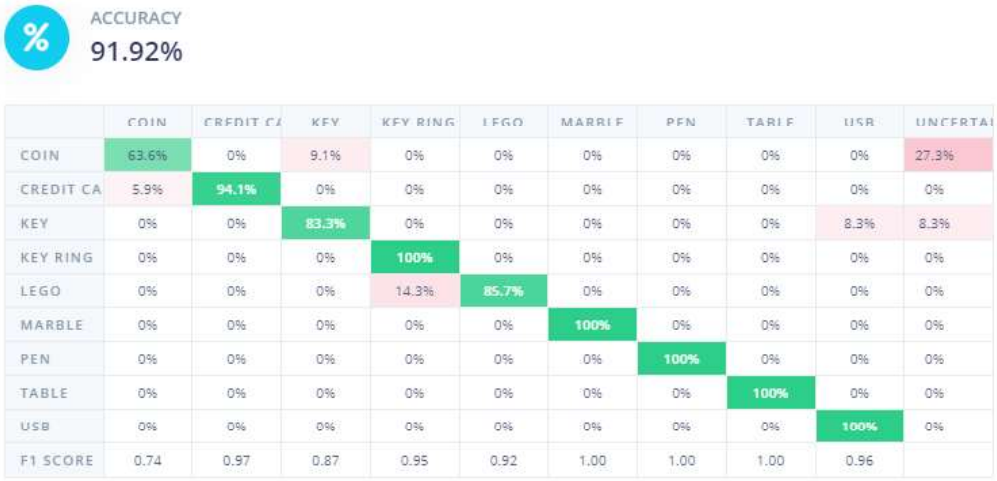

Figura 2: *Matriz de confusión del primer modelo usado. Fuente: Edge Impulse*

Como se puede observar en la Fig. 2, el modelo tiene una alta precisión para todos los objetos. Este modelo comprende los 9 objetos definidos por Yeray Navarro en su trabajo, y que corresponden a los objetos usados para la competición de Cybathlon del año 2022. Para el año 2023 se usarán solamente 4 de ellos, como se ha definido en el punto 2.2.3.3. A pesar que hay diferencia en cuanto a los elementos a clasificar mediante las imágenes, se estima conveniente usar un modelo que funciona y no hace falta reentrenar para asegurar que, en caso de error, el fallo no es del modelo y apuntar a un mal desarrollo de la aplicación.

Una vez elegido el modelo para el desarrollo inicial de la aplicación, se realiza la instalación en formato de librería Arduino, configurando la optimización en *Quantized (int8)* y con el compilador *EON Compiler* desactivado (de nuevo, teniendo en cuenta que los parámetros de uso de memoria no son críticos en este punto del desarrollo). Requiere de 350,9 kB de memoria RAM, 702,7 kB de memoria flash y un tiempo de clasificación teórico de 1328 ms. La Fig. 3 detalla las características para las diferentes optimizaciones disponibles.

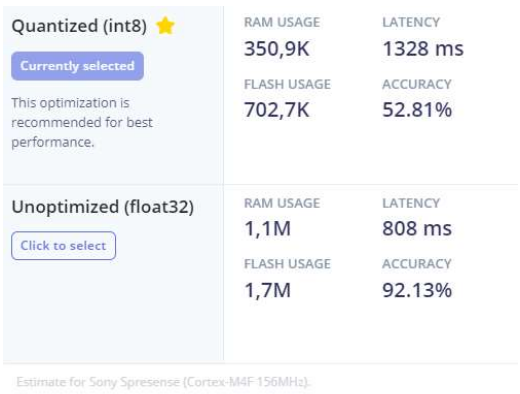

Figura 3: *Resumen de características para diferentes optimizaciones. Fuente: Edge Impulse*

Al ser un modelo entrenado en Edge Impulse y descargado en formato de librería, no tiene un aprendizaje continuo mediante su uso en Arduino. El aprendizaje es realizado únicamente en la plataforma de Edge Impulse previo a la compilación del modelo en formato de librería de Arduino.

4.1.3. Procesado de imágenes desde Arduino

Una vez elegido el formato de instalación del modelo, se configura el entorno Arduino para el uso de la librería. En el apartado 4.1.1 se ha descrito cómo se estructura de la librería de Arduino desde Edge Impulse, la cual se define en una estructura de carpetas para el correcto funcionamiento de la librería y en unos ejemplos predefinidos. Tal y como se ha definido, para la placa de desarrollo Sony Spresense, no existe un ejemplo ajustado, así que se debe usar el ejemplo genérico, llamado *static_buffer*. Dicho ejemplo realiza una clasificación de imágenes de forma estática, es decir, se deben introducir manualmente los parámetros que definen la imagen a clasificar.

Al no disponer de una documentación técnica específica a partir de la cual aprender sobre el uso básico de la librería, el aprendizaje se ha realizado a partir del estudio pormenorizado del ejemplo genérico. A continuación, se describen las funciones básicas encontradas en el ejemplo *static_buffer*, y cómo estas pueden ser extrapoladas al desarrollo del objetivo principal del proyecto.

4.1.3.1. Parámetros de entrada

```
#include <IMAGE_CLASSIFICATION_DEF_4.3_inferencing.h>

static const float features[] = {
    // copy raw features here (for example from the 'Live classification' page)
    // see https://docs.edgeimpulse.com/docs/running-your-impulse-arduino
};

/**
 * @brief      Copy raw feature data in out_ptr
 *             Function called by inference library
 *
 * @param[in]  offset   The offset
 * @param[in]  length   The length
 * @param      out_ptr  The out pointer
 *
 * @return     0
 */
int raw_feature_get_data(size_t offset, size_t length, float *out_ptr) {
    memcpy(out_ptr, features + offset, length * sizeof(float));
    return 0;
}
```

Figura 4: *Funciones de los parámetros de entrada para el modelo. Fuente: Propia.*

Tal y como se muestra en la Fig. 4, al inicio de esta declaración de parámetros de entrada, se incluye la librería de clasificación de imágenes generada por Edge Impulse mediante la directiva *#include*.

Como se ha mencionado anteriormente, el ejemplo requiere de la introducción manual de los parámetros que definen la imagen, mediante la primera línea del código (*static const float features [ ]={ }*). Dichos parámetros se denominan *raw features* y cada uno de ellos se asocia a la información de un píxel de una imagen. Al estar el modelo preliminar basado en el análisis de imágenes de 96x96 píxeles, la longitud del vector que define las *raw features*, es de 9216 (el resultado de multiplicar los píxeles de la altura por los del ancho de la imagen). Estos parámetros son fácilmente extraíbles, como se observa en la Fig. 5, a través de una ventana de la web de Edge Impulse donde se ha modelado el algoritmo, tal y como indica el texto dentro de la primera línea de código. La imagen de la cual se extraen los *raw features* se puede seleccionar del banco de imágenes destinadas al test del modelo.

Figura 5: *Ventana a partir de la cual se extraen las Raw Features desde Edge Impulse. En este caso se ha seleccionado la imagen de un USB. Fuente: Edge Impulse.*

La función contigua a la introducción de los parámetros de la imagen no es usada directamente en el código definitivo, aunque sí lo hace una función derivada de esta. A pesar de no tener un uso directo en el futuro código, esta función se resume a continuación. La función *memcpy* en lenguaje C++, se define originalmente como aparece en la Fig. 6 [15].

```
void *memcpy(void *dest, const void * src, size_t n)
```

Figura 6: *Declaración de la función memcpy en lenguaje C++. Fuente: cplusplus.*

Dicha función permite copiar n caracteres de la memoria apuntada por *src* a la memoria apuntada por *dest*. Es decir, permite copiar valores como strings, floats, ints, etc. de un array (vector de una dimensión donde se almacenan valores de diferente naturaleza, como strings o floats) a otro.

4.1.3.2. Setup

Posteriormente a la introducción de los parámetros de entrada, se debe declarar qué función básica debe iniciarse al inicio del programa, en el bloque de código *setup*. En la Fig. 7 se muestran las líneas del código que definen en el *setup*. En este caso la única función que se declara es aquella que inicia la transmisión de datos en serie y que permite el uso de las herramientas propias de Arduino, que permiten la monitorización de las variables deseadas, como son las herramientas de *Serial Monitor* y *Serial Plotter*. Para cada placa de desarrollo se debe usar aquel ratio de comunicación especificado por el proveedor y determinado por la arquitectura de la propia, en el caso de la placa Sony Spresense, el ratio de comunicación es de 115200 baudios (bits por segundo). La función *while* incorporada en este bloque tiene como única finalidad bloquear la ejecución del código mientras la placa no esté conectada.

```
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    // comment out the below line to cancel the wait for USB connection (needed for native USB)
    while (!Serial);
    Serial.println("Edge Impulse Inferencing Demo");
}
```

Figura 7: *Definición del bloque setup del código. Fuente: Propia.*

4.1.3.3. Loop

```
void loop()
{
    ei_printf("Edge Impulse standalone inferencing (Arduino)\n");

    if (sizeof(features) / sizeof(float) != EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE) {
        ei_printf("The size of your 'features' array is not correct. Expected %lu items, but had %lu\n",
            EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, sizeof(features) / sizeof(float));
        delay(1000);
        return;
    }
```

Figura 8: *Primera función del bloque loop del código Fuente: Propia.*

La función *loop* es aquella que se ejecuta un número infinito de veces. En este caso la función *loop* permite ejecutar de forma continua las funciones que activan el modelo de clasificación

de imágenes. A pesar de esta ejecución ininterrumpida, los parámetros de entrada son estáticos, mostrando una información que no varía a lo largo del tiempo. En las Figs. 8 y 9 se encuentran las líneas del código *loop*.

La primera función que se encuentra en el *loop*, tras la función de imprimir el título *Edge Impulse standalone inferencing (Arduino)* en el monitor serie (*Serial* Monitor) de Arduino, es una función condicional cuyo parámetro de comparación es el tamaño de los parámetros de entrada. Si este último es correcto frente al tamaño requerido por el modelo (*sizeof(features) / sizeof(float)*[4] *!= EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE*), se ejecuta la siguiente función. En caso contrario, el código se para y solo ejecuta el mensaje de error definido dentro de la misma función (*"The size of your 'features' array is not correct. Expected %lu items, but had %lu\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, sizeof(features) / sizeof(float))*, el cual indica que el tamaño no es el correcto y que debería ser el definido por *sizeof(features) / sizeof(float)*.

Todos los resultados obtenidos a través del análisis de los parámetros de entrada por parte del modelo, serán almacenados en la estructura *result*, definida como tal por la declaración como tipo *ei_impulse_result_t* . Dicha variable está definida como una estructura (tipo *struct* de Arduino) que permite almacenar información sobre los resultados de la clasificación y el tiempo de clasificación (para la detección de objetos, también permite almacenar la información sobre las *bounding boxes*). El tipo estructura *ei_impulse_result_t*, queda definida (y por consecuencia todas las variables que se definan bajo su declaración) de la siguiente manera[16]:

- *ei_impulse_result_bounding_box_t*: Array de valores asociados a las bounding boxes. Solo aplicable a modelos de detección de objetos
- *ei_impulse_result_classification_t*: Array de valores asociados a los resultados de la clasificación de la imagen procesada.
- *ei_impulse_result_timing_t*: Almacena información sobre el tiempo de uso del modelo, desde el muestreo hasta la inferencia.

Cada uno de los elementos de esta estructura contiene información que se puede obtener mediante la definición de subestructuras, por ejemplo, la rama de *classification* se encuentra estructurada para acceder a los valores de cada objeto posible definido en el modelo. Tomando el nombre de la estructura dada en el código, *result*, para acceder a los resultados de clasificación del valor *ix*, se usa la definición *result.classification[ix].value*, siendo

[4] *sizeof(features) / sizeof(float))* es una función que permite saber el número de elementos dentro del vector *features*. Calcula, en bytes, el tamaño del vector *features* y lo divide por el tamaño, en bytes, de un elemento *float* (4 bytes).

classification parte de la estructura *ei_impulse_result_t* y *value*, una parte de la subestructura que depende de *classification*.

Para procesar los parámetros dados por las *raw features*, estos se deben almacenar bajo una estructura que permita obtener dicha información en diversos puntos del modelo de clasificación. Tal estructura se define como *signal_t* y almacena los datos, previamente estructurados en un vector unidimensional como se define en el punto 5.1.3.1, en una localización de la memoria RAM o flash específica[17].

```
ei_impulse_result_t result = { 0 };

// the features are stored into flash, and we don't want to load everything into RAM
signal_t features_signal;
features_signal.total_length = sizeof(features) / sizeof(features[0]);
features_signal.get_data = &raw_feature_get_data;

// invoke the impulse
EI_IMPULSE_ERROR res = run_classifier(&features_signal, &result, false /* debug */);
ei_printf("run_classifier returned: %d\n", res);

if (res != 0) return;

// print the predictions
ei_printf("Predictions ");
ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
    result.timing.dsp, result.timing.classification, result.timing.anomaly);
ei_printf(": \n");
ei_printf("[");
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    ei_printf("%.5f", result.classification[ix].value);
```

Figura 9: *Definición del loop del código. Fuente: Propia.*

Como se puede comprobar en la Fig. 9, el nombre *result* se asocia a la estructura *ei_impulse_result_t* y *features_signal* se asocia a la estructura *signal_t*. La función *features_signal.get_data* apunta a la dirección de memoria donde se almacena la información del array definido por *raw_feature_get_data* para poder ser procesada por la función *run_classifier*. Esta función inicia el modelo de clasificación de imágenes con los parámetros de entrada *features_signal* y almacena los resultados en *result*. Si no hay ningún error en la clasificación la función devuelve un 0. Posterior a la ejecución de esta función, se suceden una serie de impresiones de los resultados, consultándolos mediante los sistemas de estructuras definidos anteriormente (p. ej., *result.timing.classification*, imprime el tiempo de clasificación).

Los resultados que se imprimen en el *Serial Monitor* se pueden observar en la Fig. 10.

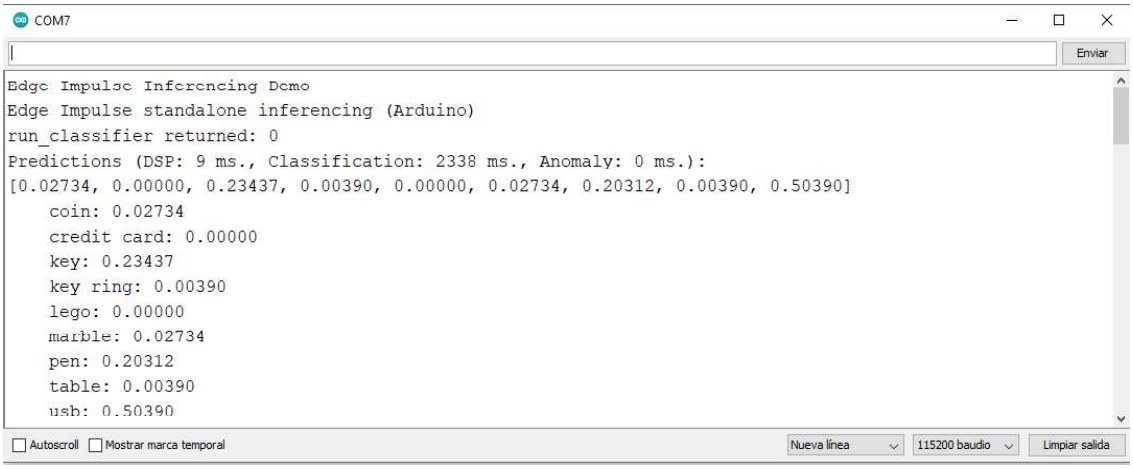

Figura 10: *Resultados impresos en el monitor serie. Fuente: Propia*

La tercera línea indica que la función *run_classifier* devuelve un 0, indicando que no ha habido ningún error en la clasificación. La siguiente línea devuelve la información sobre el tiempo de procesado y clasificación, a partir de las funciones *result.timing.dsp* y *result.timing.classification*. Se comprueba que el proceso que requiere mayor tiempo, entre la toma de la imagen y la obtención de un resultado, es el de clasificación de la imagen. Dicho proceso se lleva a cabo en 2,34 segundos. Tomando este resultado, y entendiendo que el tiempo de clasificación, siempre que se use el mismo modelo, no se reducirá, se puede afirmar que será el mismo que todos los ensayos posteriores con el mismo modelo. El resultado de la clasificación es lo último que se imprime y aparece en forma de array, primero, y en forma de lista con los objetos asociados, a continuación.

La imagen tomada es la de un USB como se muestra en la Fig. 5 y, por lo tanto, el clasificador debería clasificarla como tal. No obstante, el clasificador devuelve una probabilidad de 50,39% que sea un USB, frente a un 23,44% que sea una llave, o bien, frente a un 20,31% que sea un boli. Al ser solo una muestra y estática, no se puede determinar si la precisión del clasificador es la esperada o no. Debido a que el objetivo primero del trabajo es el desarrollo de la aplicación de visión por computador, la precisión del clasificador será tomada en cuenta en el momento de optimización de la misma.

De forma resumida, el método usado por la librería de Edge Impulse, y por consiguiente todos los códigos que deriven de ella, es romper la imagen píxel a píxel para concatenarla en forma de array en una ubicación de memoria RAM o flash. Posteriormente, a través de la función *run_classifier*, se accede al clasificador de imágenes, el cual, finalmente, devuelve los resultados de la clasificación.

4.1.4. Captura de imágenes con Sony Spresense

Como se ha mencionado anteriormente, la captura de imágenes con la placa de desarrollo

Sony Spresense, a través del Arduino IDE, se hace mediante la librería *Camera*.

Tal y como se describe en la guía de desarrollo de Arduino de Sony Spresense[18], la librería se compone de dos clases, "*theCamera*" y "*CamImage*". La primera clase contiene todas aquellas funciones que permiten modificar los parámetros de la cámara, desde el balance de blancos hasta la calidad de la imagen, pasando por la exposición y la ISO, entre otros. La clase *CamImage*, permite obtener y modificar diversos parámetros de las imágenes tomadas, como el ancho y el alto de la misma, el modelo de color, entre otros. Las funciones quedan resumidas en las tablas 6 y 7.

Nombre de la función	Descripción
setAutoWhiteBalance()	Iniciar y detener el balance de blancos automático de la cámara
setAutoWhiteBalanceMode()	Ajuste del modo en el balance de blancos automático
setAutoExposure()	Iniciar y detener la exposición automática de la cámara
setAbsoluteExposure()	Ajuste el tiempo de exposición de la cámara en una unidad de 100 microsegundos
setAutoISOSensitivity()	Iniciar y detener la sensibilidad ISO automática de la cámara
setISOSensitivity()	Establecer la configuración de sensibilidad ISO con la configuración manual de sensibilidad ISO
setColorEffect()	Establecer efecto de imagen
setHDR()	Establecer el modo HDR
setJPEGQuality()	Establecer la calidad JPEG

Tabla 6: *Funciones de control de los parámetros de la cámara. Fuente: Modificada de Sony Spresense*

Nombre de la función	Descripción
getAbsoluteExposure()	Obtener el tiempo de exposición de la cámara en una unidad de 100 microsegundos
getISOSensitivity()	Obtener sensibilidad ISO de la cámara
getHDR()	Obtener el modo HDR
getJPEGQuality()	Obtener calidad JPEG

Tabla 7: *Funciones de obtención de los parámetros de la cámara. Fuente: Modificada de Sony Spresense*

Desde que se conecta el dispositivo el visor de la cámara captura imágenes en tiempo real (llamada *Preview* por la documentación de Sony Spresense[18]), sin almacenar la información hasta que se inicia la clase de *theCamera*, a partir de la cual se puede seleccionar como se trata esta información. Para iniciar tal clase, se usa la función *begin()*, que permite establecer los siguientes parámetros:

- Número de búferes (*buffers*) de imágenes: Permite indicar cuántos búferes deben trabajar en paralelo para poder procesar diversas imágenes a la vez. El número de búferes estándar y recomendado es 1. Solo si se trabaja con un gran número de imágenes se puede ampliar a 2. Para este proyecto solo se usa 1 búfer debido a que la clasificación de varias imágenes simultáneamente carece de sentido.

- FPS: Permite indicar la tasa de fotogramas, o FPS (fotogramas por segundo), a la que se debe realizar la captura de video. Teniendo en cuenta que la clasificación de imágenes para el primer modelo usado es de 2,34 segundos, cualquier frecuencia de captura de imagen superior a una imagen cada 2,34 segundos, causará que no todas las imágenes sean clasificadas. Aun así, ninguna imagen es almacenada, con lo cual la tasa de fotogramas no tiene mayor relevancia. Se ha establecido una captura a 5 FPS debido a que, a pesar de no tener mayor relevancia, una captura de 5FPS permite capturar una imagen cada 200 milisegundos, tasa suficiente para evitar un lapso de tiempo demasiado extenso entre la obtención de los resultados de la clasificación y la obtención de una nueva imagen a clasificar.

- Ancho y alto de la imagen: Tal y como indica el nombre, permite establecer los tamaños de altura y ancho de la imagen. Deben ser tamaños estándar de imagen como QQVGA (160x120 píxeles), HD (1280x720) o FullHD (1920x1080), entre otros. A priori cualquier modelo puede ser valido pero, como se demuestra en el apartado 6.1.6.1, para transformar las imágenes al formato con el cual se ha entrenado el clasificador (imágenes cuadradas de 96x96), se debe elegir el formato QVGA (320x240 píxeles).

- Formato de píxel: Establece qué formato de píxel debe usarse. Los formatos válidos son: YUV422, RGB565 y JPEG. El formato con el cual se ha entrenado el modelo de clasificación es JPG/JPEG con lo cual este es el formato elegido.

- Divisor del tamaño de búfer JPEG: A través de una fórmula se puede determinar el divisor del tamaño del búfer. En este caso se aplica el divisor recomendado, es decir, 7.

La siguiente función necesaria para habilitar la toma de imágenes es "*startStreaming*". Esta función crea una instancia de la clase *CamImage*, necesaria para el procesado de datos de la imagen. *startStreaming()* solamente tiene dos parámetros:

- Booleano: Cuando este se especifica como verdadero ("*True*"), se inicia la toma de imágenes de *Preview*. Para detener la toma se debe llamar la función otra vez especificando el booleano como falso ("*False*").
- CamCB: Se inicia la función "*callback*" o retrollamada de la imagen. Dicha función queda descrita en el apartado 5.1.5

Una vez definidas las dos funciones anteriores, se pueden definir todas aquellas funciones que definan parámetros de la toma de la imagen como se aprecia en la Tabla 6. Las funciones que se usan en el ejemplo, que ofrece la librería *Camera,* son la función *setAutoWhiteBalance()*, que permite un balance de blancos automático en función de las

condiciones de la imagen, y la función *setStillPictureImageFormat()*. Esta función habilita la toma de imágenes estáticas y determina sus parámetros. Como *startStreaming()*, *setStillPictureImageFormat()* permite crear una instancia de *CamImage* para procesar los parámetros de la imagen, pero esta se crea a través de la función *takePicture()*. La función *takePicture()* equivale a pulsar el disparador de la cámara, por lo que solamente se crea instancia de *CamImage()* cuando se ejecuta la función.

En resumen, existen dos vías a través de las cuales se obtienen instancias de la clase *CamImage*, necesarias para obtener datos de la imagen. El primer camino es habilitar únicamente la función *startStreaming()*, la cual permite la toma de imágenes de forma continua, es decir, un vídeo, a una tasa de fotogramas definida. El segundo camino es, a parte de habilitar la función anterior, ejecutar la función *setStillPictureImageFormat()*, para definir los parámetros de una imagen estática, junto la función *takePicture()*, que toma imágenes estáticas siempre que se ejecuta. Ambos métodos conducen a la obtención de instancias de la clase *CamImage* siendo, en este caso, al no almacenarse las imágenes de forma permanente en la memoria, igual de válidas. En este primer modelo se ha decidido usar la función *startStreaming()*.

4.1.5. Función retrollamada CamCB

La función *CamCB* queda habilitada por *startStreaming()*, como se ha mencionado en el apartado 5.1.4. *CamCB* es una función de retrollamada (*callback*), es decir, es una función que se usa como argumento de otra (*startStreaming()*) de manera que cada vez que se llama a la segunda también se llama a *CamCB*. Esta función permite tratar las instancias de *CamImage*, pudiendo extraer todos aquellos parámetros de la imagen que sean útiles para el usuario. Al ser *startStreaming()* una función que habilita la toma continua de imágenes, cada vez que se tome una nueva imagen se llamará a *CamCB*, frecuencia idealmente dada por los FPS determinados en la función *begin()*.

En el uso común de la función *CamCB*, la frecuencia de llamada a esta función no se corresponde a los FPS. Esta variación es debida a que, desde el momento en que la instancia de *CamCB* se genera por la toma de una imagen, hasta que esta función acaba de procesar la información requerida de la imagen, todas las imágenes tomadas quedan descartadas. Por ejemplo, si la tasa de FPS es de 5, es decir se toma una imagen cada 200 milisegundos, y todas las funciones internas a *CamCB* tienen una duración conjunta de 0,75 segundos, en cuanto se tome la primera imagen, no será hasta la quinta imagen (0,8 segundos) que se vuelva a procesar como instancia de *CamCB* (descartando la 2ª, 3ª y 4ª imagen correspondientes a las tomas de 0,2, 0,4 y 0,6 segundos, respectivamente).

Una vez definido el funcionamiento de *CamCB*, queda demostrado que por su naturaleza es una función bucle en sí misma, siempre que se use el método de toma de imagen continua para crear instancias de la clase *CamImage*. Si bien se ha mencionado anteriormente que ambos formatos de creación de instancias de dicha clase (en video o estáticas) son igual de

validas, debido a que este trabajo incluye el clasificador de imágenes que tiene una duración mayor a cualquier tasa de FPS, es preferible usar la función *CamCB* como el bucle del código general de la aplicación. Si se usara la función *takePicture()* en la función *loop()* habitual de Arduino, cabría la posibilidad de encontrar un lapso de tiempo entre la finalización de clasificación de una imagen y la aceptación de una nueva instancia, debido a los retrasos internos de las funciones. Si se usa la primera opción, al ser la propia función *CamCB* el bucle del código, estos retrasos internos no existen ya que al finalizar una clasificación puede aceptar la nueva instancia creada.

4.1.6. Unión de los códigos de captura y clasificación de imágenes

Habiendo definido los códigos básicos que configuran los dos grandes procesos necesarios para obtener los resultados de la clasificación de imágenes, la toma de imágenes y su clasificación, el siguiente paso es unir ambos en un mismo código. La creación de dicho código unificado y funcional, conlleva a la consecución del objetivo principal de este trabajo, ya que se habrá obtenido una aplicación funcional que captura imágenes, las procesa y las clasifica en la misma placa de desarrollo.

Para obtener el código unificado con ambos procesos se presenta una gran problemática y es la transformación de una imagen, capturada por las funciones definidas en el punto 5.1.4, en valores que el clasificador de Edge Impulse permita procesar, como se indica en el punto 5.1.3.1.

4.1.6.1. Reescalado de la imagen a 96x96 píxeles

Uno de los mayores retos de crear este código de captura y clasificación de imágenes es el de extraer la información de una imagen capturada en vivo, es decir no precargada, y tratarla de manera que tenga el formato necesario para que el modelo de clasificación la interprete.

El modelo de clasificación está definido para que las imágenes de entrada tengan un tamaño de 96 píxeles de alto por 96 de ancho, una superficie total de 9216 píxeles. La resolución más baja que acepta la cámara es QQVGA de 160x120 píxeles. En cualquier caso, debido a que la resolución más baja de la cámara es mayor a la resolución de entrada de las imágenes, se ha de recortar, o bien reducir, el tamaño de las imágenes capturadas.

La propia biblioteca *Camera*, de Sony Spresense, dispone de una función llamada *clipAndResizeImageByHW()* en la cual se toma una instancia de *CamImage*, es decir, una imagen capturada, se especifican sus esquinas, izquierda superior y derecha inferior, y el ancho y altura que se desea obtener para la nueva imagen, en este caso 96 píxeles y 96

píxeles. Esta función genera una nueva instancia de *CamImage*.

Según se especifica en la documentación técnica de Sony Spresense[19], la reducción o aumento de las proporciones de la imagen, debe ser del orden de 1/2^n o 2^n y con resultados enteros. De esta manera el proceso de reducción, para poder usar el modelo, se debe realizar partiendo de una imagen cuadrada cuyos lados sean múltiplos exponenciales de 2 de 96. Es decir que se deben obtener imágenes de 192x192 píxeles (el doble de 96x96), 384x384 píxeles (el cuádruple de 96x96), etc.

Teniendo en cuenta las consideraciones y las fórmulas anteriores, se establece que el tamaño mínimo de imagen para poder hacer el recorte y reescalado de la imagen es el definido como QVGA. Este formato de imagen está definido por 320 píxeles de ancho por 240 de alto y, por consiguiente, es el tamaño mínimo más cercano a 192x192 píxeles, que permitirá reescalar la imagen a la mitad, obteniendo los 96x96 píxeles deseados.

Una vez determinado el tamaño proporcional a 96x96 píxeles deseado y el formato de imagen que permita un recorte a ese tamaño, queda determinar cuáles son las coordenadas en la imagen QVGA, a partir de las cuales se extraerá la imagen de 192x192 píxeles. La primera iteración se realizó marcando la esquina izquierda superior como 0 y 0 y la esquina derecha inferior como 192 y 192. Dicha configuración daba error, impidiendo que se generara la instancia de *CamImage*.

Dado que dicho error no se pudo resolver a pesar de cambiar diversas veces de configuración, se optó por revisar si existía algún código predefinido que permitiera realizar esta función. Ninguno de los ejemplos propios de Sony Spresense usaba dicha función, pero en la propia definición de la librería de *Camera* sí que se establecen todas las definiciones de parámetros necesarios. En la definición de la librería *Camera[20]*, se especifica en las líneas 388 y 389 que el ancho y alto de la imagen a recortar se definen como:

$$ancho\ imagen = x_2 - x_1 + 1$$

$$alto\ imagen = y_2 - y_1 + 1$$

Siendo los subíndices 1 y 2 correspondientes a la esquina superior izquierda y la esquina inferior derecha. En el caso de las imágenes, los ejes de las coordenadas se configuran tal que ambos solo están compuestos por valores positivos y enteros. El eje *x* se define como la subsección positiva del mismo, en coordenadas cartesianas, y el eje *y* se define como la subsección negativa en valor absoluto de las coordenadas cartesianas. Esta definición configura ambos ejes tal y como se muestra en la Fig.11.

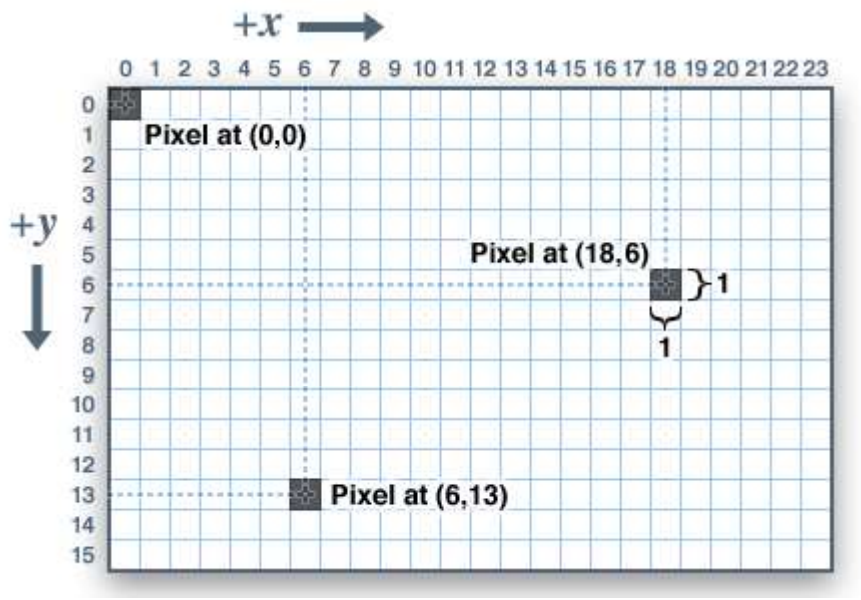

Figura 11: *Sistema de coordenadas en pantallas. Fuente: LCD's Gráficos*

El conjunto de fórmulas y definiciones anteriores, permitieron substituir las coordenadas indicadas en un inicio. Siguiendo la fórmula, si la esquina superior izquierda queda definida en (0,0), la esquina derecha inferior debe de modificarse:

$$ancho\ imagen = x_2 - x_1 + 1$$

$$192 = x_2 - 0 + 1$$

$$x_2 = 191$$

$$alto\ imagen = y_2 - y_1 + 1$$

$$192 = y_2 - 0 + 1$$

$$y_2 = 191$$

De esta manera, la esquina derecha inferior se sitúa en las coordenadas (191,191), ya que el píxel 0 de la imagen también contiene información sobre esta y forma parte del tamaño de la misma. Con esta actualización de parámetros, la función deja de dar error y la imagen ya puede recortarse y reducirse, generando una instancia de *CamImage*.

Finalizado este apartado ya se obtiene una imagen acorde al modelo, quedando únicamente el proceso de conversión de la información de los píxeles a parámetros *raw features*, para completar la unión entre el código de captura de imágenes y el de clasificación de las mismas.

4.1.6.2. Transformación de la información de la imagen a parámetros *raw features*

Como se ha mencionado anteriormente, el éxito de los procesos realizados en este apartado permitirá tomar imágenes en directo y procesarlas por el clasificador de imágenes.

La mayor dificultad de este apartado es que, a pesar de conocer los fundamentos teóricos de los *raw features*, no hay ninguna biblioteca ni función directa que permita una transformación de la información de la imagen en estos parámetros.

En apartados anteriores, se ha mencionado que la biblioteca creada por Edge Impulse ofrece ejemplos para varias placas de desarrollo, entre las que no figura Sony Spresense. Aun así, en los ejemplos proporcionados se desarrollan funciones similares a las requeridas en este apartado. El ejemplo basado en la placa de desarrollo Portenta H7, se especifica un proceso de toma de datos de la imagen y conversión a *raw features* aplicable al desarrollo para Sony Spresense. Este proceso se basa en capturar las imágenes en formato de escala de grises para, posteriormente, tratar la información píxel a píxel para que pueda ser introducida en el modelo.

La primera parte del proceso mencionado, es capturar las imágenes en escala de grises. El porqué de este formato de captura no se especifica, pero, al ser una prueba preliminar, se acepta el modelo de ejemplo y, una vez verificado su correcto funcionamiento, se realizarán pruebas con otros modelos, en la etapa de optimización, explicado en el apartado 6.3. Para la captura en este formato, la biblioteca Camera de Sony Spresense, incluye una función que permite la transformación del modelo de color aplicado a la imagen, llamada *convertPixFormat*. A dicha función se le debe especificar un parámetro, indicando el modelo de color, que, en el caso de la escala del modelo de escala de grises, tal parámetro es *CAM_IMAGE_PIX_FMT_GRAY*.

La segunda parte, cuya realización implica obtener los *raw features* necesarios para el modelo, se basa en la función *ei_camera_cutout_get_data()*, detallada en la Fig. 12.

```
int ei_camera_cutout_get_data(size_t offset, size_t length, float *out_ptr) {
    size_t bytes_left = length;
    size_t out_ptr_ix = 0;

    // read byte for byte
    while (bytes_left != 0) {

        // grab the value and convert to r/g/b
        uint8_t pixel = ei_camera_capture_out[offset];

        uint8_t r, g, b;
        mono_to_rgb(pixel, &r, &g, &b);

        // then convert to out_ptr format
        float pixel_f = (r << 16) + (g << 8) + b;
        out_ptr[out_ptr_ix] = pixel_f;

        // and go to the next pixel
        out_ptr_ix++;
        offset++;
        bytes_left--;
    }

    // and done!
    return 0;
}
```

Figura 12: *Función ei_camera_cutout_get_data. Fuente: Propia*

Esta función alimentará la estructura *signal_t*, a través de *.get_data*, así como lo hace *raw_features_get_data* en el ejemplo de *static_buffer*, como se ha definido en el apartado 5.1.3.3.

Las dos primeras líneas del código definen contadores que permiten, en conjunto, pasar de un píxel al siguiente. La siguiente línea operativa ya define una función bucle, *while()*, que finaliza una vez todos los bytes de la imagen hayan sido tratados. Dentro de esta función bucle, las funciones usadas son las siguientes:

- *ei_camera_captures_out[ ]*: Esta función permite obtener la información de un píxel, definido por la posición offset, de la imagen capturada. Es una función definida exclusivamente para la placa de desarrollo Portenta H7. Para obtener los mismos resultados para Sony Spresense, se debe buscar una función equivalente.

```
/**
 * @brief      Convert monochrome data to rgb values
 *
 * @param[in]  mono_data  The mono data
 * @param      r          red pixel value
 * @param      g          green pixel value
 * @param      b          blue pixel value
 */
static inline void mono_to_rgb(uint8_t mono_data, uint8_t *r, uint8_t *g, uint8_t *b) {
    uint8_t v = mono_data;
    *r = *g = *b = v;
}
```

Figura 13: *Función mono_to_rgb. Fuente: Propia*

- *mono_to_rgb( )*: La Fig. 13 describe la función *mono_to_rgb* y es definida de la siguiente manera:
    - La información *mono_data* se corresponde a la información del píxel extraída con la función anterior. Al ser una configuración de color de imagen monocromática, solo existe un canal de información que contiene 256 colores. Por lo tanto, los valores de *mono_data* se encuentran en el intervalo [0,255].
    - La función en sí hace una pseudo conversión de monocromático a RGB. Dado que es imposible convertir un vector unidimensional en uno tridimensional con la información completa sin disponer de más datos, se asigna el mismo valor monocromático para las tres dimensiones de colores. El resultado siempre serán tonalidades de gris, yendo desde el color negro asignado al valor RGB={0,0,0}, hasta el blanco, asignado al valor RGB={255,255,255}.
- *float pixel_f = (r << 16) + (g << 8) + b*: Esta línea de código asigna al valor *pixel_f*, formado por 32 bits (4 bytes), los valores de *r*, *g* y *b* desplazados a la izquierda 16, 8 y 0 posiciones correspondientemente. Esta conversión permite transformar tres canales de información en una sola y con el formato adecuado para la clasificación. Según la definición de los operadores de desplazamiento a la izquierda[21], siendo E1 el valor a desplazar y E2 el desplazamiento de bits, el valor final después del desplazamiento es $E1 * 2^{E2}$. Por ejemplo, si se tiene un valor *r* de 255, es decir el valor correspondiente a 8 bits, 2^8, el valor desplazado será $r' = r * 2^{16} = 2^8 * 2^{16} = 2^{24}$.
- *out_ptr[out_ptr_ix] = pixel_f*: Esta función almacena la información anterior en el espacio de memoria *out_ptr* en el punto *out_ptr_ix*.
- Para finalizar el bucle los contadores *offset* y *out_ptr_ix* pasan al siguiente valor. El contador *bytes_left* reduce en uno su valor.

Dada la complejidad de las funciones y la necesidad de adaptar *ei_camera_captures_out* a Sony Spresense, todos los intentos de adaptación a la placa de desarrollo resultaron en

múltiples errores. Es por esto que se realizó una búsqueda con el objetivo de encontrar un modelo de clasificación de imágenes a través de Edge Impulse en Sony Spresense. En la fecha de la búsqueda solo existía una publicación que cumpliera con los requisitos. Dicho trabajo permite un control y alerta sobre el estado de diferentes depósitos de aceite en Alemania[22].

Por lo que refiere a la función de obtención de los *raw features*, el trabajo sobre los depósitos de aceite también usa la función obtenida del ejemplo de la placa de desarrollo Portenta H7, *ei_camera_cutout_get_data()*. En este trabajo se realizan unos cambios sobre la función para que se adapte a Sony Spresense, como queda definida en la Fig. 14.

```
int ei_camera_cutout_get_data(size_t offset, size_t length, float *out_ptr) {
  size_t bytes_left = length;
  size_t out_ptr_ix = 0;

  uint8_t *buffer = sized_img.getImgBuff();

  // read byte for byte
  while (bytes_left != 0) {

    // grab the value and convert to r/g/b
    uint8_t pixel = buffer[offset];

    uint8_t r, g, b;
    mono_to_rgb(pixel, &r, &g, &b);

    // then convert to out_ptr format
    float pixel_f = (r << 16) + (g << 8) + b;
    out_ptr[out_ptr_ix] = pixel_f;

    // and go to the next pixel
    out_ptr_ix++;
    offset++;
    bytes_left--;
  }

  // and done!
  return 0;
}
```

Figura 14: *Función mono_to_rgb. Fuente: Propia*

Como se puede comprobar, la función crítica que impedía el uso de esta función, ha sido substituida por un parámetro obtenido directamente de una estructura de la biblioteca *Camera* de Sony Spresense, *getImgBuff*. Este parámetro se almacena en la variable *buffer* y contiene la lista de toda la información de los píxeles ordenada. La declaración *pixel = buffer [offset]*, permite elegir la información del píxel que se encuentra en la posición *offset* del *buffer*.

Para comprobar que este cambio es realmente válido, se debe añadir al código de captura de imágenes y a la biblioteca del modelo de Edge Impulse para activar la función run_classifier()

y obtener los resultados de la clasificación. El código completo se puede encontrar en el Anexo como "Codigo_captura_y_clasificacion". Una vez compilado e iniciado el código en Sony Spresense, se obtienen los resultados de la Fig. 15.

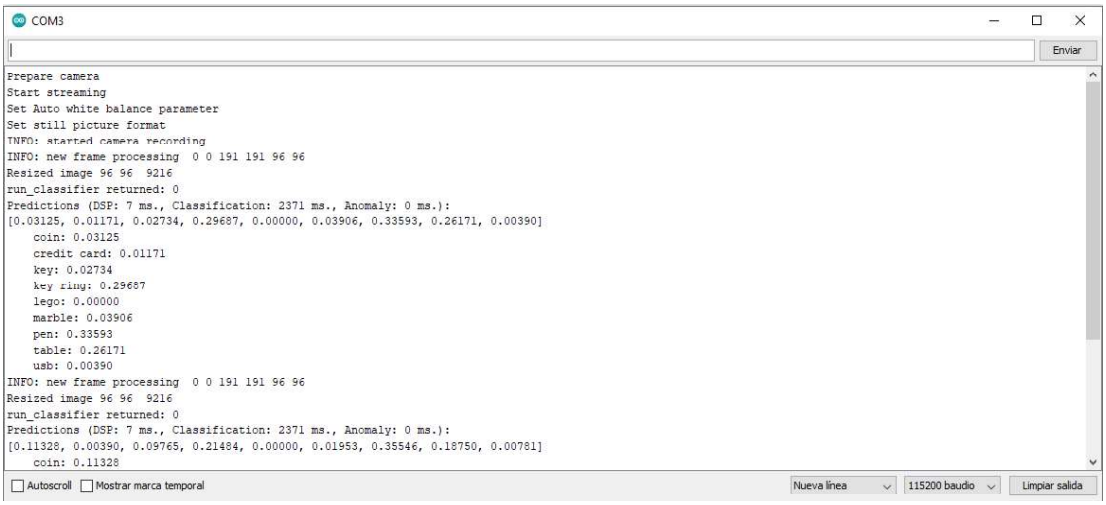

Figura 15: *Resultados obtenidos tras la compilación y ejecución del código. Fuente: Propia*

Las primeras cinco líneas indican que se están iniciando bien los parámetros establecidos de captura de imágenes. La siguiente línea indica que se realizará el proceso de recorte y reescalado de la imagen a 96x96 píxeles, tomando como referencias las esquinas (0,0) y (191,191), seguida de la línea que confirma que se ha creado correctamente la nueva instancia de *CamImage* de 96x96 píxeles[5]. El trabajo sobre los depósitos de aceite [22] configura las esquinas de forma que la imagen final corresponda al centro de la imagen inicial. Para futuras versiones de la aplicación se aplicará este segundo método, interpretando que recortando al centro de la imagen la pérdida de datos es menor. La octava línea es la que confirma que el modelo de clasificación de imágenes ha funcionado correctamente, ya que devuelve el valor 0. Las últimas líneas muestran el resultado de la clasificación, demostrando que el código es funcional en su totalidad. El resultado es satisfactorio ya que, en el momento de la captura, no se está apuntando a ningún objeto en particular y, por lo tanto, se asigna al objeto neutro *table*. Aun así, dicho resultado no es relevante ya que el tamaño muestral para valorar la precisión del modelo es insuficiente.

El código definido como "Codigo_captura_y_clasificacion" en el Anexo, derivado del desarrollo expuesto a lo largo de este apartado, se conjuga en sí mismo como una aplicación de visión por computador basada en Sony Spresense. Así que, tal y como se ha anunciado al inicio de este apartado 5.1.6, la creación del código anterior, que unifica la captura de imágenes con el

[5] Ambas líneas son reaprovechadas del trabajo sobre los depósitos de aceite [22], donde se emplean las mismas líneas de código para imprimir información sobre el reescalado de las imágenes.

procesado de estas, implica la consecución del primer objetivo de este trabajo.

4.1.7. Características empíricas del modelo preliminar

A través de diferentes pruebas con el modelo usado, se ha podido comprobar que las características del modelo especificadas en el apartado 5.1.1, no se corresponden con las características obtenidas a partir de la ejecución de este en la placa de desarrollo. Las características de rendimiento reales de este modelo para la placa Sony Spresense son las siguientes:

- Recursos de memoria requeridos:
  - 350,9 kB de RAM
- Tiempo de clasificación (latencia):
  - 2371 ms

Si bien estos resultados no influyen en el desarrollo actual de la aplicación, ya que el objetivo es conseguir una aplicación de visión por computador, una vez finalizado el desarrollo básico de esta, en la fase de optimización, el modelo debe ser mejorado en cuanto a términos de rendimiento básico (RAM y tiempo de clasificación) se refiere.

4.2. Movimiento de la prótesis y clasificación de imágenes

Como se define en el punto 3.1.2 el desarrollo de la aplicación y su aplicación en el uso de la prótesis de ARM2u puede beneficiar al desempeño de la misma. Para poder usar la aplicación desarrollada junto con el movimiento normal de la prótesis, deben unirse ambos códigos.

Tal y como se describe en el apartado 2.2.3, el movimiento de la prótesis se basa en la acción de dos servomotores a través de dos pulsadores. Cuando un pulsador es presionado de forma continua, el gripper realiza un movimiento de cierre hasta que se deja de presionar el pulsador o el servomotor llega a su límite físico de rotación. Para abrir la pinza, se presiona el pulsador un solo instante. El movimiento de pronosupinación se realiza presionando el segundo pulsador, rotando la pinza en un sentido hasta el límite físico del servomotor (hasta la posición 270º) e invirtiendo el movimiento para rotarla hasta el otro límite físico (hasta la posición 0º). El movimiento de pronosupinación es continuo siempre que se mantenga presionado el pulsador. El código de movimiento también incluye la lectura de sensores de control de temperatura y, en un futuro, deberá incluir sensores electromiográficos. Todas estas funciones son básicas y deben ser incluidas en la aplicación para tener una integración completa con el uso de la prótesis. Dicho código se encuentra comentado como “Codigo_movimiento_protesis” en el Anexo.


La idea más básica de unión de códigos es hacerlo en un mismo script y que se ejecuten consecutivamente. No es una práctica ideal si se busca una rápida reacción ante inputs externos, como puede ser la lectura de un sensor y activar un estado de alerta, pero es la práctica más básica para comprobar que ambos códigos son compatibles. La compatibilidad de dos o más códigos viene dada, sobre todo, por el no solapamiento de variables y la compatibilidad propia de librerías, así como la capacidad de procesamiento y memoria de la placa de desarrollo o microcontrolador.

Primeramente, para valorar dichas compatibilidades, se ha unido en un solo script de Arduino ambos códigos, sin crear funciones ni variables compartidas entre ambos.

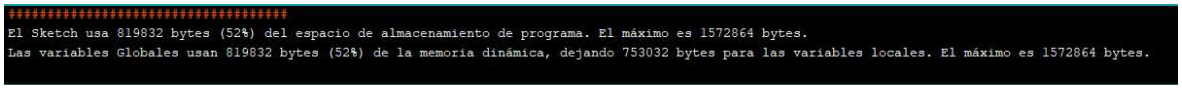

Figura 16: *Mensaje mostrado tras la compilación del código en Arduino. Fuente: Propia.*

El código compila correctamente y se ejecuta en la placa. Con la máxima memoria disponible (1536 kB), el sketch usa un 52%, aproximadamente 820 kB, mostrando el mensaje de la Fig. 16. Se comprueba que la memoria usada es tres veces superior a la teórica determinada por los valores ofrecidos por Edge Impulse descritos en el apartado 5.1.1. Tras ejecutar el código se corrobora que no hay ningún problema de falta de memoria.

Después de editar el código para crear funciones compartidas entre códigos, como la detección de un objeto y rotar el servomotor a una posición específica, y realizar las comprobaciones anteriores, se ejecuta de nuevo el código. Se hace una impresión de variables que indican el objeto que se ha detectado como *best value* y la posición del servomotor, indicada por *servo degrees*. Estas dos impresiones permiten comprobar que es posible recuperar el valor del resultado de la clasificación de imágenes, así como comprobar que se puede controlar la posición del servomotor, como se muestra en la Fig. 17. El código se encuentra en su totalidad como “Codigo_clasificacion_y_movimiento” en el Anexo.

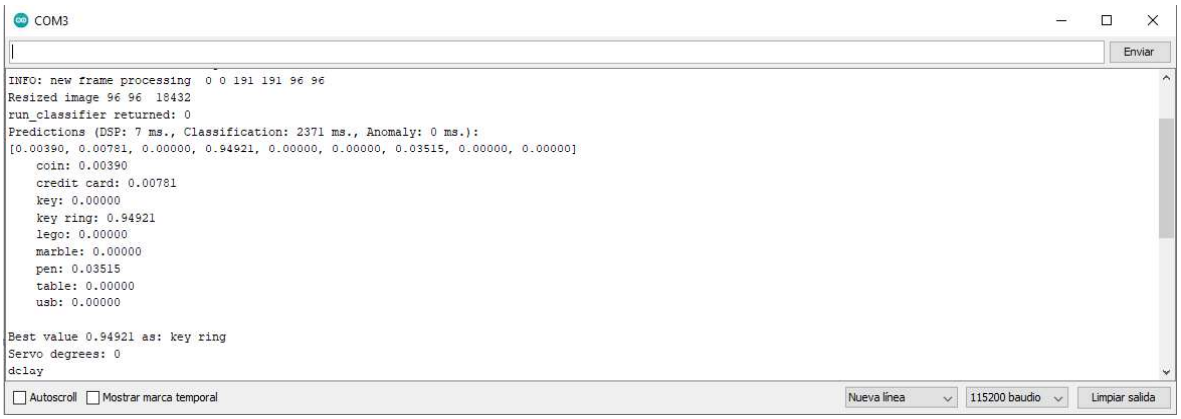

Figura 17: *Resultados obtenidos tras la compilación y ejecución del código. Fuente: Propia*

A modo de resumen, el código final captura una imagen, la clasifica y en función del resultado mueve el servomotor a una posición en concreto. En esta fase el código no tiene en cuenta si el pulsador es activado en el momento de clasificación de una imagen como objeto y, por lo tanto, al clasificar el objeto el servomotor cierra a la posición definida en el objeto. Dicha posición es definida igual para todos los objetos ya que de momento solo se busca comprobar que el código funciona, y efectivamente lo hace. El código también admite el movimiento del servomotor mediante pulsador siempre y cuando se clasifique la imagen como *table*, indicando que, supuestamente, no hay ningún objeto.

Con este código todas las acciones discurren de manera secuencial, en ningún caso pueden discurrir paralelas. En la Fig. 18 se detalla de forma esquematizada el diagrama de flujo que define el funcionamiento del código.

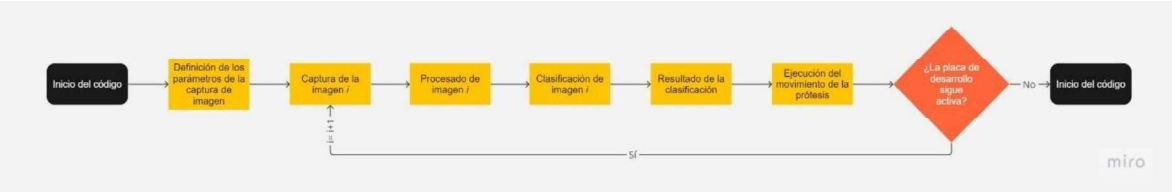

Figura 18: *Diagrama de flujo esquematizado del funcionamiento del código. Fuente: Propia*

Siempre y cuando la placa de desarrollo siga en funcionamiento, es decir, recibe corriente, el código seguirá funcionando.

Como se ha mencionado y se puede comprobar, las tareas realizadas por este código son secuenciales, es decir, el tiempo dedicado a un proceso, marcado en un recuadro amarillo de la Fig. 18, se suma al anterior y así sucesivamente hasta llegar al último. Dicha adición de tiempo va en detrimento de las necesidades de la prótesis, donde la velocidad de reacción es crucial para garantizar una similitud entre esta y un brazo completamente funcional. Por ejemplo, mediante este código si el piloto decidiera hacer una tarea para la cual no es necesaria la clasificación de imágenes, entre que la placa de desarrollo captura la señal que activa el movimiento de la prótesis y este es ejecutado, pasarían 2,6 segundos aproximadamente (teniendo en cuenta que solo la clasificación dura casi 2,4 segundos), correspondientes a la suma de todos los procesos anteriores, un tiempo de reacción increíblemente dispar al tiempo de reacción de un brazo funcional.

Dadas estas circunstancias y asumiendo que un cambio en el modelo puede conllevar a una reducción en el tiempo de clasificación, se eligen dos métodos para evitar que el tiempo de respuesta sea la suma de todos los procesos, creando así una reducción del tiempo de reacción. El primero de ellos pasa por introducir un componente que permita un cambio de

modo, como un interruptor, pudiendo, de esta manera, usar el movimiento de prótesis de manera convencional y activar dicho componente solo en momentos puntuales, donde la clasificación de imágenes es necesaria. El segundo método es aprovechar que esta placa de desarrollo tiene 6 núcleos que pueden ejecutar procesos en paralelo. La ejecución de procesos en paralelo permite usar el movimiento convencional de la prótesis cuyo tiempo de reacción es el definido por el propio código de movimiento, a la par que se clasifican las imágenes que se capturan. Se puede definir bajo qué circunstancias es necesario transmitir el resultado de la clasificación y obligar a la prótesis a realizar un movimiento específico.

La primera opción es perfectamente razonable para el uso habitual de la prótesis, pero la automatización del funcionamiento de la prótesis es clave para mejorar la experiencia del usuario. Atendiendo a la automatización de la prótesis y bajo el pretexto de la optimización de la aplicación, se elige la segunda opción, dividir los procesos en diferentes núcleos, llamada solución *MultiCore*.

Una vez desarrollado este apartado, quedan completados los dos objetivos principales de este trabajo, expuestos en el apartado 3.1. Habiendo conseguido ambos objetivos, la aplicación de visión por computador es funcional pero aún queda pendiente desarrollar el proceso de optimización de la misma. Dicha optimización, como se define en el apartado 6, aporta mejoras de carácter funcional a la aplicación, mejorando su usabilidad.

5. Optimización de la aplicación

La optimización de un código, en este caso de la aplicación de visión por computador, es un proceso inherente al desarrollo del mismo. Cualquier código puede ser optimizado una vez desarrollado, dando lugar a mejores resultados, ya sea en precisión, tiempo de ejecución o demás parámetros.

A lo largo de este trabajo se han tratado diversos puntos del desarrollo de la aplicación que son susceptibles de una gran mejora. En este apartado se expone qué procesos de mejora han sido usados para optimizar dichos puntos, correspondientes a cada uno de los subapartados.

5.1. Solución MultiCore

Habiendo completado los dos objetivos principales de este trabajo, expuestos en el apartado 3.1, se debe iterar sobre la solución propuesta con el fin de optimizarla. La primera iteración de mejora y optimización es aprovechar la estructura del procesador de la placa Sony Spresense, que permite ejecutar procesos en paralelo, usando diferentes núcleos. Esta solución, llamada solución *MultiCore*, es la primera iteración de la aplicación debido a las necesidades planteadas al final del apartado anterior.

Como se ha mencionado en el apartado 4.3, la solución de procesamiento en diversos núcleos es muy beneficiosa, pero consta de dos grandes limitaciones, las cuales deben ser estudiadas detenidamente.

5.1.1. Distribución del código en diferentes núcleos

Atendiendo a la limitación en el uso de bibliotecas, y como se ha demostrado en el apartado 4.3.2, la biblioteca *Camera* no puede ser ejecutada por un núcleo secundario. Dado que no hay alternativa, la parte del código de la aplicación descrita en los apartados 5.1.4 y 5.1.5, que corresponde a la toma de imágenes, debe ser alojada en el núcleo principal o *MainCore* de la Sony Spresense. Habiendo fijado el destino de una parte del código, queda determinar el emplazamiento de las otras dos, la parte de clasificación de imágenes y la parte del movimiento de la prótesis.

La librería de clasificación de imágenes usada en los anteriores apartados ocupa teóricamente 350,9 kB de memoria RAM, tal y como se indica en el apartado 6.1.1. Pero tal y como se ha comprobado en el apartado 6.2, la memoria usada realmente es de casi tres veces más. Para comprobarlo, se ejecuta el ejemplo *static_buffer* de nuevo, el cual incluye únicamente la librería y las funciones básicas para que esta sea de utilidad.

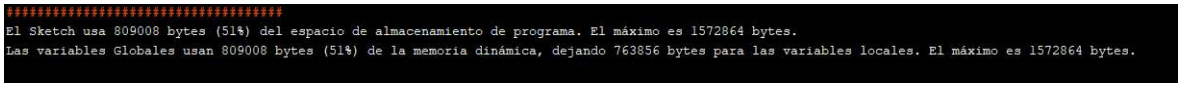

Figura 19: *Mensaje mostrado tras la compilación del código en Arduino. Fuente: Propia.*

Las funciones más básicas necesarias para el uso de la librería y la librería en sí mismo, una vez compiladas en la placa de desarrollo ocupan 809 kB, más del doble de la memoria teórica, tal y como indica el mensaje de la Fig. 19. Este resultado implica que la librería *Camera*, junto con el código de captura de imágenes y el código que permite el movimiento de la prótesis, ocupan 11 kB. A pesar de ello, la librería de clasificación de imágenes puede seguir siendo ejecutada en el primer núcleo secundario. La distribución de memoria explicada en el punto 4.3.1 puede permitir incluir la librería de *Camera* en el núcleo principal y la librería de Edge Impulse en el núcleo secundario. Esta distribución se hace de forma automática y, teóricamente, aún hay 716 kB sin ser usados (diferencia entre los 820 kB del código completo y los 1536 kB que dispone la placa de desarrollo). Una vez ejecutado el mismo ejemplo en el *SubCore1*, habiendo cargado el código de captura de imágenes en el *MainCore*, aparece el mensaje de la Fig. 20.

Figura 20: *Mensaje mostrado tras la compilación del código en Arduino. Fuente: Propia*

A pesar de no haber limitado la memoria del *SubCore1*, indicando como memoria máxima los 1536 kB de la Sony Spresense, aparecen como memoria máxima del núcleo secundario tan solo 768 kB. Al mostrar el monitor serial del Arduino, el código devuelve mensaje de error de la Fig. 21.

Figura 21: *Mensaje de error mostrado en el monitor serie de Arduino. Fuente: Propia*

Tras una búsqueda sobre el error, desde el propio equipo de Edge Impulse responden sobre la naturaleza del error, el cual solo se muestra si no hay la suficiente memoria RAM para ejecutar la librería[23].

Tras comprobar que la librería que permite la clasificación de imágenes no puede ser incluida en el núcleo secundario 1 y, atendiendo a la estructura de distribución de memoria expuesta en el apartado 4.3.1, en ninguno de los otros núcleos secundarios, dicha librería debe ser alojada en el núcleo principal junto con la librería *Camera*.

Habiendo recabado toda la información anterior, a partir de este punto solo existe una solución para completar el desarrollo en *MultiCore*. Esta opción se basa en alojar la parte del código que toma las imágenes y las clasifica, en el núcleo principal y la parte del código restante, asociada al movimiento de la prótesis, en el núcleo secundario 1.

Dado que la captura y clasificación de imágenes se realiza en un núcleo y el movimiento en otro, el primer código debe transmitir la información sobre si se ha detectado algún objeto y cuál de ellos se ha detectado. Para ello se hace uso de la librería *MP*, que habilita los núcleos secundarios, así como la comunicación entre estos y el principal.

5.1.2. Librería MultiCore MP

Tal y como se indica en la página oficial de Sony Spresense, donde se detalla la documentación técnica de todas las librerías de Arduino, la librería *MP* tiene las siguientes funcionalidades:

- Encender y apagar los núcleos secundarios
- Comunicación entre núcleos
- Control entre núcleos
- Reorganización de memoria
- Salida de registro exclusiva

Estas funcionalidades encajan perfectamente con los requerimientos planteados anteriormente, los cuales se basan en la necesidad de uso de un núcleo secundario y la comunicación entre núcleos. Como se ha demostrado anteriormente, la reorganización de memoria no es de utilidad para el caso planteado en el desarrollo de esta aplicación.

Dado que la parte del código que controla el movimiento de la prótesis debe ser alojado en el núcleo secundario, este núcleo debe ser activado al iniciar la prótesis y no debe ser apagado en ningún momento. Esta práctica no implica ningún impedimento para el correcto desarrollo de la prótesis ni perjudica en ningún caso a la velocidad de procesamiento.

5.1.2.1. Encendido del núcleo secundario 1

Una vez la librería es incluida en los códigos alojados en el núcleo principal y los secundarios deseados, se debe incluir la función *MP.begin()* en el código de todos los núcleos.

En el caso del núcleo principal se debe incluir dicha función tantas veces como núcleos se deseen activar. El argumento de *MP.begin()* es un número entero entre 1 y 5, ambos incluidos, que indica qué núcleo se desea activar. El código del núcleo secundario que se desea activar debe incluir la función sin argumentos, finalizando de esta manera el inicio del mismo.

Para la aplicación que atañe al proyecto, se incluye en el núcleo principal la función *MP.begin(1)*, con el argumento 1 indicando que el núcleo secundario que se desea activar es el *SubCore1*. Esta función envía una señal al núcleo secundario 1 para encender el núcleo en cuestión, que debe ser respondida con la ejecución de la función *MP.begin()* sin ningún argumento, desde el código alojado en el núcleo secundario. Una vez el núcleo principal reciba que la función *MP.begin()* ha sido ejecutada desde el núcleo secundario, este quedará activado. En la Fig. 22 se muestra un esquema de este funcionamiento extraído de la documentación técnica de Spresense[9].

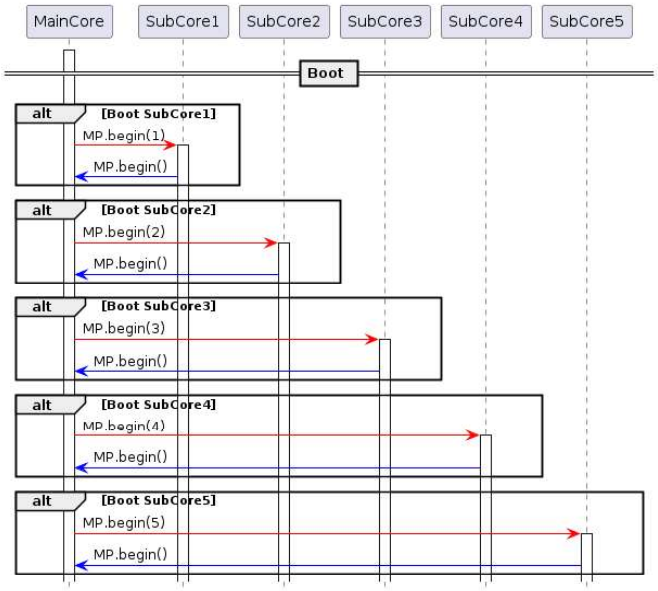

Figura 22: *Esquema del inicio de los núcleos secundarios. Fuente: Sony Spresense*

5.1.2.2. Comunicación entre los núcleos

La librería *MultiCoreMP* o *MP*, incluye dos funciones que permiten la transmisión de datos entre núcleos. Se puede realizar la comunicación a través del envío directo de datos o a través de especificar la dirección de memoria donde se almacenan. Cualquiera de los dos métodos

puede ser aplicado en la aplicación, pero para tratar más fácilmente los datos, se escoge la opción del envío directo de datos. La primera función es *MP.Send()*, la cual incluye los siguientes tres argumentos para el método de envío directo de datos:

- msgid: Asigna el identificador del mensaje a enviar. Puede ser 0 o positivo ya que el formato que acepta es de datos con signo de 8 bits.
- msgdata: Envía cualquier mensaje siempre que respete el formato de datos de 32 bits.
- subid: Especifica a qué núcleo va dirigido el mensaje (1-5). Si no se especifica ninguno o 0, se comunica con el núcleo principal.

Si el envío de información se ha realizado correctamente, devuelve un 0. De lo contrario, devuelve un valor distinto a 0. Por ejemplo si la función se define como *MP.Send(1,25,4)*, el ID del mensaje es 1, el mensaje a enviar es 25 y el núcleo al que va dirigido es el 4.

La función *MP.Recv()* habilita la recepción de los datos enviados por la función *MP.Send()*. Incluye los mismos argumentos, pero solamente debe ser especificado el argumento *subid*, con el número de núcleo del cual proviene la información que se desea recibir, ya que los otros dos argumentos deben ser incluidos como punteros. Por ejemplo, para recibir la información del ejemplo anterior desde el núcleo 4, suponiendo que viene dirigida desde el núcleo 1, la función debería ser llamada tal que *MP.Recv(&msgid,&msgdata,1)*.

Sony Spresense dispone de tres opciones de espera a la recepción de datos, tal y como se muestra en la Fig. 23. Estos tres métodos deben ser especificados como argumentos de la función *MP.RecvTimeout()*, cuyas descripciones son las siguientes:

- MP_RECV_BLOCKING: Este modo bloquea la comunicación entre núcleos hasta que se reciba una información. Ejemplo:
  - *MP.RecvTimeout(MP_RECV_BLOCKING)*
- MP_RECV_POLLING: Este modo ejecuta la función de recepción de información de manera continua, tanto si se reciben datos como si no.
  - *MP.RecvTimeout(MP_RECV_POLLING)*
- Tiempo de espera: Este modo inicia un periodo de espera determinado por el tiempo especificado como argumento en milisegundos. Si algún dato es recibido en ese periodo, se deshabilita el periodo de espera.
  - *MP.RecvTimeout(1000); tiempo de espera de 1000 ms*

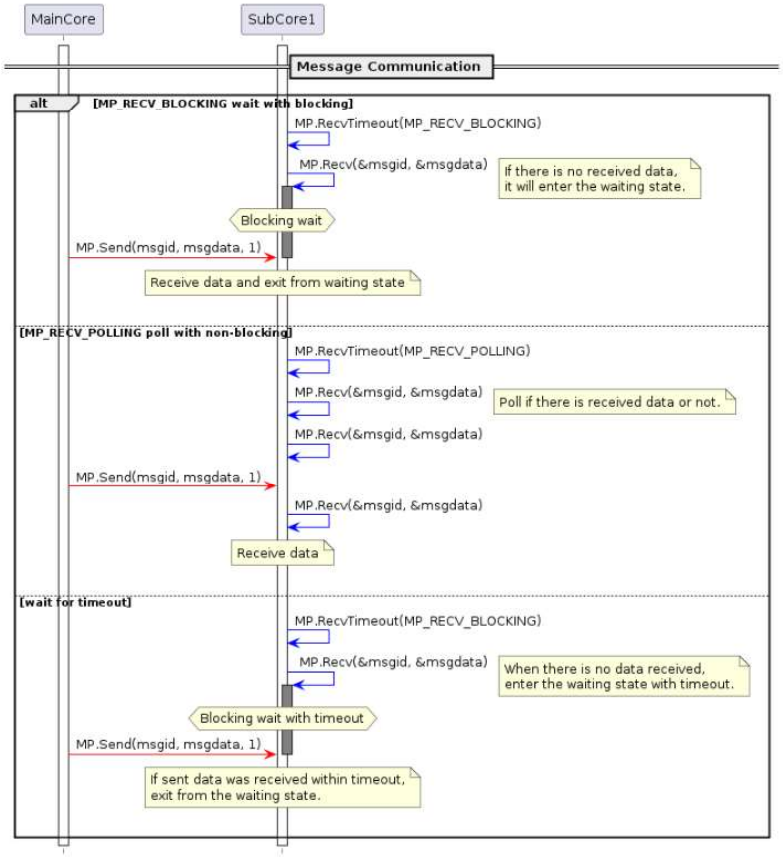

Figura 23: *Esquema de comunicación entre núcleos. Fuente: Sony Spresense*

Tras realizar diferentes pruebas de transmisión de datos, el modelo que más se ajusta a las necesidades de velocidad de la prótesis es el segundo método, *MP_RECV_POLLING*. Este método permite tener la función de recepción siempre activa, sin necesidad de considerar los tiempos de clasificación, permitiendo un movimiento de la prótesis con un tiempo de reacción igual al que se obtiene cuando la placa de desarrollo solo ejecuta el código de movimiento.

El método de transmisión de la información usado en el desarrollo de la aplicación de visión por computador, consiste en asignar un número a cada objeto que el modelo sea capaz de identificar mediante la clasificación de imágenes. Este número es enviado como *msgdata* al núcleo secundario 1, donde el código interpreta el número y ordena a los servomotores cerrar el gripper al tamaño preciso del objeto identificado. Los fragmentos del código que ejemplifican esta transmisión de información pertenecen a la Fig. 24 y Fig. 25.

```
void communication(){
        int      ret;
        int      subcore = 1;
        uint32_t snddata;
        uint32_t rcvdata;
        int8_t   sndid; /* user-defined msgid */
        int8_t   rcvid;
      if (best_value_name=="lego"){
          snddata=1;
          sndid=101;
          printf("Send: id=%d data=0x%08lx\n", sndid, snddata);
          ret = MP.Send(sndid, snddata, subcore);
          if (ret < 0) {
          printf("MP.Send error = %d\n", ret);
          }
          MP.RecvTimeout(MP_RECV_POLLING);
          ret = MP.Recv(&rcvid, &rcvdata, subcore);
          if (ret < 0) {
          printf("MP.Send error = %d\n", ret);
          }
        }
      else if (best_value_name=="credit card"){
        snddata=2;
          sndid=102;
          printf("Send: id=%d data=0x%08lx\n", sndid, snddata);
          ret = MP.Send(sndid, snddata, subcore);
          if (ret < 0) {
          printf("MP.Send error = %d\n", ret);
          }
          MP.RecvTimeout(MP_RECV_POLLING);
          ret = MP.Recv(&rcvid, &rcvdata, subcore);
          if (ret < 0) {
          printf("MP.Send error = %d\n", ret);
          }
      }
      else if(best_value_name=="key"){
          snddata=3;
          sndid=103;
          printf("Send: id=%d data=0x%08lx\n", sndid, snddata);
          ret = MP.Send(sndid, snddata, subcore);
          if (ret < 0) {
          printf("MP.Send error = %d\n", ret);
```

Figura 24: *Fragmento de la función alojada en el núcleo principal que asocia los resultados obtenidos del clasificador con números y genera el envío de dichos resultados al núcleo secundario. Fuente: Propia*

```
void closegrip_object(int msg){
  if (msg==1){
    Serial.println("Lego");
    servo_vel.write(180);
  }
  else if (msg==2){
    Serial.println("Credit Card");
    servo_vel.write(180);
  }
  else if (msg==3){
    Serial.println("Key");
    servo_vel.write(180);
  }
  else if (msg==4){
    Serial.println("Marble");
    servo_vel.write(180);
  }
```

Figura 25: *Fragmento de la función alojada en el núcleo secundario que interpreta el mensaje recibido y activa el cierre del servomotor. Fuente: Propia*

Tanto en la Fig. 25 como en el apartado 5.2, se comprueba que no hay ningún parámetro del cierre del servomotor específico a los objetos. Dichos parámetros son descritos en el apartado 7.2, tras un estudio previo realizado sobre el cierre de la prótesis.

5.2. Estudio del movimiento de la prótesis respecto los objetos

Tal y como se mencionado a lo largo del trabajo, los parámetros del ángulo de rotación del servomotor que permite el cierre del gripper de la prótesis, han sido definidos de forma genérica. Dado que el objetivo primero del trabajo era obtener una aplicación funcional, se ha considerado que dichos parámetros pueden ser definidos de forma genérica hasta que se confirme que la aplicación no tiene errores.

Para asignar el ángulo que debe cerrar el gripper, se deben tener en cuenta dos situaciones que involucren un movimiento de la pinza: agarrar objetos definidos en el modelo de clasificación, o bien, abrir/cerrar la pinza de forma independiente. En este segundo caso, el ángulo de rotación y el código de movimiento es el definido por defecto en el código de movimiento, descrito en el código llamado "Codigo_movimiento_protesis" del Anexo. Para el primer caso se debe realizar un estudio de cómo el usuario de la prótesis agarra los objetos.

Teniendo en cuenta que en el año 2022 ya se compitió en Cybathlon, se puede analizar cómo el piloto agarraba los objetos. Para el año 2023 solo se deben agarrar los 4 objetos descritos en el punto 2.2.3.3, una tarjeta, una canica, una llave y una pieza de lego. Este apartado centra el análisis del movimiento del piloto y la prótesis únicamente en estos 4 objetos. Aun así, el movimiento sobre cualquier objeto puede ser estudiado e incluido en el código siempre y cuando el modelo tenga un parámetro asociado a tal objeto. En el supuesto que en 2024 haya algún cambio en el número o la naturaleza de los objetos o se desee usar la aplicación para otro prueba, esta puede ser modificada de forma simple, adaptándose a los nuevos requerimientos.

Dado que el diseño de los engranajes que unen servomotor con el gripper ha tenido varios cambios, todos los parámetros descritos a continuación son susceptibles de ser cambiados. Para facilitar estos cambios, y atendiendo que no hay problemas de memoria en el último diseño de la aplicación, todos los parámetros asociados al ángulo de cierre de la pinza sobre los objetos son descritos como variables globales del código.

Tras realizar diversas pruebas de cierre sobre objetos, los ángulos de cierre[6] sobre los objetos de Cybathlon 2023 obtenidos son los siguientes:

- Lego: 60

[6] Son los grados indicados como valores de rotación de servomotor en Arduino. El servomotor puede rotar de 0º a 270º pero, tal y como se configura Arduino, el valor máximo que se le puede asignar a un servomotor es de 180º, significando que se crea una relación lineal entre los valores de Arduino y el valor real de rotación del servomotor.

- Llave: 114
- Tarjeta de crédito: 114
- Canica: 84

Por lo que refiere al movimiento de rotación, tras analizar los movimientos realizados por el piloto para agarrar los objetos en la prueba de Cybathlon y demás test realizados, y a pesar de ir en contra de la adaptación del usuario requerida, normalmente la rotación de la pinza nunca es activada por el piloto. Dada esta circunstancia y buscando la mayor comodidad para el piloto actual del equipo, se prescinde de parametrizar la rotación de la pinza en función de los objetos. En cualquier caso, si un futuro usuario de la prótesis desea implementar dicha funcionalidad, se puede implementar sin una gran complicación.

Una vez descritos los parámetros como variables globales del código alojado en el núcleo secundario 1, descrito en el apartado 6.1, la aplicación ya es capaz de clasificar imágenes y cerrar el gripper al tamaño requerido del objeto encontrado en estas.

La solución propuesta para combinar la clasificación de imágenes y el movimiento de la prótesis es que esta funciona de forma habitual, permitiendo el movimiento de la misma a petición del piloto. A su vez el código de clasificación de imágenes funciona en paralelo, a priori clasificando todas las imágenes como *table*, el objeto neutro del modelo de clasificación de imágenes. En cuanto el modelo clasifica dos o más veces seguidas una imagen como un objeto diferente a *table*, envía el código correspondiente al objeto al núcleo secundario 1. Si el piloto decide cerrar la pinza cuando se ha clasificado dicho objeto, la pinza cerrará directamente al tamaño del objeto. El movimiento de apertura se realiza independientemente de la clasificación ya que para todos los objetos es el mismo. Como se ha mencionado anteriormente, el movimiento de rotación también es independiente del objeto.

5.3. Uso de la escala de grises frente a color

En el apartado 5.1.6.2, se ha descrito que el primer paso para convertir imágenes en parámetros que el modelo pueda reconocer, es transformar las imágenes capturadas en formato RGB en escala de grises. El porqué de esta conversión es desconocido ya que la información asociada al color pasa de ser unidimensional a tridimensional en cuanto se asigna el valor de gris a los tres valores de color RGB, como se describe en el punto 5.1.6.2. Dado que se desconoce este factor, se decide probar ambos modelos, el modelo usado hasta ahora que se basa en la captura de una imagen en formato RGB, transformada a formato de escala de grises y extraer los valores de color desde el valor en escala de grises, y el modelo de extraer los valores de color directamente de la imagen en formato RGB.

El segundo modelo solo difiere del primero, descrito en el apartado 5.1.6.2, en tres funciones.

La primera función que los diferencia es la propia de la librería *Camera* de Sony Spresense, convertPixFormat, donde el parámetro cambia de *CAM_IMAGE_PIX_FMT_GRAY* a *CAM_IMAGE_PIX_FMT_RGB565*, pasando de convertir el formato de la imagen a escala de grises al formato RGB565. El formato RGB565 permite expresar los colores en 16 bits, 5 bits para el rojo y el azul y 6 bits para el verde[24].

La segunda función que difiere de ambos modelos es *mono_to_rgb* que, para este segundo modelo, debe transformar de RGB565 a RGB. Tras una búsqueda sobre esta transformación, un ejemplo de Edge Impulse[25], incluye una función llamada *rgb565_to_rgb*, que efectúa esta transformación. Tal y como se muestra en la Fig. 26, se usa el parámetro *color* para asignar los valores a los parámetros R, G y B, mediante una operación AND y un desplazamiento a la derecha o izquierda, en función de la posición binaria donde debe situarse la información de los colores.

```
void r565_to_rgb(uint16_t color, uint8_t *r, uint8_t *g, uint8_t *b) {
    *r = (color & 0xF800) >> 8;
    *g = (color & 0x07E0) >> 3;
    *b = (color & 0x1F) << 3;
}
```

Figura 26: *Función r565_to_rgb. Fuente: Propia*

<La última función que se modifica en el segundo modelo es *ei_camera_cutout_get_data*, donde, en lugar de llamar a la función *mono_to_rgb*, se llama a la función *r565_to_rgb*. La transformación de la información al formato *out_ptr* se realiza igual que en el primer modelo. A modo de ejemplo, se ha unido en un mismo código ambos modelos, como se muestra en el fragmento del mismo en la Fig. 27, en verde aquello que corresponde exclusivamente al modelo de escala de grises y en azul aquello que corresponde exclusivamente al modelo RGB565.

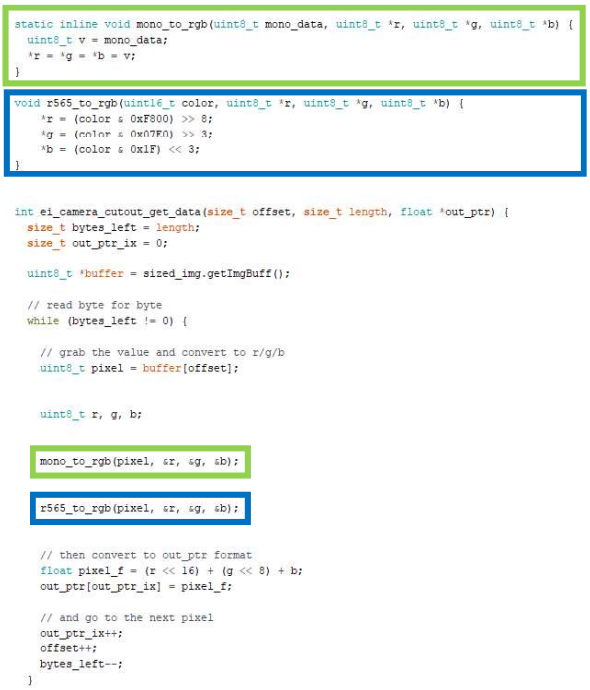

Figura 27: *Fragmento del código de ejemplo con ambos modelos. Fuente: Propia*

Los datos obtenidos con diferentes objetos[7] ratifican que el modelo de conversión a la escala de grises es claramente mejor. Ante un nuevo objeto el primer modelo clasifica la imagen acorde al objeto al segundo o tercer intento de media, incluso en el primer intento, manteniendo la clasificación en las siguientes capturas de imagen. El segundo modelo apenas consigue clasificar los objetos, variando los resultados de la clasificación en cada captura de imagen. Ambos modelos son estudiados mediante los resultados impresos en el monitor serial de Arduino.

Una vez determinado que el mejor modelo para transformar las imágenes en *raw_features* es el primero, convirtiendo el formato de imagen a escala de grises para después revertirlo y convertirlo a un pseudo RGB, queda un último paso de optimización que puede involucrar un cambio en el modelo de color que aceptan las *raw_features*. Esto se debe a que el modelo de *Transfer Learning* puede ser cambiado a un modelo que solo acepte imágenes en escala de grises.

[7] Ambos modelos son probados en las mismas condiciones de luminosidad, posición de los objetos y posición de la cámara.

5.4. Mejora del modelo de clasificación de imágenes

El modelo usado hasta este punto del proyecto es el definido en el apartado 5.1.1, estudiado por Yeray Navarro en su trabajo. Tal y como se menciona, este modelo usa el modelo de *Transfer Learning* MobileNetV2 0.35, así como 9 parámetros de entrada. El objetivo de este apartado es la mejora del modelo en dos aspectos cruciales para el buen desarrollo de la aplicación y son: la reducción de recursos de memoria necesarios, sobre todo memoria RAM, y la reducción del tiempo de clasificación.

Después de analizar qué partes del modelo pueden ser causantes de un alto uso de memoria, se determina que el modelo de *Transfer Learning* usado es aquello que augmenta más los recursos de memoria necesarios. Usando el mismo modelo de *Transfer Learning* se puede reducir sus requerimientos sobre la memoria, variando el número de neuronas del mismo (Edge Impulse permite reducir o eliminar las neuronas de la última capa). La reducción de neuronas implica, inherentemente, una reducción de la precisión del modelo. El número de parámetros de salida también influye en los recursos usados por el modelo, aunque en menor medida.

Teniendo los anteriores conceptos en cuenta, se decide reducir el número de parámetros de salida a cinco, los cuatro objetos de Cybathlon 2023 (tarjeta de crédito, llave, llavero y canica) y un objeto neutro que sigue siendo la mesa sin objetos. Debido a que este cambio no supone un cambio mayor en la memoria necesaria, se decide cambiar el modelo de *Transfer Learning* usado. Edge Impulse ofrece una herramienta llamada *EON Tuner*, que compara diferentes modelos de *Transfer Learning*, y ofrece las estadísticas básicas de cada uno de ellos, como la precisión, la latencia, la RAM necesaria o el número de neuronas de la última capa, entre otras. Previo a este análisis la herramienta requiere que se seleccione qué placa de desarrollo se va a utilizar y qué latencia se está buscando, para ajustar las estadísticas a los requerimientos de cada proyecto. En el caso de este trabajo, se ha seleccionado como placa de desarrollo Sony Spresense y una latencia de 100ms, la más baja posible, para no limitar el número de modelos estudiados.

Combinando ambos criterios, la elección de solo 5 parámetros de salida y la realización de un estudio previo de los modelos con la herramienta de *EON Tuner* se obtienen unos modelos claramente mejorados respecto el usado hasta ahora. En las Figs. 28 y 29 se muestran las estadísticas obtenidas a través de la herramienta de *EON Tuner* sobre diferentes modelos.

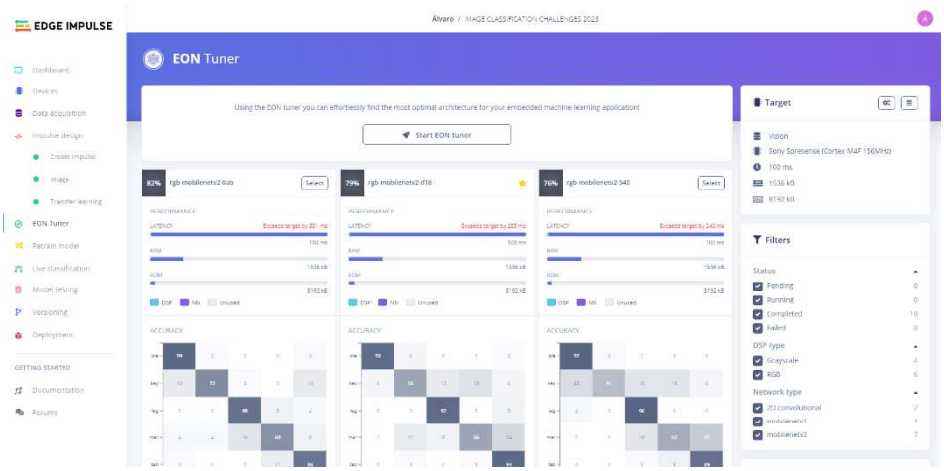

Figura 28: *Estadísticas de los modelos obtenidas con EON Turner. Fuente: Edge Impulse*

Figura 29: *Estadísticas de los modelos obtenidas con EON Turner. Fuente: Edge Impulse*

Dado que los modelos estudiados por la herramienta *EON Tuner* están ordenados por la precisión que ofrecen, de mayor a menor, y teniendo en cuenta que las variaciones entre modelos son mínimas en cuanto a latencia y RAM, se decide probar los tres primeros modelos, que ofrecen una precisión superior al 70%. El resumen de las características de cada modelo es el siguiente:

- MobileNetV2 0.05 con 64 neuronas en la capa densa final (llamado Modelo v1)
    - Modelo de color: RGB
    - Latencia teórica: 454 ms
    - RAM (int8, *EON compiler* activado): 283,4 kB
    - Precisión de entrenamiento: 91%
    - Precisión de test: 86%
- MobileNetV2 0.1 con 64 neuronas en la capa densa final (llamado Modelo v2)
    - Modelo de color: RGB
    - Latencia teórica: 380 ms
    - RAM (int8, *EON compiler* activado): 293,2 kB
    - Precisión de entrenamiento: 85%
    - Precisión de test: 88%
- MobileNetV2 0.1 con 16 neuronas en la capa densa final (llamado Modelo v3)
    - Modelo de color: RGB
    - Latencia teórica: 340 ms
    - RAM (int8, *EON compiler* activado): 293,2 kB
    - Precisión de entrenamiento: 76%
    - Precisión de test: 84%

Como se ha mostrado a lo largo del trabajo, las características teóricas que muestra Edge Impulse no corresponden con las características del mismo modelo ejecutado en la placa de desarrollo, sobre todo aquellas que hacen referencia a los recursos de memoria requeridos y la latencia. Para poder decidir qué modelo es el óptimo en este punto del proyecto es necesario instalar los tres modelos en formato de librerías y ejecutarlos. Las pruebas se realizan bajo las mismas condiciones de luminosidad y sobre el mismo fondo, para que los resultados sean comparables. Los datos más objetivos que se pueden extraer de las pruebas son precisamente aquellos que se quieren mejorar en este apartado, los recursos de memoria usados y el tiempo de clasificación. Para el caso de la precisión, se requeriría una gran cantidad de pruebas para cada modelo para extraer los porcentajes reales de precisión. Considerando los objetivos y el límite temporal del proyecto, se establece como criterio realizar cinco capturas de imagen sobre un mismo objeto en diferentes orientaciones, siendo la prueba comparativa el número de veces sobre cinco que el modelo clasifica correctamente la imagen

Usando un único código, el de captura y clasificación de imágenes, descrito en el apartado 5.1.6, se realizan las pruebas, obteniendo los siguientes resultados:

- Modelo V1
    - RAM usada: 455 kB (171,6 kB más que la teórica)
    - Latencia: 1188 ms (734 ms más que la teórica)

    - Clasificación de objetos
        - Pieza de lego: 3 veces sobre 5
        - Canica: 5 veces sobre 5
        - Llave: 1 vez sobre 5
        - Tarjeta de crédito: 5 veces sobre 5
        - Mesa: 4 veces sobre 5
- Modelo V2
    - RAM usada: 505 kB (171,6 kB más que la teórica)
    - Latencia: 1214 ms (834 ms más que la teórica)
    - Clasificación de objetos
        - Pieza de lego: 3 veces sobre 5
        - Canica: 5 veces sobre 5
        - Llave: 0 veces sobre 5
        - Tarjeta de crédito: 5 veces sobre 5
        - Mesa: 2 veces sobre 5
- Modelo V3
    - RAM usada: 496 kB (202,8 kB más que la teórica)
    - Latencia: 1166 ms (803 ms más que la teórica)
    - Clasificación de objetos
        - Pieza de lego: 2 veces sobre 5
        - Canica: 4 veces sobre 5
        - Llave: 0 veces sobre 5
        - Tarjeta de crédito: 4 veces sobre 5
        - Mesa: 2 veces sobre 5

Como se puede comprobar los datos teóricos no se corresponden con los obtenidos empíricamente. Los datos recopilados de la compilación del código en la placa de desarrollo son peores en cuanto a recursos usados y tiempo de clasificación. Aun así, en cualquier caso, los resultados en cuanto a uso de memoria y latencia son notablemente mejores que los obtenidos con el modelo usado hasta ahora.

Dentro de la mejoría de resultados, el modelo que mejor rendimiento ofrece en cuanto a la combinación de precisión, tiempo de clasificación y uso de memoria, es el Modelo V1. La Fig. 30 se corresponde con la matriz de confusión del modelo y el resumen completo de las características es el siguiente:

- Modelo V1
    - Modelo de *Transfer Learning* usado: MobileNetV2 0.05 con 64 neuronas en la capa densa final

- Modelo de color: RGB
- Latencia teórica: 454 ms
- Latencia real: 1188 ms (734 ms más que la teórica)
- RAM (int8, *EON compiler* activado): 283,4 kB
- RAM usada: 455 kB (171,6 kB más que la teórica)
- Precisión de entrenamiento: 91%
- Precisión de test: 86%
- Clasificación de objetos
  - Pieza de lego: 3 veces sobre 5
  - Canica: 5 veces sobre 5
  - Llave: 1 veces sobre 5
  - Tarjeta de crédito: 5 veces sobre 5
  - Mesa: 4 veces sobre 5

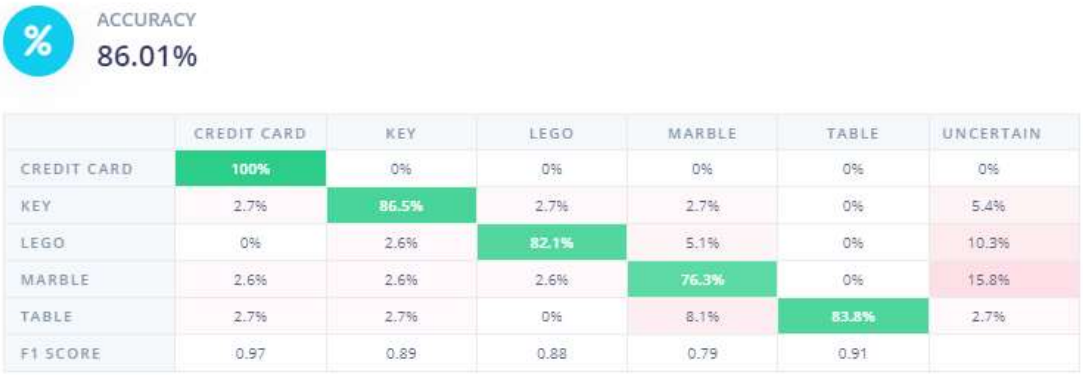

Figura 30: *Matriz de confusión del Modelo V1 tras el test. Fuente: Edge Impulse*

6. Aplicación final

Una vez completados los dos objetivos principales y abordadas las principales características sujetas a optimización, se da por terminado el desarrollo de la aplicación de visión por computador definido a lo largo de este trabajo, considerando finalizada la primera versión de la aplicación. Este apartado sintetiza las características de la aplicación, desde el algoritmo de clasificación de imágenes hasta el funcionamiento de la misma.

El modelo de clasificación de imágenes usado es el definido en el punto 6.4, cuyas características son las siguientes:

- Modelo V1
    - Modelo de *Transfer Learning* usado: MobileNetV2 0.05 con 64 neuronas en la capa densa final
    - Modelo de color: RGB
    - Latencia teórica: 454 ms
    - Latencia real: 1188 ms (734 ms más que la teórica)
    - RAM (int8, *EON compiler* activado): 283,4 kB
    - RAM usada: 455 kB (171,6 kB más que la teórica)
    - Precisión de entrenamiento: 91%
    - Precisión de test: 86%
    - Clasificación de objetos
        - Pieza de lego: 3 veces sobre 5
        - Canica: 5 veces sobre 5
        - Llave: 1 veces sobre 5
        - Tarjeta de crédito: 5 veces sobre 5
        - Mesa: 4 veces sobre 5

Dadas las características del *Transfer Learning* y de la definición propia del modelo, esta aplicación no permite un reentreno del modelo en función del uso de la misma, el único reentreno posible es mediante el uso de Edge Impulse y la ampliación del banco de imágenes, descargando, posteriormente, la librería del modelo actualizado.

Tal y como se ha mencionado a lo largo del trabajo, la precisión del modelo obtenida de forma empírica es difícilmente calculable y no se corresponde con los objetivos principales de este trabajo. Aun así se comprueba una gran precisión en los objetos *"Tarjeta de crédito*" y "*Canica*", clasificando correctamente las imágenes en el 100% de las veces a lo largo de diferentes pruebas. Dicha precisión permite validar el modelo de aplicación y su funcionalidad como aplicación de visión por computador.

Para la versión final de la aplicación se ha establecido el uso de la solución MultiCore. Se habilitan dos núcleos, el principal y el secundario, los cuales ejecutan, respectivamente, el código de captura y clasificación de imágenes y el código de movimiento de la prótesis.

El movimiento de la prótesis es ejecutado de forma habitual, pudiendo abrir, cerrar y rotar la pinza a la vez que, en una ejecución en paralelo, se toman imágenes y se clasifican. Una vez una imagen es clasificada dos veces consecutivas como un objeto particular, esta información es transmitida al código de movimiento. Si en ese instante el piloto decide cerrar la pinza, esta se cerrará al tamaño del objeto encontrado en la imagen. En la Fig. 31 se detalla el esquema de funcionamiento de la aplicación, teniendo en cuenta de manera general el movimiento de la prótesis (sin especificar todas las casuísticas de errores y modos de la prótesis ligados a la captación de señales por sensores). Esta aplicación permitiría, según su aplicación en el proyecto de ARM2u, facilitar la resolución de las diferentes pruebas de Cybathlon, sobre todo la prueba de *Clean Sweep*.

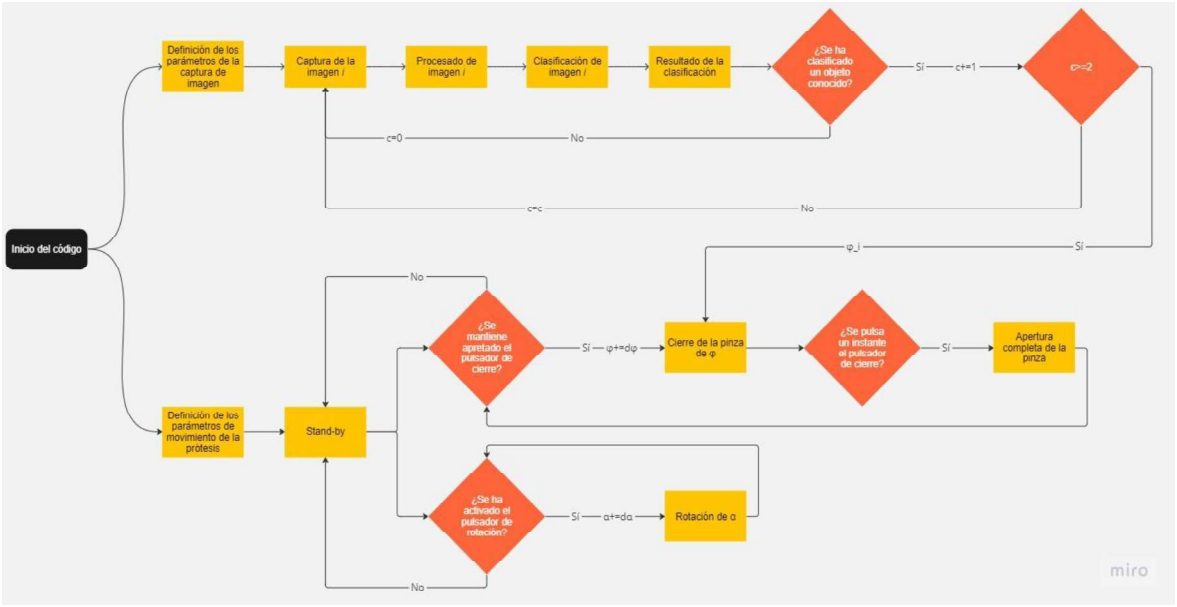

Figura 31: *Diagrama de flujo esquematizado del funcionamiento de la aplicación. Fuente: Propia*

Si bien los sensores son parámetros indispensables para el buen uso de la prótesis, su implementación en el banco de pruebas dificulta la operabilidad de la placa de desarrollo junto los servomotores. Es por esto que, para poder hacer diferentes pruebas, no se han usado todas aquellas líneas de código relacionadas con la captación de señales provenientes de sensores. De este modo, todas aquellas líneas relacionadas con la adquisición de señales de sensores y de control de estado de la prótesis han sido comentadas[8]. Ambos códigos, el del

[8] Comentar una línea de código implica que esta no se ejecuta pero es visible en el script.

MainCore y el del *SubCore1*, se encuentran en el Anexo como “Codigo_Main_Multicore” y “Codigo_Sub1_Multicore”, dentro de la carpeta “Codigos_Multicore”.

Este apartado marca el fin del desarrollo realizado sobre la aplicación de visión por computador descrito a lo largo de este Trabajo de Fin de Grado. Se han cumplido los dos objetivos principales así como el objetivo secundario basado en la optimización de la aplicación. El resultado final es una aplicación que permite el movimiento normal de la prótesis pero incorpora la clasificación de imágenes que permite la apertura y cierre de la pinza al tamaño de los objetos, siempre y cuando el piloto lo desee.

7. Diagrama de Gantt

Todos los proyectos deben estar planificados para poder realizar un seguimiento sobre el progreso y graduar el tiempo dedicado a cada etapa de los mismos. En la Fig.32 se muestra el diagrama de Gantt que corresponde a la planificación de este trabajo.

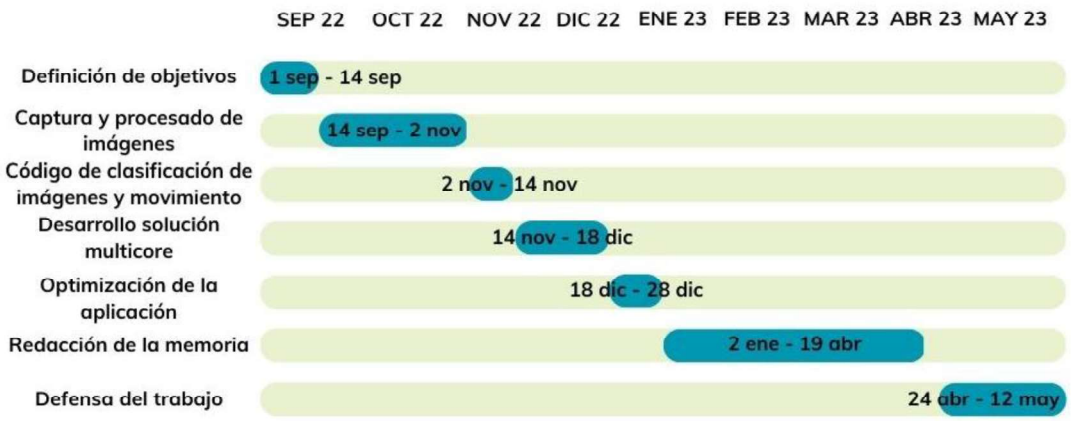

Figura 32: *Diagrama de Gantt. Fuente: Propia*

Las etapas descritas en el diagrama de la Fig. 32 son definidas de manera genérica debido a que ofrecen una previsión de cumplimiento de objetivos mucho más cercana a la realidad que en el caso de haberlas descrito de forma más específica (sobre todo por el poco conocimiento previo en algunos campos). Si bien la etapa de optimización se ubica principalmente al final del desarrollo técnico de la aplicación, se ha ido planificando y realizando en menor medida durante las etapas previas.

8. Estudio económico

El análisis económico es un componente imprescindible en cualquier proyecto ya que de él pueden depender algunas de las acciones que se deban tomar durante el transcurso del proyecto.

Para este proyecto se contemplan los gastos (incluyendo IVA) presentados en la Tabla 8.

	Cantidad	Unidades	Precio unitario (€/ud)	Coste total (€)
Material				
Sony Spresense				
Main Board	1	Unidades	63,07	63,07
Camera Module	1	Unidades	40,03	40,03
Extension Board	1	Unidades	40,14	40,14
Servomotores				
Servomotores	2	Unidades	11,99	23,98
Amortizaciones				
Ordenador PC	8	Meses	19,05[9]	152,38
Total Material				**319,60**
Mano de obra				
Ingeniero	320[10]	Horas	40	12800
Total Mano de obra				**12800**
			Total	**13119,60**

Tabla 8: *Estudio económico del proyecto. Fuente: Propia*

Este proyecto, así como todos aquellos proyectos de investigación y desarrollo, no genera ningún beneficio ya que los objetivos primarios no buscan una comercialización del producto final.

[9] El cálculo de la amortización se ha realizado asumiendo una amortización lineal mediante su vida útil, estimada en 7 años y un valor inicial de 1600 €

[10] Media de 10 horas a la semana durante 8 meses (setiembre-abril)

9. Estudio de impacto medioambiental

El estudio de impacto ambiental debe ser realizado para comprobar los efectos negativos que ha traído y/o puede traer un proyecto tanto en sus etapas de desarrollo y aplicación como en la de retirada del producto.

Teniendo en cuenta datos no cuantificables, este proyecto hace uso de componentes electrónicos, como la placa de desarrollo y sus módulos o el ordenador personal, y plásticos, encontrados, por ejemplo, en las carcasas de los servomotores, las placas de pruebas (*protoboards*) y demás componentes minoritarios. No se considera el uso de la prótesis, construida completamente en PLA, ya que no forma parte del desarrollo directo de este proyecto.

En la etapa de retirada del producto, que se debe establecer cuando las funcionalidades de los diferentes componentes impiden la correcta aplicación del proyecto, se da por supuesto que su ciclo de vida ha finalizado y, por lo tanto, deben ser llevados a un centro de recogida de residuos que no pueden ser tirados en los sistemas de recogida habituales, en el caso de Barcelona, deben ser llevados a un Punto Verde.

Durante las etapas tempranas del proyecto, el desarrollo y su aplicación, los diferentes componentes no impactan negativamente al medioambiente de forma directa, sino de forma indirecta. Dado que la huella medioambiental generada por la fabricación de los diferentes componentes se calcula para grandes producciones, se considera negligible en el caso de pocas unidades de los componentes. El ordenador personal no genera un impacto medioambiental directo pero sí de forma indirecta ya que consume electricidad que debe ser generada. Mediante una calculadora online [26], se estima que el consumo del ordenador personal usado para el desarrollo de este proyecto, tiene un consumo de 400 W que se traduce, teniendo en cuenta las 320 horas de desarrollo especificadas en el apartado 8, en un consumo eléctrico de 128 kWh. Según la Generalitat de Cataluña, consultando los datos de la Comisión Nacional de los Mercados y la Competencia (CNMC)[27], el CO2 equivalente emitido por cada kWh en 2022 fue de 259 gr. Teniendo en cuenta estos datos, la emisión de CO2 proveniente del uso del ordenador personal durante el desarrollo de este proyecto ha sido de 33,15 kg. Teniendo en cuenta que durante todo el desarrollo del proyecto la placa Sony Spresense ha recibido corriente únicamente cuando el ordenador estaba encendido, se puede asumir que las emisiones generadas por el consumo de electricidad de la placa de desarrollo están contempladas en el cálculo anterior. Dado que la aplicación de los resultados del proyecto tiene un carácter experimental y va ligado al uso de la prótesis, la estimación de consumo eléctrico y, por tanto, emisiones derivadas del uso de la placa de desarrollo, no se puede realizar.

10. Impacto social y de igualdad de género

Uno de los objetivos del desarrollo de esta aplicación, es apoyar al proyecto de ARM2u, un proyecto cuya voluntad es ofrecer prótesis electrónicas accesibles económicamente para los usuarios de las mismas. El uso de prótesis electrónicas mejora la adaptabilidad de estas al usuario, generando menos tasas de abandono en su uso. De este modo, la voluntad de mejora social del proyecto de ARM2u es la base del mismo y el desarrollo de la aplicación descrita a lo largo del trabajo puede mejorar las prestaciones de la prótesis, dando lugar a una mejor valoración de esta por parte de los usuarios y generando aún menos abandono.

Por lo que refiere al impacto sobre la igualdad de género, las funcionalidades y buen uso de la aplicación desarrollada y descrita en este trabajo no distinguen el género del usuario final, simplemente las habilidades personales de este para usarla. En cuanto al desarrollo en sí de la aplicación, el género no determina en ningún caso el éxito o fracaso de la aplicación final, son las capacidades de programación y, sobre todo, la dedicación al trabajo lo que determina el resultado de este.

Conclusiones

Este apartado pone punto y final al proyecto descrito en este trabajo, el desarrollo de una aplicación de visión por computador basada en Sony Spresense, habiendo cumplido con los objetivos descritos.

A lo largo del desarrollo de este Trabajo de Fin de Grado, se ha podido comprobar que la implementación de modelos con algoritmos complejos, como aquellos que permiten la visión por computador, no es una tarea sencilla.

Conseguir la correcta ejecución de una aplicación de visión por computador embebida en la placa de desarrollo Sony Spresense, era el primer objetivo de este proyecto. Dicho objetivo se ha cumplido, creando un código de captura y clasificación de imágenes.

El segundo objetivo se basa en acercar los resultados del primer objetivo al proyecto de ARM2u, apoyando el desarrollo de este. Analizando los diversos usos de la prótesis, se ha decidido implementar la visión por computador directamente al movimiento de esta. Dicha implementación ha sido exitosa, permitiendo una modulación del cierre de la pinza al tamaño del objeto resultante de la clasificación de imágenes.

El objetivo secundario que discurre paralelo a los dos primeros era el de mejorar los resultados de la aplicación obtenidos gracias a la compleción de los dos objetivos principales. Con la voluntad de cumplir este objetivo se ha iniciado el proceso de optimización de la aplicación cuyos resultados han sido claramente mejorados. A pesar de que cualquier proceso de optimización es iterativo, es decir se van mejorando diferentes aspectos de la aplicación de forma continua, el objetivo planteado en este proyecto se da por cumplido dada la mejoría en las funcionalidades y los resultados obtenidos tras las primeras iteraciones de este proceso.

La consecución de todos los objetivos planteados ha dado como resultado una aplicación de visión por computador embebida en la placa de desarrollo Sony Spresense, que permite la ejecución de un modelo de clasificación de imágenes y el uso de sus resultados en el propio procesador integrado en la placa.

Comentarios finales

El trabajar con código que anteriormente me era desconocido, como la captura y tratamiento de imágenes, me ha permitido expandir mis conocimientos sobre el lenguaje de programación así como de los algoritmos y funciones que permiten la clasificación de imágenes. La gran variedad de soluciones encontradas para un mismo problema ha obligado a una toma de decisiones constantes, enriqueciendo mi capacidad de decisión y dejando puertas abiertas a nuevas investigaciones.

El proyecto de ARM2u ha ocupado y sigue ocupando gran parte de mi vida como estudiante y ha marcado mi devenir profesional. Sin duda, me enorgullece poder acercar la tecnología de visión por computador al equipo, abriendo un nuevo campo de desarrollo y mejora de la prótesis.

Agradecimientos

A mi tutor, el profesor Manuel Moreno por su dedicación y guía en este trabajo, al equipo de ARM2u por su apoyo y posibilitar este trabajo, y a mi familia.

Bibliografía

Referencias bibliográficas

[1] NAVARRO SOLER Y., Placa programable Spresense de Sony. Desarrollo de una aplicación de visión por computador. Barcelona: 2022

[2] EDGE IMPULSE. *Data acquisition.* [https://docs.edgeimpulse.com/docs/edge-impulse-studio/data-acquisition, 16 de setiembre 2022].

[3] CYBATHLON. *Cybathlon.* [*https://cybathlon.ethz.ch/en*, 30 de diciembre 2022].

[4] SONY SPRESENSE. *Spresense*. [https://developer.sony.com/develop/spresense/, 14 de enero de 2023].

[5] SONY SPRESENSE. *Product specifications*. [https://developer.sony.com/develop/spresense/specifications, 14 de enero de 2023].

[6] SONY SPRESENSE. *Spresense documents*. [https://developer.sony.com/develop/spresense/docs/home_en.html, 14 de enero de 2023].

[7] SONY SPRESENSE. *CircuitPython Getting Started Guide. 6.2 Editing code* [https://developer.sony.com/develop/spresense/docs/circuitpython_set_up_en.html#_editing_code, 21 de setiembre 2022].

[8] NUTTX. *NuttX Documentation* [https://nuttx.apache.org/docs/latest/, 27 de diciembre de 2022].

[9] SONY SPRESENSE. *Spresense Arduino Library Developer Guide, 2.11 MulticoreMP Library*. [https://developer.sony.com/develop/spresense/docs/arduino_developer_guide_en.html#_mp_library, 21 de noviembre 2022].

[10] ZHU M. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. [https://arxiv.org/abs/1704.04861, 30 de diciembre de 2022].

[11] MICROSOFT.*Representación IEE del punto flotante*. [https://learn.microsoft.com/es-es/cpp/build/ieee-floating-point-representation?view=msvc-170, 8 de enero 2023]

[12] CHERUKURI, R., MATHWORKS. *What Is int8 Quantization and Why Is It Popular for*

Deep Neural Networks? [https://es.mathworks.com/company/newsletters/articles/what-is-int8-quantization-and-why-is-it-popular-for-deep-neural-networks.html, 28 de diciembre 2022].

[13] NEURAL NETWORK DISTILLER. *Quantization, "Conservative" Quantization: INT8*. [https://intellabs.github.io/distiller/quantization.html, 28 de diciembre 2022].

[14] TENSORFLOW. *TensorFlow Lite 8-bit quantization specification*. [https://www.tensorflow.org/lite/performance/quantization_spec?hl=en, 28 de diciembre 2022].

[15] CPLUSPLUS. *Function memcpy*. [https://cplusplus.com/reference/cstring/memcpy/, 4 de octubre 2022]

[16] EDGE IMPULSE. *ei_impulse_result_t*. [https://docs.edgeimpulse.com/reference/c++-inference-sdk-library/structs/ei_impulse_result_t, 7 de octubre 2022]

[17] EDGE IMPULSE. *signal_t*. [https://docs.edgeimpulse.com/reference/c++-inference-sdk-library/structs/signal_t, 7 de octubre 2022]

[18] SONY SPRESENSE. *Spresense Arduino Library Developer Guide, 2.2 Camera Library*. [https://developer.sony.com/develop/spresense/docs/arduino_developer_guide_en.html#_camera_library, 17 de setiembre 2022].

[19] SONY DEVELOPER WORLD. *Spresense Arduino Library. clipAndResizeImagebyHW()* [https://developer.sony.com/develop/spresense/developer-tools/api-reference/api-references-arduino/group__camera.html#ga3df31ea63c3abe387ddd1e1fd2724e97, 24 de setiembre de 2022]

[20] GITHUB. SONYDEVWORLD. *Spresense-arduino-compatible, Camera.cpp* [https://github.com/sonydevworld/spresense-arduino-compatible/blob/master/Arduino15/packages/SPRESENSE/hardware/spresense/1.0.0/libraries/Camera/Camera.cpp, 17 de setiembre 2022].

[21] MICROSOFT. REFERENCIA DEL LENGUAJE C++. *Operadores de desplazamiento a la izquierda y a la derecha (<< y >>)* [https://learn.microsoft.com/es-es/cpp/cpp/left-shift-and-right-shift-operators-input-and-output?view=msvc-170, 13 de octubre 2022].

[22] MITHUN DAS. *Oil Tank Measurement and Delivery Improvement Using Computer Vision.* [https://docs.edgeimpulse.com/experts/prototype-and-concept-projects/oil-tank-

gauge-monitoring, 25 octubre 2022].

[23] EDGE IMPULSE. *Image Classification Wio Terminal.* [https://forum.edgeimpulse.com/t/image-classification-wio-terminal/4775, 18 de noviembre 2022].

[24] CHRIS HEWETT. *A true RGB565 colour picker.* [*https://chrishewett.com/blog/true-rgb565-colour-picker/#:~:text=RGB565%20is%20used%20to%20represent,pages%20use%20to%20specify%20colours., 12 de diciembre 2022*].

[25] ALEX-EEE. *example-signal-from-rgb565-frame-buffer. [https://github.com/edgeimpulse/example-signal-from-rgb565-frame-buffer/blob/master/source/main_using_cutout.cpp, 12 de diciembre de 2022*].

[26] GEEKNETIC. *CALCULADORA DE FUENTES DE ALIMENTACIÓN. [https://www.geeknetic.es/calculadora-fuente-alimentacion/, 10 de abril de 2023*].

[27] GENCAT. *Factor de emisión de la energía eléctrica: el mix eléctrico*. *[https://canviclimatic.gencat.cat/es/actua/factors_demissio_associats_a_lenergia/#:~:text=El%20mix%20de%20la%20red%20el%C3%A9ctrica%20espa%C3%B1ola%20publicado%20por%20la,259%20g%20CO2eq%2FkWh., 10 de abril de 2023*].