Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Sistema de Detección y Alertamiento de Somnolencia en Conductores a Partir del Uso de Machine Learning en un Sistema Empotrado

T E S I S

que para obtener el título de:

Licenciatura en Ingeniería en Sistemas Automotrices

presenta:

Sergio Ángel Rosas González

Asesores:

M.C. Nicolás Quiroz Hernández

M.C. Leonardo Delgado Toral

Puebla Pue. Junio 2025

Dedicado a.....

A mis padres, Rosario y Sergio, por su sacrificio y apoyo incondicional. A mi abuela Victoria, por ser mi ejemplo de fortaleza y sabiduría. Finalmente a Alan y Farias, mis mentores y amigos. !Gracias a todos ustedes!

Agradecimientos

Agradezco profundamente a mis asesores, M.C. Nicolás Quiroz Hernández y M.I. Leonardo Delgado Toral, por su invaluable apoyo, guía y dedicación a lo largo de la realización de este proyecto de tesis sobre el Sistema de Detección y Alertamiento de Somnolencia en Conductores a partir del uso de Machine Learning en un Sistema Empotrado. Su conocimiento y compromiso fueron esenciales para el desarrollo de esta investigación.

A mis padres, Cristina del Rosario González Bravo y Sergio Rosas Hurtado, quienes con su amor, paciencia y constante apoyo hicieron posible que llegara hasta este punto. No hay palabras suficientes para agradecerles todo lo que han hecho por mí.

A mi abuela Victoria Bravo Villarruel, cuya sabiduría y cariño han sido siempre una fuente de inspiración, y a mi hermana Alejandra Rosas González, por su apoyo incondicional y su confianza en mí. A toda mi familia, que de una u otra manera ha estado presente en este camino, les agradezco de corazón.

Finalmente, a mis grandes amigos Alan Jiménez, Eduardo Farías y Gerardo Benítez, quienes me acompañaron en cada paso de esta aventura universitaria, su amistad y ánimo fueron fundamentales para superar los desafíos que se presentaron en el camino. Gracias por estar siempre ahí.

Índice general

Índice de figuras

Índice de tablas

Abreviaturas

SP	*Serial Port*, Perfíl de Puerto Serial
AS	*Android Studio*
JDK	*Java Development Kit*, Herramienta de Desarrollo Java
API	*Application Programming Interface*, Interfaz de Programación de Aplicaciones
ML	*Machine Learning*, Aprendizaje de MáquinaAutomático
AI	*Artificial Intelligence*, Inteligencia Artificial
DL	*Deep Learning*, Aprendizaje Profundo
ID	*Identifier*, Identificador
UUID	*Universally Unique Identifier*, Identificador Único Universal
RFCOMM	*Radio Frequency Communications*, Comunicaciones de Radio Frecuencia
SDPTOOL	*Service Discovery Protocol Tool*, Herramienta de Servicio de Descubrimiento
OS	*Operative System*, Sistema Operativo
BLE	*Bluetooth* Low Energy, Bluetooth de Baja Energía
Wi-Fi	*Wireless Fidelity*, Fidelidad Inalámbrica
IoT	*Internet of Things*, Internet de las cosas
SBC	*Single Board Computer*, Computadores de una sola placa
SADS	Sistema de Alertamiento y Detección de Somnolencia
JVM	*Java Virtual Machine*, Máquina Virtual de Java
SDS	Sistemas de Detección de Somnolencia
ADC	*Analogic-Digital Conversor*, Conversor Analógico-Digital
DAC	*Digital-Analogic Conversor*, Conversor Digital-Analógico
GPIO	*General Purpose Input-Output*, Entrada y Salida de Propósito General.

Resumen

La inteligencia artificial está presente en casi todas las áreas de la tecnología, y la industria automotriz no es la excepción. El desarrollo de sistemas de visión artificial presentes en el auto desempeñan acciones como la conducción automática o la detección de peatones. Sin embargo, los sistemas de monitoreo al conductor siguen siendo de gran estudio ya que algunos métodos son intrusivos y no permiten la conducción adecuada y segura. Una alternativa a este tipo de sistemas es emplear la visión artificial como método no intrusivo.

Este proyecto presenta el desarrollo de un sistema de detección y alertamiento de somnolencia en conductores empleando técnicas del aprendizaje automático enfocado a sistemas de visión. Este sistema realiza detecciones de somnolencia y emite alertas sonoras, ejecutando al mismo tiempo una llamada telefónica y una alerta en forma de voz como métodos de alerta en caso de la disponibilidad, o no, de una conexión a internet.

La implementación recae en el uso de un computador de una sola tarjeta para el procesamiento del software, una cámara web para la captura de vídeo en tiempo real para el monitoreo y detección de somnolencia y el uso de un *smart phone* para compartir el número telefónico del conductor a monitorear.

Capítulo 1

Introducción

En este capítulo se aborda la problemática que genera la somnolencia en conductores. Además, se presenta una breve descripción del prototipo a desarrollar en este proyecto, los objetivos y la justificación del mismo. Por último, se explican algunos antecedentes que abordan la solución de dicha problemática mediante sistemas de detección facial y monitoreo al conductor. Finalmente, se presenta una descripción de cada capítulo de este trabajo de tesis.

1.1. Introducción

En la actualidad, el automóvil y el transporte terrestre se han convertido en una parte esencial de la vida del ser humano, pero así como se ha incrementado la manufactura y el uso del automóvil, también se han incrementado los accidentes viales a causa de múltiples factores. Según una estimación de la Organización Mundial de la Salud (OMS), alrededor de 1.35 millones de personas en todo el mundo han muerto a causa de traumatismos causados por el tránsito cada año [1].

Una de las causas más importantes de accidentes viales es la somnolencia en conductores, y se menciona que conductores que no son capaces de descansar más de 4 horas, es 10.2 veces más probable que ocurra un accidente [2]. De acuerdo con algunos reportes, en EU más de 71,000 personas sufren lesiones cada año por accidentes de tránsito provocados por la somnolencia en conductores [3].

A hoy en día, las expresiones faciales ofrecen profundos enfoques sobre las condiciones psicológicas y fisiológicas del cuerpo humano y últimamente, gracias al desarrollo de la visión artificial, se han desarrollado sistemas inteligentes que detectan emociones a partir de patrones faciales [4], la detección automática del dolor [5] o la predicción de la personalidad [6] partiendo del procesamiento de imágenes con rostros humanos, por nombrar algunos ejemplos.

Así mismo, la idea del proyecto parte de la problemática causada por la somnolencia en conductores, ya que esta provoca accidentes fatales en caminos y carreteras. Debido a la falta de normativas viales y sistemas que detecten e informen a pasajeros sobre el estado del conductor [7], el número de accidentes viales sigue siendo muy alto. De acuerdo con el informe más reciente del Secretariado Técnico del Consejo Nacional para la Prevención de Accidentes (STCONAPRA) de México, en el año 2019 fallecieron 14 mil 673 personas por lesiones a causa de accidentes viales [8] y el número de defunciones por siniestros viales en 2020 fue de 13 mil 630 [9].

1.2. Objetivos

1.2.1. General

Desarrollar un sistema para la detección de signos o patrones faciales de somnolencia y alertamiento al conductor mediante el uso de algoritmos de Aprendizaje de Máquina.

1.2.2. Específicos

1. *Investigar el estado del arte de la detección de somnolencia a partir del aprendizaje de máquina.*

2. *Determinar e implementar un algoritmo adecuado disponible para la detección de somnolencia.*

3. *Investigar e Implementar una API que realice llamadas a teléfonos inteligentes vía servicio de internet y otra API que permita ejecutar alertas en forma de voz.*

4. *Diseñar una aplicación móvil que comparta el número telefónico del usuario que conduce el automóvil, a la API de de llamadas.*

5. *Examinar y establecer el hardware del sistema (cámara, tarjeta de desarrollo).*

6. *Ejecutar el código de detección de somnolencia y simultáneamente la API de llamadas y alerta sonora estableciendo una comunicación Bluetooth entre el teléfono inteligente y la tarjeta de desarrollo.*

7. *Probar el desempeño del prototipo.*

1.3. Justificación

Algunos de los trabajos más recientes sobre sistemas para la detección de somnolencia no han sido implementados simulando entornos reales, como el trabajo de Shinfeng D. Lin et al. [10], que a pesar de que su sistema detecta variaciones de luz, no contempla la integración en un sistema vehicular. La investigación de Jang Woon et al. [11] contempla la detección de somnolencia en patrones oculares y es interesante su integración en un sistema empotrado usando una tarjeta de desarrollo de bajo presupuesto. La implementación de algunas de estas características permite desarrollar un diseño de *hardware* y *software* con el fin de observar el desempeño del mismo en un sistema para una posible integración vehicular. Llevar a cabo la tarea de alertamiento al conductor, o a los pasajeros, es una aplicación que muchos estudios no desarrollan después de haber concluido con la detección de somnolencia.

El diseño y la implementación de este sistema alentará a que la sociedad esté más consciente que al monitorizar variables que en el pasado era impensable medir, se pueden resolver problemas que la innovación genera, como es el caso del automóvil y los accidentes viales. Así mismo, el estudio futuro de este tipo de investigaciones exhortará a las grandes industrias manufactureras de automóviles a que integren mejores medidas y sistemas de seguridad similares que apoyen a la disminución de accidentes, pérdidas económicas y bienes materiales para concluir en una mejor convivencia vial.

1.4. Descripción

El proyecto se centra en el desarrollo e implementación de un Sistema de Detección y Alertamiento de Somnolencia (SDAS), cuyo propósito es identificar signos de somnolencia en el rostro del conductor y generar alertas. Para ello, el sistema emplea una cámara instalada en el interior del vehículo y un parlante que emite alertas sonoras. La detección se basa en técnicas de procesamiento de imágenes y aprendizaje automático (*Machine Learning*), que permiten analizar las expresiones faciales del conductor y activar el mecanismo de alertamiento en caso de somnolencia.

El funcionamiento general del SDAS realiza la detección de somnolencia y posteriormente emite un par de alertas, una en forma de llamada telefónica y otra alerta sonora en forma de voz que es emitida por un parlante o el estéreo del auto. Se utiliza una aplicación móvil para teléfonos inteligentes que permite editar el número telefónico al que se le pretende realizar la llamada usando una comunicación *Bluetooth*.

Para desarrollar el SDAS, primero se realizó una investigación que aborda el tema de detección de somnolencia facial y algoritmos de prototipos desarrollados y publicados en artículos científicos.

Seguidamente, a través de un análisis, se hizo la selección y prueba de un algoritmo de detección de somnolencia de código abierto disponible, considerando los patrones o rasgos faciales más importantes para determinar si el conductor se encuentra dispuesto a conducir correctamente.

Después, se procedió a la selección del *hardware* mediante un análisis de cada componente que cumpla con los requisitos del algoritmo de detección de somnolencia, como la tarjeta de desarrollo que alojará y ejecutará el *software*, el sensor para la captura de imágenes y un parlante para emitir alertas. Al haber terminado con la selección, se realizaron pruebas del *software* y sistema operativo en la tarjeta seleccionada.

Posteriormente, se diseñó la arquitectura del software, en la cual se programaron, integraron y ejecutaron las aplicaciones correspondientes al algoritmo de detección de somnolencia, la comunicación *Bluetooth*, el alertamiento mediante llamadas telefónicas y la generación de alertas en forma de voz.

Simultáneamente, se implementó una aplicación móvil utilizando la plataforma de desarrollo *Android Studio*, destinada a dispositivos con sistema operativo *Android*. El propósito de esta aplicación consiste en la transmisión del número de teléfono del usuario al sistema que ejecuta el *software* de alertamiento y detección de somnolencia a través de la conexión *Bluetooth*. Esta funcionalidad permite la realización de llamadas al número previamente ingresado desde la aplicación móvil.

Finalmente, se llevó a cabo la prueba del prototipo en un ambiente controlado simulando un entorno real en el automóvil para mostrar qué tan funcional es el prototipo para este tipo de detección, realizando análisis de desempeño cuantitativo del SDAS.

El diagrama de bloques 1.1 muestra el sistema desarrollado:

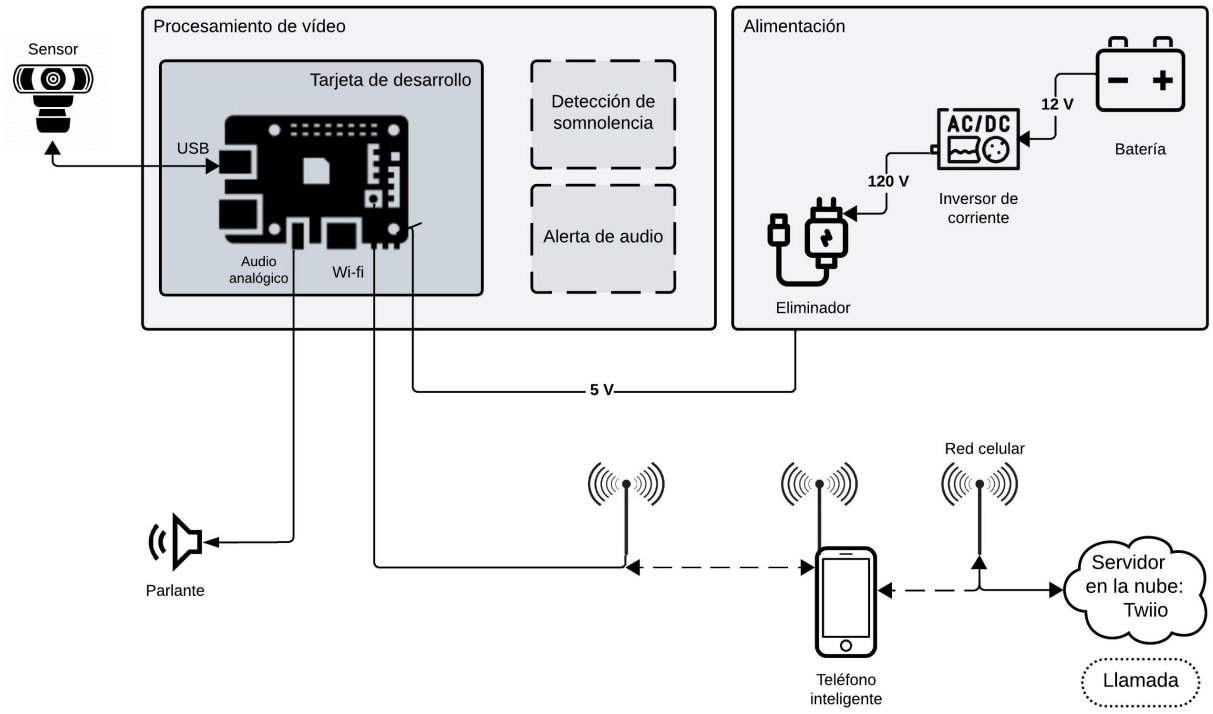


Figura 1.1: Diagrama de bloques del sistema propuesto.

En el diagrama de la figura 1.1 se observa que el sistema se compone de dos bloques importantes: el de procesamiento de vídeo y el bloque de alimentación. El primero contempla el *hardware* y *software* que se encarga de procesar el vídeo que el sensor proporciona. Los bloques de líneas punteadas representan las aplicaciones de *software* que ejecuta la tarjeta de desarrollo. Además, se muestra el tipo de conexión que se emplea para conectar otros

dispositivos que realizan la tarea de alertamiento, por ejemplo: audio analógico para el parlante que emite alertas en forma de voz y el uso de la comunicación *Wi-fi* para realizar una llamada telefónica donde la tarjeta ejecuta una solicitud en un servidor en la nube que se conecta a una red celular para transmitir la llamada al teléfono inteligente.

El bloque de alimentación del sistema usa elementos eléctricos que representan la toma de energía de la batería del auto, usando un inversor de corriente y, finalmente, un eliminador que provee de energía a la tarjeta de desarrollo, donde la cámara se alimenta directamente de la tarjeta.

Por otro lado, en la figura 1.2 se muestra un diagrama de flujo que representa la estructura general del *software* diseñado para ejecutar la tarea de alertamiento cuando se detecta una expresión facial de somnolencia excesiva.

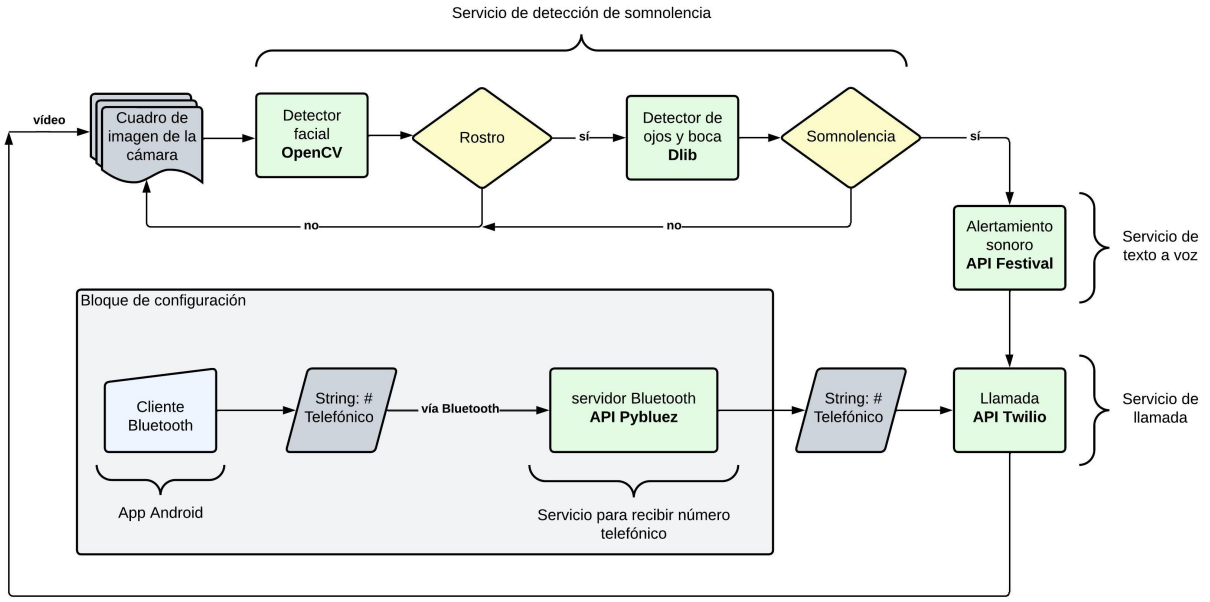


Figura 1.2: Diagrama de flujo del sistema de alertamiento y detección de somnolencia del apartado de *software*.

La figura 1.2 muestra que el *software* consta de 4 servicios y una aplicación para sistemas operativos *Android*. Cada servicio representa una aplicación que se ejecuta en el sistema empotrado, siguiendo la lógica de cada etapa. El proceso inicia con la entrada de vídeo en tiempo real desde una cámara, que se traduce en una secuencia de cuadros de imagen. Cada cuadro se somete a un algoritmo de detección facial que determina la

presencia de un rostro. Si se detecta un rostro, se procede a la detección de elementos faciales, como ojos y boca, para evaluar posibles síntomas de somnolencia, como ojos cerrados o boca abierta. Si la lógica confirma la somnolencia, se activan los servicios de texto a voz y llamadas como métodos de alertamiento. Una vez completado este proceso de alerta, se reanuda la captura de cuadros de imagen para seguir monitorizando al conductor.

Es relevante destacar que para realizar las llamadas telefónicas, se debe enviar un número telefónico al servicio de llamadas en la tarjeta de desarrollo. Por ello, se implementó un proceso de configuración que permite editar y enviar, vía *Bluetooth*, un número telefónico desde una aplicación para teléfonos inteligentes con sistema operativo *Android* a la tarjeta que ejecuta el servicio de llamadas.

Estructura

La estructura de este trabajo de tesis se compone de 5 capítulos y un apéndice. El capítulo 1 presenta una breve introducción donde se aprecia la problemática que genera la somnolencia en conductores y se incluye un apartado que muestra algunos antecedentes de sistemas propuestos por investigadores que pretenden dar una solución al problema que ocasiona la somnolencia durante la conducción del automóvil, usando tecnología de visión artificial. Además, se presenta la descripción del prototipo desarrollado en este proyecto de tesis usando un diagrama de bloques y un diagrama de flujo; este último representa el software del sistema.

En el capítulo 2 se despliega teoría necesaria para comprender los capítulos posteriores y el desarrollo de la elaboración del SDAS. Del mismo modo, se define el estado del arte de temas relacionados con la investigación y diseño de sistemas de detección de somnolencia, como el uso de diversas técnicas y algoritmos empleados para la detección de rostros y detección de elementos del mismo (ojos, boca, nariz), propios de la visión por computador y el aprendizaje automático como ramas y aplicaciones de la inteligencia artificial. También se presenta teoría relacionada con la implementación de estos sistemas en computadoras de una sola tarjeta y sistemas empotrados, lenguajes de programación empleados y características de la comunicación inalámbrica *Bluetooth* como vía para intercambiar datos

entre dispositivos electrónicos.

El capítulo 3 expone el desarrollo práctico del sistema de detección de somnolencia. Se divide en dos apartados: *hardware* (3.2) y *software* (3.3).

El apartado de *hardware* (3.2), da a conocer una justificación de uso y las características de los componentes electrónicos principales que forman parte del sistema de detección de somnolencia, como el computador de una sola tarjeta o componentes integrados adicionalmente, la cámara como sensor, el parlante y el teléfono inteligente empleado como enrutador para obtener acceso a Internet. En este capítulo se presenta un análisis que responde a la elección de estos componentes.

Por otro lado, en el apartado de *software* (3.3) se describe la función y partes importantes de los códigos relacionados con la programación de los servicios o aplicaciones en lenguaje de programación *Python*, como: el servicio de detección de somnolencia, el servicio de llamadas vía internet, el servicio de alertas en forma de voz y el servicio de comunicación *Bluetooth*. También se muestra el apartado relacionado con la programación de la aplicación para teléfono inteligente que envía el número telefónico al sistema de detección, desarrollado en la plataforma *Android Studio* en lenguaje de programación *Kotlin*. También, se hace mención del programa que permite la vinculación de dispositivos *Bluetooth* o la ejecución automática del sistema, programados en lenguaje *Bash*.

El capítulo 4 presenta una serie de evidencias y análisis de los resultados obtenidos durante la evaluación de desempeño del prototipo.

Finalmente, el capítulo 5 plantea las conclusiones y otros posibles enfoques para abordar este proyecto desde otra perspectiva, impulsando un trabajo a futuro.

1.5. Antecedentes: Trabajo relacionado

Esta sección presenta un resumen de los antecedentes, técnicas y trabajos previos de diferentes proyectos relacionados con la detección de fatiga y somnolencia facial. Se darán a conocer los elementos más importantes de cada diseño científico para informar al lector cómo y por qué del proceso de la elaboración de un sistema detector de somnolencia, desde su algoritmo hasta la implementación.

1.5.1. Detección en tiempo real de fatiga en conductores basado en comportamiento facial con enfoques de aprendizaje de automático

En el estudio de Sanjay Dey et al. [12], presenta el diseño de un detector de somnolencia que detecta características faciales que contemplan el monitoreo de los ojos, la nariz y la boca. El área de oportunidad que este estudio menciona es la presencia de valores bajos en las métricas de evaluación, como la baja especificidad, y que el sistema puede marcar alerta aún cuando no se tiene una detección real de somnolencia, y que no fue probado en un ambiente real con *hardware* móvil, sino que las pruebas se establecieron sólo en el computador de escritorio.

La figura 1.3 describe, de forma general, el algoritmo que se usó para el procesamiento de detección de somnolencia en conductores.

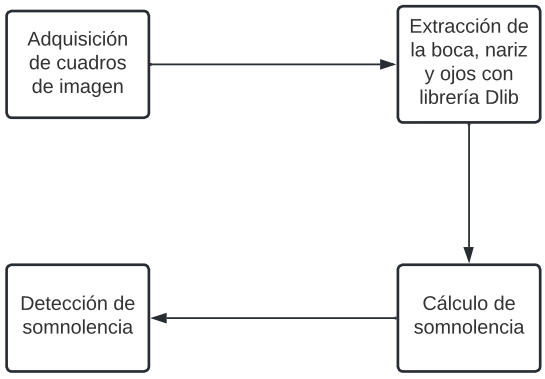
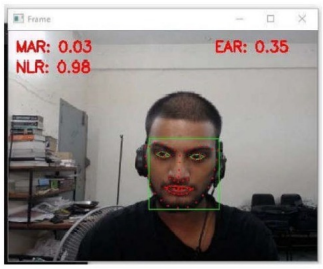
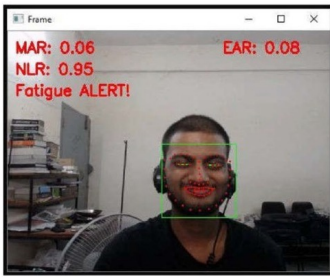


Figura 1.3: Diagrama a bloques del algoritmo del sistema propuesto por el autor.

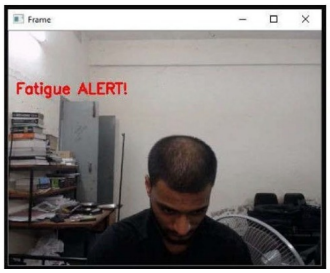
La figura 1.4 muestra las evidencias del detector. Donde la imagen 1.4a muestra condiciones normales, la imagen 1.4b alerta de fatiga por pestañeo. La imagen 1.4c alerta de fatiga por cabeceo y la imagen 1.4d alerta de fatiga por bostezo.



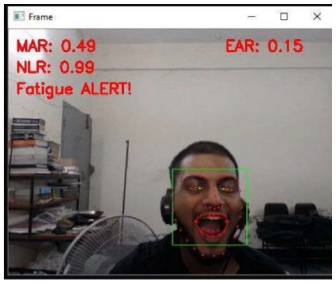
(a) Condiciones normales.



(b) Alerta por pestañeo.



(c) Alerta por cabeceo.



(d) Alerta por pestañeo.

Figura 1.4: Cuadro de detección facial de ojos, nariz y boca.

1.5.2. Algoritmo de detección de somnolencia en tiempo real para sistemas de monitoreo del estado del conductor

La investigación de Jang Woon et al. [11] presenta un algoritmo de detección de somnolencia implementado en una tarjeta de desarrollo i.MX6QUAD, junto con una cámara infrarroja para la detección facial en condiciones de baja iluminación y en tiempo real. El estudio se centra en la precisión y la velocidad de detección del sistema, evitando el uso de algoritmos de alta complejidad computacional debido a las limitaciones de la tarjeta de desarrollo, la cual no cuenta con una *GPU* dedicada. No obstante, se señala que la precisión del sistema está influenciada por la complexión fisiológica de los ojos de los individuos y que no se consideró la detección de bostezos ni la inclinación de la cabeza.

En la figura 1.5 se observa el diagrama de flujo del algoritmo para la detección de somnolencia que el autor propone.

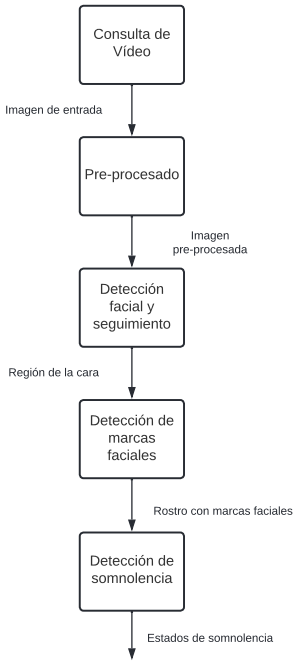
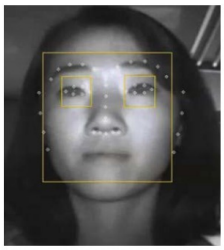
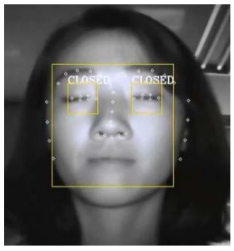


Figura 1.5: Proceso de detección del algoritmo.

La figura 1.6 especifica los posibles estados de detección de somnolencia basados exclusivamente en la detección ocular. En particular, la figura 1.6a muestra la detección del estado de ojos abiertos, mientras que la figura 1.6b ilustra la detección de ojos cerrados.



(a) Detección del estado ojos abiertos.



(b) Detección del estado ojos cerrados.

Figura 1.6: Estados de somnolencia en los ojos.

1.5.3. Reconocimiento de ojos somnolientos para la detección de somnolencia

El trabajo de Shinfeng D. Lin et al. [10] introduce un sistema de detección de somnolencia que monitoriza el estado de los ojos (abierto/cerrado) usando una metodología que controla la variación de iluminación que pueda influir en la precisión del sistema. A pesar de que este sistema detecta variaciones de iluminación, no fue probado en condiciones de baja iluminación (oscuridad) y es óptimo para sistemas de bajo poder computacional.

La figura 1.7 demuestra el proceso propuesto que realiza el algoritmo de detección de somnolencia, representado en un diagrama de flujo.

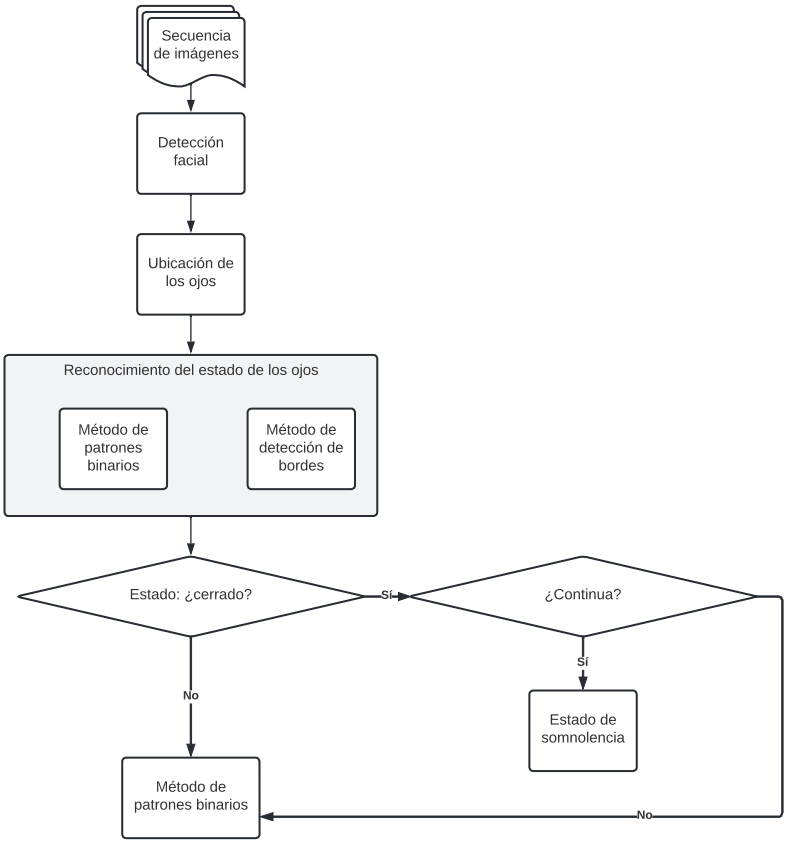


Figura 1.7: Diagrama de flujo del método propuesto por los autores.

En la figura 1.8 la imagen 1.8a describe solamente la detección de la región de la cara y los ojos en condiciones normales. La imagen 1.8b exhibe el monitoreo de un individuo

que se encuentra usando anteojos.

(a) Detección de la región facial; condiciones normales.

(b) Detección de la región facial con anteojos.

Figura 1.8: Resultado de la detección de la región de la cara y los ojos.

1.5.4. Sistema de detección de somnolencia usando aprendizaje profundo

Avigyan Sinha et al. [13] realizó una comparación de algoritmos y así desarrollar un sistema de detección que, después de la detección de signos de somnolencia, emite una alarma para alertar al conductor y este tome las acciones necesarias. Se usó una cámara infrarroja y el entrenamiento del algoritmo fue con un conjunto de datos enfocado en condiciones de baja iluminación y detección nocturna. El autor menciona que es posible desarrollar un proyecto más robusto en cuanto al factor de iluminación; implementando una mejor resolución de cámara puede ayudar a aumentar el desempeño del sistema. Añadió que se pueden mejorar apartados como un canal de audio junto con la velocidad de cuadros por segundo del vídeo para crear un aprendizaje de máquina multimodal y mejorar el desempeño del prototipo.

El diagrama de la figura 1.9 representa el proceso algorítmico que los autores de esta obra proponen.

Figura 1.9: Sistema propuesto.

Figura 1.10: Cuadros del conjunto de datos.

La figura 1.10 muestra capturas del conjunto de datos; imágenes de individuos presentando signos de somnolencia en condiciones de baja iluminación empleando una cámara infrarroja.

1.5.5. Diseño de un sistema de detección de somnolencia en tiempo real usando dlib

El estudio de Shruti Mohanty et al. [14] propone un sistema de detección de somnolencia basado en el algoritmo *Dlib*, que monitorea en tiempo real al conductor mediante video, considerando el estado de los ojos y la boca. Los resultados indican que la eficiencia del algoritmo es mayor en condiciones de buena iluminación, mientras que su precisión disminuye en videos en tiempo real. Además, se sugiere como trabajo futuro la incorporación de condiciones de iluminación adversas y la detección del movimiento de cabeceo.

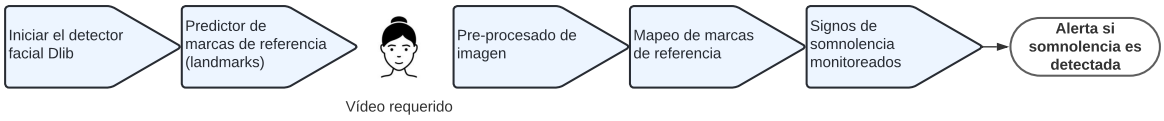
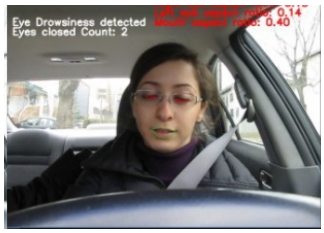


Figura 1.11: Diagrama a bloques del sistema de detección propuesto.

La figura 1.12 muestra los resultados de detección de somnolencia que se obtuvieron durante el estudio. Donde la imagen 1.12a se detecta somnolencia por bostezo, la imagen 1.12b es el estado de somnolencia por pestañeo, la figura 1.12c es otra posibilidad de

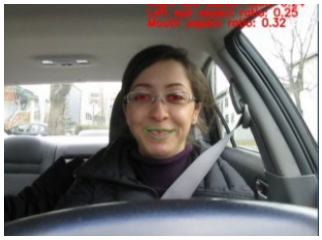
detección de somnolencia por bostezo y finalmente la figura 1.12d muestra la detección de la región facial (ojos y boca) en condiciones normales (sin somnolencia).

(a) Detección de ojos cerrados y boca abierta.



(b) Detección de ojos cerrados y boca cerrada.

(c) Detección de ojos abiertos y boca abierta.



(d) Detección de ojos cerrados y boca cerrada.

Figura 1.12: Detección facial de los 4 estados posibles para la detección de somnolencia.

Capítulo 2

Marco Teórico

En este capítulo se expone una serie de conceptos, definiciones, análisis y teoría relacionada con la elaboración de este proyecto, con el fin de que el lector tenga conocimiento y entienda de manera más concreta el desarrollo y los apartados de esta obra.

Primero se presentan conceptos del *hardware* empleados para sistemas de detección, como sistemas empotrados y redes. Luego, se define el concepto general de somnolencia desde una perspectiva médica, abordando sus tipos y causas. Posteriormente, se muestra un enfoque que aborda sistemas inteligentes, en específico los de visión por computador para la detección de objetos, así como las técnicas del aprendizaje automático que permiten desarrollar sistemas de detección.

Finalmente, se analizan los conceptos y herramientas utilizados en el desarrollo del SDAS en este trabajo de tesis.

2.1. Sistemas empotrados

Un sistema empotrado es un sistema electrónico que incluye uno o más microcontroladores y está configurado para desempeñar una aplicación dedicada especial. Un sistema empotrado se caracteriza por ser un dispositivo con entradas y salidas, permitiendo interactuar con el mundo real [15].

El software que controla el sistema programado o fijado en la memoria de sólo lectura *ROM*, no está disponible para el usuario del dispositivo. A pesar de ello, el mantenimiento del software es importante.

Generalmente, los sistemas embebidos desempeñan una sola función y resuelven un rango limitado de problemas para ser empleados solamente con un propósito, sin ser posible la alteración de este por el usuario final.

Estos sistemas son estrictamente limitados ya que deben operar dentro de unos parámetros de desempeño definidos por el desarrollador.

Deben operar en tiempo real ya que deben garantizar una respuesta de operación hasta en el peor de los casos; desde que la información de entrada está disponible, hasta cuando esta información sea procesada [16].

Existen dos clasificaciones de sistemas embebidos, el sistema transformativo reúne datos de las entradas y toma decisiones. Estas afectan el ambiente a su alrededor mediante el control de actuadores.

Por otro lado, un sistema reactivo recolecta datos de forma continua y proporciona salidas también de forma continua.

2.1.1. Arquitectura de computadoras

Una computadora combina un procesador, memoria de acceso aleatorio (RAM), memoria de sólo lectura (ROM) y puertos de entrada y salida (I/O). El bus más común corresponde a la configuración de arquitectura de tipo *Von Newmann* mostrada en el diagrama de la figura 2.1 [15]:

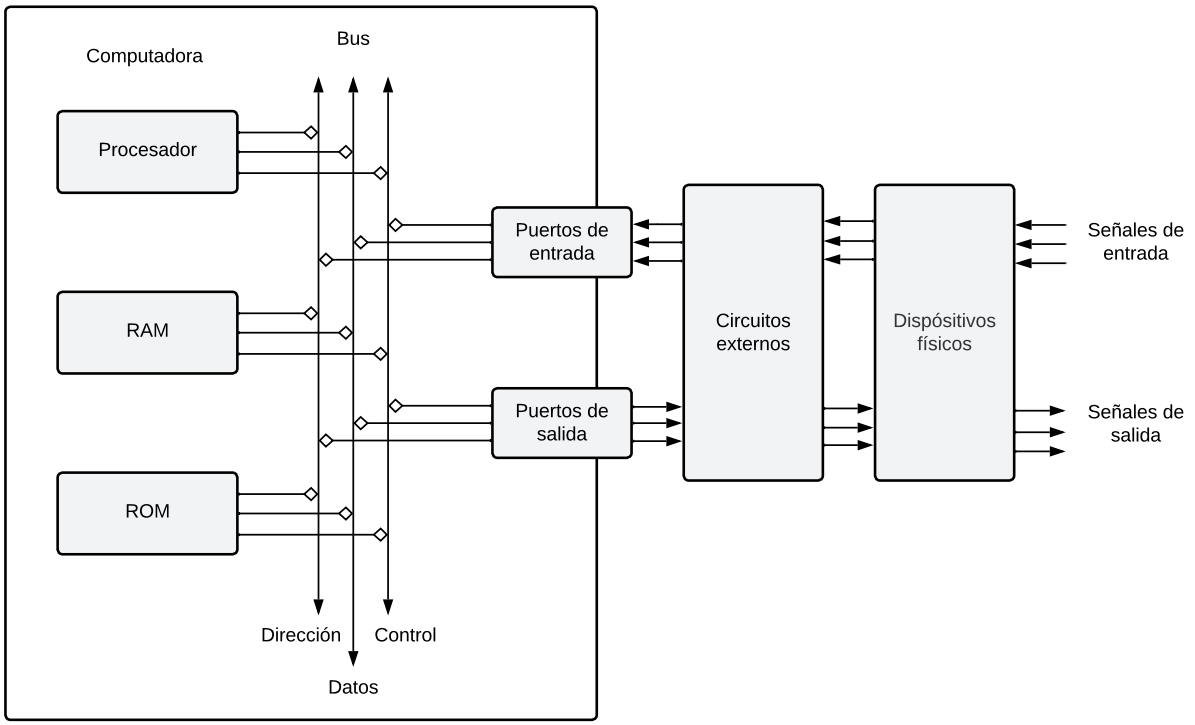


Figura 2.1: Diagrama a bloques de los componentes básicos de un sistema de computadora de la configuración *Von Neuman*.

En la arquitectura de *Von Neumann*, las instrucciones se obtienen de la ROM en el mismo bus, como los datos se obtienen de la memoria de acceso aleatorio. El *software* es una secuencia ordenada de instrucciones específicas que son almacenadas en memoria, definiendo cuándo y qué tareas específicas se van a desempeñar. El procesador ejecuta al *software* al recuperar e interpretar estas instrucciones. El microprocesador es un procesador de menor tamaño, sin considerar el poder de cómputo.

Una micro-computadora es una computadora de tamaño pequeño, también conocida como microcontrolador, contiene los componentes de una computadora (procesador, memoria, *I/0*) en un solo chip.

La mayoría de las *RAM* son volátiles, quiere decir que si la fuente de energía es interrumpida y después restaurada, la información en la *RAM* se perderá.

La información es programada en la memoria de sólo lectura usando técnicas más complicadas que escribir en una *RAM*. La *ROM* son no volátiles, significa que si la energía es interrumpida y después restaurada, la información es retenida [15].

2.1.1.1. Microprocesador

Un microprocesador es una implementación integrada de una parte de la unidad de control y procesamiento lógico (CPU) de una máquina. Generalmente se le conoce como CPU o ruta de datos. Los microprocesadores presentan características que contemplan un rango variado de costos y consumo de energía. La arquitectura de sistema basado en microprocesador debe incluir subsistemas de entrada y salida y un sistema de memoria externo. También incluye un reloj o sistema de referencia de tiempo. Todos estos componentes son conectados vía sistema bus o buses. La figura 2.2 muestra un diagrama a bloques de un sistema basado en microprocesador [17].

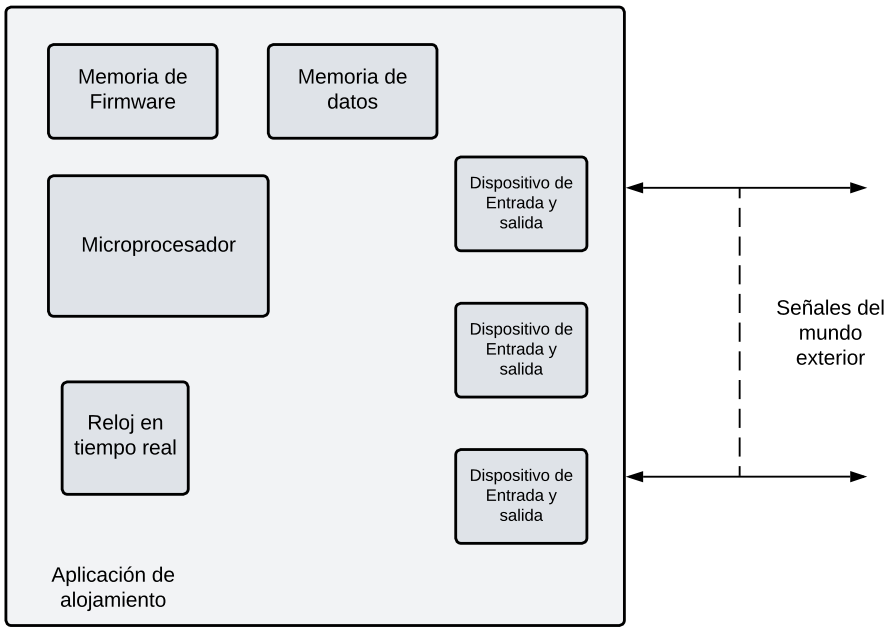


Figura 2.2: Diagrama de bloques de un sistema basado en microprocesador.

El bloque de memoria de *firmware* de la figura anterior (2.2) refiere al programa alojado que contiene el código de aplicación y el bloque de memoria de datos corresponde a los datos que son manipulados, enviados o traídos del exterior. En sistemas empotrados, el firmware se define como código de aplicación porque generalmente se almacena en la ROM. A la memoria de datos se conoce generalmente como RAM [17].

2.1.1.2. Microcomputador

Una microcomputadora es un sistema de computadora completo que emplea un microprocesador como su núcleo de cómputo. Estas poseen un largo número de circuitos integrados que proporcionan la funcionalidad periférica necesaria. La complejidad de microcomputadoras varía desde unidades simples que son implementadas en un sólo chip con pequeñas cantidades de memoria sobre el chip y un sistema de entradas y salidas tan complejo que permite un amplio arreglo de circuitos de soporte periféricos [17].

2.1.1.3. Microcontrolador

Un microcontrolador reúne el núcleo de microprocesador y una gran colección de capacidades periféricas I/O integradas a un sólo circuito. Como adicionales, incluye temporizadores, convertidores digitales-analógicos, convertidores analógicos-digitales, I/O digitales, canales de comunicación seriales o paralelos y memoria de acceso directo (DMA). Puede, o no, incluir un subsistema de memoria. En la figura 2.3 se visualizan componentes de un sistema basado en microcontrolador [17].

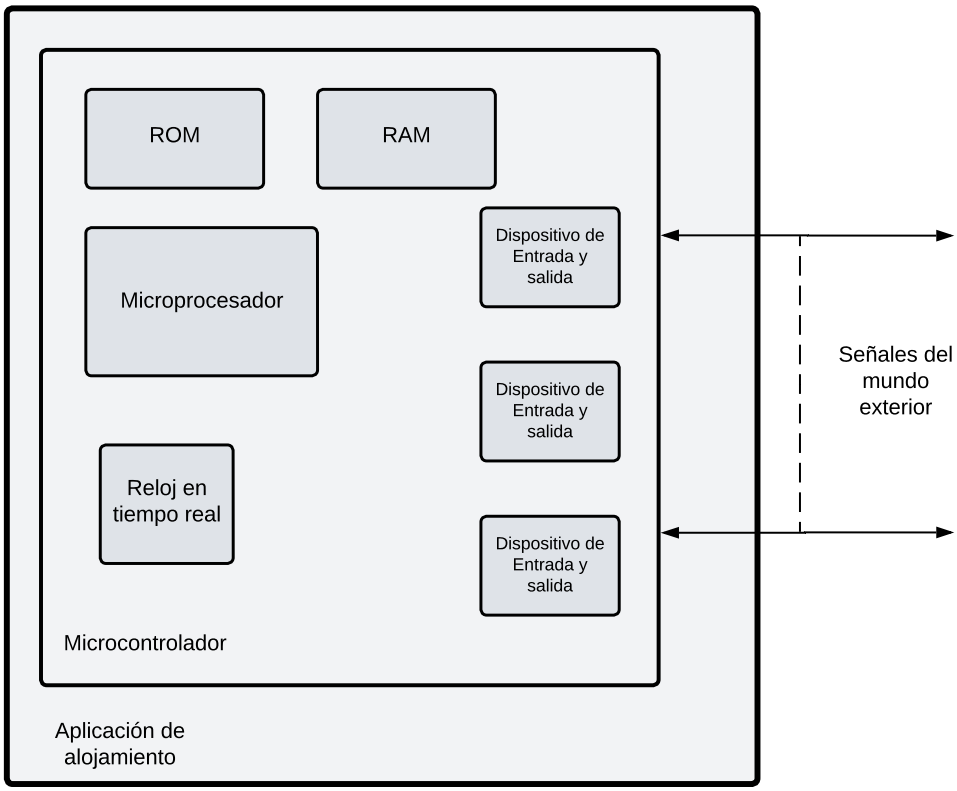


Figura 2.3: Diagrama de bloques de los componentes básicos de un sistema basado en microcontrolador.

2.1.2. Sistemas en tiempo real

El tiempo es una restricción esencial en una aplicación de sistemas embebidos en contraste con una aplicación de escritorio. Un sistema en tiempo real demanda que el sistema responda a eventos externos o internos ya designados dentro de un intervalo de tiempo específico. La respuesta esperada es típicamente la ejecución de una tarea asociada con el evento detonante [17].

Significa que el sistema embebido debe responder a eventos críticos dentro de un tiempo definido estrictamente, llamado tiempo máximo. La predicción del comportamiento del sistema operativo es aquel que garantiza alcanzar estos tiempos máximos de la ejecución de tareas donde el tiempo es determinista [16].

Otro requerimiento en tiempo real que debe existir en estos sistemas es la ejecución de tareas periódicamente. Significa que debe ser ejecutada en un intervalo de tiempo igual.

2.1.3. Computadoras de una sola placa (*SBC*) para desarrolladores

Un sistema empotrado basado en microcontrolador también se le conoce como tarjeta de desarrollo y, en algunos casos, computadores de una sola tarjeta, dependiendo de la definición de sistema empotrado. Para que una *SBC* o tarjeta de desarrollo pueda ser empleada como sistema embebido, debe considerar los requerimientos y características mencionadas en el apartado de 2.1, como presentar sistema operativo en tiempo real o ser usado para una aplicación especial con las restricciones pertinentes.

Una tarjeta de este tipo es un dispositivo electrónico enfocado a crear prototipos, probar y programar sistemas electrónicos mediante una plataforma de *software* y *hardware* ya adaptado a una placa que incluye microcontroladores, reguladores de voltaje, puertos de entrada y salida, con convertidores *ADC* y *DAC*, conocidos como *GPIO*, entre otros componentes o módulos. Estas tarjetas o placas cuentan con interfaces de comunicación, para conectarse con otros dispositivos, como *I2C, SPI, 1-Wire, Ethernet, Wi-Fi, Bluetooth*, entre otras. [18]. A hoy en día existe una amplia cantidad de plataformas de placas de *hardware* disponibles en el mercado como placas basadas en microcontroladores, microprocesadores o en unidad de procesamiento gráfico (GPU). Algunas de las plataformas de desarrollo más populares y aceptadas del mercado son:

- *Raspberry Pi*: Plataforma que ofrece distintos modelos de placas de desarrollo de bajo costo y tamaño reducido, ideales para proyectos de electrónica y computación con microprocesadores de arquitectura ARM.
- *Arduino*: Ofrece un entorno de desarrollo versátil y sus placas están diseñadas para proyectos de electrónica y robótica básica.
- *BeagleBone*: Placas de código abierto con una potente capacidad de conectividad y procesamiento.
- *NVIDIA Jetson*: Placas de alto rendimiento diseñadas para aplicaciones de inteligencia artificial y *ML*.

- *Intel Edison, Galileo*: Placas de desarrollo de Intel que combinan capacidades de procesamiento con conectividad y expansión.
- ESP32 y ESP8266: Microcontroladores con *Wi-Fi* de bajo costo y bajo consumo de energía, ideales para proyectos de Internet de las cosas (IoT).

A diferencia de las computadoras de una sola tarjeta (SBC), las placas de desarrollo permiten una mejor versatilidad para agregar *hardware* y conectividad que se adapta a las necesidades de los proyectos; en cambio, una SBC es una solución de *hardware* y conectividad predeterminada por el fabricante y está más orientada al uso de computador personal.

Es importante mencionar que algunos modelos de las distintas plataformas de desarrollo pueden ser también catalogados como SBC, ya que la capacidad de *hardware* y conectividad define el uso de estas y puede ser usadas de las dos formas, como computador de escritorio o para aplicaciones en sistemas empotrados.

A continuación, se presentan las características de tres modelos de placas SBC empleadas específicamente en prototipos de sistemas de detección de somnolencia (SDS) presentados en publicaciones científicas [19], [20], [21].

Tabla 2.1: Tabla comparativa entre distintos modelos de placas de desarrollo (*SBC*).

Características	**Raspberry Pi 4**	**BeagleBone Black**	**Jetson Nano**
Procesador	quad-core 1.8 GHz ARM Cortex-A72	quad-core 1 GHz ARM Cortex-A8	quad-core 1.4 GHz Cortex-A57
GPU	3D VideoCore VI 1 GB	SGX 3D Graphics Engine 16 MB	NMaxwell 4 GB
Memoria RAM	8 GB LPDDR4	512 MB DDR3	4 GB LPDDR4
Almacenamiento	Ranura tarjeta microSD	eMMC, microSD	eMMC, microSD
Conectividad	Eth, Wi-Fi 802.11ac, BLE 5.0	Eth, Wi-Fi 802.11b/g/n, BLE 4.1	Eth, Ranura M.2 Key E
Puertos	USB 3.0, USB 2.0, micro HDMI	USB, micro HDMI	USB 3.0, USB 2.0, HDMI, DP
GPIO	40 pines	2 x 46 pines	40 pines
OS	Raspbian (basado en Linux)	Linux	Linux
Precio (USD)	$115-$125	$130-$150	$170-$200

De la tabla 2.1 se observa que la placa de desarrollo *Nvidia Jetson Nano* (figura: 2.4) presenta una característica de procesamiento distinta de las otras plataformas, ya que está diseñada para proyectos que involucran aprendizaje automático o inteligencia artificial y presenta una unidad de procesamiento gráfico dedicada (*GPU*), por lo cual es ideal para sistemas de visión, ya que las *GPU* desempeñan un mejor procesamiento de imágenes y su

rendimiento es mucho mayor para el cálculo de una gran cantidad de datos. Su costo va de los 170 a 200 dólares americanos y para la conectividad de redes de telecomunicación posee una ranura *M.2 Key E* que permite agregar módulos o tarjetas de red *Wi-fi 5* y *BLE 4.1*. Cabe mencionar que cuenta con puertos *USB 2.0 y 3.0, HDMI* y *Display Port*.

Figura 2.4: Placa de desarrollo *Nvidia Jetson nano*.

La tarjeta *BeagleBone Black* (figura: 2.5) está basada en un microprocesador de arquitectura *ARM*; su costo se debe a la cantidad de pines *GPIO*. Está diseñada para agregar la conexión y comunicación con otros componentes, como sensores, actuadores, módulos y otros dispositivos electrónicos. Presenta la conectividad *BLE 4.1*, *Wi-Fi 5* sin implementarlos externamente.

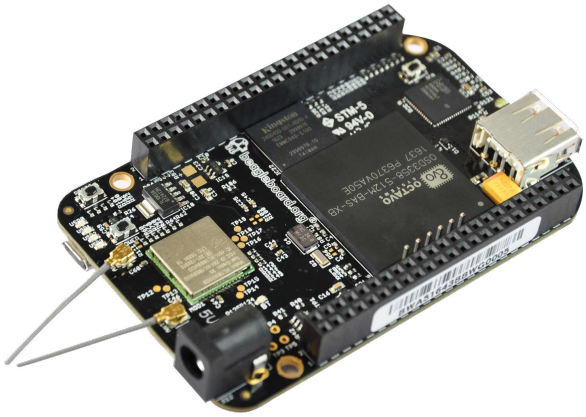


Figura 2.5: Placa modelo *BeagleBone Black wireless*.

Por el contrario, el modelo *Raspberry Pi 4 Model B* (figura:2.6) implementa módu-

los inalámbricos, pero al igual que la placa *BeagleBone Black*, presentan un poder de procesamiento limitado sin poseer una *GPU* dedicada. No obstante, el procesador *ARM* Cortex-A72 ofrece un rendimiento del 50 % mayor en comparación con los procesadores Cortex-A8 de acuerdo con el análisis de desempeño de procesadores *ARM* del autor Athow D. [22].

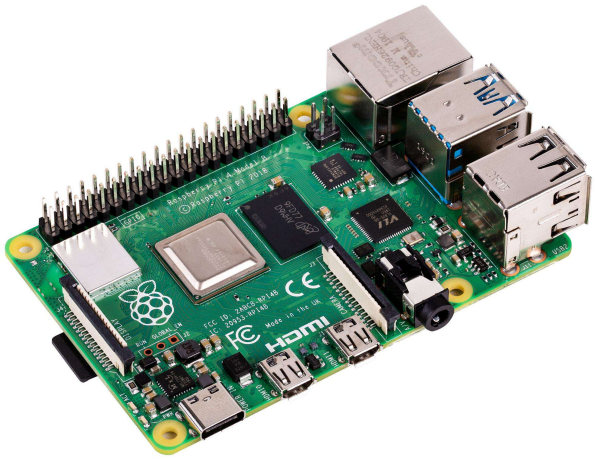


Figura 2.6: Placa *Raspberry Pi 4 Model B*.

2.1.4. Redes inalámbricas

Para el desarrollo de la comunicación *Bluetooth* entre un *smartphone Android* y el *SBC Raspberry*, con el fin de enviar el número de celular del usuario a la *Raspberry*, es necesario analizar algunas de las redes inalámbricas más populares en proyectos que requieren conexiones inalámbricas.

Un proyecto de *IoT* implica un *software* que controla sensores, actuadores, controladores, etc., y este establece la interacción entre ellos mediante una conexión. En la actualidad, la mayoría de implementaciones presentan conexiones inalámbricas, pero es común encontrar combinaciones de medios cableados e inalámbricos.

Los tres tipos de conexión más empleados dependen del nivel de integración del dispositivo *IoT* [18].

1. Conexión con dispositivos integrados y *app* de administración.
2. Conexión con dispositivos integrados y registro en un servidor.
3. Conexión con elementos integrables.

La conexión con elementos integrables permite a la placa de desarrollo realizar el control del actuador y esta placa puede ser administrada por la aplicación del cliente. El diagrama de la figura 2.7 muestra el escenario de este tipo de conexión con elementos integrables.

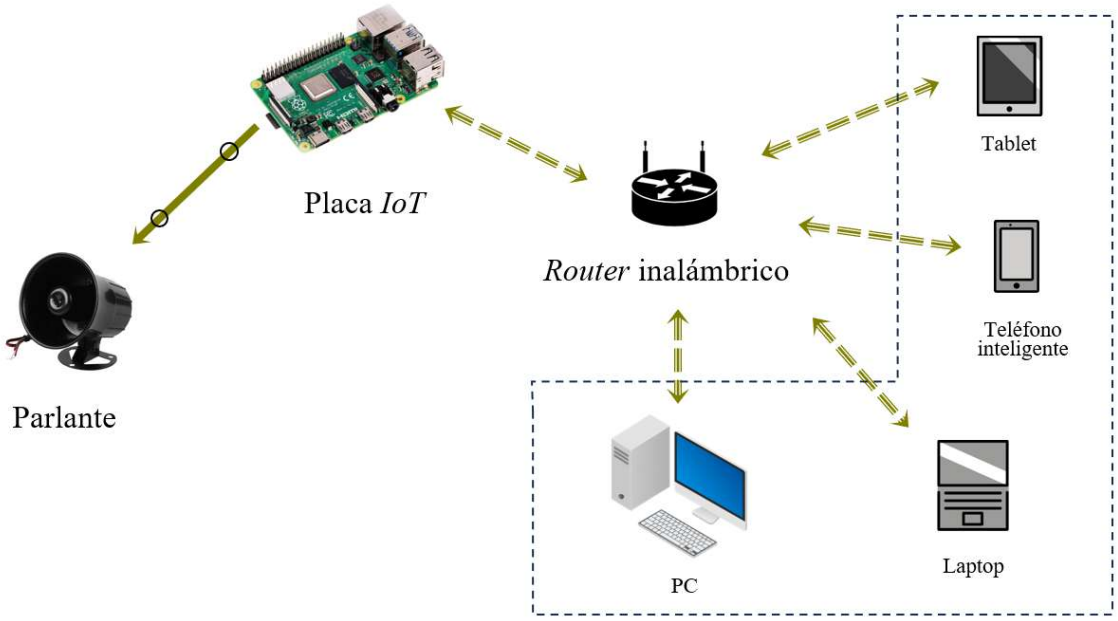


Figura 2.7: Escenario *IoT* con dispositivos integrables.

La conectividad entre los dispositivos *IoT* se da gracias a diferentes tecnologías de red, las más populares son las redes inalámbricas *Wi-Fi* y *Bluetooth LE*, o tecnologías más especializadas de corto alcance como *NFC* (*Near Field Communication*), *Z-wave*, *ZigBee* y otras redes de área de cobertura más extensa, como *5G, LTE, LoRaWAN, SigFox* y *LPWAN (Low Power-WAN)*. Gracias a la gran variedad de redes existentes, las principales características que se deben tener para las necesidades de la implementación de un proyecto de *IoT* son: el rango de cobertura o alcance, la velocidad de la transferencia de datos, el consumo de energía, frecuencia de operación y seguridad [18].

2.1.4.1. *Zigbee*

Es una tecnología de red inalámbrica con topología de malla especializada para escenarios locales que implican bajo consumo de potencia. Su alcance efectivo en interiores va de 75 a 100 metros. Soporta hasta 65 mil nodos con una velocidad de transferencia de 40 a 250 Kbps en la frecuencia de 2.4 GHz. Un nodo dentro de la red *Zigbee* puede tomar los roles de:

- **Coordinador**: nodo que administra la red, enruta y verifica nodos de entrada y salida.
- ***Router***: sólo enruta los dispositivos de la red.
- **Dispositivo final**: envían o reciben paquetes de la red.

Este tipo de red es frecuentemente usada en la administración de escenarios de domótica *IoT* o automatización industrial.

2.1.4.2. *Bluetooth Low Energy (BLE)*

Es una versión actualizada de la tecnología *Bluetooth* diseñada para trabajar con niveles bajos de potencia conservando las características propias del *Bluetooth* clásico (versiones anteriores a *Bluetooth 4.0*). Transmite datos en la frecuencia 2.4 GHz. Puede alcanzar velocidades de transmisión de datos de hasta 1.4 Mbps. Las versiones más recientes utilizan una topología de malla donde cada dispositivo tiene una conexión directa con todos los demás dispositivos de la red. Permite la detección de dispositivos dentro de un radio determinado.

Su implementación está centrada en el intercambio de datos en bajas cantidades, consumiendo una potencia relativamente baja. Las aplicaciones más comunes están presentes en el sector automovilístico, la domótica, monitoreo inalámbrico o el cuidado de la salud.

2.1.4.3. *Wi-Fi*

Las diferentes versiones de *Wi-Fi* han mejorado con el tiempo ya que se basa en un estándar de comunicación IEEE 802.11 *WLAN Working Group* que permite la interoperabilidad, seguridad y confiabilidad de red. *Wi-Fi 6* es retrocompatible con las versiones 4 y 5.

Wi-Fi 5,6 IEEE 802.11ac/ax puede operar en la frecuencia 2.4 o 5 GHz. La frecuencia de 5 GHz provee mayor velocidad de transferencia sacrificando alcance, y la red de 2.4 GHz comprende mayor alcance sacrificando velocidad de transferencia de datos.

- **2.4 GHz**

- Alcance: 45 m. interiores, 90m. exteriores.
- Velocidad: 60 Mbps.

- **5 GHz**
  - Alcance: 15m. interiores, 30m. exteriores.
  - Velocidad: 867 Mbps

A continuación se presentan las diferentes versiones de *Wi-Fi* con sus nombres en la tabla 2.2:

Tabla 2.2: Estándares y versiones de *Wi-Fi*

Wi-Fi	**Estándar**
Wi-Fi 7	802.11be (en desarrollo)
Wi-Fi 6	802.11ax
Wi-Fi 5	802.11ac
Wi-Fi 4	802.11n
Wi-Fi 3	802.11g
Wi-Fi 2	802.11b
Wi-Fi 1	802.11a

La tabla 2.3 muestra una comparación de las características de algunas de las redes de comunicación inalámbrica más empleadas del mercado.

Tabla 2.3: Tabla comparativa de redes inalámbricas

Características	**Wi-Fi 5.0**	**Bluetooth 3.0 HS**	**BLE 4.0**	**Zigbee**
Velocidad de transferencia de datos	11 Gbps	24 Mbps	1.4 Mbps	250 Kbps
Rango de alcance	15-45m	10m	50m	75-100m
Consumo de energía	2-6 W	0.01-0.5 W	0.001-0.01 W	0.01-0.1 W
Frecuencia de operación	2.4 GHz-6 GHz	2.4 GHz	2.4 GHz	2.4 GHz
Seguridad	Alto nivel	Moderada	Moderada	Alto nivel
Interferencia	Susceptible	Susceptible	Susceptible	Menor susceptibilidad
Aplicaciones	Redes domésticas, empresariales	Periféricos, auriculares	Dispositivos IoT de baja potencia	Domótica, automatización industrial
Costo	20 USD	30 USD	5-20 USD	40 USD

2.2. Somnolencia y sus tipos

Generalmente la somnolencia se define como la tendencia a quedarse dormido. Es una necesidad fisiológica básica para la supervivencia humana, como el hambre o la sed. La causa más común es la privación del sueño. Se puede presentar por efectos de algunos medicamentos o como un síntoma de enfermedades como la depresión. Su intensidad depende de la cantidad de sueño, las interrupciones del sueño y qué tan rápido se inicia este.

Otros autores definen a la somnolencia en optativa o excesiva; la primera es la facilidad de quedarse dormido en momentos socialmente aceptables y la somnolencia excesiva ocurre cuando el individuo debería estar despierto [23].

2.2.1. Somnolencia objetiva y subjetiva

La somnolencia objetiva se define como la propensión al sueño o la tendencia a quedarse dormido. Su medición se realiza mediante el Test de Latencia Múltiple del Sueño, un instrumento estandarizado que evalúa el tiempo necesario para conciliar el sueño (latencia del sueño) y la aparición de la fase REM. Este análisis se basa en el registro de señales electroencefalográficas, electrooculográficas de ambos ojos y electromiográficas del mentón, durante cinco siestas de veinte minutos programadas a lo largo del día.

Por otro lado, la somnolencia subjetiva considera la percepción individual de la necesidad de dormir o la transición entre vigilia y el sueño. Esta condición se manifiesta a través de síntomas subjetivos, como la sensación de fatiga o pesadez, y de signos objetivos, como el bostezo, la constricción pupilar, la ptosis palpebral, la relajación de los músculos extensores del cuello, así como la disminución de la atención, el rendimiento psicomotor y la función cognitiva.

Las escalas de la somnolencia son cuestionarios estandarizados para cuantificar la somnolencia subjetiva y no miden ningún parámetro objetivo, pero emplean manifestaciones físicas objetivas [23].

2.2.2. Somnolencia excesiva diurna

La somnolencia excesiva diurna (SED) se define como la incapacidad de permanecer despierto y alerta durante el período de vigilia, con episodios no intencionados de somnolencia o sueño.

La somnolencia diurna es un estado caracterizado por la reducción de la atención, el incremento en los tiempos de reacción y la aparición de errores por omisión, lo que afecta la memoria y la calidad de vida. Su causa principal es la mala calidad del sueño.

Los síntomas del insomnio pueden derivar en somnolencia diurna, cuya causa más frecuente es el déficit de sueño asociado a alteraciones en el ritmo circadiano. Entre estas alteraciones se incluyen el despertar precoz, el retraso en la hora de dormir, el insomnio y la disrupción del patrón sueño-vigilia [24].

A continuación se presenta una breve descripción del enfoque de algunas publicaciones o investigaciones recientes sobre la detección de somnolencia en conductores.

2.3. Estado del arte: detección de somnolencia

Este apartado presenta una recopilación de distintas publicaciones que abordan la detección de somnolencia a partir de distintos enfoques, por lo cual es importante identificar el método científico y el modelo prototipo que algunas publicaciones exponen.

En la publicación de Sanjay Dey et al.: "Detección de síntoma de somnolencia en conductores en tiempo real basado en comportamiento facial a través de un enfoque de aprendizaje automático" [12], se analiza la detección de somnolencia que causa accidentes fatales. Su principal objetivo es resolver el problema de la detección del pestañeo o parpadeo excesivo, bostezo y el cabeceo.

La investigación de Jang Woon Baek et al.: "Algoritmo de detección de somnolencia en tiempo real para sistemas de monitoreo del estado del conductor" [11], retoma la definición de fatiga ocular como un factor que induce a la somnolencia.

El artículo de Shinfeng D. Lin et al.: "Reconocimiento de ojos cansados para la detección de Somnolencia" [10], se aborda la detección de solamente un rasgo facial de la somnolencia: parpadeo por tiempo prolongado.

Los autores de la obra: "Sistema de detección de somnolencia usando aprendizaje profundo" [13], hacen énfasis en que las expresiones faciales de la somnolencia se definen como una condición de la fatiga.

En la obra de Shruti Mohanty et al.:"Diseño de un sistema de detección de somnolencia en tiempo real usando Dlib" [14], este diseño se enfoca en realizar una detección de rasgos de somnolencia que provocan la inatención al momento de conducir. Y se caracteriza por la detección de las siguientes expresiones faciales: ojos cerrados por más tiempo, bostezo excesivo, inclinación del cuello simulando un cabeceo.

En el artículo: "Implementación de monitoreo del conductor en un ambiente adaptativo AUTOSAR (Arquitectura de sistema abierto automotriz)" [25], esta implementación se basa en la premisa de que la fatiga y la somnolencia son condiciones relacionadas entre sí y pueden presentar un alto riesgo dependiendo del nivel a detectar (bajo, medio o alto).

En la investigación de Adochiei Ioana et al.: "Sistemas de detección y alertamiento de somnolencia en conductores para infraestructuras críticas" [19], publicada por miembros

de la facultad de medicina de la universidad de Popa en Rumanía, se cataloga de igual manera la somnolencia y la fatiga como dos conceptos que comparten patrones o rasgos faciales entre sí.

Por otra parte, la publicación: "Diseño de sistema de alertamiento del sueño a través de reconocimiento de imágenes y procesamiento en Python, Dlib y OpenCV" [20], los investigadores participantes abordan la detección de rasgos faciales que denotan falta de concentración al conducir un auto, provocados por la fatiga o somnolencia.

El estudio titulado "Sistema de detección de somnolencia usando KNN y OpenCV" [26] señala que el monitoreo de los ojos es un indicador claro para identificar la somnolencia o la falta de sueño en una persona. En este sentido, características como el cansancio ocular, el pestañeo lento o la permanencia de los ojos cerrados por un tiempo prolongado son rasgos faciales asociados a la somnolencia.

En el contenido del estudio de Petchara Inthanon et al. "Detección de somnolencia por imágenes faciales en medios de vídeo en tiempo real usando Nvidia Jetson Nano" [21], se detalla el concepto de "micro-sueño" como un síntoma de la somnolencia y que es un período de sueño en el que no es posible diferenciar si una persona está dormida o despierta, y el intercambio de estas dos etapas ocurre en un período de tiempo muy corto.

Los autores de la obra: "Discriminación de somnolencia moderada y aguda en base a expresiones faciales espontáneas" [27], detallan que discernir entre somnolencia aguda o moderada beneficia a la seguridad al momento de detectar los niveles de somnolencia para simplemente alertar o informar sobre una muy probable colisión. La publicación realiza el análisis de rasgos faciales de somnolencia menos notorios.

"Modelo de detección de niveles de somnolencia integral combinando información multimodal" [28]. Este artículo realiza la detección de rasgos característicos del comportamiento facial y corporal de las etapas de somnolencia subjetiva definidos en una escala propuesta por los investigadores Hiroki Jatajima y Walter W. Wierwille, en la que se incluye la evaluación de somnolencia en humanos.

A continuación, se presentan las tablas comparativas 2.4 y 2.5 que exhiben los rasgos y comportamientos faciales asociados a tres categorías de somnolencia: somnolencia subjetiva, somnolencia moderada y somnolencia excesiva. Estos rasgos y comportamientos faciales han sido recopilados a partir de investigaciones publicadas en fuentes científicas. Es importante destacar que algunos autores no especifican claramente el tipo de somnolencia que sus sistemas se proponen detectar.

El propósito de estas tablas radica en destacar las diferencias y similitudes observadas en los rasgos o comportamientos faciales analizados para la detección de somnolencia en cada uno de los prototipos descritos en los artículos y publicaciones revisados. A través de esta comparación, se busca categorizar estos rasgos dentro de una de las tres categorías de somnolencia, basándose en los síntomas característicos que definen cada una de ellas.

Este análisis tiene como finalidad determinar los rasgos o comportamientos faciales más relevantes para la detección de somnolencia en el Sistema de Alerta y Detección de Somnolencia (SDAS) desarrollado en el contexto de este trabajo de tesis. Todos estos criterios se establecieron a partir de las definiciones formales de somnolencia, con el objetivo de brindar un enfoque más objetivo en la detección de esta condición en conductores.

Tabla 2.4: Comparación entre artículos y publicaciones científicas sobre SDS.

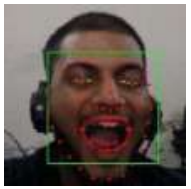
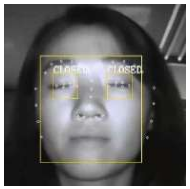
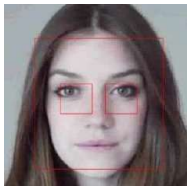
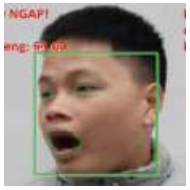
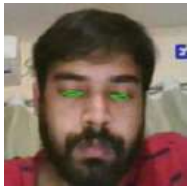
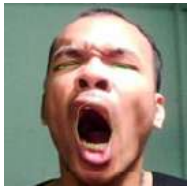
ARTÍCULO	**[12]**	**[11]**	**[10]**	**[13]**	**[14]**	**[25]**
DETECCIÓN DE RASGOS FACIALES.	-Parpadeo excesivo. -Bostezo. -Cabeceo.	-Pestañeo o parpadeo (ojos cerrados).	-Estado del ojo (abierto o cerrado).	-Estado del párpado (abierto o cerrado). -Bostezo excesivo.	-Ojos cerrados por más tiempo. -Bostezo.	-Detección del iris. -Estado de los ojos (abiertos o cerrados)
TIPO DE SOMNOLENCIA A PARTIR DE EXPRESIONES FACIALES.	Combina síntomas característicos de somnolencia moderada y excesiva.	El pestañeo excesivo es un síntoma de somnolencia excesiva.	Somnolencia excesiva o aguda.	El bostezo es un síntoma de la fatiga. Somnolencia excesiva por pestañeo.	Combinación de síntomas de la somnolencia moderada y excesiva.	Somnolencia excesiva; dificultad para enfocar la vista.
EVIDENCIA GRÁFICA						

Tabla 2.5: Continuación de comparación entre artículos y publicaciones científicas sobre SDS.

ARTÍCULO	**[19]**	**[20]**	**[26]**	**[21]**	**[27]**	**[28]**
DETECCIÓN DE RASGOS FACIALES.	-Estado de los ojos (cerrados o abiertos; considerable periodo de tiempo).	-Estado de los ojos (cerrados o abiertos; considerable periodo de tiempo). -Bostezo. -Cabeceos.	-Estado de los ojos (cerrados o abiertos; considerable periodo de tiempo).	-Monitoreo del estado de los ojos. -Detección del bostezo con ojos cerrados.	-Frunción en labios. -Arrugas en nariz. -Cabeceo lateral. -Tensión en parpados. -Cierre de ojos.	-Pestañeo o estado de los ojos. -Bostezo o movimiento en la boca. -Postura al manejo. -Cabeceo.
TIPO DE SOMNOLENCIA A PARTIR DE EXPRESIONES FACIALES.	- Característica de somnolencia excesiva.	-Combinación de rasgos faciales, somnolencia excesiva y fatiga moderada.	-Expresión facial de somnolencia excesiva.	-Combinación de expresiones de somnolencia extrema y moderada.	-Rasgos como el fruncir los labios y tensión en párpados son catalogados como somnolencia moderada.	Diferentes niveles de somnolencia subjetiva (. . . moderada, significativa, extremadamente somnoliento).
EVIDENCIA GRÁFICA					X	

Ahora bien, el estudio del concepto de detección de somnolencia comprende aspectos de temas computacionales que son la base para comprender este tipo de sistemas que detectan de signos o patrones faciales de somnolencia, ya que mediante su análisis, será posible llevar a cabo a una implementación real. No obstante, se requiere también del uso de la electrónica y manipulación de señales, más enfocada en el uso de sistemas empotrados para realizar pruebas controladas.

2.4. Inteligencia artificial (IA)

Según la definición de Alan Turing en 1950, las máquinas se definen como inteligentes si pueden realizar actividades que el humano puede hacer. Partiendo del test de Turing, "se entabla una comunicación entre una persona y una máquina en cuartos distintos y si la persona no es capaz de distinguir si lo que hay en otra habitación es un humano o una máquina, entonces, en caso de ser una máquina, la podemos considerar inteligente".

La definición de inteligencia artificial recae en una máquina que sea capaz de pasar el Test de Turing y ha de tener las siguientes capacidades: reconocimiento del lenguaje natural, aprendizaje, representación del conocimiento, razonamiento. Más tarde se definió el test de Turing total, donde se agrega una cámara de vídeo y elementos que le permitan realizar otras capacidades (visión y robótica) [29].

Una definición más adaptada a la actualidad del concepto de Inteligencia Artificial es la de Russell y Norvig donde "La Inteligencia Artificial se refiere al estudio y diseño de sistemas informáticos inteligentes que sean capaces de realizar tareas que requieren inteligencia humana, como el aprendizaje, la percepción, el razonamiento y la toma de decisiones" [30].

La definición resalta la habilidad de los sistemas informáticos para realizar tareas que anteriormente sólo podían ser ejecutadas por personas, y la importancia de desarrollar algoritmos y modelos de aprendizaje para lograr que los sistemas puedan adaptarse y perfeccionar su rendimiento en una tarea.

Las ramas más populares de esta área de la informática son los siguientes apartados:

- Visión artificial o por computadora.
- Procesado de lenguaje natural.
- Robótica.
- Aprendizaje Automático.

Aprendizaje automático o de máquina (*ML*)

Por otro lado, el *Machine Learning* (*ML*) es una rama de la inteligencia artificial que abarca el análisis estadístico de los datos y se centra en la aplicación de diversas técnicas, entre las que se incluyen:

- Clasificación.
- *Clustering* o agrupamiento.
- Redes neuronales.

Aprendizaje profundo

Se le conoce como la "Revolución del ML". El *Deep learning* o aprendizaje profundo es un apartado del *ML* donde las computadoras realmente aprenden y toman decisiones inteligentes por sí mismas. Además de que involucra algoritmos más complejos y niveles más profundos de automatización en comparación del *ML* convencional [31].

En concreto, la IA es un área de la informática que emplea distintas técnicas del *ML* para desarrollar sistemas de distintos tipos, ya sean aplicaciones que usan software para controlar *hardware*, sistemas mecantrónicos, visión artificial o aplicaciones solamente de *software*, como *chatbots* o los sistemas de recomendación de series o películas para dispositivos de entretenimiento. El mapa de la figura 2.8 muestra una relación entre las herramientas necesarias para crear sistemas inteligentes en distintas áreas de la tecnología.

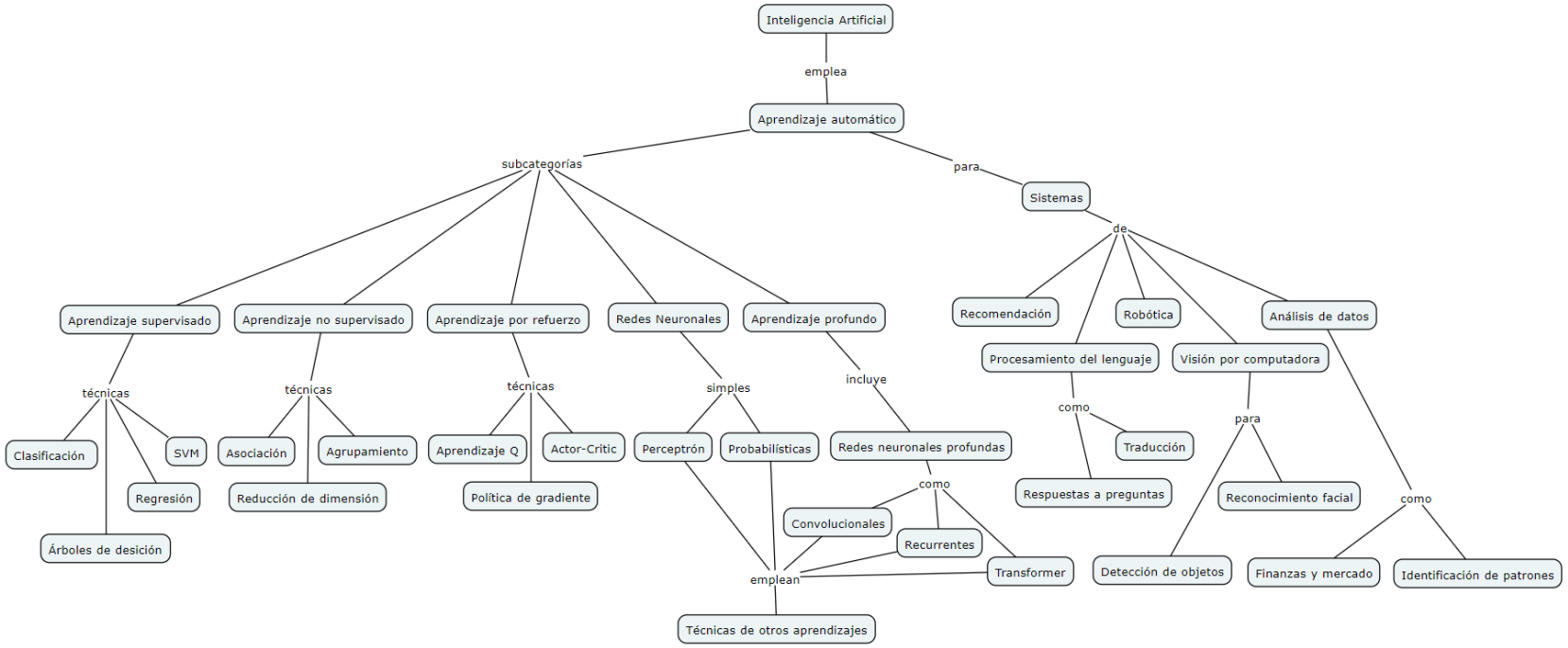


Figura 2.8: Mapa conceptual de las técnicas empleadas para desarrollar aplicaciones de IA.

Como dice el título de esta obra, este proyecto se desarrolla partiendo del uso de técnicas del *Machine Learinig* para sistemas de detección o visión artificial. Por ello, es necesario conocer la relación entre el concepto de *ML* y visión artificial.

2.4.1. Aprendizaje automático (Machine Learning)

El aprendizaje de máquina, aprendizaje automático o *ML*, es una subárea de la ciencia de la computación que otorga a las computadoras la habilidad de aprender sin ser explícitamente programadas.

Arthur Samuel

Pionero estadounidense en el campo de los juegos en computadora e inteligencia artificial, acuñando el término *Machine Learning* en 1995 en IBM [31].

En otras palabras, *Machine Learning* sigue el mismo proceso que un niño de 4 años usa para aprender, entender y diferenciar. Los algoritmos del aprendizaje de máquina están inspirados en el proceso de aprendizaje humano e iterativamente aprenden mediante datos y permiten a las computadoras encontrar puntos de vista ocultos. Estos modelos ayudan en una variedad de tareas, como el reconocimiento de objetos, recomendaciones, a resumir, etcétera.

El *ML* impacta a la sociedad de una manera muy profunda, un ejemplo son los algoritmos de recomendación en vídeos, películas o programas de TV que las aplicaciones de *streaming* sugieren a los usuarios, en base a sus gustos o al contenido digital que consumen habitualmente.

Existen muchas otras aplicaciones que vemos en la vida diaria como lo son: *chatbots* para realizar procesamiento del lenguaje natural, como traducciones, chats, ingresar al teléfono celular mediante detección facial o juegos de computadora que agregan personajes que interactúan con el usuario. Cada una de estas aplicaciones utiliza diferentes técnicas y algoritmos de *ML*. Algunas de las técnicas más populares son:

- Regresión/Estimación
    - Predice valores continuos, por ejemplo: predecir el valor de una casa o estimar las emisiones de CO_2 de un auto.
- Clasificación

    - Predice el elemento, clase/categoría de un caso. Ejemplo: si una célula puede ser benigna o maligna.

- Agrupamiento o *Clustering*

    - Encuentra la estructura de datos; resumir. Ejemplo: encontrar pacientes con síntomas similares, o la segmentación de clientes de un banco.

- Asociación

    - Encuentra elementos o eventos que ocurren simultáneamente. Ejemplo: artículos que se compran usualmente juntos por los clientes.

- Detección de anomalías

    - Se usa para descubrir anomalías o casos inusuales como: detección de fraudes de tarjetas de crédito.

- Minado secuencial

    - Predice eventos siguientes, por ejemplo: flujo de «clicks» en los sitios web.

- Reducción de dimensión

    - Reduce el tamaño de datos (PCA)

- Sistemas de recomendación

    - Recomienda artículos a partir de sus preferencias o las preferencias de sus similares más cercanos como libros o anuncios.

El *ML* es importante porque permite usar y combinar técnicas o herramientas de sus distintas subcategorías en las que se dividen en el tipo de aprendizaje que las necesidades del proyecto requieren, como el aprendizaje profundo, supervisado o no supervisado.

Las aplicaciones de visión artificial emplean principalmente técnicas del aprendizaje supervisado y el aprendizaje profundo (deep learning). Estas técnicas permiten entrenar modelos de visión artificial para reconocer patrones y características en imágenes y vídeos.

En el aprendizaje supervisado, se utilizan conjuntos de datos etiquetados, donde se proporciona al modelo imágenes junto con las etiquetas que indican las clases o categorías a las que pertenecen. El modelo aprende a partir de estos ejemplos y luego puede generalizar y clasificar nuevas imágenes en función de las características aprendidas.

El aprendizaje profundo es una forma de aprendizaje automático basada en redes neuronales profundas. Estas redes están compuestas por múltiples capas de neuronas interconectadas, lo que les permite aprender representaciones jerárquicas de los datos. En el caso de la visión artificial, las redes neuronales convolucionales (CNN) son ampliamente utilizadas, ya que son eficientes en la extracción de características visuales y el reconocimiento de patrones en imágenes.

Estos enfoques de aprendizaje permiten a las aplicaciones de visión artificial realizar tareas como detección de objetos, reconocimiento facial, clasificación de imágenes, seguimiento de objetos o la detección de somnolencia, entre otras. La capacidad de aprender de manera automatizada a partir de datos visuales ha impulsado avances significativos en áreas como la conducción autónoma, la seguridad, la medicina, la robótica y más [31].

2.4.1.1. Aprendizaje supervisado

El término supervisar significa observar y direccionar la ejecución de una tarea, proyecto o actividad. En este caso, se supervisará un modelo de *ML* que podrá ser capaz de clasificar regiones, estado de atención y estado de somnolencia. Se dice que se supervisa un modelo de *ML* "enseñándole" al modelo. Que es cuando se carga al modelo con conocimiento, así que este podrá predecir situaciones futuras. Para poder "enseñarle" al modelo se necesita entrenarlo con algunos datos de un conjunto ya etiquetado. Es importante mencionar que los datos deben ser etiquetados o definidos en una clase. Se le llama atributos a los datos que describen al objeto de interés. En la figura 2.9, la imagen exhibe un ejemplo de atributos:

ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
1000025	5	1	1	1	2	1	3	1	1	benign
1002945	5	4	4	5	7	10	3	2	1	benign
1015425	3	1	1	1	2	2	3	1	1	malignant
1016277	6	8	8	1	3	4	3	7	1	benign
1017023	4	1	1	3	2	1	3	1	1	benign
1017122	8	10	10	8	7	10		7	1	malignant
1018099	1	1	1	1	2	10	3	1	1	benign
1018561	2	1	2	H	2	1	3	1	1	benign
1033078	2	1	1	1	2	1	1	1	5	benign
1033078	4	2	1	1	2	1	2	1	1	benign

Figura 2.9: La primera fila indica los Atributos y la columna señalada en rojo son las etiquetas.

En la figura 2.10 se mira que el contenido de cada columna es una "característica", pero el contenido de la primera fila describe su atributo.

ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
1000025	5	1	1	1	2	1	3	1	1	benign
1002945	5	4	4	5	7	10	3	2	1	benign
1015425	3	1	1	1	2	2	3	1	1	malignant
1016277	6	8	8	1	3	4	3	7	1	benign
1017023	4	1	1	3	2	1	3	1	1	benign
1017122	8	10	10	8	7	10		7	1	malignant
1018099	1	1	1	1	2	10	3	1	1	benign
1018561	2	1	2	H	2	1	3	1	1	benign
1033078	2	1	1	1	2	1	1	1	5	benign
1033078	4	2	1	1	2	1	2	1	1	benign

Figura 2.10: La primera fila es el atributo y a cada atributo se le asigna una característica.

En el ámbito del *Machine Learning* (*ML*), los datos más utilizados son de tipo numérico. Sin embargo, también se emplean datos categóricos, los cuales contienen caracteres en lugar de valores numéricos.

Las figuras 2.11 y 2.12 ilustran gráficamente los dos tipos de aprendizaje supervisado: clasificación y regresión.

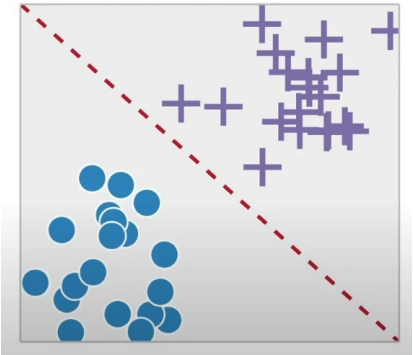


Figura 2.11: Aprendizaje supervisado; clasificación.

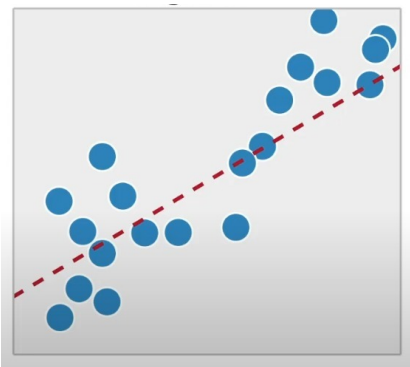


Figura 2.12: Aprendizaje supervisado; regresión.

La clasificación es el proceso de predecir una etiqueta de clase discreta o categoría. Por otro lado, la regresión es el proceso de predecir valores continuos en lugar de predecir valores categóricos [31]. Por lo que

2.4.1.2. Clasificación

En el aprendizaje automático, la clasificación es un enfoque del aprendizaje supervisado que permite categorizar elementos desconocidos dentro de un conjunto discreto de clases. Su objetivo es establecer la relación entre un conjunto de variables características y una variable objetivo de interés, la cual es de tipo categórico con valores discretos.

A partir de un conjunto de datos de entrenamiento con etiquetas asignadas, un clasificador es capaz de predecir la clase correspondiente a nuevos casos no etiquetados. Existen dos tipos principales de clasificadores: binarios, cuando solo hay dos categorías posibles, y de multiclase, cuando se trabaja con más de dos categorías.

La clasificación tiene aplicaciones en diversas industrias, particularmente en problemas donde es posible asociar características con una variable objetivo dentro de un conjunto de datos etiquetado. Entre sus aplicaciones más comunes se encuentran los filtros de correo electrónico, el reconocimiento de voz, la identificación de escritura manuscrita y los sistemas biométricos, entre otros [31].

Algunos de los tipos de algoritmos de clasificación más populares son:

- Arboles de decisión (ID3 C4.5 C5.0)

- Clasificador bayesiano ingenuo (Naïve Bayes)
- Análisis del discriminante lineal.
- Vecino más cercano (*K-nearest neighbor*).
- Regresión Logística.
- Redes Neuronales.
- Máquinas de soporte vectorial (*SVM*).

2.4.2. Visión artificial

Partiendo del significado de sentido de la vista o visión humana, este es el sentido más importante y complejo de todos los sentidos, ya que aproximadamente el 75 % de la información procesada por el cerebro proviene de este sentido.

Según González y Penedo, "la visión artificial es una rama de la inteligencia artificial que se encarga del desarrollo de técnicas y métodos que permiten a las computadoras o sistemas electrónicos capturar y procesar información visual para llevar a cabo tareas tales como la identificación de objetos, reconocimiento de patrones, seguimiento de movimientos, entre otros" [32].

La visión por computador es el análisis de imágenes a través de computadoras para obtener una descripción de los objetos físicos que son captados por una cámara. Haciendo énfasis en que la entrada (input) a un sistema de visión son imágenes y la salida (output) es información sobre estas imágenes [33].

Algunas aplicaciones más empleadas de la visión artificial son:

- Aplicaciones industriales:
  - Control de calidad.
  - Robots industriales.
- Sociedad:

- Sistemas de seguridad y detección facial.
- Fotografía.

Según Duque A., un sistema de visión se compone de tres bloques principales: sensor (cámara), módulo de procesamiento y control, y finalmente, actuadores externos. La figura 2.13 muestra la interacción entre los elementos de cada bloque, donde el sensor realiza una captura de imágenes o vídeo en tiempo real, el módulo de procesamiento ejecuta un algoritmo que procesa y analiza las imágenes capturadas por el sensor y a partir de este análisis se ejecuta el *software* que controla a los actuadores externos que realizan acciones físicas [34].

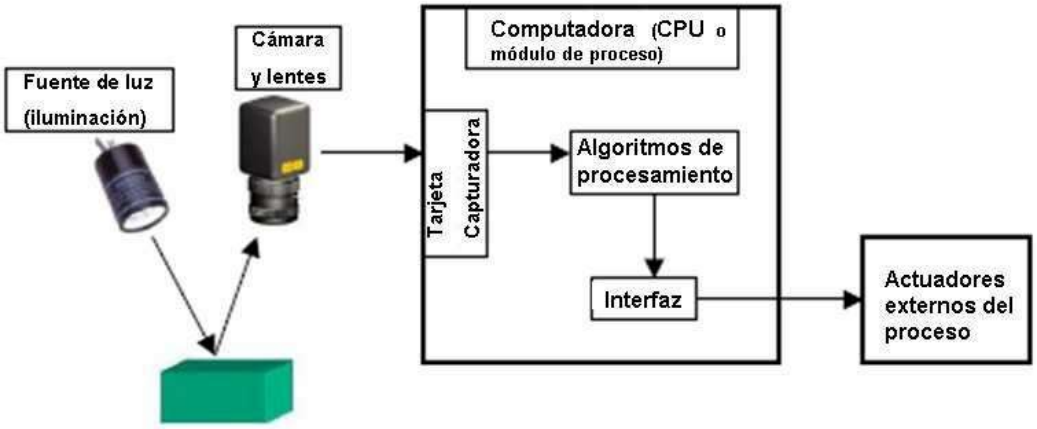


Figura 2.13: Diagrama de bloques de un sistema de visión.

2.4.3. Relación entre visión artificial y aprendizaje automático (ML)

La visión artificial, como aplicación, utiliza técnicas de *ML* como la clasificación, redes neuronales, entre otros, para analizar y comprender imágenes y vídeos para la toma de decisiones. El aprendizaje automático es una de las técnicas más importantes en la visión artificial, ya que permite a los sistemas aprender de los datos y mejorar su capacidad de detectar y reconocer objetos, rostros, emociones, patrones y otros elementos visuales. El *ML* se utiliza en la visión artificial para entrenar modelos y algoritmos que puedan

reconocer patrones, clasificar objetos y tomar decisiones en función de las imágenes que se les presentan como entrada (*input*) [35], como es el caso de la detección de somnolencia en conductores.

2.4.4. Detección de rostros con clasificador *Haar cascade*

La detección de rostros en visión artificial permite localizar regiones faciales en imágenes o vídeo, siendo clave para aplicaciones como el reconocimiento facial o la detección de somnolencia en conductores. Para ello, se emplean algoritmos de aprendizaje automático, destacando el clasificador *Haar cascade* por su eficiencia en tiempo real. Su uso facilita la identificación de fatiga mediante el análisis de los ojos y otros rasgos faciales, contribuyendo a la toma de decisiones basadas en la información visual.

La detección de objetos mediante clasificadores en cascada basados en funciones de *Haar* es un método eficaz de detección de objetos propuesto por Paul Viola y Michael Jones en su artículo, "Detección rápida de objetos mediante una cascada potenciada de funciones simples" en 2001 [36]. Es un enfoque basado en el aprendizaje automático en el que una función de cascada se entrena a partir de muchas imágenes positivas y negativas. Luego se usa para detectar objetos en otras imágenes [37].

Después de haber entrenado millones de imágenes y ser introducidas al sistema, el clasificador comienza extrayendo características de cada imagen. Las ondas o características de *Haar* son *kernels* convolucionales usados para extraer características. Los *kernels* convolucionales son filtros que se emplean para distintas tareas, como mejorar la calidad de la imagen, detección de bordes, difuminar el fondo, entre otras.

Enfocado a la detección de rostros, el algoritmo emplea imágenes positivas con rostros e imágenes negativas, que no contienen rostros o la misma imagen sólo incluyendo el fondo, para entrenar el modelo. Luego se necesita extraer características de ellas. Las características se definen como el resultado de agregar filtros (*kernels*) a una imagen.

Para esto, se utilizan las características *Haar*. Cada característica es un valor único obtenido al restar la suma de los píxeles debajo de un rectángulo blanco de la suma de los píxeles debajo del rectángulo negro. Las ondas de *Haar* extraen información sobre: bordes, líneas, ejes diagonales.

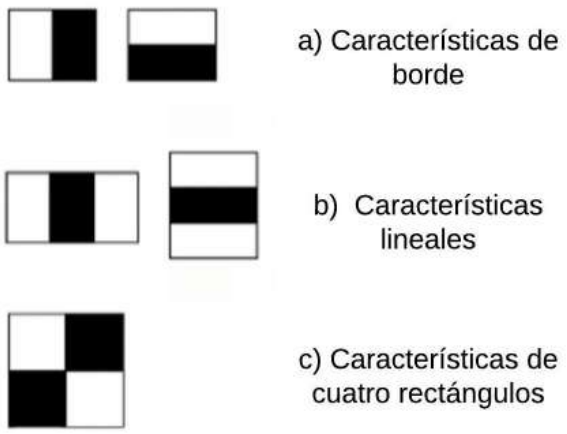


Figura 2.14: Características *haar*.

El algoritmo selecciona algunas características importantes de un conjunto largo para proporcionar clasificaciones altamente eficientes mediante el uso de un potenciador, otro algoritmo llamado *AdaBoost*.

Todos los tamaños y ubicaciones de cada *kernel* se utilizan para calcular diversas características. Para cada cálculo, es necesario determinar la suma de los píxeles en las regiones definidas por los rectángulos negros y blancos. Para optimizar este proceso, se emplea el concepto de imagen integral, lo que permite reducir los cálculos a una operación que involucra solo cuatro píxeles, sin importar el tamaño de la imagen.

Pero entre todas estas características calculadas, la mayoría son irrelevantes. Ejemplo, de la figura 2.15, las filas superiores muestran dos buenas características. La primera parece enfocarse en la propiedad de que la región de los ojos es frecuentemente más oscura que la región de la nariz o las mejillas. La segunda característica seleccionada recae en la propiedad de que los ojos son más oscuros que el puente de la nariz, pero las mismas ventanas que aplican a las mejillas o a otro lugar son irrelevantes. Para seleccionar las mejores características de más de 160,000, se hace uso de otro algoritmo de potenciación llamado *AdaBoost*.

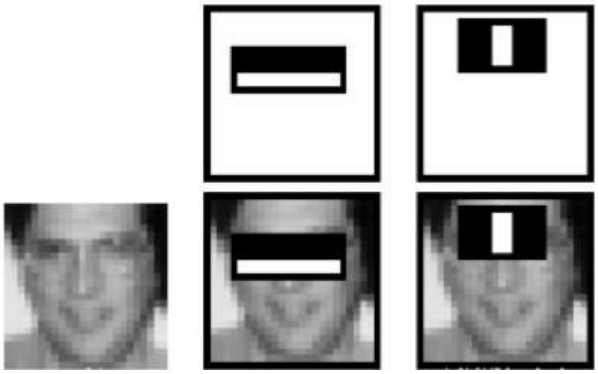


Figura 2.15: Características *haar* en rostros.

Para esto, cada característica se aplica sobre el conjunto de entrenamiento de imágenes, determinando el umbral óptimo que clasifica los rostros como positivos o negativos. Durante este proceso, pueden ocurrir errores o desclasificaciones. Se seleccionan las características con la menor tasa de error, es decir, aquellas que clasifican con mayor precisión tanto los rostros como las imágenes sin rostros.

El procedimiento es el siguiente: inicialmente, todas las imágenes reciben el mismo peso. Tras cada clasificación, los pesos de las imágenes mal clasificadas se incrementan y el proceso se repite. Se recalculan las tasas de error y los nuevos pesos hasta alcanzar la precisión deseada, minimizar la tasa de error o determinar el número óptimo de características.

El clasificador final es una combinación ponderada de clasificadores débiles. Se denominan débiles porque, individualmente, no pueden clasificar una imagen con precisión, pero en conjunto forman un clasificador robusto. Según el artículo, al menos 200 características proporcionan una precisión del 95 %, mientras que la configuración final utiliza alrededor de 6,000 características.

El siguiente paso consiste en evaluar una imagen dividiéndola en ventanas de 24×24 píxeles y aplicando las 6,000 características para determinar la presencia de un rostro. Este enfoque es ineficiente, ya que en la mayoría de los casos la imagen no contiene rostros. Para optimizar el proceso, se implementa un método preliminar que descarta rápidamente las regiones sin rostro, enfocándose solo en aquellas con mayor probabilidad de contener uno.

Aquí es donde entra en juego el concepto de cascada de clasificadores. En lugar de aplicar las 6,000 características a cada ventana, estas se agrupan en múltiples etapas de

clasificación, aplicadas de forma secuencial. Las primeras etapas contienen pocas características y sirven como filtro inicial: si una ventana no supera la primera etapa, se descarta sin procesar el resto de características. Si la supera, avanza a la siguiente etapa, y así sucesivamente. Una ventana que pase todas las etapas se considera una región de rostro.

El detector descrito en el artículo utiliza más de 6,000 características distribuidas en 38 etapas, con 1, 10, 25, 25 y 50 características en las primeras cinco etapas. Las dos características mostradas en la figura 2.15 fueron seleccionadas como las más relevantes mediante *Adaboost*. Según los autores, en promedio, solo 10 de las más de 6,000 características son evaluadas por subimagen [37].

2.4.5. Histograma de gradientes orientados *HOG*

El Histograma de Gradientes Orientados (*HOG*) es una técnica utilizada en la detección de rostros por su capacidad para capturar la estructura y contornos de los objetos en una imagen. Su eficacia radica en la extracción de características basadas en la distribución de gradientes de intensidad, lo que permite diferenciar patrones faciales con precisión ante variaciones de iluminación y pose. Esta teoría es clave, ya que el algoritmo *DLib*, empleado en la detección de somnolencia, utiliza *HOG* para localizar el rostro antes de analizar características faciales específicas, como el estado de los ojos y la boca.

HOG se define como un descriptor de características en visión artificial y procesamiento de imágenes para la detección de objetos. Este método cuenta las ocurrencias de orientación de gradiente en secciones localizadas de una imagen, enfocándose en la estructura y forma de los objetos.

A diferencia de los descriptores de borde tradicionales, *HOG* emplea la magnitud y el ángulo del gradiente para calcular características, lo que mejora la detección al ser más sensible a cambios en textura y contorno. Para cada región de la imagen, se generan histogramas que representan la distribución de la magnitud y orientación del gradiente, facilitando la detección de patrones visuales específicos [38].

Pasos para calcular características *HOG*

1. Tomar una imagen como entrada y cambiar el tamaño de la resolución a 128x64 píxeles.

2. El segundo paso es calcular el gradiente de la imagen. Se obtiene a partir de la combinación de la magnitud y el ángulo de la imagen. Considerando bloques de 3x3 píxeles. Primero se calcula G_x y G_y para cada píxel mediante la fórmula 2.1.

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \quad G_y = I(r-1,c) - I(r+1,c) \tag{2.1}$$

Donde r refiere a filas y c a las columnas respectivamente.

Después de haber calculado G_x y G_y, se calcula la magnitud y el ángulo de cada píxel usando la expresión 2.2:

$$Magnitud(\mu) = \sqrt{G_x^2 + G_y^2} \quad Ángulo(\theta) = |\tan^{-1}(G_y/G_x)| \tag{2.2}$$

3. Una vez obtenido el gradiente de cada píxel, las matrices de gradientes se dividen en celdas de 8x8 para formar un bloque. Para cada bloque, se calcula un histograma de 9 puntos y cada contenedor contiene un rango de ángulo de 20°. La tabla 2.6 representa un histograma de 9 contenedores en el cual los valores son alojados después de los cálculos. Cada uno de los histogramas de 9 puntos puede ser graficado como histogramas de con contenedores sacando la intensidad del gradiente en ese contenedor. Como un bloque contiene 64 diferentes valores, para todos los 64 valores de gradiente y de magnitud, el siguiente cálculo se lleva a cabo.

   Representación de un histograma de 9 contenedores. Este histograma es único para bloques de 8x8 para compensar 64 celdas. Estas 64 celdas agregarán sus valores Vj y $Vj + 1$ a la j-ésima y a la $(j + 1)$-ésima indexado de el arreglo respectivamente.

Tabla 2.6: Histograma de 9 contenedores.

Valor									
Contenedor	0	20	40	60	80	100	120	140	160

$$Número\ de\ contenedores = 9(rango\ de\ 0^o\ a\ 180^o) \tag{2.3}$$

$$Tamaño\ de\ paso(\Delta\theta) = 180^o\ /\ Número\ de\ contenedores = 20^o \tag{2.4}$$

Para cada contenedor j, el contenedor tendrá limites desde:

$$[\Delta\theta \cdot j, \Delta\theta \cdot (j + 1)] \tag{2.5}$$

El valor del centro de cada contenedor será:

$$C_j = \Delta\theta(j + 0.5) \tag{2.6}$$

4. Para cada celda en un bloque, primero se calculará el j-ésimo contenedor y después el valor que se proporcionará al j-ésimo y $(j+1)$-ésimo contenedor respectivamente. El valor viene dado por las expresiones 2.7, 2.8, 2.9:

$$j = [(\frac{\theta}{\Delta\theta} - \frac{1}{2})] \tag{2.7}$$

$$V_j = \mu \cdot [\frac{\theta}{\Delta\theta} - \frac{1}{2}] \tag{2.8}$$

$$V_{j+1} = \mu \cdot [\frac{\theta - C_j}{\Delta\theta}] \tag{2.9}$$

5. Un arreglo es tomado como un contenedor para un bloque y valores de V_j y V_{j+1} es adjuntado en el arreglo en el j-ésimo indexado y $j + 1$-ésimo contenedor calculado para cada píxel.

6. La matriz resultante después de los cálculos anteriores tendrá una forma de 16x8x9.

7. Una vez que la computación del histograma esté terminado para todos los bloques, 4 bloques de la matriz de histograma de 9 puntos se juntan para formar un nuevo bloque de 2x2, este proceso de combinación está hecho de manera superpuesta con un paso de 8 píxeles, para todas las 4 celdas en 1 bloque, se concatenan los histogramas de 9 puntos para cada celda de 1 vector de 36 elementos.

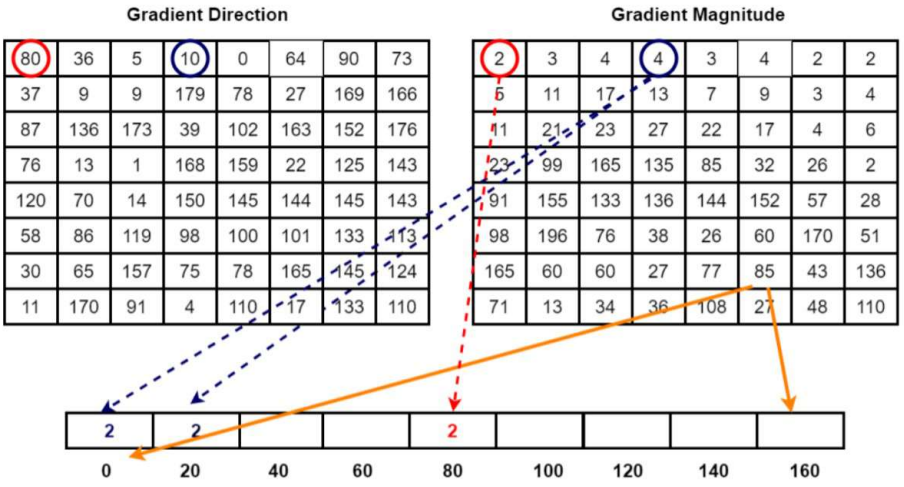


Figura 2.16: Histograma de Gradientes

$$f_{bi} = [b_1, b_2, b_3, ..., b_{36}] \tag{2.10}$$

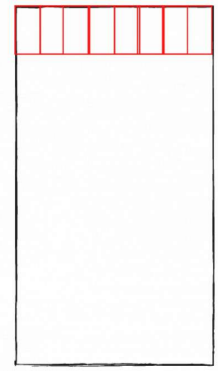


Figura 2.17: Travesía de un cuadro de cuadrícula de 2x2 alrededor de la imagen para hacer un fbi combinado de 4 bloques.

8. Los valores de fb para cada bloque es normalizado por la norma de $L2$:

$$f_{bi} \leftarrow \frac{f_{bi}}{\sqrt{||f_{bi}^2|| + \epsilon}} \tag{2.11}$$

Donde ϵ es un pequeño valor agregado al cuadrado de fb para evitar que la división sea 0 y ocurra un error. En código, el valor es: $\epsilon = 1e^{-05}$

9. Para normalizar, el valor de k se calcula primero por la ecuación 2.12 y 2.13:

$$k = \sqrt{b_1^2 + b_2^2 + b_3^2 + ... + b_{36}^2} \tag{2.12}$$

$$f_{bi} = [(\frac{b_1}{k}), (\frac{b_2}{k}), (\frac{b_3}{k}), ..., (\frac{b_{36}}{k})] \tag{2.13}$$

10. Esta normalización se hace para reducir el efecto de cambio en contraste entre imágenes del mismo objeto. De cada bloque, se recolecta un vector de características de 36 puntos.

    En la dirección horizontal hay 7 bloques y en la vertical hay 15 bloques. Entonces la longitud total del *HOG* será: 7x15x36 = 3780. Se obtienen las características *HOG* de la imagen seleccionada.

Figura 2.18: Visualización de las características *HOG* de una imagen usando librería *Skimage*

Visión general del algoritmo

Para computar un *HOG*:

1. Normalización opcional de imagen global

2. Calcular el gradiente de imagen en "x" y en "y"

3. Calcular los histogramas de gradiente

4. Normalizar a través de bloques

5. Convertir a un vector de características

La primera etapa aplica una ecualización de normalización de imagen global que reduce la influencia de los efectos de iluminación. En la práctica se usa compresión gamma calculando, ya sea, la raíz cuadrada o el logaritmo a cada canal de color. Esta compresión ayuda a reducir los efectos de las variaciones de sombra e iluminación local.

La segunda etapa consiste en calcular gradientes de imagen de primer orden. Capturan el contorno, la silueta y cierta información de las texturas y brindan mayor resistencia a las variaciones de iluminación. Los métodos variantes también pueden incluir derivadas de imágenes de segundo orden, una función útil para capturar, *p.* barras como estructuras en bicicletas y extremidades en humanos.

La tercera etapa tiene como objetivo producir una codificación sensible al contenido de la imagen mientras permanece resistente a pequeños cambios en la pose o apariencia. Este método agrupa la información de orientación del gradiente localmente de la misma manera que la función SIFT[2]. La ventana de imagen se divide en dos regiones espaciales llamadas celdas. Para cada celda se acumula un histograma local 1-D de gradiente o orientación de borde sobre todos los píxeles en la celda. Este histograma combinado representa un histograma de orientación. Este histograma 1-D divide el rango del ángulo de gradiente en un número fijo de contenedores. Las magnitudes de gradiente de los píxeles en la celda se emplean para votar en el histograma de orientación.

En la cuarta etapa se calcula la normalización, que toma grupos locales de celdas y el contraste normaliza sus respuestas generales antes de pasar a la siguiente etapa. La normalización brinda mejor invarianza a la iluminación, sombreado y contraste a bordes. Se desempeña acumulando una medida de "energía" de histograma local sobre grupos locales de celdas llamados "bloques". El resultado se usa para normalizar cada celda en el bloque. Por lo tanto, la celda aparece varias veces en el vector de salida final con diferentes normalizaciones y así mejorar su desempeño. A los descriptores de bloque normalizados se les llama descriptores de histograma de gradiente ordenado (HOG).

El último paso es recopilar los descriptores *HOG* de todos los bloques de una densa cuadrícula superpuesta de bloques que cubren la ventana de detección en un vector de características combinadas para usar en el clasificador de ventanas [38].

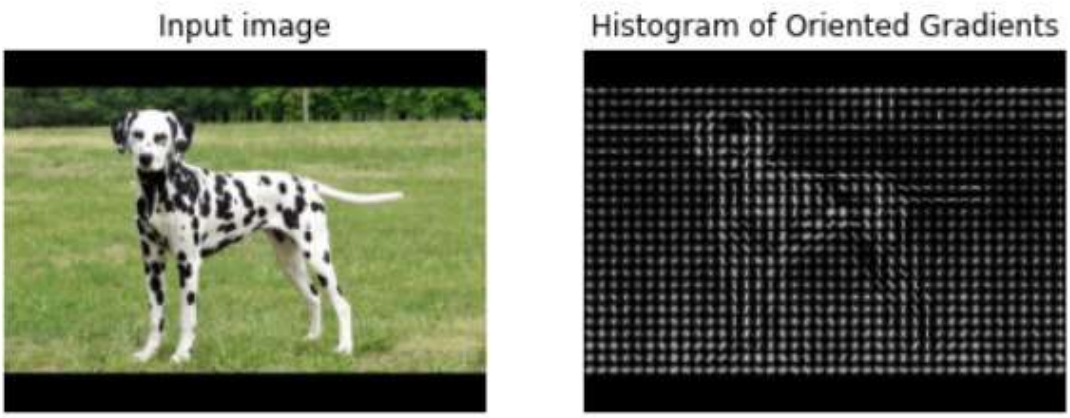


Figura 2.19: Visualización de las características *HOG* de un perro.

2.4.6. Máquinas de soporte vectorial *SVM* para clasificación

Es un algoritmo de aprendizaje supervisado que clasifica casos mediante la búsqueda del separador. Este modelo trabaja mapeando datos en un espacio de características de alta dimensión en el que los puntos de esos datos pueden ser categorizados incluso si los datos no son separables linealmente. Después, se estima un valor que se le conoce como separador. Los datos se transforman de tal manera que el separador se puede dibujar como un hiperplano, como en la figura 2.20, que muestra la clasificación de dos categorías en una dimensión R3.

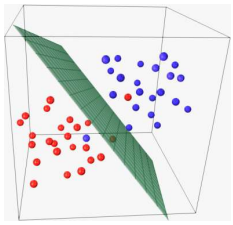


Figura 2.20: Clasificación de dos categorías en una dimensión R3

Este plano se emplea para clasificar nuevos casos o datos desconocidos. Por lo tanto, un algoritmo de máquina de soporte vectorial ofrece como salida un hiperplano óptimo

que categoriza nuevas muestras.

Si el comportamiento gráfico de los datos no es linealmente separable, es posible transferirlo a otro espacio dimensional donde al incrementar las dimensiones del conjunto de datos graficando X en un nuevo espacio se haga linealmente separable. A este proceso de graficar datos aumentando el espacio dimensional se le conoce como *Kernelling*. La función matemática usada para esta transformación se le llama función de *Kernel* y pueden ser de tipo lineal, polinomial, función de base radial (RBF) o sigmoide. El mejor separador a elegir debe contar con el margen más largo posible entre las dos clases. Las muestras que se encuentren más cerca al separador se les conoce como vectores de soporte y estos definen la longitud del margen. El margen se compone de dos líneas llamadas líneas de decisión y cada una posee sus propias ecuaciones donde los vectores de soporte definen la dirección de estas líneas del margen.

La ecuación del separador corresponde a la expresión 2.14:

$$W^T x + b = 0 \tag{2.14}$$

El margen más óptimo se obtiene generalmente partiendo del entrenamiento de datos y se resuelve con descenso del gradiente, como en otros problemas.

Por otro lado, la salida del algoritmo son el valor W y b para la línea. Primero se introducen los valores de entrada en la ecuación de línea y así se puede calcular si un valor desconocido se encuentra debajo o por encima del separador y definir una clasificación.

Algunas de las ventajas y desventajas de SVM son:

Ventajas:

- Ventajas
  - Buena precisión en altos espacios dimensionales
  - Eficiencia en memoria gracias al uso de vectores de soporte
- Desventajas
  - Propenso al sobre-ajuste (overfitting) si el número de características es mayor a las muestras

- No proporcionan directamente estimaciones de probabilidad en la clasificación.
- *SVM* no es muy eficiente al memento del computo si el conjunto es muy grande.

Los *SVM* son buenos para tareas de procesamiento y clasificación de imágenes y reconocimiento de patrones escritos a mano. También es bueno para la minería de texto como la detección de *spam* [39].

Para implementar un sistema de detección de somnolencia en conductores, es fundamental comprender los principios de Máquinas de Soporte Vectorial (*SVM*), Histograma de Gradientes Orientados (*HOG*) y el clasificador *Haar cascade*. Estas técnicas son importantes para la detección y clasificación de patrones faciales, permitiendo el análisis del estado del conductor. *HOG* facilita la extracción de características clave al representar la distribución de gradientes en una imagen, mientras que *Haar cascade* utiliza clasificadores en cascada para detectar rostros de manera eficiente en tiempo real. Las *SVM* proporcionan un modelo robusto para clasificar estados de alerta y somnolencia.

Estos enfoques se integran en herramientas como los 68 *facial landmarks* de *DLib*, los cuales permiten identificar puntos faciales específicos, como ojos y boca, fundamentales para el análisis de somnolencia. Por otro lado, el clasificador *Haar cascade* de *OpenCV* propicia la detección rápida de rostros.

A continuación, se abordarán temas y conceptos clave para la implementación de estos modelos en código, utilizando *Python* y las principales bibliotecas de *Machine Learning*, como *OpenCV*, *DLib* y *scikit-learn*, esenciales para desarrollar un sistema de detección de somnolencia en conductores.

2.4.7. Python

En esta sección es importante conocer las herramientas más populares empleadas en el área del aprendizaje automático y visión artificial, ya que el SDAS hace uso de dependencias o bibliotecas del lenguaje de programación *Python* junto con sus herramientas, funciones, clases y algoritmos para llevar a cabo el desarrollo e implementación de este prototipo.

Python es un lenguaje de programación muy popular y robusto ya que es de propósito

general y recientemente se ha convertido en el lenguaje preferido entre los científicos de datos. Por lo tanto, se pueden crear algoritmos de *Machine learning* y se trabaja muy bien. Actualmente ya se encuentran implementadas una gran variedad de módulos y librerías que hacen el trabajo más sencillo para aplicaciones de sistemas inteligentes que emplean *ML* o *Deep Learning* [31].

Algunos de los paquetes más usados en *Python* son:

- ***NumPy***: Es una librería de matemáticas para trabajar con arreglos de n dimensión en *Python*. Permite hacer cálculos eficiente y efectivamente. Con *NumPy*, se trabaja con datos de tipo: arreglos, diccionarios, funciones, "tipado" de datos e imágenes.

- ***SciPy***: Es una colección algoritmos numéricos y domina variedad de herramientas, incluyendo procesamiento de señales, optimización, estadísticas y mucho más. Es una librería de alto desempeño computacional.

- ***Mathplotlib***: *Mathplotlib* es un paquete para diseño de gráficos muy popular que provee de gráficos 2-D y 3-D.

- ***Pandas***: Esta librería de alto nivel desempeña fácilmente el uso de estructuras de datos. Contiene diversas funciones para importar datos, manipularlos y analizarlos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series de tiempo.

- ***SciKitLearn***: Es una colección de algoritmos y herramientas para *ML*. Contiene la mayoría de algoritmos de clasificación, regresión y agrupamiento (*Clustering*). Está diseñado para trabajar con librerías de tipo numéricas y científicas como *NumPy* y *SciPy*. Algunas de las tareas más empleadas en *SciKit* son: pre-procesado de datos, selección y extracción de características, división de pruebas de entrenamiento, definición de algoritmos, modelos apropiados, personalizado de parámetros, evaluación de predicción y el exportado de modelos.

- ***OpenCV* y *Dlib***: Estas bibliotecas son un conjunto de funciones, clases, algoritmos, modelos pre-entrenados y muchas otras herramientas para el análisis y procesamiento de imágenes enfocadas a la visión artificial.

2.4.8. Matriz de confusión para determinar métricas de evaluación del rendimiento de un algoritmo de *ML*

Las métricas de análisis cuantitativo son esenciales para evaluar el rendimiento de un modelo de *Machine Learning*, ya que permiten medir su capacidad de realizar predicciones precisas y confiables. Entre estas métricas, la matriz de confusión es una herramienta clave para comparar las predicciones con los valores reales, proporcionando información detallada sobre la tasa de aciertos y errores. En el contexto del Sistema de Detección y Alertamiento de Somnolencia (SDAS), su uso es esencial para determinar la efectividad del algoritmo en la clasificación del estado del conductor, diferenciando correctamente entre estados de alerta y somnolencia. A través de métricas derivadas como la precisión, sensibilidad y exactitud, la matriz de confusión permite optimizar el rendimiento del SDAS, asegurando que las detecciones sean confiables y se minimicen falsos positivos y negativos, lo que resulta crucial para la seguridad en la conducción.

La matriz de confusión es una tabla que permite visualizar el rendimiento de un algoritmo de clasificación. Contiene cuatro valores categóricos básicos:

- **Verdaderos Positivos *TP (True Positives)***: casos correctamente identificados como positivos.
- **Falsos Positivos *FP (False Positives)***: casos incorrectamente identificados como positivos.
- **Verdaderos Negativos *TN (True Negatives)***: casos correctamente identificados como negativos.
- **Falsos Negativos *FN (False Negatives)***: casos incorrectamente identificados como negativos.

Precisión (*Precision*)

- **Definición**: La precisión mide la proporción de verdaderos positivos entre el total de predicciones positivas realizadas por el modelo.

- **Fórmula**:

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{2.15}$$

- **Interpretación**: Una alta precisión indica que el modelo tiene pocos falsos positivos.

Sensibilidad o Tasa de Verdaderos Positivos (*Recall*)

- **Definición**: El *recall* mide la proporción de verdaderos positivos entre el total de casos positivos reales. Es la capacidad del modelo para identificar correctamente los casos positivos.

- **Fórmula**:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \tag{2.16}$$

- **Interpretación**: Un alto *recall* indica que el modelo tiene pocos falsos negativos y que el modelo es efectivo en detectar los casos positivos.

Exactitud (*Accuracy*)

- **Definición**: La exactitud mide la proporción de predicciones correctas (tanto positivas como negativas) sobre el total de casos evaluados.

- **Fórmula**:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.17}$$

- **Interpretación**: La exactitud da una visión general del rendimiento del modelo, pero puede ser engañosa en conjuntos de datos desbalanceados.

2.4.9. Dependencias para detección de somnolencia

En esta sección, se presentan las bibliotecas *OpenCV* y *Dlib* con el fin de proporcionar al lector una comprensión completa de las herramientas utilizadas para la implementación

del código en la detección de somnolencia, para saber cómo se combinan e interactúan en el *software*.

En el ámbito de la detección de somnolencia, se han desarrollado herramientas avanzadas que hacen uso de potentes bibliotecas como *OpenCV* y *Dlib*. Éstas proporcionan un conjunto de funciones y algoritmos que permiten identificar patrones y características asociadas con la somnolencia a partir de la detección de rostros y sus elementos.

2.4.9.1. Detección de rostros con dependencia *OpenCV*

OpenCV ofrece la posibilidad de entrenar clasificadores en cascada o utilizar modelos previamente entrenados disponibles en su instalación y documentación oficial. Para este proyecto, se emplea un modelo preentrenado especializado en la detección frontal de rostros: *haar_cascade_frontalface_default.xml*. Este modelo ha sido seleccionado debido a su implementación en el código desarrollado por *Arijit Das* [40], quien lo utiliza como primer paso para la detección facial antes de aplicar un segundo algoritmo de la biblioteca *Dlib* (2.4.9.2). La detección de rostros en este enfoque se basa en el uso del clasificador en cascada de *Haar*, un método ampliamente utilizado por su eficiencia en la identificación de estructuras faciales en tiempo real.

2.4.9.2. Detección de elementos del rostro; dependencia *Dlib*

Dlib es un conjunto de herramientas en lenguaje *C++* que contiene algoritmos de *ML* y modelos para crear *software* complejo en *C++* y también disponible para lenguaje *Python*. Cuenta con herramientas y modelos preentrenados para el procesamiento de imágenes, en específico para la detección de objetos, incluyendo la detección frontal de rostros con una alta calidad para el reconocimiento de rostros.

Uno de los modelos más utilizados para la detección de elementos faciales es *shape_predictor_68_face_landmarks*, perteneciente a la clase *shape_predictor*. Este modelo toma como entrada una región de la imagen que contiene un rostro y genera como salida un conjunto de 68 puntos de referencia que definen la pose y el contorno facial. Su objetivo es identificar con precisión las ubicaciones clave del rostro, como las esquinas de la boca y los ojos, la punta de la nariz y el contorno general de la cara, boca, ojos y cejas. En la

figura 2.21 se muestra un ejemplo de la salida esperada de este modelo.

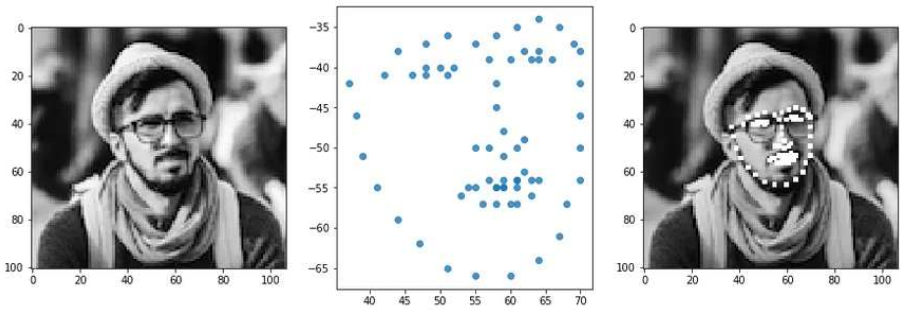


Figura 2.21: Procesamiento de imágenes faciales con el modelo *shape predictor 68 face landmarks*.

Este modelo de preentrenado usa el clásico histograma de gradientes orientados para la toma de características, combinando un clasificador lineal, imagen piramidal y un esquema de detección de ventanas deslizantes. Este estimador o predictor de pose se creó usando la implementación *Dlib* por los autores *Vahid Kazemi* y *Josephine Sullivan* en su artículo *One Millisecond Face Alignment with an Ensemble of Regression Trees* de 2014 [41].

2.4.10. Modelo 68 puntos de referencia faciales de *Dlib* para detección de somnolencia

Algunas publicaciones citadas en este documento presentan prototipos de SDS con un enfoque no intrusivo, basados en la monitorización del estado de los ojos o párpados mediante algoritmos de detección de somnolencia y modelos preentrenados para el procesamiento de imágenes. Entre ellas, "Detección de fatiga del conductor en tiempo real basada en el comportamiento facial junto con enfoques de aprendizaje automático" [12], "Diseño de un sistema de detección de somnolencia en tiempo real utilizando *Dlib*" [14], "Diseño de un sistema de advertencia del sueño en conductores mediante reconocimiento y procesamiento de imágenes en *Python*, *Dlib* y *OpenCV*" [20], "Sistema de detección de somnolencia mediante *KNN* y *OpenCV*" [26] y "Detección de somnolencia a partir de imágenes faciales en medios de vídeo en tiempo real utilizando *Nvidia Jetson Nano*" [21], han implementado el modelo *68 face landmark shape predictor* de la biblioteca *Dlib*. Este modelo permite mapear elementos faciales clave, como la boca, los ojos, la nariz y el contorno

del rostro, a través de 68 puntos de referencia, facilitando la detección de somnolencia a partir del análisis de estos rasgos.

Este modelo no detecta la somnolencia de manera directa, ya que no ha sido entrenado específicamente para identificar estados de fatiga a partir de imágenes. Su entrenamiento no incluyó un proceso de extracción de características basado en una colección de datos con rostros somnolientos y alertas. En cambio, el modelo de la biblioteca *Dlib* fue diseñado y entrenado exclusivamente para la detección facial y la identificación de elementos clave del rostro [41].

En la figura 2.22 se observan los 68 puntos de referencia que mapean el contorno de los elementos del rostro.

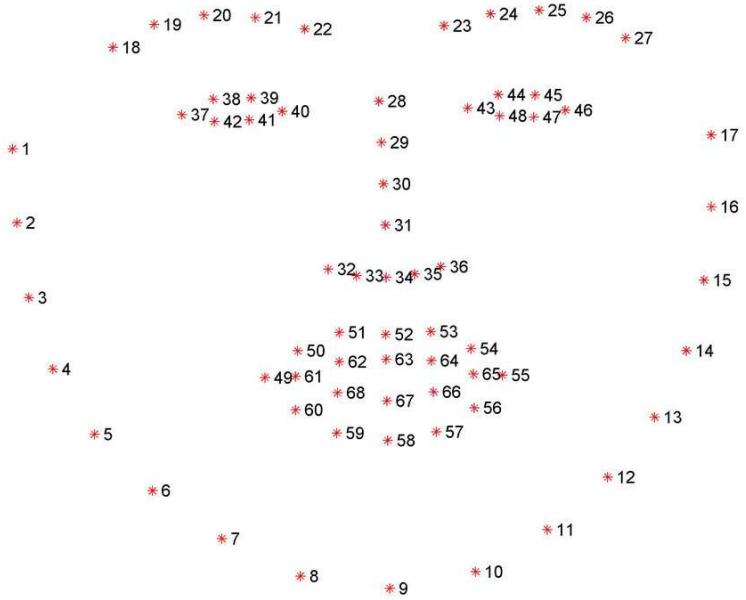


Figura 2.22: 68 puntos de referencia que mapean el rostro.

Estos puntos de referencia o también llamados *landmarks*, posibilitan el cálculo de distancias euclidianas entre diversos puntos. Estas distancias se utilizan para determinar dos métricas conocidas como: "tasa de aspecto del ojo" (*EAR*, por sus siglas en inglés, *Eye Aspect Ratio*) y la "tasa de aspecto de la boca" (*MAR*, por sus siglas en inglés, *Mouth Aspect Ratio*). Estas métricas permiten discernir si los ojos y la boca están abiertos o cerrados y así determinar si existen rasgos de somnolencia.

La expresión matemática que define el *EAR* toma 6 puntos de referencia que mapean

al ojo tales como la figura 2.22 lo muestra. Para el ojo izquierdo son: el 37, 38, 39, 40, 41 y 42. Y para el ojo derecho son: 43, 44, 45, 46, 47, 48. Cabe mencionar que estos puntos se denotan con la letra "p" y se enumeran hasta el 6 como lo muestra la figura 2.23.

2.4.10.1. Tasa de aspecto del ojo *EAR*

La expresión que define el *EAR* (*Eye Aspect Ratio*) utiliza 6 puntos de referencia que están relacionados con la estructura del ojo, como se muestra en la figura 2.22. Para el ojo izquierdo, estos puntos corresponden con las numeraciones 37, 38, 39, 40, 41 y 42, mientras que para el ojo derecho, se relacionan con las numeraciones 43, 44, 45, 46, 47 y 48. Es importante destacar que estos puntos se identifican mediante la letra "p" y están enumerados del 1 al 6, como se muestra en la figura 2.23.

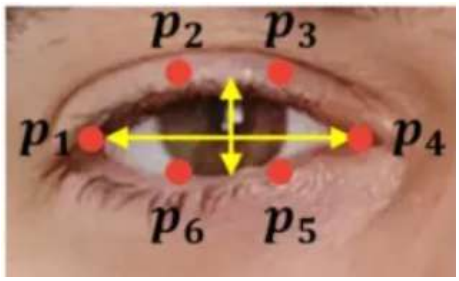


Figura 2.23: Puntos de referencia *Landmarks* del ojo.

A continuación se muestra la expresión algebraica que calcula el valor *EAR*:

$$EAR = \frac{||p2 - p6|| + ||p3 - p5||}{2||p1 - p4||} \tag{2.18}$$

Con respecto a la definición de somnolencia, se emplea el valor *EAR* para definir el estado de los ojos, tal que si este valor se aproxima a cero, el estado del ojo es cerrado y si este valor es mayor, se cataloga que el ojo sigue abierto.

2.4.10.2. Tasa de aspecto de la boca *MAR*

En cuanto a la detección de bostezos, el proceso es similar. Diferentes autores proponen una definición del *MAR* empleando distintos puntos de referencia que rodean a la boca y labios. El autor *Thi Hien* et al., [20] propone en su sistema la toma de los puntos de

referencia que corresponden al exterior de los labios con la numeración 49 a 55 para el labio superior, y 56 a 60 para el labio inferior. Igualmente se denota cada *landmark* con la letra "p" como lo muestra la figura 2.24.

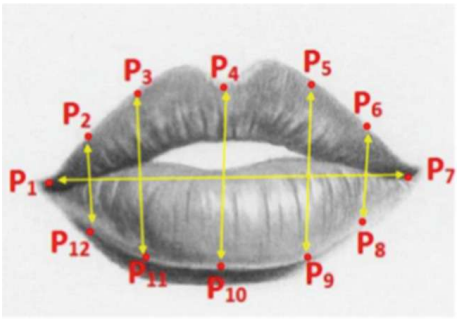


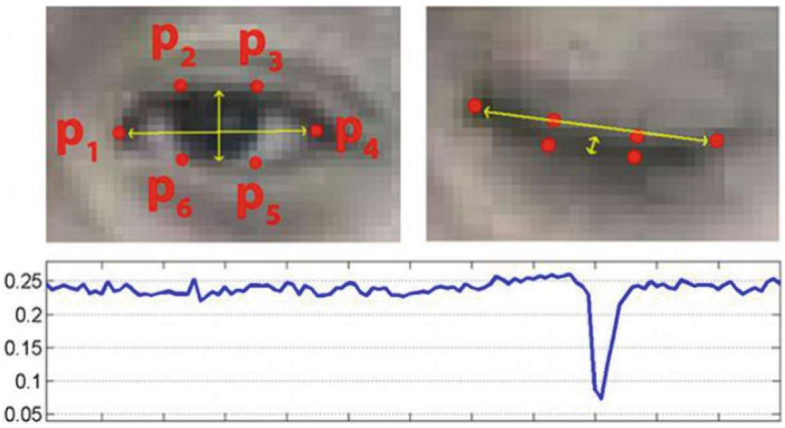
Figura 2.24: Puntos de referencia del contorno de los labios exteriores.

La fórmula que calcula el valor MAR es la siguiente:

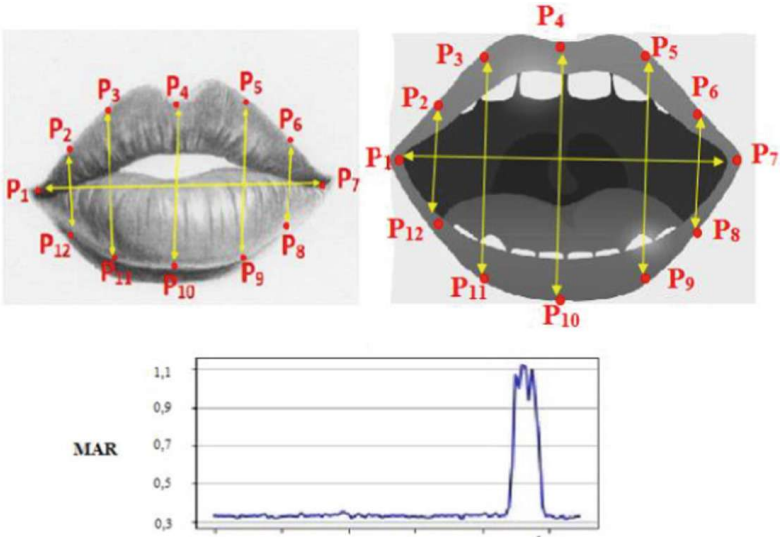
$$MAR = \frac{||p2 - p12|| + ||p3 - p11|| + ||p4 - p10|| + ||p5 - p9|| + ||p6 - p8||}{5||p1 - p7||} \tag{2.19}$$

Diversas publicaciones, como el trabajo realizado por el autor *Thi Hien* et al., [20], han establecido valores umbrales como referencia para la limitación de tasas específicas. Cuando las tasas no cumplen con el umbral predeterminado, se considera que el individuo se encuentra en un estado de somnolencia, lo que activa alertas por parte del prototipo. En la figura 2.25, se presentan estos dos valores umbrales: 0.25 para el EAR (2.25a) y 0.7 para el MAR (2.25b).

Es importante destacar que cuando el ojo se encuentra cerrado, la tasa de EAR disminuye por debajo de 0.1 como se observa en la figura 2.25a. En lo que respecta al valor del MAR, cualquier valor que supere el umbral de 0.7 se clasifica como un bostezo, como se muestra en la figura 2.25b, lo que indica la importancia de estos valores umbrales en la detección efectiva de somnolencia.



(a) Gráfico del valor umbral *EAR*.



(b) Gráfico del valor umbral *MAR*.

Figura 2.25: Valores umbrales y tasas de aspecto para los ojos y boca.

Este capítulo presentó la base teórica para el desarrollo del *software* y la selección del *hardware* del SDAS. Se abordaron conceptos fundamentales en inteligencia artificial y aprendizaje automático, incluyendo el uso y la descripción breve de algoritmos como las Máquinas de Soporte Vectorial y la importancia de métricas como la matriz de confusión para evaluar el desempeño del modelo. Asimismo, se exploraron técnicas clave en visión artificial, como la detección de rostros mediante *Haar cascade* y *HOG*, y el uso de modelos preentrenados, como *shape predictor 68 facial landmarks* de *DLib*, para la identificación de rasgos faciales relevantes en la detección de somnolencia.

Además, se resaltó la importancia del lenguaje de programación *Python* y sus bibliotecas especializadas en aprendizaje automático y procesamiento de imágenes, como *OpenCV* y *DLib*, para la implementación eficiente del sistema. También se enfatizó el papel de los sistemas embebidos y la arquitectura de computadoras en el desarrollo de un sistema de detección de somnolencia en tiempo real.

En el próximo capítulo (3), se detallará la selección del *hardware* y *software* implementado y la justificación de cada componente del SDAS, además de la integración de los elementos teóricos expuestos en este capítulo para consolidar el prototipo de SDAS.

Capítulo 3

Desarrollo del Hardware y Software

A continuación, se aborda en detalle el desarrollo del sistema de detección y alerta de somnolencia en conductores, dividiéndolo en dos secciones. Primero se aborda el apartado físico correspondiente al *hardware* donde se muestra el proceso de diseño en base a los requerimientos del algoritmo de detección y alertamiento de somnolencia seleccionado. Posteriormente se describe la implementación del *software*, destacando apartados importantes de código para la detección y alertamiento de somnolencia.

3.1. Diseño de la solución

La problemática de los accidentes automovilísticos originados por la somnolencia en conductores representa un desafío significativo en la seguridad vial. La somnolencia al volante es un factor que puede mermar la capacidad de respuesta y el juicio de los conductores, poniendo en peligro su vida, la de los pasajeros y la de otros usuarios de la carretera.

En este contexto, el propósito del desarrollo de un prototipo de un sistema de detección y alertamiento de somnolencia (SDAS) en conductores pretende brindar una advertencia temprana a los conductores cuando su nivel de somnolencia se torna peligroso, con el fin de prevenir potenciales tragedias viales y de contribuir a la reducción de accidentes automovilísticos causados por somnolencia en conductores.

Este SDAS busca mitigar esta problemática mediante un prototipo de monitoreo facial del conductor, que analiza patrones de comportamiento en busca de signos de somnolencia. Al detectar estos síntomas, el sistema genera alertas a través de llamadas telefónicas y avisos por voz mediante un parlante o el estéreo, advirtiendo sobre el peligro de quedarse dormido al volante.

El SDAS diseñado en este trabajo de tesis parte de la definición de un sistema de visión artificial, enfocado en la detección de somnolencia, donde un sistema de visión se compone de tres elementos clave: sensor óptico para captura de imagen, un módulo de procesamiento de video para el análisis de patrones faciales y un sistema de control encargado de gestionar los actuadores y dispositivos externos [34].

El diseño propuesto en esta obra se basa en una combinación e integración de *hardware* y *software* seleccionados para realizar la acción de alertamiento mediante alertas en forma de voz y llamadas telefónicas después de haber realizado una detección verdadera de patrones faciales de somnolencia en el conductor.

La figura 3.1 muestra el diagrama de bloques que representa la configuración de *hardware* y *software* seleccionados para este diseño, considerando los tres principales bloques de un sistema de visión y un bloque extra que describe el proceso de alimentación de energía del sistema.

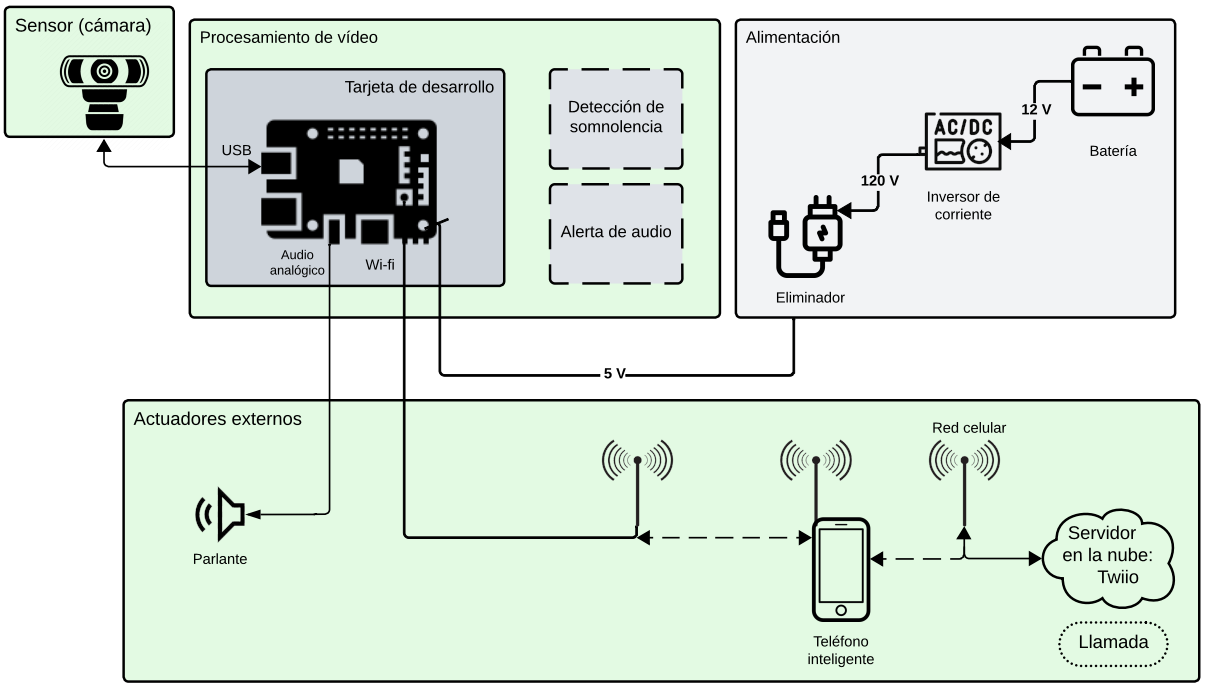


Figura 3.1: Diagrama de bloques del sistema propuesto.

Por lo tanto, fue fundamental seleccionar cuidadosamente cada componente del sistema de visión. En cuanto al *hardware*, se eligió una cámara como sensor de vídeo para monitorizar al conductor a través de la captura continua de imágenes, una tarjeta *SBC* para el procesamiento del algoritmo de detección de somnolencia dentro de un sistema empotrado y, como actuadores, un parlante para la emisión de alertas en forma de voz y un teléfono inteligente para la recepción de llamadas de advertencia, como se muestra en la figura 3.1.

Para el funcionamiento de un sistema de visión y de este diseño SDAS, es necesario garantizar la alimentación de energía continua del sistema en un entorno automotriz. Entonces, se requirió energizar al sistema empotrado con un eliminador conectado a un inversor de corriente que conecta el sistema a la batería del automóvil. La figura 3.2 presenta el bloque de alimentación con los elementos eléctricos para el energizado del SDAS.

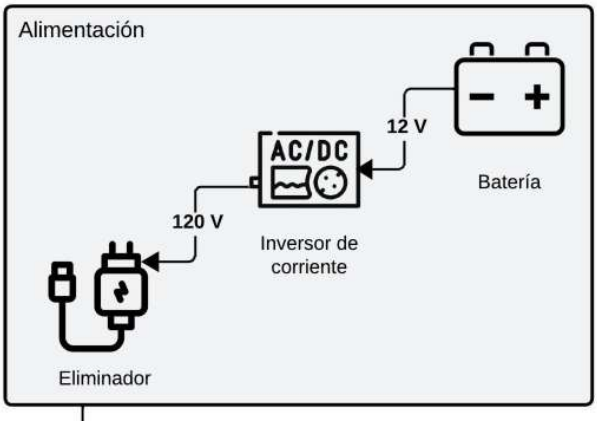


Figura 3.2: Bloque de energizado continuo del sistema empotrado.

En cuanto al *software*, se emplea un programa de detección de somnolencia desarrollado por un colaborador de la plataforma *GitHub*, el cual se adapta a la arquitectura del algoritmo propuesto en este trabajo de tesis. El diseño planteado se basa en una arquitectura modular, descomponiendo el *software* en una lógica de servicios para mejorar su escalabilidad y mantenimiento. La aplicación se compone de múltiples subaplicaciones, denominadas servicios, que trabajan en conjunto para la detección y alertamiento de somnolencia. Entre estos servicios se incluyen:

- Detección de somnolencia.
- Alertas en forma de voz.
- Llamadas.
- Servidor *Bluetooth*.

Éste último encargado de recibir el dato del número telefónico del conductor a través de una aplicación cliente *Bluetooth* diseñada para dispositivos *Android*. Esta arquitectura modular se propone como la base para el sistema de detección y alertamiento de somnolencia en conductores.

El proceso de desarrollo del SDAS comienza con la elección de un algoritmo de detección de somnolencia, ya que este paso inicial posibilita la selección del *hardware* adecuado y orienta el diseño de la arquitectura del *software* correspondiente. En este contexto, se

incorpora una sección que, a través de un análisis, propicia la selección apropiada de un algoritmo, haciendo hincapié en la utilización de modelos previamente estudiados y publicados en fuentes o artículos científicos sobre la detección de somnolencia.

3.1.1. Análisis para la selección del algoritmo para detección de somnolencia

De acuerdo con las observaciones presentadas en las tablas 2.4 y 2.5 del capítulo 2, a continuación se analizan las principales similitudes y diferencias entre las publicaciones que abordan la detección de somnolencia.

La similitud más destacada es la detección del estado de los ojos o párpados, presente en todos los estudios revisados. Esto se debe a que la visión es esencial para la conducción, por lo que el monitoreo del estado del párpado (abierto o cerrado), el iris o el ojo en general es un factor clave para identificar la somnolencia. Estos rasgos faciales están directamente relacionados con la somnolencia excesiva y constituyen un criterio fundamental en los sistemas de detección.

Por otro lado, algunos sistemas combinan rasgos faciales característicos de somnolencia excesiva y niveles bajos de somnolencia, tal como el monitoreo de párpados y bostezo. Este último rasgo facial se emplea para la prevención y alertamiento temprano de somnolencia de bajo nivel, pero el bostezo es un rasgo facial causado por otros comportamientos del cuerpo humano y no solamente por somnolencia.

El cabeceo o inclinación del cuello es otro patrón común en los síntomas de somnolencia de alto nivel, generalmente acompañado por la constricción ocular. Varias publicaciones han implementado esta característica en sus modelos de detección.

Finalmente, la detección de microexpresiones y otros comportamientos físicos y faciales de somnolencia es un concepto que sigue en desarrollo y experimentación con modelos que requieren un entrenamiento con bases de datos más personalizadas y alto poder de cómputo.

Con base en este análisis, se concluyó que el rasgo facial más significativo en la detección de somnolencia excesiva es el monitoreo del estado de los párpados o los ojos, ya que estos

indicadores están presentes en la mayoría de los episodios de somnolencia de alto nivel. Por ello, el SDAS implementa un algoritmo que detecta el pestañeo y la apertura o cierre de los párpados, al ser un indicador clave de la somnolencia excesiva.

3.1.2. Algoritmo seleccionado para detección de somnolencia

La detección de somnolencia puede abordarse desde dos enfoques principales: intrusivo y no intrusivo. El enfoque intrusivo implica la utilización de sensores que pueden medir parámetros como la frecuencia cardíaca o la postura del cuerpo del conductor. Sin embargo, en el contexto de la detección de somnolencia en conductores, este enfoque presenta limitaciones significativas, ya que la presencia de sensores y cables en el monitoreo del conductor puede dificultar su capacidad para conducir de manera adecuada.

Por otro lado, el enfoque no intrusivo se centra en la detección y procesamiento de imágenes, lo que permite monitorizar al conductor mediante la captura de vídeo en tiempo real. Este enfoque busca minimizar cualquier interferencia con la maniobrabilidad del conductor al utilizar sensores que no requieren contacto físico y que no limitan su libertad de movimiento.

En el contexto de la detección de somnolencia en conductores, el enfoque no intrusivo se presenta como una alternativa más viable y práctica, ya que propicia la detección de manera efectiva sin comprometer la capacidad del conductor para operar el vehículo de manera segura y adecuada.

En este trabajo de tesis, el propósito de la selección de un algoritmo no radica en la creación de un modelo de *Machine Learning* (ML) o *Deep Learning* (DL) a través de la recopilación y procesamiento de datos para su posterior entrenamiento, evaluación y despliegue.

En cambio, se fundamenta en la selección de un modelo que cumpla con tres características principales:

- Enfoque no intrusivo: procesamiento de imágenes.
- Monitoreo del estado de los ojos o párpados (cerrado/abierto).

- Modelo previamente entrenado (preentrenado).

La decisión de emplear un modelo preentrenado se respalda en evitar crear un modelo desde cero, acelerando el proceso de detección, lo que contribuye a la optimización de recursos y tiempos en la investigación.

En este contexto, diversas publicaciones recientes ([12], [14], [20], [26], [21]) han adoptado el uso de modelos preentrenados de la biblioteca *Dlib* para la detección de somnolencia. En particular, el "predictor de forma del rostro de 68 puntos de referencia" ha demostrado ser una opción eficiente para identificar elementos faciales y evaluar distintos niveles de somnolencia, proporcionando una base sólida para la implementación de sistemas de monitoreo basados en visión artificial.

El "predictor de forma del rostro de 68 puntos de referencia" ofrece la capacidad de discernir la posición de los párpados, un aspecto clave para identificar la somnolencia de alto nivel cuando estos se encuentran cerrados. Además, este algoritmo facilita la evaluación del estado de la boca, permitiendo la detección de signos de somnolencia de bajo nivel, como el bostezo. Para ello, se emplea el cálculo de dos métricas principales: *EAR* y *MAR*.

Este modelo fue seleccionado debido a que satisface los requerimientos y necesidades para su integración en el *software* del SDAS, que sea un modelo previamente entrenado y que realice el monitoreo de los párpados mediante un enfoque no intrusivo.

Un programa clave para el despliegue del algoritmo en el SDAS es el desarrollado por el usuario "Arijit1080", quien creó un sistema en *Python* basado en el modelo "predictor de forma del rostro de 68 puntos de referencia" de la biblioteca *Dlib*. Su proyecto, titulado "*Drowsiness-and-Yawn-Detection-with-voice-alert-using-Dlib*", está disponible en su repositorio de *GitHub* [40].

Este programa combina el modelo de "detección frontal de rostros" de *OpenCV* con el modelo de "68 puntos de referencia faciales" para identificar primero el rostro y luego analizar en tiempo real signos de somnolencia, como el cierre de párpados o el bostezo. Además, integra alertas de voz mediante la biblioteca *Text2Speak*, lo que permite advertir al usuario sobre su estado.

El algoritmo desarrollado por *Arijit Das* [40] sigue un diseño similar al de Thi Hien et al. [20], con dos diferencias clave. Primero, *Arijit Das* incorpora un filtro basado en el

modelo de detección de rostros de *OpenCV* para verificar la presencia de un rostro antes de aplicar el modelo de "68 puntos de referencia faciales".

Además, define la tasa de aspecto de la boca (MAR) considerando los puntos 61-64 del labio superior y 65-68 del labio inferior, evaluando así el contorno interior de los labios para calcular su distancia. Si MAR supera un umbral predefinido, se detecta un bostezo y se activa una alerta de voz.

El trabajo de *Arijit Das* fue seleccionado por su implementación del modelo de 68 puntos de referencia, su código abierto y su capacidad para monitorear párpados y detectar bostezos de manera no intrusiva.

Para ejecutar la detección de somnolencia en vídeo en tiempo real, el código requiere ciertos componentes de *hardware*, cuya selección se detalla en la sección 3.2.

3.2. Análisis del *hardware* requerido

A continuación, se describe el diseño del *hardware* propuesto, así como la justificación de la selección de cada componente mostrado en el diagrama de bloques de la figura 3.1. La elección de los elementos se basa en los requerimientos del algoritmo desarrollado por *Arijit Das* (sección 3.1.2) y en el análisis de trabajos previos que emplean el modelo "predictor de forma del rostro de 68 puntos de referencia" de la biblioteca *Dlib* ([12], [14], [20], [26], [21]).

El diseño del *hardware* se estructura a partir de la implementación de dicho modelo de detección facial, el cual requiere un sistema de visión artificial compuesto por los siguientes elementos:

- Sensor: cámara para la captura de imágenes.
- Unidad de procesamiento: análisis y procesamiento de imágenes.
- Sistema de alertamiento: notificar al usuario ante la detección de somnolencia.

3.2.1. Captura, procesamiento y análisis de imágenes

El estudio de literatura muestra que diversas investigaciones ([11], [19], [20], [21]) emplean computadoras de tarjeta única (*SBC*, por sus siglas en inglés) como unidad de procesamiento para sistemas de detección de somnolencia. Estas tarjetas integran procesadores de al menos 1 GHz, múltiples núcleos y una unidad gráfica (*GPU*) integrada o dedicada. Además, permiten la conexión de cámaras *USB* y dispositivos de alerta, como alarmas sonoras, vibración o luces *LED*.

Para el desarrollo del Sistema de Detección y Alertamiento de Somnolencia (SDAS), se requiere una tarjeta *SBC* con las siguientes características mínimas:

- **Procesamiento**: procesador de al menos 1 GHz, 4 núcleos y *GPU* integrada o independiente, capaz de ejecutar simultáneamente el sistema de detección y las alertas.
- **Conectividad**: puertos *USB* para cámaras digitales, salida de audio y preferiblemente conexión a internet para llamadas o notificaciones remotas.

Respecto a la captura de imágenes, se requiere una cámara digital con resolución mínima de 640x480 píxeles, suficiente para un correcto funcionamiento de los algoritmos *Haar cascade* de *OpenCV* y *68 facial landmarks* de *Dlib* [37], [42]. Aunque resoluciones mayores mejoran la precisión, también incrementan el tiempo de procesamiento.

Finalmente, algunos prototipos ([19], [20], [21]) ya han validado con éxito la implementación de estos algoritmos en tarjetas *SBC*, aprovechando su capacidad de conexión con cámaras digitales y sistemas de alertamiento mediante audio, vídeo o redes inalámbricas.

3.2.1.1. Sistema empotrado para el procesamiento de imágenes y ejecución de aplicaciones

Para realizar el procesamiento de imágenes y la ejecución de aplicaciones, se requirió una unidad de procesamiento *SBC*. Se seleccionó la tarjeta de una sola placa *Raspberry Pi 4 model B* como sistema empotrado porque cumple con los requerimientos necesarios para el procesamiento y análisis de imágenes, ofrece la conectividad para la transmisión de audio analógico, conectividad *USB* para la conexión de la cámara digital (sensor), acceso a redes inalámbricas y conexión *bluetooth*.

Entre los factores más importantes que implican la selección de una unidad de procesamiento son realizar un análisis de alcance o cobertura de comunicaciones, poder de procesamiento de datos y puertos *GPIO*, de aplicación o consumo y gestión de energía.

La selección de este modelo se basó en un análisis comparativo, resumido en la tabla 2.1, donde se muestran las principales diferencias entre las tarjetas empleadas en prototipos de detección de somnolencia reportados en estudios previos ([19], [20], [21]) del capítulo 2.

En la tabla comparativa 2.1 del capítulo 2, se presentan las características de tres modelos de tarjetas pertenecientes a las plataformas *Raspberry Pi*, *NVidia* y *BeagleBone* que contemplan los requerimientos mínimos para el procesamiento de imagen y multitarea. La elección de estos entornos se fundamentó en factores clave, incluyendo el poder de procesamiento de imágenes, su popularidad, asequibilidad y disponibilidad en el mercado, soporte de *software*, así como la adopción de estas plataformas en el desarrollo de prototipos de SDS que han sido documentados en la literatura científica.

Al evaluar la relación costo-beneficio, conectividad y capacidad de procesamiento, se

optó por elegir la tarjeta *Raspberry Pi 4 model B* por las siguientes razones:

- **Conectividad**: Este modelo presenta los módulos inalámbricos necesarios integrados ya en la placa, sin adquirirlos e integrarlos externamente y que agrega un costo adicional en comparación con la placa *Nvidia Jetson nano* o *BeagleBone*. Además, esta *SBC* presenta un puerto de salida de audio *Jack* 3.5 para conectar un parlante o sistema de audio.

- **GPIO**: Para el diseño del prototipo no es necesario el uso de una gran cantidad de puertos si se implementan redes inalámbricas.

- **Costo-Procesamiento**: La relación costo-desempeño es muy distante entre la tarjeta de la marca *Nvidia* y *Raspberry* ya que estos dispositivos están diseñados para aplicaciones diferentes y que *Jetson nano* presenta una unidad de procesamiento gráfico dedicada.

3.2.1.2. Sensor (Cámara) para la captura de imágenes

Para la implementación de los algoritmos de detección facial *Haar Cascade* y *Dlib*, se recomienda una resolución mínima de 640x480 píxeles. No obstante, resoluciones superiores, como 1280x720 o 1920x1080 píxeles (*Full HD*), permiten capturar con mayor detalle características clave del rostro, como el contorno de los párpados, mejorando así la precisión en tareas exigentes como la detección de somnolencia.

Asimismo, se recomienda una tasa mínima de 15 cuadros por segundo (fps) para lograr detección en tiempo real. Sin embargo, una tasa igual o superior a 30 fps mejora significativamente la fluidez y precisión del seguimiento facial, especialmente ante movimientos rápidos o cambios de pose, según la documentación oficial de *OpenCV* [37] y *Dlib* [42].

Otro criterio fundamental es la compatibilidad de los controladores de la cámara con el sistema operativo utilizado. En este caso, se requiere compatibilidad con sistemas basados en *Linux Debian*, ya que este es el núcleo de *Raspberry OS*, el sistema operativo de la tarjeta *SBC* seleccionada.

Por lo anterior, se eligió una cámara web con salida *Full HD* (1920x1080 píxeles) y una tasa de captura de 30 fps, como se muestra en la figura 3.3. Este modelo supera amplia-

mente los requisitos mínimos, permitiendo un procesamiento de imágenes más preciso a costa de una mayor demanda de recursos computacionales. La conexión con la tarjeta *SBC* se realiza mediante un puerto *USB*. Además, su ficha técnica garantiza compatibilidad con los principales sistemas operativos, incluyendo *Windows OS*, *iOS* y *Linux OS*.

Figura 3.3: Sensor: cámara web *Full HD* a 30 cuadros por segundo.

3.2.2. Alertamiento auditivo: alertas en forma de voz y llamada telefónica

El proceso de alertamiento se activa una vez que el sistema de detección determina un estado positivo de somnolencia en el conductor. Ante esta condición, se emite una advertencia sonora con el objetivo de informar al conductor, o en su caso a los pasajeros, que se han detectado características faciales asociadas a un nivel alto o bajo de somnolencia.

El algoritmo propuesto en este trabajo de tesis contempla un sistema de alertamiento auditivo basado en dos mecanismos principales: generación de alertas por medio de mensajes de voz locales y la realización de llamadas telefónicas de advertencia. En consecuencia, los requerimientos mínimos que debe cumplir el sistema de cómputo embebido son los siguientes:

- Entradas y salidas de audio analógico o de propósito general (GPIO), para la integración de módulos de audio en el dispositivo de cómputo para la conexión de un actuador que presente condiciones de audio aceptables para un alertamiento.
- Conectividad inalámbrica, para realizar llamada telefónica consumiendo un servicio

en la nube desde el dispositivo de cómputo como método de alertamiento inalámbrico (*Bluetooth o wifi*).

Por ello, se eligió el sistema *SBC Raspberry pi 4 model B* ya que cuenta con una salida de audio estéreo de 4 polos para la transmisión de audio analógico usando su salida de tipo AUX, a la cual se le conectó un amplificador de sonido para realizar pruebas experimentales de audio de las alertas en forma de voz.

Adicionalmente, para la ejecución del segundo mecanismo de alertamiento —la llamada telefónica—, el modelo *Raspberry Pi 4 Model B* resulta adecuado debido a que incorpora módulos de conectividad inalámbrica integrados (*Wi-Fi* y *Bluetooth*), permitiendo la ejecución de un programa capaz de consumir un servicio en la nube especializado en llamadas automatizadas y, posteriormente, enviar la alerta a través de la red celular.

3.2.2.1. Alertamiento auditivo mediante un amplificador de audio

Para el sistema de alertamiento sonoro, se integró un amplificador de audio conectado a la salida *Jack* de 3.5 mm de la tarjeta *Raspberry Pi 4 Model B*, permitiendo la reproducción de alertas en formato de voz cuando se detecta un patrón de somnolencia facial.

Esta solución se eligió por su facilidad de implementación, bajo costo y efectividad para emitir alertas audibles sin afectar la ergonomía del vehículo ni generar distracciones visuales, a diferencia de otras alternativas como luces LED o vibradores en el asiento.

La potencia específica requerida de la bocina para el sistema de detección y alertamiento de somnolencia depende de varios factores, pero para la experimentación del prototipo, se utilizó una bocina genérica de 5 Watts de potencia ya que es una alternativa viable porque la distancia entre el conductor y la bocina no es mayor a 1.5 metros y se recomienda de 5 a 10 watts. Una opción más acertada a la realidad es la utilización del equipo de audio del automóvil para transmitir las alertas, pero para fines de experimentación y prueba del prototipo, las necesidades para la transmisión de alertas sonoras no son de alta prioridad, sino observar el desempeño del sistema de detección y tomar en cuenta un posible trabajo futuro en este aspecto.

La figura 3.4 muestra la conexión del amplificador de audio a la tarjeta *SBC* mediante un cable *Jack* de 3.5 mm.

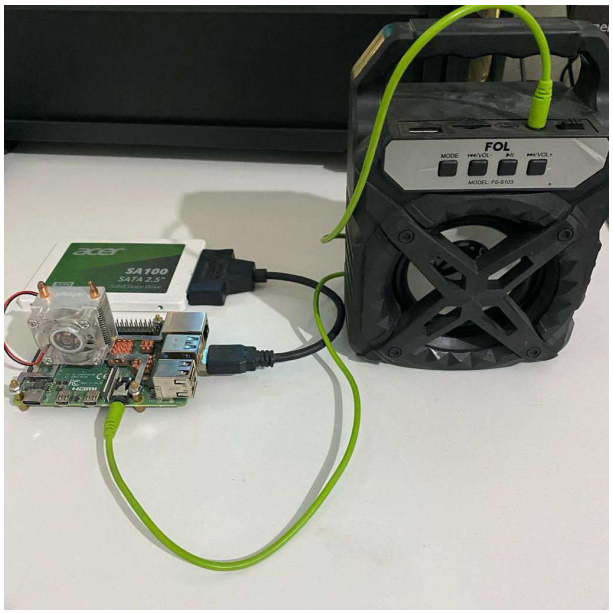


Figura 3.4: Amplificador de audio como actuador externo para emitir alertas de voz.

3.2.2.2. Alertamiento auditivo mediante una llamada telefónica

Como método adicional de alerta, se integró una llamada telefónica, ya que esta tiene la ventaja de interrumpir cualquier sonido activo en un teléfono inteligente, incluso en modo silencioso o vibración. En caso de no contestar, las alertas de voz se reproducen directamente desde la *Raspberry Pi*.

Se evaluaron distintas alternativas para realizar esta llamada:

- Incorporar un módulo celular en la tarjeta *SBC*, lo cual se descartó por su complejidad y costo.
- Emular un tono de llamada, opción innecesaria debido a las alertas de voz ya existentes.
- Simular una llamada a un *smartphone* vía red local, descartada por requerir programación específica de *Android* o *iOS*.

La solución adoptada fue el uso de un servicio de llamadas en la nube, integrable mediante una *API* en *Python*. Este método permite realizar llamadas reales a un número telefónico definido, facilitando el proceso de alertamiento remoto a través de la red y con un bajo costo para pruebas experimentales del prototipo.

3.2.2.3. Comunicación *Bluetooth* como medio para transmisión del número telefónico de destino

Partiendo del objetivo de enviar el número telefónico a la tarjeta *SBC*, se requirió la tecnología de red *BLE* ya que presenta diferentes ventajas significativas en comparación con *Zigbee, Wi-Fi o Bluetooth 3.0*.

En primer lugar, muchas de las tarjetas disponibles en el mercado ya integran a la placa módulos *Wi-Fi* y *BLE*; sin embargo, aunque existen tarjetas que incluyen módulos *Zigbee* o *Bluetooth 3.0*, sigue siendo poco frecuente y es necesario adquirir el módulo de *hardware* de forma externa para adaptarlo a un *GPIO*, debido a que estos tipos de red inalámbrica (*Zigbee* y *Bluetooth 3.0*) están pensados para aplicaciones específicas en proyectos que requieren conexiones inalámbricas con una velocidad de transferencia de datos más alta o redes más seguras para evitar interferencias.

De la tabla 2.3 presentada en el capítulo 2, se observa que la tecnología *BLE* consume menos energía que *Wi-Fi* en relación al valor de potencia y *BLE* está enfocada para aplicaciones de bajo consumo si se piensa emplear para dispositivos portátiles o de batería. Por otro lado, es importante mencionar que enviar un número telefónico no es un dato que requiere una alta velocidad de transferencia; no obstante, este valor de velocidad de transferencia de *BLE* es superior con un consumo de potencia más bajo en comparación con *Zigbee*.

Además, resulta sencillo de configurar y emparejar dispositivos mediante *BLE*, ya que no requiere una infraestructura de red como *Wi-Fi* para su programación. En definitiva, *BLE* es la conexión inalámbrica que se eligió para enviar un número telefónico a la tarjeta de desarrollo que estará ejecutando el *software* para el SDAS debido a que muchas tarjetas del mercado ya cuentan con el módulo *BLE* y el módulo *Wi-Fi* integrado a la placa. Otras razones importantes son: el SDAS no requiere un alcance superior a los 10 metros, *BLE* cuenta con bajo consumo de energía, la velocidad de transferencia es aceptable en relación a los datos que se piensan transferir y por la facilidad de uso y programación en comparación con *Zigbee, Wi-Fi* o *Bluetooth 3.0 HS*.

3.2.2.4. Comunicación *Wi-fi* para acceder a servicios en la red

Se requirió el uso de *Wi-fi* para establecer conectividad a Internet desde una tarjeta *Raspberry Pi 4 model B* y, por ende, realizar solicitudes a un servidor con el propósito de ejecutar una *API* de llamadas telefónicas.

Esta elección surge por la necesidad de realizar una llamada telefónica real y automatizada utilizando un servicio en la nube y sencillo de implementar. Una opción más viable es emular el comportamiento de una llamada, pero requiere una investigación más profunda sobre la programación y la comprensión del proceso que realiza una llamada en un sistema operativo *android*.

La utilización de *Wi-fi* se justifica por las siguientes razones:

1. **Conectividad inalámbrica:** *Wi-fi* proporciona una conectividad inalámbrica útil para el funcionamiento del sistema de alertamiento en forma de llamadas en un entorno móvil como un vehículo. Esto propicia que el dispositivo pueda acceder a Internet, independientemente de su ubicación.

2. **Acceso a servicios remotos:** La conexión a Internet a través de *Wi-fi* habilita la posibilidad de acceder a servicios y recursos remotos, como la ejecución de una *API* de llamadas telefónicas. Esto es necesario para activar el proceso de alertamiento en forma de llamada telefónica cuando se detecta somnolencia en el conductor.

3. **Rapidez en la comunicación:** *Wi-fi* ofrece una velocidad de comunicación adecuada para la transmisión de datos en tiempo real, lo que es esencial para la detección y respuesta rápida a episodios de somnolencia en el conductor.

Por lo tanto, la elección de *Wi-fi* como medio de acceso a Internet desde la tarjeta *SBC* se basa en su capacidad para proporcionar una conectividad inalámbrica confiable y versátil, permitiendo así la ejecución de una *API* de llamadas telefónicas como parte del proceso de alertamiento de somnolencia en forma de llamada.

3.2.2.5. *Smartphone* como *Router wi-fi*

El uso de un servicio de llamadas automatizadas en la nube requiere que el sistema empotrado tenga acceso a Internet. Actualmente, muchos automóviles ofrecen planes de pago que permiten el acceso a Internet desde el propio vehículo. Sin embargo, una alternativa eficiente y de bajo costo es utilizar el *smartphone* del conductor como punto de acceso *Wi-Fi* para proporcionar conectividad a la tarjeta *SBC*.

Al utilizar un *smartphone* como enrutador *Wi-Fi* para proporcionar acceso a Internet a la tarjeta *Raspberry Pi*, se proporciona la conectividad incluso en áreas sin infraestructura de Internet directa. Además, para propósitos experimentales, esta opción resulta más económica que adquirir un plan de Internet a bordo y evita costos adicionales.

Por lo tanto, el uso del *smartphone* como enrutador *Wi-Fi* representa una alternativa práctica y rentable para la conexión a Internet de la *Raspberry Pi*, permitiendo al *SDAS* realizar llamadas telefónicas automatizadas a través del software desarrollado.

La figura 3.5 ilustra cómo el *smartphone* funciona como enrutador para acceder a Internet y realizar llamadas utilizando el servicio de llamadas en la nube.

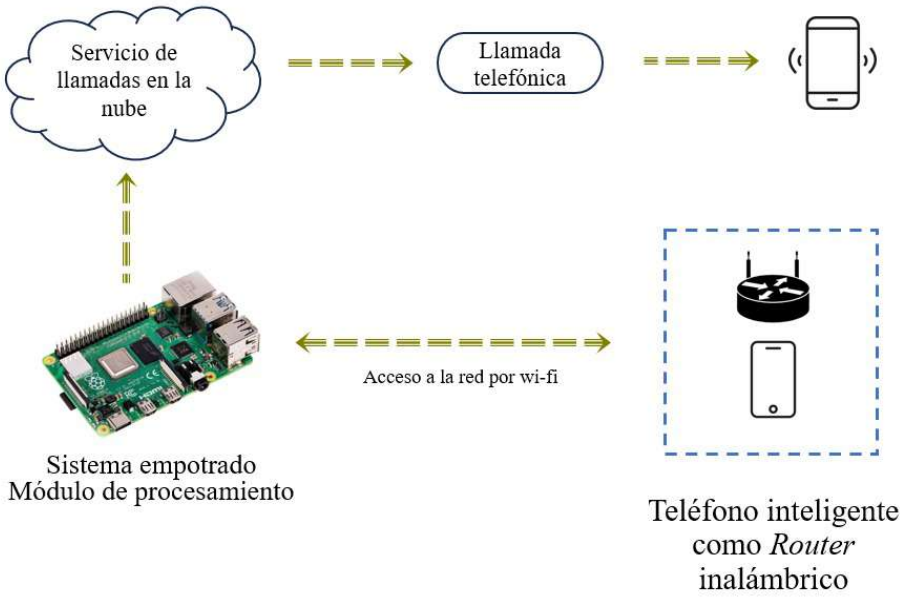


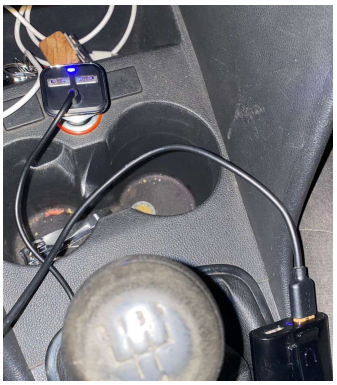
Figura 3.5: Proceso de conectividad y acceso a la red para ejecutar una llamada telefónica como alerta de somnolencia.

3.2.3. Alimentación del SDAS

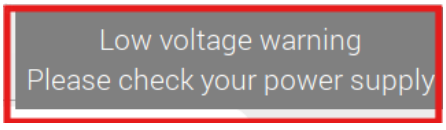
La *Raspberry Pi 4 Model B* es una placa base de cómputo de bajo costo que puede alimentarse de una variedad de fuentes, incluyendo una batería de automóvil. Para ello, se requiere un sistema de alimentación que convierta el voltaje y la corriente de la batería del automóvil a 5 voltios y 3 amperios, según los requerimientos de alimentación de la ficha técnica de *Raspberry Pi 4*.

Durante las pruebas del sistema de detección y alertamiento de somnolencia, se observó que el uso de un regulador de voltaje genérico 3.6b, del tipo conector para encendedor de cigarrillos, no proporciona una alimentación adecuada a la tarjeta cuando se utiliza un cable USB como medio de conexión. Mediante mediciones realizadas con multímetro en los pines *GPIO* 2 (5V) y 6 (GND), se registraron picos de voltaje que apenas alcanzaban los 4.71V, como se observa en la figura 3.6a, con caídas intermitentes que activaban mensajes de advertencia de bajo voltaje en el escritorio del sistema operativo mostrado en la figura 3.6c.

(a) Voltaje en pines 2 y 6 en multímetro.



(b) Regulador de voltaje con salida USB.

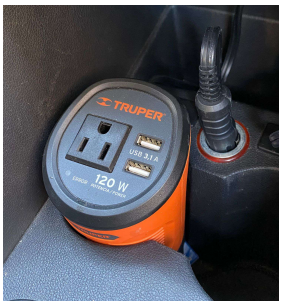


(c) Alerta de bajo voltaje en escritorio remoto.

Figura 3.6: Prueba de alimentación de la *SBC* usando regulador de voltaje con salida USB.

Esta condición afectaba directamente el rendimiento de la tarjeta, generando lentitud

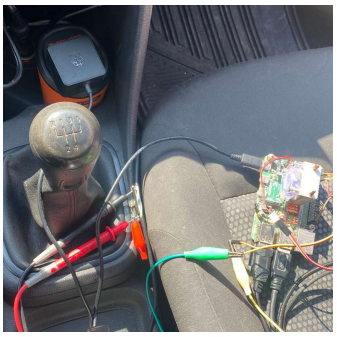
o apagones intermitentes durante la ejecución del *software* de detección y alertamiento. En contraste, al emplear un inversor de corriente de 12V a 120V, figura 3.7a, acompañado por la fuente oficial de *Raspberry Pi* (5.1V 3A), figura 3.7b, se logró una entrega estable de energía, con valores medidos entre 5.00V y 5.12V en los mismos pines, como lo marca la figura 3.7c, y con una variación mínima incluso durante la ejecución de la aplicación de detección y alertamiento de somnolencia. Cabe señalar que la *Raspberry Pi 4 Model B* puede llegar a requerir corrientes superiores a los 2.5A en carga alta, por lo que el uso de componentes de baja calidad incapaces de suministrar al menos 3A sostenidos compromete la estabilidad y la fiabilidad de las detecciones realizadas por el sistema.



(a) Inversor de corriente.

(b) Eliminador.

(c) Voltaje en pines 2 y 6 usando fuente oficial *raspberry*.



(d) Prueba de medición inversor, fuente, *SBC*.

Figura 3.7: Sistema de alimentación para la placa *Raspberry Pi 4 Model B*.

Este sistema de alimentación de una *Raspberry Pi 4 Model B* conectada a una batería de automóvil es eficiente y confiable.

3.3. Implementación de Software

En la figura 3.8 se muestra el diagrama de flujo del software del SDAS que describe el proceso del algoritmo para la detección y alertamiento de somnolencia.

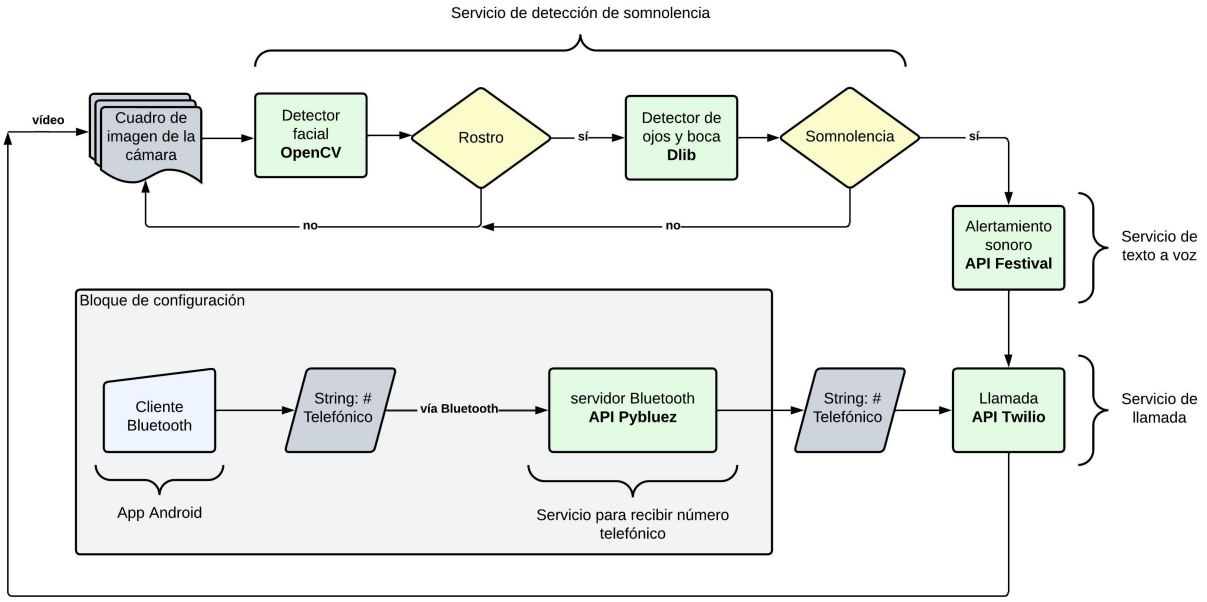


Figura 3.8: Diagrama de flujo del sistema de detección y alertamiento de somnolencia del apartado de *software*.

El *software* de la aplicación de detección y alertamiento de somnolencia se basa en una arquitectura de servicios, que es un conjunto de aplicaciones, donde cada servicio (aplicación) realiza una o varias tareas y se ejecuta de forma independiente. Estos servicios se interconectan usando interfaces de aplicación programables, mejor conocidas como *API's*, con la finalidad de generar una aplicación de aplicaciones y estas aplicaciones intercambian datos específicos entre sí.

La integración de este tipo de arquitectura propicia una mejor compatibilidad y ejecución simultánea en la tarjeta de desarrollo, además de mantener un orden y distinguir las diferentes partes del sistema, pero principalmente para que la ejecución de las aplicaciones sea de forma independiente. Así se mejora la trazabilidad de errores en comparación con un código de arquitectura monolítica, ya que si falla alguna parte o un servicio, esta no afecta a toda la aplicación.

La aplicación se compone de los siguientes servicios:

1. Servicio de detección de somnolencia (3.4.2).
2. Servicio de llamadas telefónicas (3.4.4).
3. Servicio de alarma texto a voz (3.4.3).
4. Servicio *Bluetooth server* (3.4.5).

En primer lugar, se ejecuta el servicio de detección de somnolencia que realiza la captura de cuadros de imagen en tiempo real mediante el sensor (cámara web). Seguidamente, cada cuadro de imagen entra a un proceso de detección del rostro, donde el modelo preentrenado de detección frontal de rostros *Haar cascade*, de la biblioteca *OpenCV*, determina si en el cuadro de imagen se encuentra un rostro, por lo que si esta sentencia es correcta, se procede a la ejecución del siguiente proceso y si no es así, el servicio sigue tomando cuadros de imagen hasta encontrar un rostro en un siguiente cuadro.

Posteriormente, el cuadro entra al proceso de detección de los ojos y la boca usando el modelo de predicción de forma del rostro de 68 puntos de referencia de la biblioteca *Dlib*, donde una lógica de decisión determina la presencia o ausencia de síntomas de somnolencia a partir de la evaluación de las métricas EAR y MAR para definir el estado de los ojos y la boca.

En caso de una detección incorrecta de síntomas de somnolencia, se procede a capturar otro cuadro de imagen distinto. Sin embargo, si la detección indica la presencia de síntomas de somnolencia, se activan los servicios de texto a voz y llamadas telefónicas. Una vez completado este proceso de alerta, se reanuda la captura de cuadros de imagen para seguir monitoreando al conductor.

Para habilitar la ejecución de llamadas telefónicas, se requiere el envío de un número telefónico al servicio de llamadas en la tarjeta de desarrollo. Con el propósito de evitar la interacción directa del usuario con el sistema empotrado, se implementó una sección de configuración asíncrona. En este bloque, se programó un servicio cliente-servidor a través de una conexión *Bluetooth*, que transmite y recibe cadenas de caracteres correspondientes a números telefónicos. La introducción del número telefónico se lleva a cabo mediante una

aplicación diseñada para dispositivos con sistemas operativos *Android*. Luego, el dato es enviado desde el cliente *Bluetooth* (aplicación *Android*) al servidor (servicio para recibir cadenas vía *Bluetooth*) en la tarjeta de desarrollo, donde posteriormente es utilizado por el servicio de llamadas.

Por otro lado, la aplicación *Android* no se contempla como servicio, ya que es un proceso indirecto del sistema, pero es necesario para el funcionamiento de las alertas en forma de llamadas.

3.4. Arquitectura de servicios

El desarrollo de la aplicación se basa en dos bloques principales: el bloque de desarrollo de servicios y el bloque de producción. En el bloque de desarrollo, se prioriza el diseño, prueba y contenedorización (empaquetado) de cada servicio como una aplicación individual, como se muestra en la figura 3.9

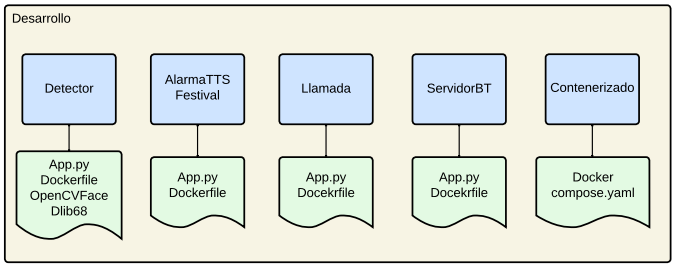


Figura 3.9: Diagrama del bloque de desarrollo de los servicios (aplicaciones).

En primer lugar, en el bloque de desarrollo se implementó el código de cada servicio y luego se diseñó la arquitectura del software utilizando interfaces de programación programables, así como una base de datos como un bus de comunicación e intercambio de datos entre servicios. El siguiente paso fue la contenedorización mediante la herramienta *Docker*. Inicialmente, se construyó la imagen de cada servicio, que es una plantilla que permite ejecutar comandos para la instalación de bibliotecas junto con su versionado o archivos de configuración necesarios para el servicio; esta plantilla se programa en el archivo llamado $Docker file$. Luego, se crearon los contenedores, los cuales empaquetan la imagen de un servicio asignándole un nombre, una red, una dirección IP, un límite de memoria y la cantidad de CPU asignadas a cada contenedor, utilizando la imagen de cada servicio. Para concluir este proceso, se utilizó la herramienta *Docker Compose* para permitir que los contenedores sean ejecutados y detenidos simultáneamente, además de establecer la conexión al bus de datos (base de datos) a través de la implementación de una red.

El bloque de producción implica reubicar los archivos de aplicación *Python* de cada servicio en una ruta diferente para mejorar la trazabilidad y el orden. Esta ubicación específica se definió en el archivo *Docker Compose yaml* para que estos archivos sin errores sean los que va a ejecutar cada contenedor y no los archivos de prueba.

Finalmente, los contenedores son activados por un bloque orquestador, el cual, al ejecutar el archivo *main.py*, inicia simultáneamente cada contenedor. Estos intercambian datos a través de una base de datos almacenada en la memoria caché, haciendo uso de la biblioteca *Redis*.

La figura 3.10 representa un diagrama de flujo donde cada servicio se muestra con su conjunto de documentos asociados. En este esquema, el bloque orquestador inicia la rutina y ejecuta los contenedores del bloque de producción. Este bloque orquestador envía variables iniciales a la base de datos, desde donde los contenedores del bloque de producción recuperan y reenvían los datos necesarios para cada uno. Por otro lado, el bloque "Cliente Android" corresponde a la aplicación presente en el teléfono inteligente que ingresa el número telefónico al que se realizará la llamada. Este dato es recibido por el servicio de *servidorBT* y almacenado en la base de datos, evitando la necesidad de reintroducirlo cada vez que el dispositivo se reinicia, aunque se puede modificar desde la misma aplicación si el usuario así lo desea.

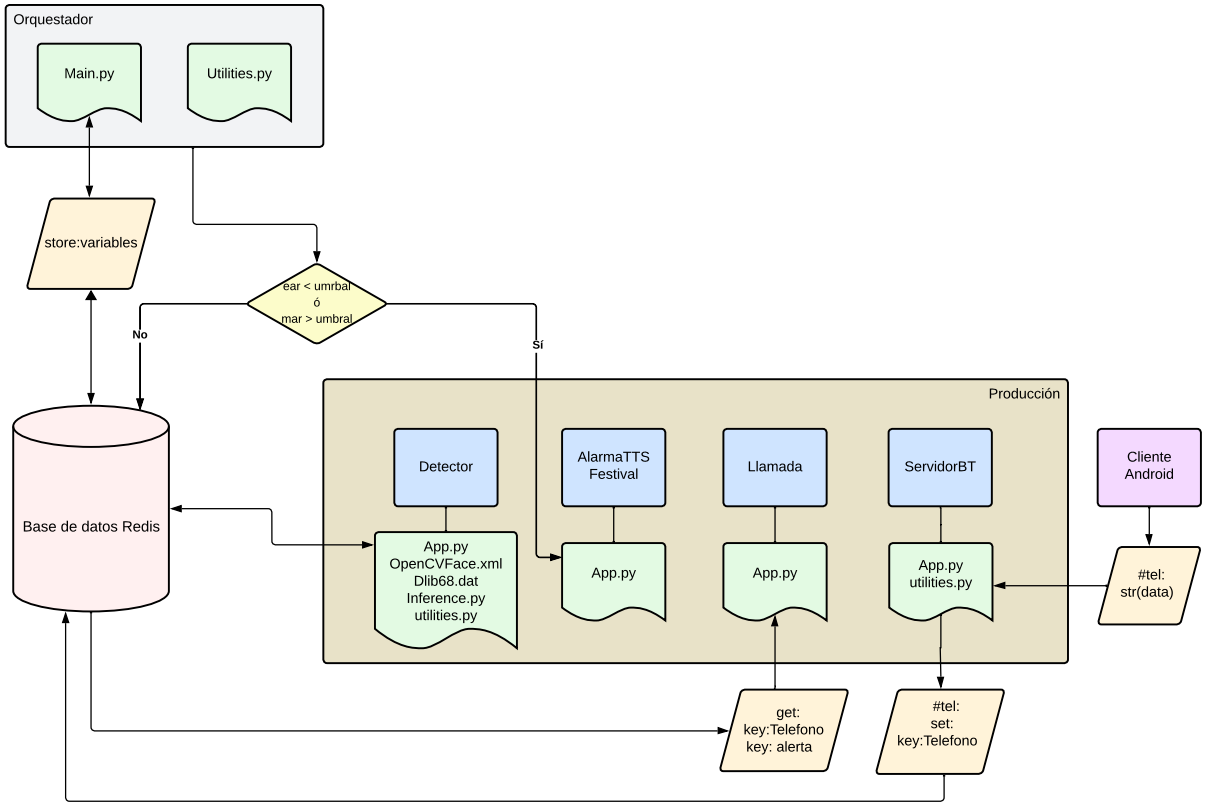


Figura 3.10: Diagrama de flujo de la arquitectura de servicios del software para detección y alertamiento de somnolencia.

El bloque de desarrollo comprende el proceso de prueba y construcción de la arquitectura de servicios. Por otro lado, el bloque orquestador posibilita ejecutar la aplicación junto con el bloque de producción, donde se ejecutan los servicios simultáneamente y realizan el intercambio de datos con *Redis*.

3.4.1. Bloque orquestador

A partir del diagrama de flujo de la figura 3.10, la rutina del algoritmo de detección y alertamiento de somnolencia comienza con el bloque orquestador. El programa principal (*main.py*) inicia todas las aplicaciones o servicios. Además, aquí se gestiona la toma de cuadros de imagen y se establecen valores de variables en condiciones iniciales que se depositan en la base de datos *redis* para que los otros servicios o archivos puedan consumir estos datos.

En la figura 3.11 se observa cómo se corren los diferentes servicios (*containers*) a partir de la ejecución del programa principal en el intérprete de comandos de la tarjeta. Si un servicio contiene un error, el estado (*status*) de cada servicio lo indica.

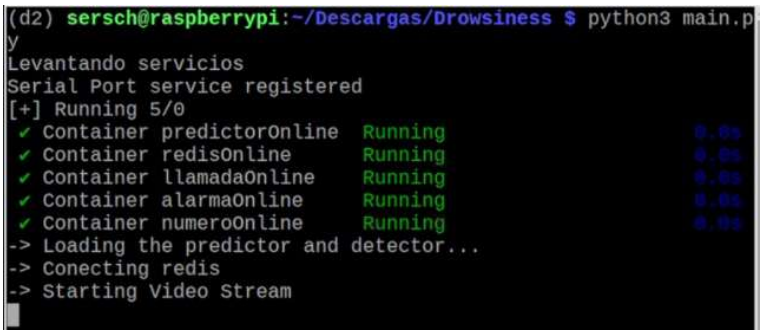


Figura 3.11: Iniciando la aplicación mediante el programa principal en el interprete de comandos de *RaspberryPi OS*.

3.4.1.1. Programa principal

En el programa *main.py*, se establece la lógica de programación para determinar alertas de somnolencia y activar los servicios de texto en forma de voz y llamadas, utilizando los valores almacenados en la base de datos *Redis*.

Una parte fundamental de esta implementación se presenta en la figura 3.12a, donde se muestra el código de programación para correr los comandos de los servicios de texto a voz y llamadas mediante el editor de código fuente *Visual Studio Code*. Se definió una variable booleana llamada "alerta", la cual, al cambiar de falso a verdadero ("True"), desencadena una impresión en la ventana que muestra los cuadros de imagen mediante la función *putText* de *OpenCV*, indicando "ALERTA de somnolencia!" como lo muestra la figura 3.12b. Esta

misma variable ($alerta == True$) se utiliza para activar la API de llamadas a través de la biblioteca *Twilio* (ver Sección 3.4.4).

Además, un contador de alertas ($count\_alert$) establece que si se detectó una alerta verdadera, éste realiza un conteo hasta 200 con el fin de que no se ejecuten más alertas durante este conteo, esto para poder diferenciar un cambio de estado de somnoliento a atento sin realizar una excesiva repetición de alertas. Cuando el conteo es mayor a 200, se vuelve a establecer su valor a cero y, cuando ocurre este restablecimiento, se ejecuta un subproceso llamado *query*, siempre que la condición de la variable alerta sea verdadera.

Este subproceso, en el intérprete del servicio de alerta en voz, inicia una secuencia de comandos que activa la alerta (ver Sección 3.4.3).

```
if alerta:
    lack_alert=True

    cv2.putText(frame, "ALERTA de somnolencia!", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    if count_alert==0:
        subprocess.Popen([querry], shell=True)
```

(a) Código de la lógica que inicia el servicio de alertas en forma de voz.

(b) Leyenda: "ALERTA de somnolencia!" en el cuadro de imagen cuando se detecta somnolencia.

Figura 3.12: Programa principal del orquestador y lógica para la ejecución de los comandos de los servicios de llamadas y alertas en forma de voz.

3.4.2. Servicio de detección de somnolencia

El servicio de detección de somnolencia se encarga de analizar imágenes provenientes de una cámara web. Estas imágenes, convertidas a escala de grises, se almacenan en la base de datos para su procesamiento por el programa principal. Este programa utiliza modelos de detección facial para calcular tasas de aspecto y distancias entre rasgos faciales, identificando señales de somnolencia.

Los resultados obtenidos actualizan la base de datos y pueden activar alertas auditivas o llamadas según la lógica de detección establecida. En general, el servicio monitoriza continuamente signos de somnolencia, como bostezos o cierres prolongados de ojos durante 30 cuadros consecutivos, activando alertas específicas al detectar estas señales.

3.4.2.1. Implementación del código del servicio para detección de somnolencia

Retomando la lógica del diagrama de flujo de la figura 1.2, la rutina del servicio de detección de somnolencia comienza capturando cuadros de imagen a través de la cámara web por parte del programa principal del bloque orquestador. Cada cuadro se convierte a escala de grises y se almacena en una variable que se agrega a la base de datos *Redis*. Hasta este punto, este proceso es realizado por el programa principal del bloque orquestador. Es importante mencionar que las variables que contienen las condiciones iniciales y el preprocesamiento del cuadro de imagen (escala de grises) son depositadas en la base de datos *Redis*, ya que el contenedor del servicio de detección requiere estos datos iniciales para cumplir con su tarea.

Posteriormente, el programa principal de detección, *aplicacion.py*, recoge los valores de condiciones iniciales de alertas, valores umbrales y cuadros de imagen en escala de grises de la base de datos utilizando el método *get HTTP*, como es ilustrado en la Figura 3.13. Los modelos de detección facial (*Haar cascade* y *68 facial landmarks*) se almacenan y utilizan en el programa de procesamiento, $inference.py$, donde se desarrollan funciones que computan y comparan las tasas de aspecto oculares, la distancia entre labios y umbrales predefinidos para detectar signos de somnolencia. Una vez ejecutadas las funciones desde *aplicacion.py*, los resultados se actualizan en la base de datos, lo que permite activar alertas

en forma de voz o llamadas según la lógica de detección establecida desde las funciones y el programa principal del orquestador.

```
detector = cv2.CascadeClassifier("/artefacts/haarcascade_frontalface_default.xml")
predictor = dlib.shape_predictor('/artefacts/shape_predictor_68_face_landmarks.dat')

app = FastAPI()

@app.get("/predictor")
async def get_response():

    data=rd.get_dict_from_redis("variables")
    gray=decode_image(data["gray"])

    COUNTER=data["COUNTER"]
    bostezo=data["bostezo"]
    alarm_status=data["alarm_status"]
    alarm_status2=data["alarm_status2"]
    alerta=data["alerta"]

    leftEyeHull, rightEyeHull, lip, COUNTER, alerta, bostezo, ear, distance,\
    alarm_status, alarm_status2 = deteccion_somnolencia(gray, COUNTER, bostezo,
    detector, predictor, alarm_status,alarm_status2, alerta, rd)

    data_umbral=rd.get_dict_from_redis("dataUmbral")
    bostezo_list=data_umbral["bostezo"]
    ojos_list=data_umbral["ojos"]
```

Figura 3.13: Extracto del programa principal del servicio de detección de somnolencia que importa los modelos de detección facial *detector* y *predictor*; método *get* para tomar valores de la base de datos; y ejecución de la función *deteccion_somnolencia* y sus argumentos.

Este servicio analiza constantemente la información de los cuadros de imagen para identificar un cierre de ojos prolongado durante 30 cuadros consecutivos y bostezos, ofreciendo una detección de signos de somnolencia.

A continuación se describen las funciones que permiten la discriminación de un cierre de ojos prolongado o un bostezo presentes en el programa $inference.py$ y ser utilizadas en el programa principal de este servicio.

3.4.2.2. Funciones para detección de somnolencia

En la figura 3.14 se presentan las cuatro funciones principales que conforman el programa $inference.py$ para el servicio de detección de somnolencia. Las primeras tres funciones calculan las tasas de aspecto de los ojos y la boca. La cuarta función, denominada

deteccion_somnolencia, lleva a cabo la comparación de estas tasas de aspecto con valores umbrales para determinar una lógica de alerta. Esta alerta se guarda en una variable en *Redis* con el programa principal *aplicacion.py* para el programa principal del orquestador que permite ejecutar tareas de otros servicios, como el servicio de llamadas y el de alertas en forma de voz si existe una alerta verdadera.

```
> def eye_aspect_ratio(eye): ···

> def final_ear(shape): ···

> def lip_distance(shape): ···

> def deteccion_somnolencia(gray, COUNTER, bostezo,
```

Figura 3.14: Funciones para determinar un estado de somnolencia.

3.4.2.3. Función *eye_aspect_ratio*; tasa de aspecto de un ojo

En la figura 3.15a, se exhibe la función *eye_aspect_ratio*, encargada de calcular la distancia euclidiana entre cada punto de referencia marcado con la letra p, como se detalla en la figura 3.15b. Las distancias A, B y C se utilizan en el cálculo de la tasa de aspecto de un ojo, empleando la fórmula *ear* (2.4.10.1). Esta fórmula se basa en los valores de diferentes posiciones extraídas de una lista que almacena las *landmarks* o puntos de referencia correspondientes a cada ojo.

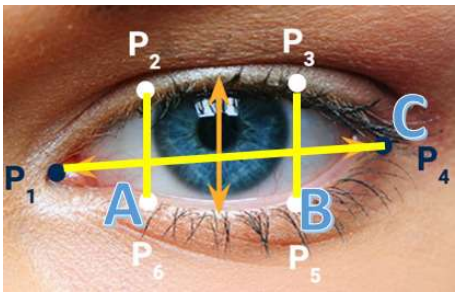
```
def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])

    C = dist.euclidean(eye[0], eye[3])

    ear = (A + B) / (2.0 * C)

    return ear
```

(a) Código de la fórmula para calcular la tasa de aspecto de un ojo.



(b) Distancias A, B, C para realizar el cálculo de la tasa de aspecto de un ojo.

Figura 3.15: Función que determina la tasa de aspecto de un ojo y distancias para calcular la tasa *ear*

3.4.2.4. Función *final_ ear*; promedio de la tasa de aspecto para los ojos

En la figura 3.16, se muestra la función $final\_ear$, la cual toma como argumento todos los puntos faciales representados en la variable *shape*. Utilizando la función $face\_utils$, se seleccionan únicamente los puntos de referencia de los dos ojos y se almacenan en variables de lista: $leftEye$ para los puntos del ojo izquierdo (37 al 42) y $rightEye$ para los del ojo derecho (43 al 48). Posteriormente, se calcula el promedio de las tasas de aspecto de ambos ojos mediante la función eye_aspect_ratio mostrada en la figura 3.15a.

```
def final_ear(shape):
    (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
    (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

    leftEye = shape[lStart:lEnd]
    rightEye = shape[rStart:rEnd]

    leftEAR = eye_aspect_ratio(leftEye)
    rightEAR = eye_aspect_ratio(rightEye)

    ear = (leftEAR + rightEAR) / 2.0
    return (ear, leftEye, rightEye)
```

Figura 3.16: Función que determina el promedio de la tasa de aspecto de los dos ojos.

3.4.2.5. Función *lip_ distance*; distancia entre los puntos medios de los labios

La función $lip_distance$ de la figura 3.17a calcula la distancia entre los labios. Esta función inicializa los puntos de referencia del labio superior e inferior, calcula los promedios de cada contorno y determina la distancia D (variable *distance*) entre los puntos medios de los labios, superior e inferior, como se observa en la figura 3.17b.

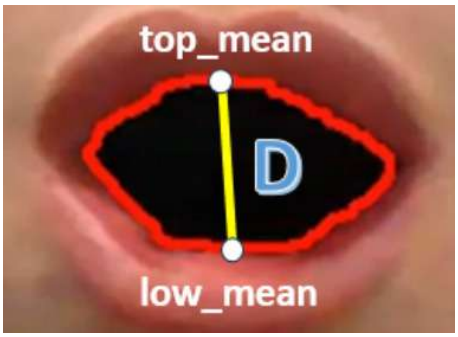
```python
def lip_distance(shape):
    top_lip = shape[50:53]
    top_lip = np.concatenate((top_lip, shape[61:64]))

    low_lip = shape[56:59]
    low_lip = np.concatenate((low_lip, shape[65:68]))

    top_mean = np.mean(top_lip, axis=0)
    low_mean = np.mean(low_lip, axis=0)

    distance = abs(top_mean[1] - low_mean[1])
    return distance
```

(a) Código de la fórmula para calcular la tasa de aspecto de un ojo.



(b) Distancia D que define el monitorio del estado de la boca.

Figura 3.17: Función que calcula la tasa de aspecto de la boca personalizada y la distancia D mostrada en la boca.

3.4.2.6. Función principal; determinar somnolencia

La función *deteccion_somnolencia* de la figura 3.18, primero establece los umbrales de las tasas de aspecto EYE_AR_THRESH para los ojos y $YAWN\_THRESH$ para la boca, tomando valores de la base de datos *Redis*. Durante 30 cuadros consecutivos, se promedian y multiplican estos umbrales por factores específicos para posteriormente realizar la comparación entre tasas y umbrales para determinar alertas de cierre de ojos o bostezos.

```python
def deteccion_somnolencia(gray, COUNTER, bostezo, detector

    data_umbral=rd.get_dict_from_redis("dataUmbral")
    bostezo_list=data_umbral["bostezo"]
    ojos_list=data_umbral["ojos"]

    EYE_AR_THRESH =  np.mean(ojos_list)*0.8
    EYE_AR_CONSEC_FRAMES = 30
    YAWN_THRESH =  np.mean(bostezo_list)*1.2
```

Figura 3.18: Valores de umbrales de tasas de aspecto y número de cuadros de imagen a evaluar.

En el contexto de la función *deteccion_somnolencia*, se retoma el cuadro de imagen en escala de grises para ser procesado por el modelo de detección de rostros *haarcascade frontal face* de la biblioteca *OpenCV*. Utilizando el descriptor de características de *Haar* y

un clasificador en cascada, este modelo examina la imagen en escala de grises para detectar rostros. En la figura 3.19, se ilustra cómo el cuadro de imagen en escala de grises es el primer argumento de la función *detector.detectMultiScale* en el objeto *rects*, permitiendo la identificación del número de rostros presentes en cada cuadro.

```
rects = detector.detectMultiScale(gray, scaleFactor=1.1,
    minNeighbors=5, minSize=(30, 30),
    flags=cv2.CASCADE_SCALE_IMAGE)
```

Figura 3.19: Variable que almacena el número de rostros.

Si la variable *rects*, que almacena el modelo con el cuadro de imagen, detecta uno o más rostros, se procede a aplicar el modelo *shape predictor 68 face landmarks* para la detección de elementos faciales. Este modelo, basado en un descriptor de características de tipo *HOG* y utilizando máquinas de soporte vectorial para la clasificación, identifica los puntos de referencia faciales. En la implementación, se define un bucle for que examina la variable *rects* y su contenido. Si esta variable almacena al menos un rostro (*rects* es mayor o igual a 1), se ejecutan los cálculos de tasas de aspecto y la lógica de comparación con los valores umbrales correspondientes a cada tasa. De lo contrario, si no se detecta más de un rostro en un cuadro de imagen, no se procede con los cálculos de tasas, con las funciones anteriores y la lógica de alerta, como se muestra en la figura 3.20. Las funciones *convexHull* de la biblioteca *OpenCV* se utilizan para dibujar el contorno de los ojos en la ventana del vídeo, mientras que la variable *lip* hace lo propio para los labios. Estos métodos permiten resaltar y visualizar de manera gráfica los bordes y formas de los elementos faciales.

```
for (x, y, w, h) in rects:
    rect = dlib.rectangle(int(x), int(y), int(x + w),int(y + h))

    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)

    eye = final_ear(shape)
    ear = eye[0]
    leftEye = eye [1]
    rightEye = eye[2]

    distance = lip_distance(shape)

    leftEyeHull = cv2.convexHull(leftEye)
    rightEyeHull = cv2.convexHull(rightEye)

    lip = shape[48:60]
```

Figura 3.20: Bucle para detección de somnolencia

En la figura 3.21a se presentan las condiciones de comparación entre la tasa de aspecto de los ojos y el valor umbral EYE_AR_THRESH. Si la tasa *ear* es menor que este valor umbral, las variables de alerta cambian a verdadero durante un intervalo de evaluación de 30 cuadros de imagen.

En cuanto a otro aspecto, si la variable *distance* supera el umbral $YAWN\_THRESH$, se activan los estados de alerta, como se muestra en la figura 3.21b.

```
if ear < EYE_AR_THRESH:
    COUNTER += 1
    print(COUNTER)

    if COUNTER >= EYE_AR_CONSEC_FRAMES:
        if alarm_status == False:
            alarm_status = True
            alerta=True
            print("Alarma ojos")
```

(a) Lógica de comparación para determinar una alarma por cierre de ojos.

```
if (distance > YAWN_THRESH):
    print("alarma bostezo")
    if alarm_status2 == False:
        alarm_status2 = True
        alerta=True
        bostezo+=1
else:
    alarm_status2 = False
```

(b) Lógica de comparación para determinar una alarma por bostezo.

Figura 3.21: Lógica de programación de la comparación de tasas de aspecto de los ojos y distancia de los labios para determinar alertas por somnolencia.

Cuando la variable *alerta* es verdadera (*True*) es retomada de la base de datos *Redis* por el programa del orquestador que inicia el subproceso mencionado en la figura 3.12a (ver

sección 3.4.1.1) para iniciar el servicio de alertas en forma de voz. El servicio de llamadas también toma de la base de datos esta misma variable cuando es verdadera para iniciar el programa de aplicación del mismo servicio como se observa en la figura .

3.4.3. Servicio de alarma de texto a voz

El servicio de alertas en forma de voz se encarga de reproducir una frase en forma de audio cuando se detecta un bostezo o un cierre de ojos prolongado por parte del servicio de detección.

El programa principal de este servicio se inicia desde la lógica del programa principal del bloque orquestador. Su función consiste en buscar el valor verdadero del objeto *alerta* en la base de datos. Posteriormente, se inicia un subproceso denominado *querry*. Este subproceso utiliza el método *POST http* para lanzar una secuencia de comandos en el *shell* del contenedor que alberga el servicio de alertas en forma de voz.

3.4.3.1. *API* para alertas en forma de texto a voz servicio local

Se optó por la *API* de la biblioteca *Festival* para convertir texto personalizable a audio, útil como advertencia de somnolencia al conductor. Aunque *espeak* y *GoogleTTS* también convierten texto a audio, *GoogleTTS* destaca por su calidad vocal natural, pero requiere conexión a Internet. Por otro lado, *espeak* funciona $offline$, pero su calidad de voz es menos fluida y carece de voces en español. En contraste, $FestivalTTS$, al ser $offline$, ofrecer voces multilingües y permitir personalización de voz, resulta adecuado para advertencias de somnolencia en sistemas locales sin conexión a Internet.

3.4.3.2. Implementación del código del servicio de alerta con la biblioteca Festival

En el programa principal del servicio de alertamiento de texto a voz, denominado *aplicacion.py*, se define una ruta de tipo *POST http* en *execute_command*, esperando recibir un parámetro *command* en forma de cadena, que corresponde al texto a convertir en audio que se escuchará como alerta. Al recibir esta solicitud *POST*, se construye una cade-

na *baseString* que utiliza el comando *echo* junto con la aplicación *festival* para convertir el texto proporcionado en voz, en idioma español. Posteriormente, ejecuta este comando en el sistema operativo (*shell* del *kernel* del contenedor) a través de *subprocess.check_output*(), almacenando el resultado e implementando la lógica para mostrar posibles errores mediante excepciones *HTTPException*, en caso de fallos en la ejecución del comando, comunicando el estado de la operación con un código 400. En la figura 3.22 se muestra el código del programa que ejecuta el contenedor del servicio de alertas en forma de voz.

```python
from fastapi import FastAPI, HTTPException
import subprocess

app = FastAPI()

@app.post("/execute_command/")
def execute_command(command: str):
    baseString=f"echo '{command}' | festival --language spanish --tts"

    try:
        # Execute the command in the shell using subprocess
        result = subprocess.check_output(baseString, shell=True, text=True)
        return {"output": result}
    except subprocess.CalledProcessError as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Figura 3.22: Código del programa para emitir alarmas en forma de voz.

Este código crea un servicio que convierte texto en voz en español para alertas, ofreciendo una *API* mediante solicitudes *POST* desde la lógica de la variable *alerta* del programa principal del bloque orquestador (Figura 3.12a). Esto activa la ejecución de comandos en la aplicación *Festival* para reproducir el texto como voz. El texto a convertir está codificado en el parámetro *command* del objeto *querry* en el programa *main.py*, es "ALERTA DE SOMNOLENCIA, TE ESTAS QUEDANDO DORMIDO" (Figura 3.23).

```
querry="curl -X 'POST' 'http://localhost:8080/execute_command/?command=%22ALERTA%20DE%20SOMNOLENCIA
```

Figura 3.23: Objeto *querry*; usa el parámetro *command* que contiene la frase en texto a convertir en audio.

3.4.4. Servicio de llamadas telefónicas

Esta sección aborda el servicio de alertas a través de llamadas. Incluye una descripción de las características de *Twilio* para comunicaciones automáticas. Luego, se muestra el código y la adaptación de la interfaz de programación de aplicaciones de llamadas, también en el editor *Visual Studio Code*.

El servicio se ejecuta usando la variable *alerta* como detonador, al igual que el servicio de alertas en forma de voz. Esta variable es procesada desde el servicio de detección, donde si su valor booleano cambia a verdadero, se ejecuta el programa principal del servicio de llamadas.

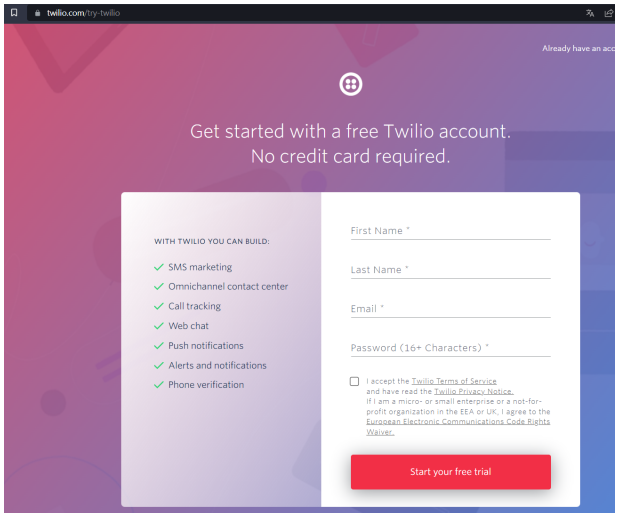
3.4.4.1. *API* para llamadas vía servicio internet

Twilio es una plataforma de comunicaciones en la nube que permite construir e implementar aplicaciones de comunicación automatizada. *Twilio* ofrece un conjunto de *API's* para realizar llamadas telefónicas, enviar y recibir mensajes de texto, *whatapp* o iniciar y administrar video llamadas. Estas *API's* están disponibles para diferentes lenguajes de programación como *JavaScript*, *Python*, *C#*, entre otros [43]. Algunos de los beneficios de esta plataforma son:

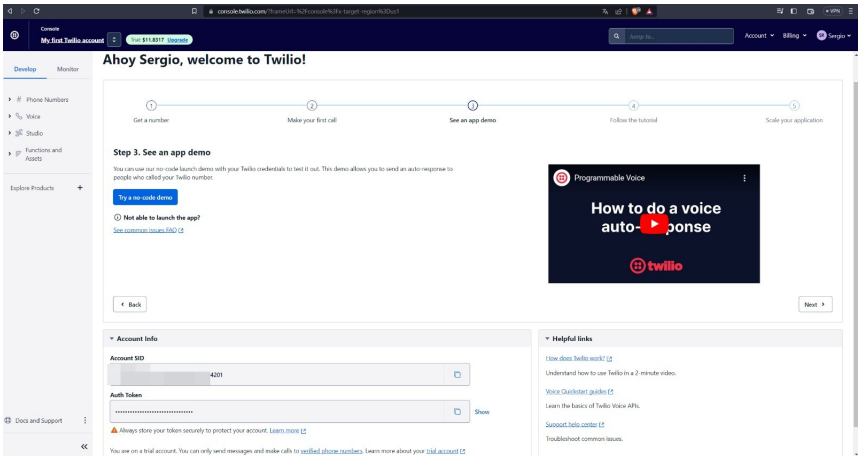
- Facilidad de uso: la integración de las *API's* a distintos lenguajes de programación es simple e intuitiva.
- Seguridad en la comunicación: cuenta con los estándares más altos en la industria simplemente por proveer de un servicio único para cada usuario (*SID, token*, número telefónico, verificación de identidad).
- Gratis por tiempo limitado: efectivo si se piensa experimentar o realizar pruebas sin adquirir por un plan que brinde prestaciones más personalizadas para industrias o empresas.

Se empleó *Twilio* para desarrollar una *API* de llamadas automatizadas en *Python* sin adquirir un plan de pago para facilitar la experimentación. Se inició el proceso accediendo

al sitio oficial de *Twilio* en *https* : *www.Twilio.comenus* y registrando una cuenta para obtener un número telefónico, token de autenticación y ID de cuenta. Estos datos son cruciales para realizar solicitudes seguras desde la *API* hacia el servicio en la nube. Las Figuras 3.24a y 3.24b muestran el proceso de registro y los datos esenciales obtenidos como SID, token y crédito de prueba.



(a) Sito web de registro de la plataforma de comunicaciones *Twilio*.



(b) Página principal de usuario *Twilio*.

Figura 3.24: Registro y datos de usuario de la plataforma de comunicaciones de *Twilio*

.

Una vez realizado el registro, en el apartado de *phone numbers* se puede apreciar el número telefónico que proporciona el servicio como ejemplo. En la figura 3.25 se muestra el número. Estos datos se muestran con el objetivo de agregarlos posteriormente en la

programación e implementación de la *API*.

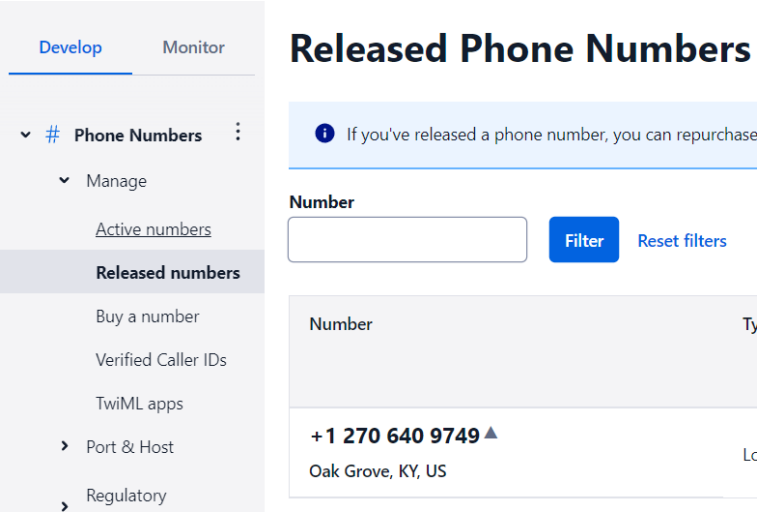


Figura 3.25: Número telefónico que ofrece el servicio *Twilio* aleatoriamente.

3.4.4.2. Implementación del código del servicio para llamadas en *Python*

El servicio de llamadas telefónicas se basa en la implementación de la interfaz de aplicación programable de *Twilio*. Retomando el diagrama de flujo de la Figura 3.8, una vez que el servicio de detección realiza una alerta verdadera por un cierre de ojos prolongado o un bostezo por somnolencia, se ejecuta el programa principal del servicio de llamadas, denominado *aplicacion.py*.

En el programa principal, se emplea la dependencia *Client* de *Twilio.rest* para acceder a los datos del cliente y usuario, así como las dependencias *VoiceResponse* y *Say* de *Twilio.twiml.voice_response* para funciones específicas de llamadas.

La Figura 3.26 muestra la importación de las dependencias mencionadas y el código del programa. El programa contiene un bucle *while* que verifica el valor booleano de la variable *alerta* en la base de datos *Redis*. Si es verdadero, busca el valor de *numerobt* en la base de datos y realiza una llamada utilizando la función almacenada en el objeto *call*. El objeto *numerobt* corresponde al número telefónico configurado desde la aplicación *android* cliente *bluetooth* del *smartphone*.

Para utilizar este servicio, se requiere el identificador único de seguridad (*SID*) asignado a la variable *account_sid*, el token de autenticación asignado a *auth_token*, y la creación

de un objeto *client* que contenga esta información. La variable *call* utiliza el método *client.calls.create*, requiriendo atributos como la URL del audio en formato $MP3$ para reproducir al contestar la llamada, el número de celular al que se desea llamar (*numerobt*), y el número proporcionado por *Twilio*, que puede modificarse desde su plataforma web.

```
from twilio.rest import Client
from twilio.twiml.voice_response import VoiceResponse, Say
#
import redis
rd=redis.Redis("redis",6379,decode_responses=True)

while True:
    try:
        if int(rd.get("alerta"))==True:

            numerobt=rd.get("telefono")#
            numerobt=numerobt.replace('b','')
            print(numerobt)
            # Find your Account SID and Auth Token at twilio.com/console
            # and set the environment variables. See http://twil.io/secure
            account_sid = 'AC0534338ceafc08aeb0f6b1e36b9d4201'
            auth_token = '37b686b1cbd3013df4aa6bd06791c7f6'
            client = Client(account_sid, auth_token)

            call = client.calls.create(
                                    url='https://daffodil-dragonfly-4559.
                                    to=numerobt,
                                    from_='+17175368513'
                                )
            print(f'llamada realizada correctamente {call.sid}')
            rd.set("alerta",int(False))
    except:
        pass
```

Figura 3.26: Programa principal del servicio de llamadas e implementación de la API para llamadas del servicio *Twilio* en *Python*.

Finalmente, se ejecuta el programa al momento de recibir una alerta de somnolencia verdadera y en la figura se observa la llamada entrante al número de celular que se especificó en la variable *to*. Al haber realizado una llamada exitosa, se actualiza la variable *alerta* a su estado inicial ($False$) para seguir repitiendo el proceso de detección. Cabe mencionar que, como se experimentó con un plan de prueba, el número entrante cambia, ya que tiene un vencimiento.

3.4.5. Servicio *Bluetooth Server*

Este servicio actúa como un servidor de datos a través de una comunicación serial emulada mediante *Bluetooth*. Su función primordial consiste en recibir, a través de *Bluetooth*, el número telefónico al que se dirigirá una llamada como alerta, almacenándolo en la base de datos *Redis* en caché para su uso por el servicio de llamadas. Este número se ingresa manualmente desde la aplicación cliente *Bluetooth* en un teléfono *Android*.

Esta sección detalla la implementación del servicio *Bluetooth* como servidor, incluyendo el código correspondiente en *Python* utilizando el entorno de desarrollo Visual *StudioCode*. Se comienza describiendo las configuraciones previas necesarias en el sistema operativo para asegurar la correcta ejecución de la aplicación o servicio.

Posteriormente, se expone el uso de la biblioteca *PyBluez* en *Python* para establecer una comunicación inalámbrica serial emulada tipo $RS-232$, específicamente para la recepción de datos. Se emplean *sockets* de tipo servidor en el computador *Raspberry 4 model B* para habilitar este proceso.

3.4.5.1. Configuraciones en sistema operativo (*OS*) para comunicación serial vía *Bluetooth*

En seguida se exhibe el proceso de configuraciones preliminares en el sistema operativo *Raspbery Pi OS* con el objetivo de establecer una comunicación serial emulada vía *Bluetooth* entre el computador de una sola tarjeta *Raspberry Pi 4 Model B* y un teléfono inteligente, mediante una aplicación cliente-servidor.

El módulo de extensión *PyBluez* permite usar los recursos del sistema *Bluetooth* mediante comandos y funciones en lenguaje *Python*.

Primero se instalan dos dependencias necesarias que son: *the Python Development Library* y *the Bluetooth Development Library*. Se instalan junto con su versionado de forma automatizada mediante el contenerizado en el archivo $Dockerfile$.

Consecuentemente, se activa el perfil de puerto serial (SSP o SP), permitiendo al servicio *Bluetooth* establecer comunicación serial inalámbrica con dispositivos que utilizan la misma tecnología. Esto requiere la activación del modo de compatibilidad del servicio

Bluetooth daemon, encargado de gestionar funciones, soporte y el inicio de dispositivos *Bluetooth* entre aplicaciones de terceros en la *Raspberry Pi*.

Para realizar esta configuración, se ejecuta el comando en la terminal de la tarjeta: *sudo nano /etc/systemd/system/dbus-org.bluez.service*, que permite editar el archivo de configuración *dbus-org.bluez.service* utilizando el editor de texto *nano*. La figura 3.27 muestra la ventana de edición *nano* con este archivo.

Este archivo de configuración almacena los parámetros de arranque del control *Bluetooth* y se encuentra en los directorios de la dependencia *PyBluez*.

Se añade un guion y la letra -C al final de la línea $ExecStart = /usr/lib/bluetooth/bluetoothd$ para habilitar el modo de compatibilidad, tal como se muestra en la figura 3.27.

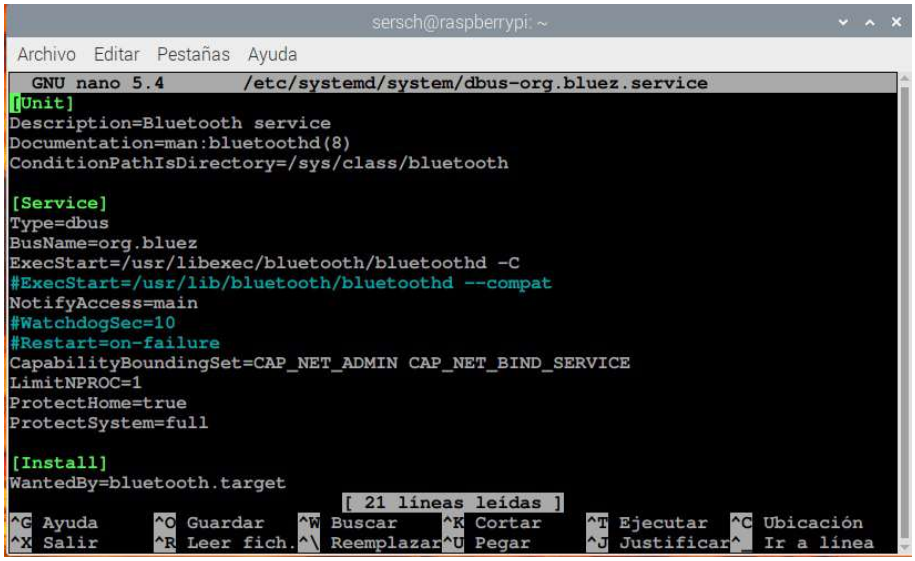


Figura 3.27: Ventana de configuración de arranque usando editor *nano* y activación del modo compatibilidad en el apartado de servicio (*Service*).

Una vez realizadas las modificaciones, se guardan los cambios en la configuración desde la terminal utilizando la combinación de teclas $Ctrl + O$, y se reinicia el servicio *Bluetooth Daemon* mediante los comandos: *sudo systemctl daemon-reload* y *sudo systemctl restart dbus-org.bluez.service*, como se observa en la figura 3.28. Es recomendable reiniciar el ordenador para asegurar la aplicación de los cambios con el comando: *reboot*.

Figura 3.28: Reinicio del servicio *Bluetooth daemon* y *Pybluez*.

Después se agrega el perfil *Bluetooth* para comunicación serial inalámbrica, conocido como *SP*, al sistema operativo de la tarjeta. Esto se realiza ejecutando el comando: *sudo sdptool add SP* en la terminal, tal como se muestra en la figura 3.29.

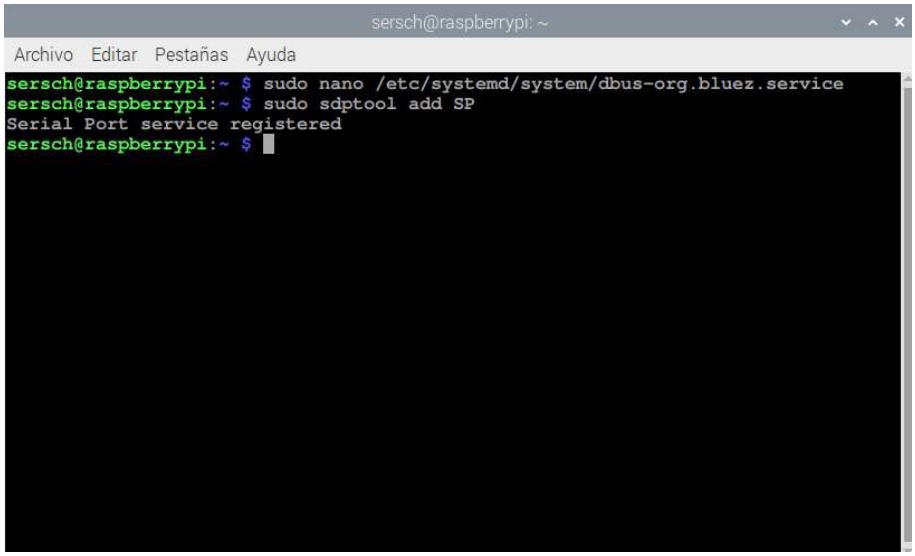


Figura 3.29: Perfil *Bluetooth* puerto serial.

Ya agregado el perfil *SP*, es posible crear un programa que permita el intercambio y transferencia de datos secuenciales usando *Bluetooth* emulando una comunicación serial (*RS-232*). El problema de esta configuración recae en que cada vez que el computador se enciende, se debe agregar el perfil *SP* con el comando: *sudo sdptool add SP* manualmente. Este detalle se resuelve directamente automatizando el proceso en el programa orquestador, como se observa en la figura 3.30.

```
os.popen("sudo -S %s"%( "sudo sdptool add SP" ), 'w').write('1234')
```

Figura 3.30: Función *popen* de la biblioteca *os*; permite ejecutar comandos *linux* en el *shell* de *raspberry OS* desde *Python*.

La línea de código de la figura 3.30 ejecuta el comando *sudo sdptool add SP* desde

Python utilizando la función *os.popen*. Este comando de terminal generalmente se utiliza para agregar un perfil de servicio (*Service Profile*, *SP*, también *SSP*) al sistema *Bluetooth*. El comando *sudo* permite ejecutar el comando con privilegios de superusuario.

El número 1234 escribe una contraseña en el flujo de entrada estándar (*stdin*). Esto puede estar destinado a proporcionar la contraseña si es requerida por *sudo* para obtener los privilegios necesarios para ejecutar el comando. Por lo tanto, cada vez que la tarjeta se reinicia o se enciende, el perfil se agrega automáticamente agregando esta línea a la programación.

3.4.5.2. Implementación del código del servicio *Bluetooth* en *Python*

A continuación se describe el apartado de código en *Python* que favorece establecer una conexión serial emulada vía *Bluetooth* con dispositivos que usen el protocolo de transporte *RFCOMM*, usando la dependencia *PyBluez* para *Python*.

PyBluez es una dependencia para código *Python* que propicia acceder a los recursos *Bluetooth* de la máquina usuario. Soporta el escaneo de dispositivos y apertura de *sockets* nativos de *Windows* y *Linux*.

Se usó esta dependencia porque la teoría y ejemplos de código de programación de esta biblioteca *Bluetooth* permiten hacer actuar al dispositivo *Raspberry* como servidor, cliente o ambos, mediante la administración de *sockets* usando el protocolo de transporte *RFCOMM*, todo en lenguaje *Python*.

Importación de módulos y configuración inicial

Primero se importan las bibliotecas *Bluetooth*, *sys* y la clase $support\_redis$. La biblioteca *sys* se usa en este programa para el almacenamiento de datos en el $buffer$ y la clase $support\_redis$ proviene de un archivo denominado: utilidades, y se emplea para leer y escribir valores de diferentes variables a la base de datos $redis$, en caché. En la figura 3.31 se observan las dependencias importadas.

```
from utilities import support_redis
import bluetooth
import sys
```

Figura 3.31: Dependencias empleadas para el desarrollo del código en *Visual Studio Code*.

Configuración de la conexión *Bluetooth*

Después, se crea un objeto que contiene una dirección IP y el puerto para comunicación entre servicios y almacenar el valor de la variable que corresponde al número telefónico. Se define la función $recieve\_data()$ que contiene la lógica para configurar el *socket* de tipo servidor, como se observa en la figura 3.32.

```
rd=support_redis("172.16.0.4",6379)

def receive_data():#recibir numero···
```

Figura 3.32: Objeto *rd* para la conexión con *redis* y la función *recieve_data* para la lógica del servicio.

Creación del *socket Bluetooth*

En la función *recieve_data*(), se asigna el protocolo de transporte *RFCOMM* para el *socket* como argumento de la función *BluetoothSocket* de la clase *bluetooth* en el objeto *server_sock*. Se elige el puerto 1 y se enlaza con la función *bind*, donde el primer argumento corresponde a la accesibilidad con cualquier IP y el segundo es el puerto asignado al objeto *port*, como lo expone la figura 3.33.

```
def receive_data():#recibir numero
    server_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
    port = 1
    server_sock.bind(("", port))
```

Figura 3.33: Creación del *socket* a partir de la asignación del protocolo de transporte *RFCOMM*.

Espera y aceptación de de conexiones

El *socket* se pone en modo de escucha (*listen*) para aceptar una única conexión entrante usando el método *listen*, indicando el número de conexiones como parámetro. El programa imprime un mensaje indicando que está esperando una conexión *Bluetooth* en la terminal del servicio, como lo muestra la figura 3.34.

```
server_sock.listen(1)
print("Waiting for a connection...")
client_sock, client_addr = server_sock.accept()
print(f"Accepted connection from {client_addr}")
```

Figura 3.34: Aceptación de una única conexión usando el método *listen* de la biblioteca *bluetooth*.

Manejo de la conexión entrante

Cuando se establece la conexión, el programa imprime la dirección del cliente conectado y entra en un bucle *while* para recibir datos del cliente a través del *socket*. Los datos recibidos se almacenan en una variable llamada *data*, que corresponde al número telefónico que el cliente (aplicación *Android*) envía.

Si se recibe el dato, se utiliza la instancia *rd.redis_client* para establecer la clave *telefono* en la base de datos *Redis* con el valor del dato recibido, como se observa en la figura 3.35.

```
try:
    while True:
        data = client_sock.recv(1024)
        if not data:
            break
        print(f"Received data: {data}")
        rd.redis_client.set("telefono", str(data))# numero redis
except OSError as e:
    print("Error:", e)
finally:
    print("Closing the connection...")
    client_sock.close()
    server_sock.close()
```

Figura 3.35: Administración de funciones del *socket bluetooth*.

Finalmente, una condición define que si se inicia el programa principal de la aplicación, se procede a la ejecución de la función *recieve_data*() que corresponde al servicio *Bluetooth server* como lo muestra la figura 3.36.

```
if __name__ == "__main__":
    receive_data()
```

Figura 3.36: Ejecución de la función que ejecuta el *socket bluetooth*.

3.4.6. Aplicación móvil cliente *Bluetooth* en *Android Studio*

Esta sección detalla la implementación de una aplicación móvil que permite al usuario enviar el número telefónico para recibir alertas de somnolencia del detector. El diseño se desarrolló en *Android Studio*, una plataforma que compila aplicaciones en lenguajes *Java* o *Kotlin*, utilizando *Kotlin* para la lógica y *HTML* con *widgets* para la interfaz gráfica.

La configuración de esta aplicación cliente *Bluetooth* permite al usuario interactuar con el Sistema de Detección y Alerta de Somnolencia (SDAS), ingresando a través del teclado digital el número de su teléfono celular, que consta de 10 dígitos más el código de país. Este dato se envía al servidor *Bluetooth* en el dispositivo *Raspberry Pi 4 Model B*, posibilitando la generación de alertas en forma de llamadas cuando el sistema detecta somnolencia.

Esta adaptación facilita la transmisión de datos en forma de cadenas de caracteres (*String*) a través del *Bluetooth* de un teléfono inteligente. El *Bluetooth* en la tarjeta *Raspberry Pi 4 Model B* actúa como servidor, mientras que el teléfono inteligente *Android* opera como cliente emisor de datos mediante la interfaz gráfica de la aplicación. En este contexto, la información transmitida es una cadena de 13 caracteres.

3.4.6.1. Implementación de la aplicación cliente *Bluetooth*

Se utilizó una aplicación previa diseñada por Loachamin Luis, instructor del curso "Android Studio y Arduino – Comunicación Bluetooth – Kotlin – BASICO" [44]. Su app, creada originalmente para controlar luces con *Arduino*, se adapta fácilmente al SDAS gracias a su estructura basada en un *socket* cliente y el protocolo *RFCOMM*.

La plantilla, de código abierto, fue modificada en *Android Studio* usando *HTML* para la interfaz y *Kotlin* para la lógica de comunicación. La Figura 3.37 muestra el diseño original.

Figura 3.37: Plantilla de la app creada por el autor Loachamin.

3.4.6.2. Diseño de interfaz gráfica

La interfaz se diseñó desde *activity_ main.xml* usando componentes de la pestaña *palette*. Los elementos principales son:

- ***Button***: Encendido/apagado de *Bluetooth*, búsqueda, conexión y envío.
- ***Spin***: Lista de dispositivos emparejados.
- ***TextView***: Indicaciones al usuario.
- ***PlaneText***: Entrada del número telefónico.
- ***ImageView***: Inserción de imágenes.

La Figura 3.38 presenta los componentes, y en la Figura 3.39, el árbol de componentes y sus atributos.

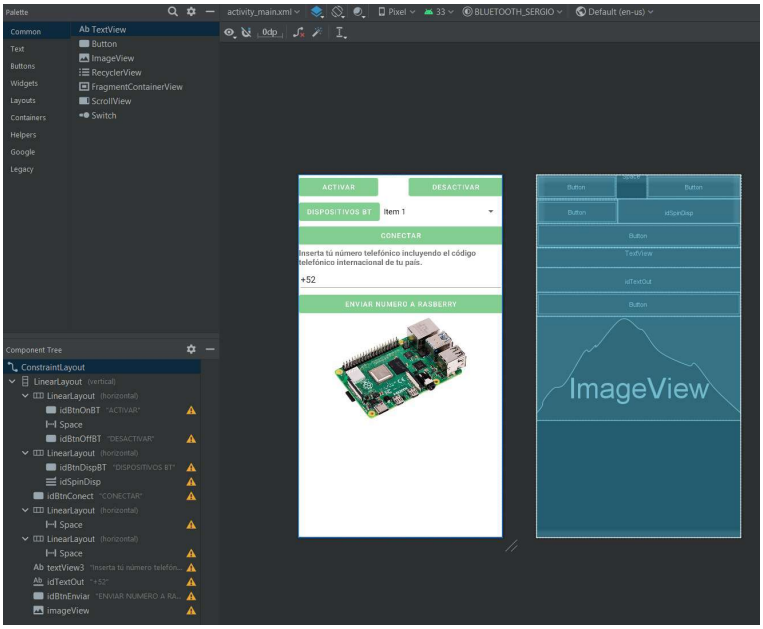
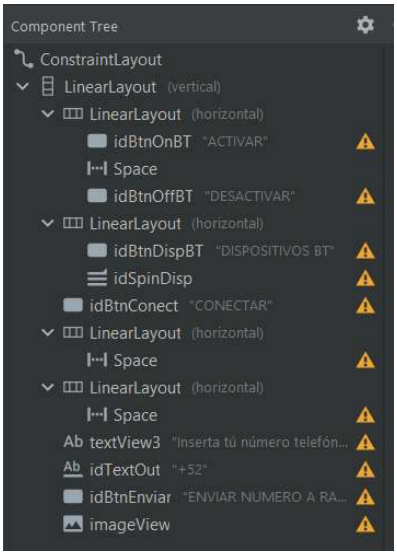
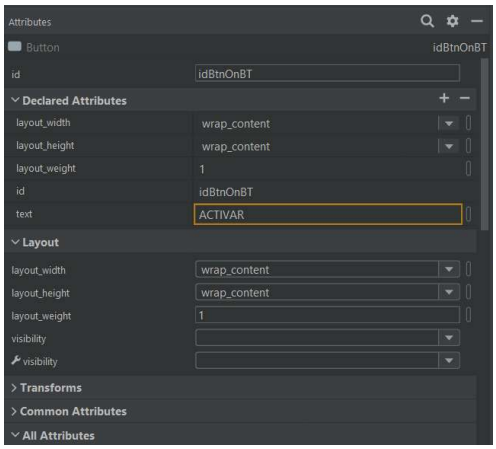


Figura 3.38: Componentes para la interfaz gráfica de la aplicación.



(a) Árbol de componentes.



(b) Atributos del botón *idBtnOnBT*.

Figura 3.39: Árbol de componentes y atributos personalizados.

3.4.6.3. Lógica de aplicación y configuración del *socket* cliente

En la lógica programada en *MainActivity.kt*, se añadieron permisos desde *manifest.xml* para el uso de *Bluetooth* (ver Figura 3.40).

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_logo"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.BLUETOOTH_SERGIO"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 3.40: Permisos requeridos en el archivo *manifest.xml*.

Se reemplazó el UUID base por el estándar RFCOMM: “00001101-0000-1000-8000-00805F9B34FB” (Figura 3.41).

```
companion object {
    var m_myUUID: UUID = UUID.fromString( name: "00001101-0000-1000-8000-00805F9B34FB")
    private var m_bluetoothSocket: BluetoothSocket? = null
```

Figura 3.41: *UUID* estándar para comunicación *Bluetooth* RFCOMM.

La Figura 3.42 muestra la inicialización del adaptador *Bluetooth* con *BluetoothManager*.

```
mBtAdapter = (getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager).adapter

if (mBtAdapter == null) {
    Toast.makeText(
        context: this,
        text: "Bluetooth no está disponible en este dipositivo",
        Toast.LENGTH_LONG
    ).show()
} else {
    Toast.makeText( context: this, text: "Bluetooth está disponible en este dispositivo", Toast.LENGTH_LONG)
        .show()
}
```

Figura 3.42: Inicialización del adaptador *Bluetooth*.

Cada componente de la interfaz tiene su función programada con *setOnClickListener*:

Encender *Bluetooth*: Solicita permisos y activa el adaptador (Figura 3.43).

Apagar *Bluetooth*: Desactiva el adaptador (Figura 3.44).

Dispositivos Emparejados: Lista los dispositivos disponibles (Figura 3.45).

Conectar: Establece conexión usando la MAC seleccionada (Figura 3.46).

Enviar Número: Captura e interpreta el número insertado (Figura 3.47).

```
idBtnOnBT.setOnClickListener { it: View!
    if (mBtAdapter.isEnabled) {

        Toast.makeText( context: this,  text: "Bluetooth ya se encuentra activado", Toast.LENGTH_LONG).show()
    } else {

        val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
        if (ActivityCompat.checkSelfPermission(
                context: this,
                Manifest.permission.BLUETOOTH_CONNECT
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            Log.i( tag: "MainActivity",  msg: "ActivityCompat#requestPermissions")
        }
        someActivityResultLauncher.launch(enableBtIntent)
    }
}
```

Figura 3.43: Encendido del adaptador *Bluetooth*.

```
idBtnOffBT.setOnClickListener { it: View!
    if (!mBtAdapter.isEnabled) {

        Toast.makeText( context: this,  text: "Bluetooth ya se encuentra desactivado", Toast.LENGTH_LONG)
            .show()
    } else {
        //Encender Bluetooth
        mBtAdapter.disable()
        Toast.makeText( context: this,  text: "Se ha desactivado el bluetooth", Toast.LENGTH_LONG).show()
    }
}
```

Figura 3.44: Apagado del adaptador *Bluetooth*.

```
idBtnDispBT.setOnClickListener { it: View!
    if (mBtAdapter.isEnabled) {

        val pairedDevices: Set<BluetoothDevice>? = mBtAdapter?.bondedDevices
        mAddressDevices!!.clear()
        mNameDevices!!.clear()

        pairedDevices?.forEach { device ->
            val deviceName = device.name
            val deviceHardwareAddress = device.address // MAC address
            mAddressDevices!!.add(deviceHardwareAddress)

            mNameDevices!!.add(deviceName)
        }


        idSpinDisp.setAdapter(mNameDevices)
    } else {
        val noDevices = "Ningun dispositivo pudo ser emparejado"
        mAddressDevices!!.add(noDevices)
        mNameDevices!!.add(noDevices)
        Toast.makeText( context: this,  text: "Primero vincule un dispositivo bluetooth", Toast.LENGTH_LONG)
            .show()
    }
}
```

Figura 3.45: Dispositivos *Bluetooth* emparejados.

```
idBtnConect.setOnClickListener { it: View!
    try {
        if (m_bluetoothSocket == null || !m_isConnected) {

            val IntValSpin = idSpinDisp.selectedItemPosition
            m_address = mAddressDevices!!.getItem(IntValSpin).toString()
            Toast.makeText( context: this, m_address, Toast.LENGTH_LONG).show()
            // Cancel discovery because it otherwise slows down the connection.
            mBtAdapter.cancelDiscovery()
            val device: BluetoothDevice = mBtAdapter.getRemoteDevice(m_address)
            //m_bluetoothSocket = device.createInsecureRfcommSocketToServiceRecord(m_myUUID)
            m_bluetoothSocket = device.createRfcommSocketToServiceRecord(m_myUUID)
            //m_bluetoothSocket = device.listenUsingRfcommWithServiceRecord(m_myUUID)
            m_bluetoothSocket!!.connect()

        }

        Toast.makeText( context: this,  text: "CONEXION EXITOSA", Toast.LENGTH_LONG).show()
        Log.i( tag: "MainActivity",  msg: "CONEXION EXITOSA")

    } catch (e: IOException) {
        //connectSuccess = false
        e.printStackTrace()
        Toast.makeText( context: this,  text: "ERROR DE CONEXION", Toast.LENGTH_LONG).show()
        Log.i( tag: "MainActivity",  msg: "ERROR DE CONEXION")
    }
}
```

Figura 3.46: Conexión con dispositivo seleccionado.

```
idTextOut.inputType = InputType.TYPE_CLASS_NUMBER
idTextOut.setOnClickListener { it: View!
    idTextOut.requestFocus()
    idTextOut.setRawInputType(InputType.TYPE_CLASS_PHONE)
    val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
    imm.showSoftInput(idTextOut, InputMethodManager.SHOW_IMPLICIT)
    //Botón enviar número
    idBtnEnviar.setOnClickListener { it: View!
        //if(idTextOut.text.toString().isEmpty()){
        if (idTextOut.text.toString().isEmpty()) {
            Toast.makeText( context: this,  text: "El nombre no puede estar vacío", Toast.LENGTH_SHORT)
        } else {
            val inputText = idTextOut.text.toString()//nueva
            //var mensaje_out: String = idTextOut.text.toString()
            if (inputText.length == 10) {
                val mensaje_out: String = "+52$inputText"

                // Resto del código
                sendCommand(mensaje_out)
            } else {
                Toast.makeText( context: this,  text: "Ingrese un número valido", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

Figura 3.47: Entrada y envío del número telefónico.

El sistema permite gestionar el adaptador *Bluetooth*, conectarse a un dispositivo *Raspberry Pi* y enviar un número validado como mensaje. Las Figuras 3.48a y 3.48b muestran una conexión exitosa y la validación del número ingresado.

(a) Conexión exitosa entre tarjeta *SBC* y *smartphone*.

(b) Inserción del número telefónico.

Figura 3.48: Proceso de conexión y envío de datos por *Bluetooth*.

El proceso de ejecución comienza cuando se enciende y se inicia la aplicación de detección y alerta de somnolencia en la tarjeta *Raspberry*. Si el usuario opta por compartir su número telefónico, deberá vincular el dispositivo *Bluetooth* del *smartphone* con la tarjeta presionando un botón. Una vez que ambos dispositivos están vinculados, el usuario puede iniciar la aplicación *Android* en el *smartphone* para compartir el número, estableciendo la conexión con la tarjeta *Raspberry* vinculada, como se muestra en la imagen de conexión exitosa 3.48a. El servicio de *Bluetooth server* se activa junto con los demás servicios cuando la tarjeta *Raspberry* se enciende, lo que permite recibir el número telefónico. Este dato se guarda en la base de datos de la aplicación para su reutilización cuando se ejecuta el servicio de llamadas telefónicas, el cual realiza la llamada al número ingresado desde la aplicación en un principio, solo si se detecta un patrón de somnolencia facial.

En este capítulo se ha expuesto el proceso de desarrollo e implementación del sistema de detección y alertamiento de somnolencia. Se comenzó con el diseño integral de la solución, abarcando tanto el *software* como el *hardware* necesario para llevar a cabo la detección de somnolencia. Se analizaron diferentes algoritmos para seleccionar el más adecuado, seguido de la integración del *hardware* necesario para la captura, procesamiento y análisis de imágenes, con énfasis en el uso de cámaras y sistemas empotrados. Además, se detallaron los mecanismos de alertamiento auditivo, tanto mediante voz como llamadas telefónicas, apoyados en comunicaciones *Bluetooth* y *Wi-Fi*. Finalmente, se abordó la implementación del *software*, incluyendo la arquitectura de servicios y las funciones específicas para la detección de somnolencia, así como los servicios de alertas por voz y llamadas telefónicas.

En el siguiente capítulo, se presentarán los resultados de las pruebas del prototipo del sistema, evaluando su desempeño mediante métricas clave del modelo de aprendizaje automático implementado tomando un enfoque cualitativo para trabajo futuro.

Capítulo 4

Resultados

En este capítulo se presentan los resultados obtenidos de las pruebas del sistema de detección y alerta de somnolencia (SDAS) a través de la evaluación del prototipo integrado con el *hardware* y *software*. Para validar el desempeño del sistema, se realizaron pruebas a cinco individuos que simularon somnolencia en un entorno controlado. El sistema emitió alertas de texto a voz o llamadas telefónicas basadas en la detección de somnolencia.

Se registraron los resultados de las alertas emitidas, categorizándolas en distintos casos, lo que permitió evaluar el desempeño del sistema mediante métricas de evaluación utilizando matrices de confusión. Cada evidencia obtenida de los sujetos es analizada y documentada en forma de tablas, mostrando el comportamiento del sistema ante una simulación de somnolencia.

4.1. Prueba del prototipo

Después de haber realizado la integración los distintos servicios y apartados del *software* (sección 3.3) en el *hardware*, se evaluó el desempeño del sistema mediante una vídeo-grabación del escritorio remoto de la tarjeta, usando la herramienta *VNC viewer*, para realizar un conteo y categorización del número de alertas que se ejecutaron (llamada o alerta de texto a voz) congruentemente con la detección de somnolencia.

Se grabó a 5 individuos donde cada uno de ellos simuló una situación de somnolencia excesiva, cerrando los ojos durante 10 segundos cada comienzo de minuto durante un periodo aproximado de 5 a 7 minutos. Por lo que se espera observar un mínimo de 5 simulaciones de somnolencia.

Por otra parte, el entorno donde se realizaron las evaluaciones fue en un automóvil estático donde se representaba una acción de conducción en un momento (aproximadamente 12:00 horas) en el que la luz natural permitiera la mejor captación de luz por parte de la cámara web. Cabe mencionar que la cámara se encontraba en la misma posición, sobre el tablero, frente al conductor.

Figura 4.1: Integración del prototipo del Sistema de Detección y Alertamiento de Somnolencia en un automóvil.

A continuación se presenta la evidencia de cada individuo evaluado, mostrando el re-

gistro de alertas y detecciones en forma de tablas. Estas tablas registran las alertas en momentos donde se simula somnolencia y también si el sistema realiza una alerta sin que haya una simulación de somnolencia.

El objetivo del registro es clasificar cada alerta en una de las siguientes cuatro situaciones posibles.

- **Verdadero Positivo (TP)**: El sistema emite una alerta cuando el individuo simula somnolencia.
- **Falso Positivo (FP)**: El sistema emite una alerta cuando el individuo no simula somnolencia.
- **Falso Negativo (FN)**: El sistema no emite una alerta cuando el individuo simula somnolencia.
- **Verdadero Negativo (TN)**: El sistema no emite una alerta cuando el individuo no simula somnolencia.

Estas categorizaciones permiten evaluar el rendimiento cuantitativo del SDAS, mediante el cálculo de métricas como sensibilidad, precisión y exactitud, a partir de una matriz de confusión generada para cada evidencia. Finalmente, se presenta una matriz de confusión total con el número acumulado de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos, lo cual permite determinar las métricas globales del modelo.

4.1.1. Evidencia 1

El sujeto etiquetado como evidencia 1 exhibe una edad aproximada de 60 años y usa anteojos. En esta prueba, el individuo simuló somnolencia excesiva cerrando los ojos en 7 ocasiones, y en 6 ocasiones realizó bostezos como síntoma de somnolencia de bajo nivel en un período de 6 minutos.

La figura 4.2 muestra un cuadro de imagen donde se detectó una alerta de somnolencia tanto sonora como por llamada telefónica, en la cual el individuo es monitoreado simulando somnolencia de alto nivel, cerrando los ojos.

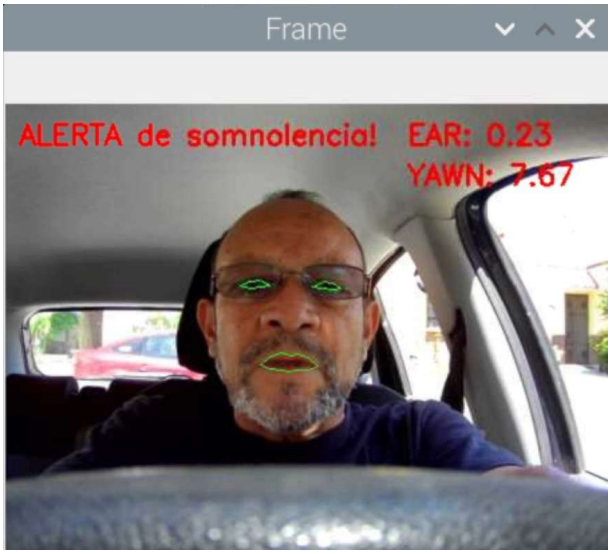


Figura 4.2: Cuadro de imagen captando una alerta por somnolencia de la evidencia 1.

En la tabla 4.1 se muestra el registro de las alertas al evaluar las simulaciones de somnolencia excesiva del sujeto en el SDAS.

Somnolencia		Alerta por detección	Minuto
S	N		
	X	SI	1:10
X		NO	1:17
X		NO	1:24
	X	SI	1:48
X		NO	2:14
	X	SI	2:52
X		SI	3:24
X		SI	3:57
X		NO	4:44
X		SI	5:39

Tabla 4.1: Registro de alertas de la evidencia 1.

De la Tabla 4.1, se observa que:

- La columna “Somnolencia” presenta dos estados: “S” denota simulación de somnolencia y “N” denota ausencia de somnolencia.
- La columna “Alerta por detección” indica si el SDAS emitió una alerta por somnolencia.
- La columna “Minuto” describe el momento en que se observó la alerta en el videoclip grabado durante la evaluación del individuo.

La Tabla 4.1 permite categorizar las muestras en las diferentes situaciones cuando el sistema emite una detección y determinar si son:

1. Verdaderos Positivos (TP).
2. Falsos Positivos (FP).
3. Falsos Negativos (FN).
4. Verdaderos Negativos (TN).

Estos casos facilitan la creación de la matriz de confusión. En la Tabla 4.2, se muestra la matriz de confusión correspondiente a la evidencia 1. Se registraron 3 muestras como verdaderos positivos, 4 como falsos negativos, 3 como falsos positivos y 7 como verdaderos negativos. Los verdaderos negativos representan casos en los que el modelo no emite una alerta cuando no se simuló somnolencia, lo cual son 7 casos de ausencia de somnolencia.

Predicción Ojos			
ACTUAL		Positivo	Negativo
	Positivo	3 TP	4 FN
	Negativo	3 FP	7 TN

Tabla 4.2: Matriz de confusión de la evidencia 1.

Por otro lado, el desempeño del modelo en esta prueba se muestra a continuación. La tasa correspondiente a la precisión del modelo se determinó a partir de la siguiente expresión:

Precisión:

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{3}{3+3} = 0.5 \tag{4.1}$$

Para la exactitud:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3+7}{3+7+3+4} = 0.5882 \tag{4.2}$$

Finalmente la sensibilidad o *recall* muestra un resultado de:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{3}{3+4} = 0.4285 \tag{4.3}$$

4.1.2. Evidencia 2

En esta prueba, el sujeto es el mismo que se evaluó en la evidencia 1, pero sin anteojos.

La figura 4.3 muestra un cuadro de imagen donde el individuo está siendo monitoreado con el SDAS; se aprecia el momento en que el sistema arroja una alerta de somnolencia cuando el individuo simula cerrar los ojos.

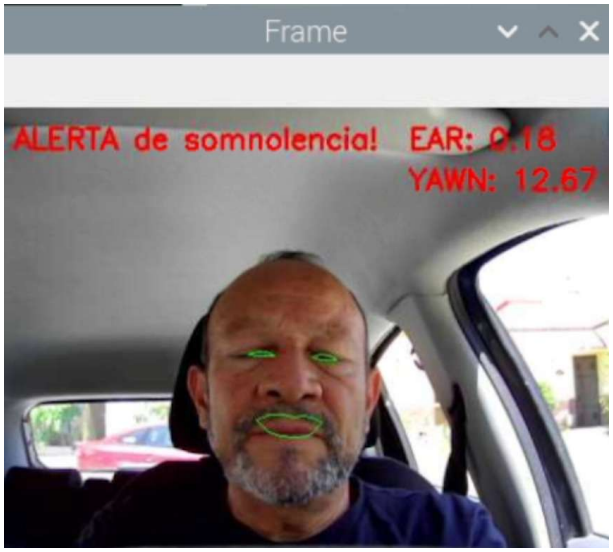


Figura 4.3: Cuadro de imagen captando una alerta por somnolencia de la evidencia 2.

En esta evidencia, el individuo realizó 6 simulaciones de somnolencia de alto nivel cerrando los ojos durante 10 segundos cada minuto. La tabla 4.3 muestra las diferentes

situaciones que se capturaron de la grabación del individuo evaluado en el sistema. Cabe mencionar que se encuentran más de 6 muestras, ya que también se presentan casos en donde el individuo no simuló somnolencia, pero el sistema detectó que sí la hubo.

Somnolencia		Alerta por detección	Minuto
S	N		
X		NO	1:34
X		NO	2:38
X		SI	3:38
	X	SI	4:12
X		SI	4:45
	X	NO	5:35

Tabla 4.3: Registro de alertas de la evidencia 2.

Por otro lado, la tabla 4.4 presenta los resultados finales de la matriz de confusión, con un total de 2 verdaderos positivos, 4 falsos negativos, 2 falsos positivos y 5 verdaderos negativos.

PREDICCIÓN Ojos			
ACTUAL		Positivo	Negativo
	Positivo	2 TP	2 FN
	Negativo	1 FP	5 TN

Tabla 4.4: Matriz de confusión de la evidencia 2.

A continuación se muestra el cálculo de desempeño para la evidencia 2.

Precisión:

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{2}{2+1} = 0.6666 \tag{4.4}$$

Para la exactitud:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2+5}{2+5+1+2} = 0.7 \tag{4.5}$$

Finalmente la sensibilidad o *recall* muestra un resultado de:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{2}{2+2} = 0.5 \tag{4.6}$$

Interpretación de la evidencia 1 y 2:

El mismo sujeto fue evaluado con y sin anteojos. Los resultados indican una mejora en todas las métricas cuando no usa anteojos, sugiriendo que el uso de anteojos podría afectar negativamente el rendimiento del modelo en la detección de características faciales, especialmente en la sensibilidad (del 0.4285 al 0.5) y en la precisión.

4.1.3. Evidencia 3

El individuo a evaluar es un femenino adulto de 29 años de edad. El individuo simuló en 5 ocasiones somnolencia de alto nivel, cerrando los ojos durante 10 segundos antes de finalizar cada minuto, durante 5 minutos.

La figura 4.4 muestra al individuo monitorizado con el SDAS, donde se observa una alerta verdadera emitida mientras el sujeto simulaba un episodio de somnolencia de alto nivel al cerrar los ojos.

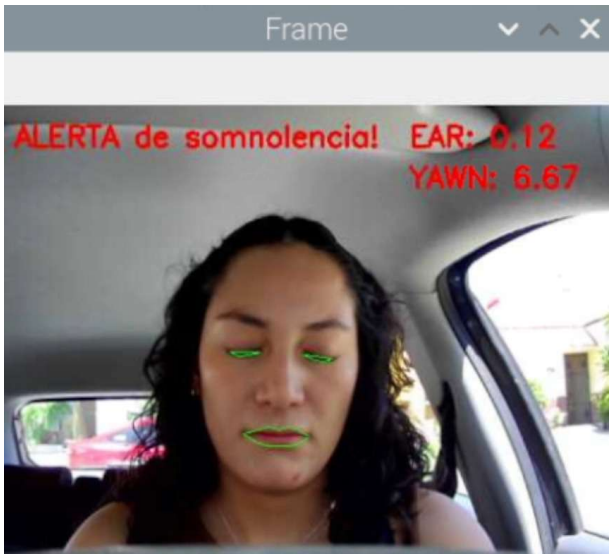


Figura 4.4: Cuadro de imagen captando una alerta por somnolencia de la evidencia 3.

La tabla 4.5 registra los momentos en los que el individuo simuló somnolencia. Se aprecia que el sistema no emitió alertas cuando el individuo no estaba emulando somnolencia; sin embargo, durante las cinco simulaciones realizadas, solo se presentaron dos de los cuatro posibles casos de alerta.

Somnolencia		Alerta por detección	Minuto
S	N		
X		SI	1:58
X		SI	3:01
X		NO	3:54
X		NO	5:05
X		SI	6:14

Tabla 4.5: Registro de alertas de la evidencia 3.

Por lo tanto, la tabla 4.6 presenta el conteo de los posibles casos generados durante la monitorización del individuo. Tal que, se registraron 3 muestras verdaderas positivas, 2 falsas negativas, 0 falsos positivos y 5 verdaderos negativos.

PREDICCIÓN Ojos			
ACTUAL		Positivo	Negativo
	Positivo	3 TP	2 FN
	Negativo	0 FP	5 TN

Tabla 4.6: Matriz de confusión de la evidencia 3.

Una vez construida la matriz de confusión, se procedió al cálculo de métricas de desempeño del sistema para esta evidencia.

Precisión:

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{3}{3 + 0} = 1 \tag{4.7}$$

Para la exactitud:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3 + 5}{3 + 5 + 0 + 2} = 0.8 \tag{4.8}$$

Finalmente la sensibilidad o *recall* muestra un resultado de:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{3}{3 + 2} = 0.6 \tag{4.9}$$

Interpretación:

Para este sujeto más joven, la precisión indica que el modelo no tuvo falsos positivos. La exactitud es alta (0.8) y la sensibilidad también es aceptable (0.6). Estos resultados sugieren que el modelo tiene un mejor rendimiento en sujetos más jóvenes, posiblemente porque la edad avanzada puede introducir variaciones en las características faciales (arrugas, párpados caídos, uso de anteojos) que afectan la precisión.

4.1.4. Evidencia 4

La persona evaluada en esta prueba es un femenino adulto mayor de 60 años de edad que emuló 5 episodios de somnolencia de alto nivel durante 10 segundos cada minuto, durante aproximadamente 5 minutos.

La figura 4.5 describe al individuo monitoreado por el SDAS en un momento en que simula somnolencia de alto nivel, cerrando los ojos y el sistema arroja una alerta de detección positiva de somnolencia.

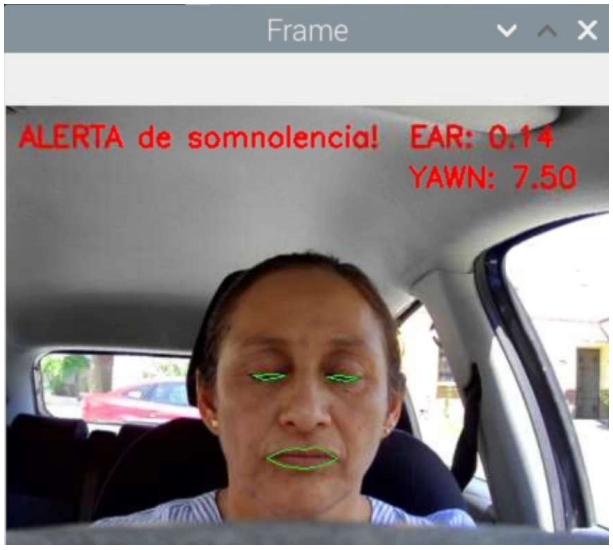


Figura 4.5: Cuadro de imagen captando una alerta por somnolencia de la evidencia 4.

La tabla 4.7 presenta el registro de las alertas generadas durante el tiempo de prueba, mostrando el momento de la grabación del monitoreo.

Somnolencia		Alerta por detección	Minuto
S	N		
X		NO	1:30
X		NO	2:40
X		SI	3:27
X		NO	3:50
X		SI	4:25
	X	SI	4:58
X		NO	5:35
	X	SI	4:41

Tabla 4.7: Registro de alertas de la evidencia 4.

En consecuencia, el conteo de las alertas permite generar la matriz de confusión correspondiente para este individuo. La tabla 4.8 presenta 5 verdaderos negativos, 2 falsos positivos, 4 falsos negativos y 2 verdaderos positivos.

PREDICCIÓN Ojos			
ACTUAL		Positivo	Negativo
	Positivo	2 TP	4 FN
	Negativo	2 FP	5 TN

Tabla 4.8: Matriz de confusión de la evidencia 4.

A continuación se presentan las métricas de desempeño para esta prueba.

Precisión:

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{2}{2+2} = 0.5 \tag{4.10}$$

Para la exactitud:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2+5}{2+5+2+4} = 0.5384 \tag{4.11}$$

Finalmente la sensibilidad o *recall* muestra un resultado de:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{2}{2+4} = 0.3333 \tag{4.12}$$

Interpretación:

Este sujeto, también de edad avanzada, muestra métricas más bajas que el sujeto más joven, con una precisión de 0.5, una exactitud baja de 0.5384 y una sensibilidad pobre de 0.3333. Esto respalda la idea de que el rendimiento del modelo disminuye para personas de mayor edad.

4.1.5. Evidencia 5

El individuo evaluado en esta evidencia corresponde a una mujer de aproximadamente 65 años. Al igual que en pruebas anteriores, simuló somnolencia de alto nivel en cinco ocasiones, cerrando los ojos durante 10 segundos cada minuto. La figura 4.6 muestra un cuadro

de imagen donde el sistema emitió una alerta positiva de somnolencia en un momento en que la persona no presentó simulación de somnolencia.

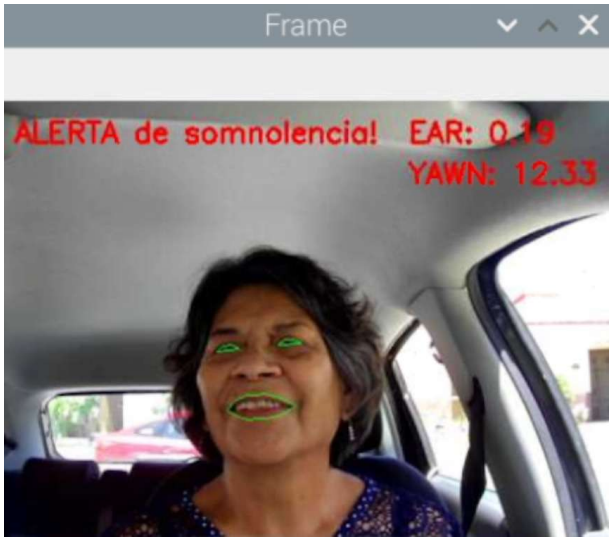


Figura 4.6: Cuadro de imagen captando una alerta por somnolencia de la evidencia 5.

La tabla 4.9 muestra el registro de las alertas generadas por el SDAS.

Somnolencia		Alerta por detección	Minuto
S	N		
X		SI	0:49
	X	SI	1:50
	X	SI	2:23
X		SI	3:32
X		SI	3:32

Tabla 4.9: Registro de alertas de la evidencia 5.

El conteo de las alertas anteriores permite generar la matriz de confusión que cataloga las muestras en los cuatro casos posibles. La tabla 4.10 presenta el número de muestras catalogadas para cada caso, donde se presentan 3 muestras positivas verdaderas, 1 falso negativo, 1 falso positivo y 5 verdaderos negativos.

PREDICCIÓN Ojos			
ACTUAL		Positivo	Negativo
	Positivo	3 TP	1 FN
	Negativo	2 FP	5 TN

Tabla 4.10: Matriz de confusión de la evidencia 5.

Finalmente, se muestran las métricas de evaluación de desempeño:

Precisión:

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{3}{3+2} = 0.6 \tag{4.13}$$

Para la exactitud:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3+5}{3+5+2+1} = 0.72 \tag{4.14}$$

Finalmente la sensibilidad o *recall* muestra un resultado de:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{3}{3+1} = 0.75 \tag{4.15}$$

Este individuo, a pesar de ser mayor, muestra una mejora notable en la sensibilidad (0.75), junto con una buena exactitud (0.72) y una precisión razonable (0.6). Esto sugiere que, aunque la edad puede afectar el rendimiento del modelo, hay variabilidad en los resultados, lo que podría depender de características individuales específicas, como la claridad en la simulación de somnolencia.

4.2. Resultados de desempeño de las pruebas

A continuación se muestran gráficamente los resultados de las métricas de desempeño de cada prueba realizada.

Figura 4.7: Gráfica de desempeño de las 5 evidencias.

De la figura 4.7 se observa que el rendimiento del modelo tiende a ser mejor en sujetos más jóvenes (29 años), con una precisión correcta en la evidencia 3. En personas de mayor edad, como en las evidencias 1, 4 y 5, la precisión y la sensibilidad tienden a ser más bajas, aunque existen excepciones, como en la evidencia 5. El uso de anteojos (evidencias 1 y 2) afecta negativamente las métricas, lo que sugiere que el modelo puede tener dificultades para manejar obstrucciones faciales. Por ello, el desempeño del modelo parece estar influenciado por la edad del individuo y otros factores como el uso de anteojos, siendo más efectivo en individuos más jóvenes y sin obstáculos visuales o complexión facial afectada por las arrugas.

4.3. Resultado total del modelo

Para obtener las métricas de desempeño del algoritmo en general, se determinó una matriz de confusión total que suma todos los valores *TP*, *FN*, *FP*, y *TN* de las matrices individuales de cada evidencia. Como resultado, la tabla 4.11 muestra la matriz de confusión total, donde se muestran 13 predicciones verdaderas positivas, 13 falsos negativos, 7 falsos positivos y 27 verdaderos negativos.

PREDICCIÓN Ojos			
ACTUAL		Positivo	Negativo
	Positivo	13 TP	13 FN
	Negativo	7 FP	27 TN

Tabla 4.11: Matriz de confusión total.

Tomando estos valores, se calcularon las métricas que determinan el desempeño cuantitativo del modelo de detección de somnolencia. A partir de la definición de precisión, el puntaje obtenido mediante la expresión matemática fue el siguiente:

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{13}{13 + 7} = 0.65 \tag{4.16}$$

Para la exactitud:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{13 + 27}{13 + 27 + 7 + 13} = 0.67 \tag{4.17}$$

Finalmente la sensibilidad o *recall* muestra un resultado de:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{13}{13 + 13} = 0.5 \tag{4.18}$$

La tabla 4.12 muestra los resultados de desempeño expresados en porcentaje.

Desempeño del Algoritmo		
Precisión	Exactitud	Sensibilidad
65 %	67 %	50 %

Tabla 4.12: Valores de métricas de evaluación de desempeño para el algoritmo implementado.

La precisión indica qué tan bien el modelo identifica correctamente los casos positivos, considerando los falsos positivos. El modelo tiene una precisión del 65 %, lo que significa que, cuando predice ojos cerrados, es correcto en un 65 % de los casos. Esto sugiere que el modelo tiene una tasa moderada de falsos positivos.

La exactitud mide el porcentaje total de predicciones correctas, tanto para positivos como para negativos. La exactitud del modelo es del 67 %, lo que indica que en un 67 % de las veces el modelo predice correctamente tanto los casos de ojos cerrados como abiertos.

La sensibilidad mide la capacidad del modelo para detectar correctamente los casos positivos (ojos cerrados), ignorando los falsos negativos. La sensibilidad es del 50 %, lo que significa que el modelo solo detecta correctamente la mitad de los casos donde los ojos estaban cerrados, fallando en el otro 50 % (falsos negativos). Esto es relativamente bajo, indicando que el modelo tiene problemas para identificar correctamente los casos positivos.

En este capítulo se ha presentado la metodología empleada para evaluar el desempeño del modelo de detección de somnolencia. A través de la captura y análisis del registro de alertas, se construyeron matrices de confusión que permitieron calcular métricas clave, como la precisión, sensibilidad y exactitud del modelo. La interpretación de estos resultados proporcionó una visión clara del comportamiento del sistema en diferentes individuos evaluados, identificando puntos clave para diferentes áreas de mejora.

En el siguiente capítulo, se ofrecerán las conclusiones finales del prototipo, donde se resumirán los logros alcanzados y se señalarán diversas mejoras potenciales para futuras implementaciones, basadas en las observaciones obtenidas durante las pruebas y el análisis del desempeño.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusión general del rendimiento del modelo

A partir de los resultados obtenidos de la matriz de confusión total del capítulo 4, el desempeño del algoritmo de detección de 68 puntos de referencia (*landmarks*) en combinación con el clasificador *Haar* de *OpenCV* para la detección de rostros lo determinan las métricas de evaluación ya calculadas. Éstas proporcionan una visión cuantitativa del rendimiento del modelo en la detección de somnolencia.

Precisión (0.65): El modelo es razonablemente bueno al identificar los casos de ojos cerrados, pero hay un número significativo de falsos positivos (7 en total).

Exactitud (0.67): La capacidad del modelo para hacer predicciones correctas en general es decente, con un 67 % de precisión en todas las predicciones.

Sensibilidad (0.5): La capacidad del modelo para detectar casos de ojos cerrados es

baja, con un 50 % de sensibilidad. Esto sugiere que el modelo tiene una tendencia a no detectar algunos casos de ojos cerrados, lo que puede ser problemático en contextos donde es crucial no perder ninguna detección. En conclusión, el modelo tiene un rendimiento aceptable, pero hay espacio para mejorar en cuanto a la detección de ojos cerrados (mayor sensibilidad) y reducir la cantidad de falsos positivos.

5.2. Conclusiones finales

5.2.1. Reflexión crítica

Los resultados obtenidos reflejan que, si bien el modelo presenta un desempeño aceptable en términos generales, aún existe un margen importante de mejora, particularmente en la reducción de falsos negativos y en la precisión de la detección ocular, aspecto esencial para una identificación efectiva de la somnolencia en niveles avanzados. Esta limitación señala la necesidad de continuar optimizando tanto el algoritmo de aprendizaje automático como la calidad del procesamiento de imágenes.

Como reflexión final, el desarrollo del prototipo cumplió con los objetivos planteados, demostrando la viabilidad de implementar un sistema funcional de detección y alertamiento de somnolencia a partir de herramientas de *machine learning*. Este trabajo aporta a la disciplina al proponer una metodología integral que combina componentes de *hardware* y *software* en un entorno embebido, documentando cada fase del proceso de desarrollo. La tesis ofrece una guía para investigadores y desarrolladores interesados en abordar esta problemática, particularmente en la literatura hispana, donde la disponibilidad de bibliografía especializada es limitada. Este aporte amplía el acceso al conocimiento técnico en la región y sienta las bases para investigaciones futuras o mejoras en sistemas similares.

5.2.2. Limitaciones del estudio

Si bien el propósito principal del proyecto no fue alcanzar resultados de alta precisión en el desempeño del sistema, es importante señalar que el prototipo no representa aún una solución inmediata ni completamente fiable para abordar la problemática de la somnolencia

al volante. Las pruebas se realizaron en un entorno controlado, sin considerar variables críticas como las condiciones de iluminación cambiantes o las limitaciones en la velocidad de procesamiento de la tarjeta de desarrollo. Estas condiciones podrían afectar de manera significativa el rendimiento en situaciones reales, especialmente considerando que se trata de un sistema cuyo objetivo es prevenir accidentes.

Otra limitación relevante fue el uso de redes inalámbricas para el envío de alertas, que si bien representan una alternativa práctica y flexible, no garantizan plena confiabilidad en situaciones extremas donde una alerta oportuna es fundamental. En este sentido, la dependencia de una red Wi-Fi o de un teléfono inteligente puede comprometer la eficacia del sistema. Una solución más robusta podría consistir en la implementación de alertas locales mediante tonos o llamadas simuladas directamente desde la tarjeta de desarrollo, sin necesidad de intermediarios externos. Sin embargo, dicho enfoque requeriría un desarrollo más especializado y un estudio técnico más profundo.

Asimismo, la calidad de las alertas generadas por voz sintetizada representa una limitación adicional. Aunque se emplearon herramientas como *Festival* y *text2speak*, estas bibliotecas producen voces en español que suenan artificiales y poco naturales. La integración de voces más realistas y humanizadas implicaría investigar alternativas más avanzadas, lo cual excedía el alcance de esta investigación inicial.

5.2.3. Trabajo futuro

Una alternativa a mejorar es cambiar el valor umbral y realizar pruebas para comprobar cuáles valores arrojan un mejor desempeño y se adaptan a la complexión facial para la mayoría de conductores. Ya que en situaciones donde el individuo a evaluar presenta los párpados caídos o los músculos que rodean al ojo están muy relajados, el detector realiza predicciones falsas negativas. Un ejemplo claro es la comparación de los resultados de desempeño (4.7) de la evidencia 3 respecto a las demás evidencias. Lo cual, determinar los umbrales correctos permitirá que el sistema sea capaz de detectar estos cambios de distancia de una mejor manera para cada individuo, tenga o no, los párpados caídos o cortos.

El algoritmo pierde la capacidad de realizar detecciones correctas cuando el conduc-

tor mueve la cabeza repentinamente a cualquier dirección, generando falsos positivos o verdaderos negativos. Para mejorar el desempeño del modelo, se puede considerar la implementación de un detector de rostros más preciso basado en aprendizaje profundo (*Deep Learning*) combinando técnicas de aprendizaje automático como las versiones más actualizadas de ***YOLO*** (*Convolutional Network*) e intentar adaptarlo a un sistema empotrado, ya que algunos prototipos arrojan resultados prometedores en computadores de escritorio. Estos modelos, aunque más demandantes en términos de recursos computacionales, tienden a ofrecer una mayor precisión en la detección de rostros en condiciones variadas de luz, ángulos de posición y expresiones faciales.

Retomando el prototipo diseñado, se podrían ajustar los parámetros de umbral y número de *frames* consecutivos para la detección de somnolencia, optimizando así la sensibilidad y precisión del modelo para reducir falsos negativos y mejorar la detección temprana de signos de somnolencia.

Por otro lado, una posible mejora en el sistema sería la vinculación automática por *Bluetooth* de la tarjeta *Raspberry* con el teléfono inteligente. También, se podría automatizar la conexión de la *Raspberry* a una red *Wi-Fi* abierta o conectarla a un enrutador mediante *Ethernet* para obtener acceso a Internet a bordo, como lo traen integrado algunos modelos de autos.

Además, es posible redefinir la integración del sistema contemplando la emulación de una llamada telefónica desde la misma tarjeta *raspberry*, sin depender de servicios de la red celular o internet, tal que la emulación sea local mediante el uso de un perfil *bluetooth* para el uso de tonos de llamada.

Finalmente, el *software* del prototipo puede optimizarse eligiendo un sistema operativo que incluya únicamente las herramientas esenciales para ejecutar la aplicación, sin un entorno gráfico ni aplicaciones *preinstaladas*. Esto reduciría significativamente el consumo de recursos y mejoraría el rendimiento, ya que el proyecto no requiere una interfaz gráfica ni el uso de un monitor. Algunas alternativas adecuadas son *Raspberry Pi OS Lite* (64 bits).

Apéndice A

Instalación de *OS*

A.1. Instalación del sistema operativo *Raspberry Pi OS*

Primero se instaló el sistema operativo propio de la marca, denominado *Raspberry Pi OS*, en su versión *Bullseye* de 64 bits. Esta elección se fundamenta en que la mayoría del *software* disponible actualmente está desarrollado para arquitecturas de 64 bits.

La versión de 32 bits presenta limitaciones importantes, como la falta de algunas dependencias de *Python* o errores de compatibilidad, especialmente con bibliotecas clave como *OpenCV*, esenciales para proyectos de visión artificial. Además, el sistema operativo de 64 bits permite direccionar memoria para procesos que demandan hasta 1TB, superando ampliamente a la versión de 32 bits.

Cabe destacar que la compañía *Raspberry Pi* garantiza soporte para la mayoría de sus placas hasta al menos el año 2026 o 2028, según el modelo. No obstante, hasta la fecha no se ha especificado una declaración de obsolescencia para el modelo *Raspberry Pi 4 Computer Model B 8GB RAM*.

A.1.1. Carga e instalación de *Raspberry Pi OS*

El primer paso consiste en ingresar al sitio oficial de *Raspberry Pi* para descargar el instalador o la imagen del sistema operativo desde la sección *Software*. La figura A.1 muestra el sitio web donde se selecciona la versión para *Windows*, indicada con un recuadro rojo y la leyenda "*Download for Windows*".

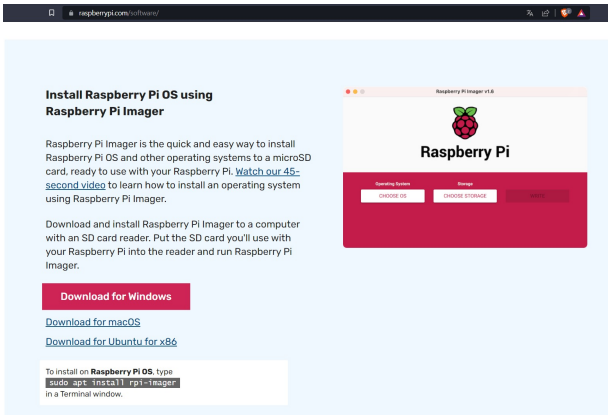


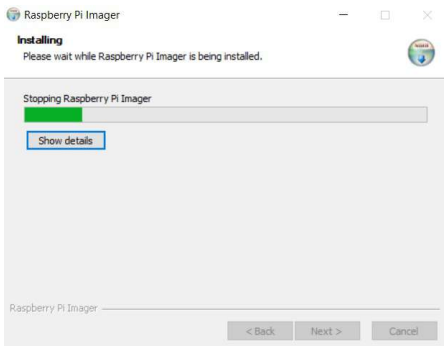
Figura A.1: Sitio web para descarga de imagen de *Raspberry Pi OS* para *Windows OS*.

Finalizada la descarga, se ejecuta el archivo *.exe* (imager_1.7.4), aceptando los permisos de administrador. La figura A.2 muestra el ícono del archivo.

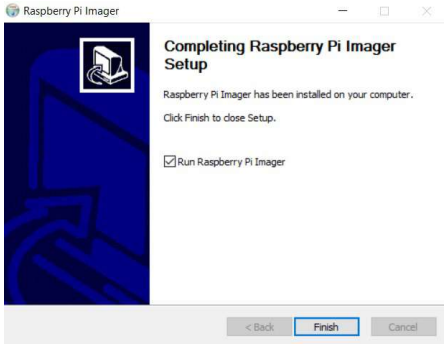
Figura A.2: Logo del archivo instalador de *Raspberry Pi imager*.

El proceso de instalación es sencillo: se selecciona "*Install*", se espera su finalización y se concluye con "*Finish*", como se observa en la figura A.3.

(a) Iniciar proceso de instalación.



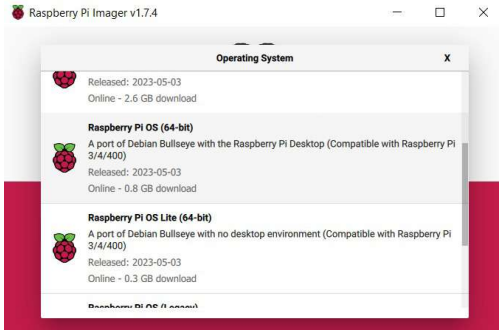
(b) Espera de instalación.



(c) Finalizar instalación.

Figura A.3: Proceso para la instalación del software *Raspberry Pi Imager*

Al concluir la instalación, se abre automáticamente el programa *Raspberry Pi Imager*. Este permite seleccionar el sistema operativo a instalar y el medio de almacenamiento. En la figura A.4a se observa la selección del sistema *Raspberry Pi OS (64 bit)*; en la figura A.4b, el almacenamiento, una memoria *micro SD* de 16 GB, representada en la figura A.4c.



(a) Opción del sistema operativo *Raspberrry Pi OS (64 bit)*.

(b) Escribir y cargar el sistema operativo en almacenamiento seleccionado.

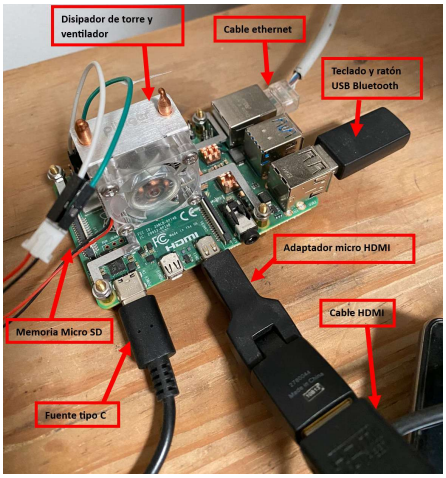
(c) Memoria *micro SD*, almacenamiento seleccionado.

Figura A.4: Proceso de selección de sistema operativo, almacenamiento y escritura del OS.

Para completar la carga de la imagen, se selecciona "*Write*". El sistema operativo se escribe en la memoria *micro SD*, previamente formateada.

Antes de encender la tarjeta *Raspberry Pi 4 Model B*, se inserta la memoria *micro SD*, se conecta un monitor al puerto *micro HDMI*, un cable *ethernet*, teclado y ratón a los puertos *USB*. Esta configuración permite operar localmente, ya que no se configuró acceso remoto durante la carga de la imagen.

Posteriormente, se alimenta la tarjeta con un eliminador de 5.1V y 3.0A. La figura A.5a muestra las conexiones; la figura A.5b, la fuente de alimentación.



(a) *Hardware*

(b) Eliminador

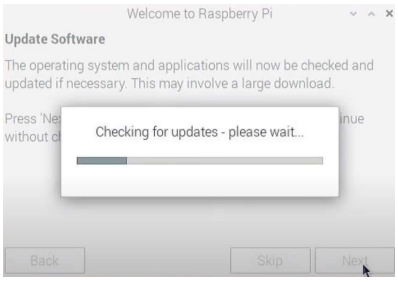
Figura A.5: Componentes y conexiones para *Raspberry Pi 4 Model B*.

Al encender el dispositivo, se despliega en el monitor una ventana de bienvenida (figura A.6a), seguida de configuraciones iniciales: país, idioma, zona horaria y teclado (figura A.6b), cambio de contraseña (figura A.6c) y actualización de *software* (figura A.6d). Finalmente, se reinicia el sistema para aplicar los cambios.

(a) Bienvenida a *Raspberry Pi OS*.

(b) Configuración regional y de teclado.

(c) Cambio de contraseña.



(d) Actualización de *software*.

Figura A.6: Configuraciones iniciales del sistema operativo *Raspberry Pi OS*.

Para visualizar las propiedades del sistema instalado, se abre una terminal (Ctrl+Alt+T), se ejecutan los comandos “sudo apt install neofetch” y “neofetch”. La figura A.7 muestra información como: versión del sistema (*Debian GNU/Linux 11 (bullseye) aarch64*), uso de *RAM* en reposo (407MiB), frecuencia del *CPU* (1.8 GHz) y resolución del monitor (1920x1080).

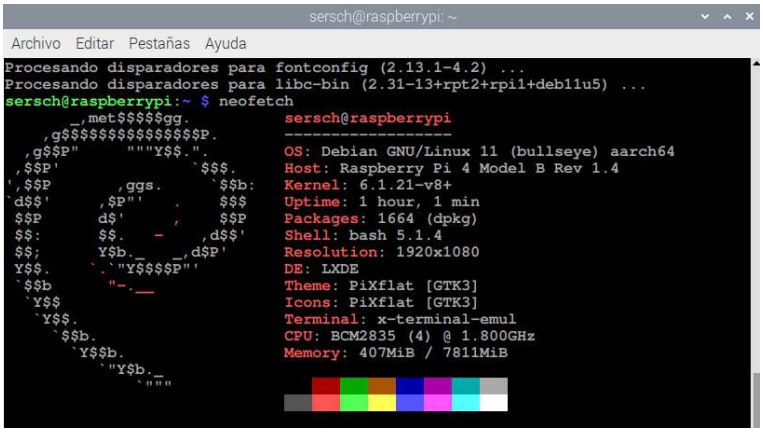


Figura A.7: Comando *neofetch* que despliega propiedades de la tarjeta *Raspberry Pi 4 Model B*.

A.1.2. *SSD* como dispositivo de almacenamiento para *SBC Raspberry Pi 4 Model B*

En esta sección se describe la implementación de un disco de estado sólido, el cual mejora significativamente la velocidad de lectura y escritura del computador de una sola tarjeta *Raspberry Pi 4 Model B*, al reemplazar la memoria micro *SD* por un *SSD*. Este cambio incrementa notablemente el rendimiento en escritura secuencial y en operaciones aleatorias (*IOPS*), permitiendo ejecutar aplicaciones de mayor demanda con mayor fluidez.

Durante la ejecución del sistema operativo *Raspberry Pi OS* en la *Raspberry Pi 4 Model B*, se detectó un bajo desempeño asociado a las limitaciones de transferencia de la tarjeta micro *SD* utilizada (ver sección A.1.1). Esta deficiencia afectaba la fluidez del sistema y la ejecución de aplicaciones.

Para evaluar el rendimiento de la tarjeta, se empleó la herramienta nativa *Raspberry Pi Diagnostics*, la cual permite medir la velocidad de lectura y escritura, determinando si

cumple con los estándares mínimos para un funcionamiento óptimo. La figura A.8 muestra la interfaz de la herramienta y la opción *Run Test* para iniciar la prueba.

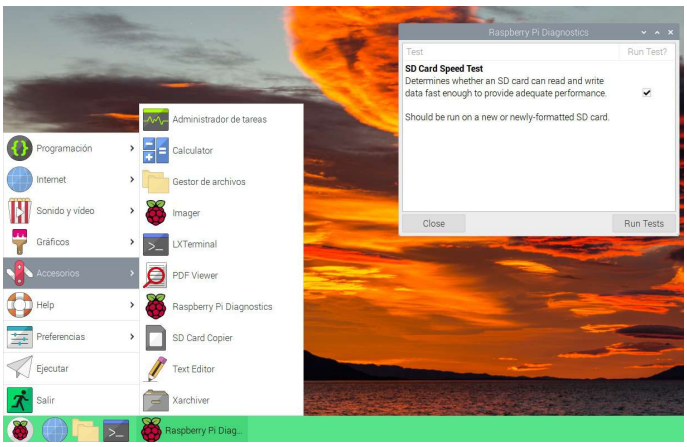


Figura A.8: Aplicación *Raspberry Pi Diagnostics*.

Al finalizar la prueba, el sistema arroja un resultado con la leyenda *PASS* (figura A.9), lo cual indica que la unidad de almacenamiento cumple con los parámetros necesarios para un desempeño adecuado en el sistema operativo.

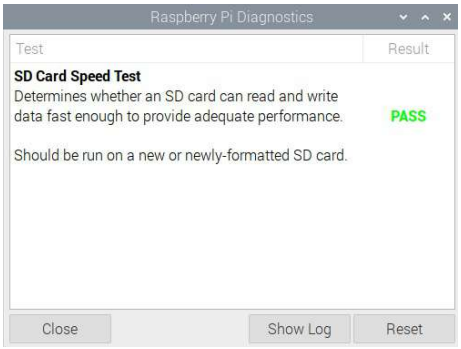


Figura A.9: Test de Raspberry Pi Diagnostics.

Al seleccionar la opción *Show Log*, se abre un archivo de tipo *.txt* denominado *rp-diags.txt*, donde se detallan parámetros clave del desempeño de la tarjeta *SD*, como las operaciones aleatorias por segundo (*IOPS*) en lectura y escritura, cuyos valores mínimos aceptables son 500 y 1,500 respectivamente. También se incluye la velocidad de escritura secuencial en *KB/s*, con un umbral mínimo de 10,000.

La figura A.10 muestra que la memoria *SD* alcanza una velocidad secuencial de 15,976 *KB/s*, junto con 917 *IOPS* en escritura y 2,679 *IOPS* en lectura, superando todos los

umbrales mínimos con la leyenda *PASS*.

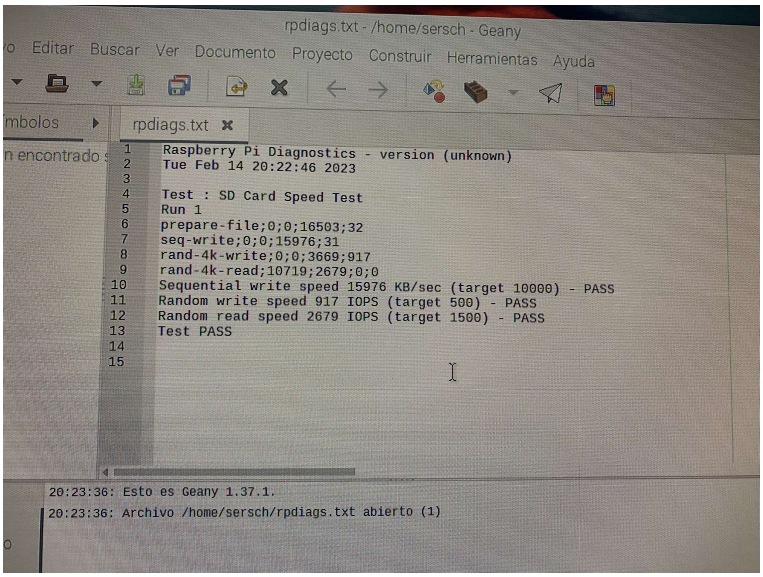


Figura A.10: Desempeño de escritura y lectura de la tarjeta de memoria *SD*.

Si bien los valores de operaciones aleatorias por segundo (*IOPS*) superan los umbrales mínimos, lo hacen por márgenes relativamente estrechos: 417 *IOPS* en escritura y 1,179 en lectura. Esto indica que el rendimiento, aunque aceptable, no duplica los valores mínimos requeridos, lo cual limita una experiencia fluida para el usuario. De manera similar, la velocidad de escritura secuencial supera por 5,975 *KB/s* el umbral establecido.

A mayor cantidad de *IOPS*, mayor será la velocidad de transferencia y, en consecuencia, el rendimiento general del sistema.

Uno de los principales inconvenientes detectados al utilizar una tarjeta micro *SD* como unidad principal fue el prolongado tiempo de arranque y la lentitud en la ejecución de aplicaciones e intercambio de archivos. Ante esta situación, se optó por una alternativa con mejor desempeño: un disco de estado sólido (*SSD*).

Se dispuso de una unidad *SSD* previamente utilizada como almacenamiento externo en una computadora portátil. Esta unidad, al usar la interfaz *SATA*, ofrece velocidades de lectura y escritura significativamente superiores a las de una memoria micro *SD*. Aunque la *Raspberry Pi 4* no cuenta con un conector *SATA* nativo, se empleó un adaptador *USB-SATA* para conectar el dispositivo. La tarjeta incluye puertos *USB* 3.0, capaces de transferir datos a una velocidad de hasta 5 Gbps (aproximadamente 625 MB/s) [45].

La figura A.11 muestra la conexión entre el *SSD* y la *Raspberry Pi 4 Model B*, donde también se emplea un disipador de calor tipo torre para evitar la limitación térmica del sistema, proteger el *hardware* y prolongar su vida útil.

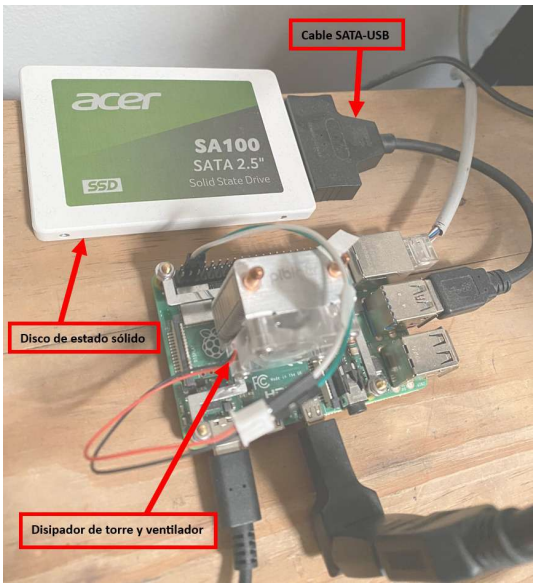


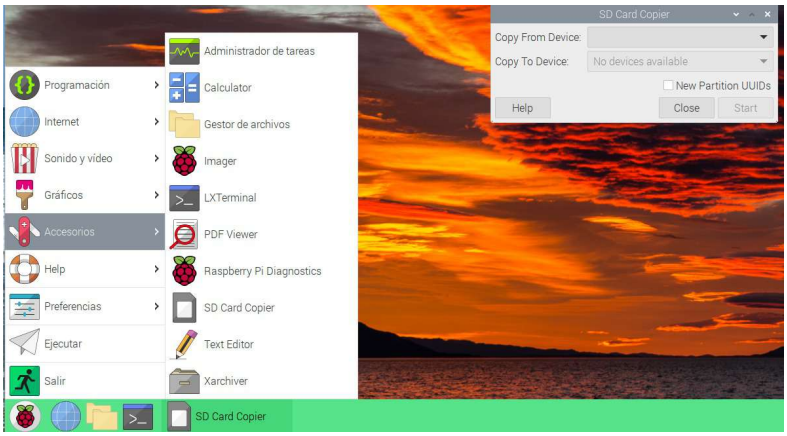
Figura A.11: Conexión de *SSD* y *Raspberry Pi model 4 B*.

Para grabar la imagen del sistema operativo *Raspberry Pi OS* en la nueva unidad *SSD*, se deben seguir los pasos descritos en el apartado A.1, seleccionando el *SSD* como unidad de destino en el programa *Raspberry Pi Imager*.

No obstante, en el diseño del SDAS también se optó por una alternativa más práctica: clonar el contenido de la micro *SD* al *SSD*, incluyendo el sistema operativo, mediante la herramienta *SD Card Copier*, integrada en *Raspberry Pi OS*. Este proceso se realiza directamente desde la *Raspberry Pi*, sin necesidad de una computadora externa.

La figura A.12a muestra el acceso a la herramienta desde el menú principal, mientras que en la figura A.12b se observan las opciones de origen y destino. La micro *SD* se selecciona como dispositivo fuente y el *SSD* como destino.

Para iniciar el proceso de copiado, que incluye la creación de particiones y transferencia del sistema operativo, se selecciona la opción *Start*.



(a) Herramienta *SD Card Copier*.

(b) Selección de dispositivos de almacenamiento en la aplicación *SD Card Copier*.

Figura A.12: Componentes y conexiones para *Raspberry Pi 4 Model B*.

En la figura A.13, la ventana final demuestra que el proceso de copiado ha concluido y se finaliza con el botón *OK*.

Figura A.13: Fin del proceso de copiado.

Es necesario modificar el orden de arranque desde la *BIOS* de la *Raspberry Pi 4 Model B* para establecer el *SSD* como dispositivo principal de arranque del sistema operativo.

Para realizar este cambio, se abre una terminal con el comando *Ctrl+Alt+T* y se ejecuta *sudo raspi-config*, lo que abre la ventana de configuración de la *BIOS*, como se muestra en la figura A.14.

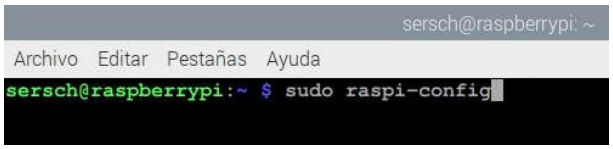
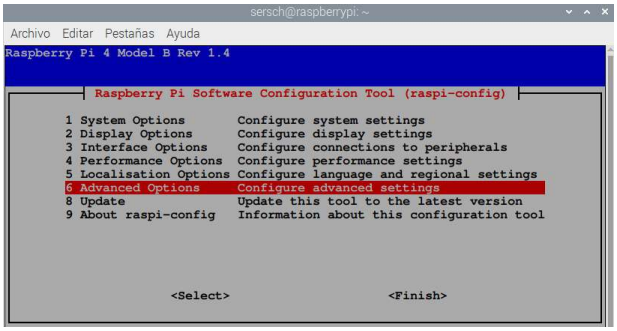


Figura A.14: Comando del lenguaje *bash* para abrir la *BIOS* desde terminal *shell*.

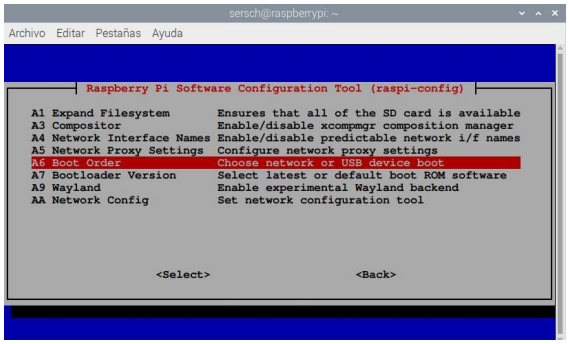
La figura A.15a muestra la ventana de configuración de la *BIOS* del computador *Raspberry Pi 4 Model B*. Se selecciona la opción 6, *Advanced Options*, desplazándose con las flechas del teclado y presionando *Enter*. Luego, se accede a la opción *A6 Boot Order*, como se observa en la figura A.15b.

Posteriormente, se selecciona *B2 USB Boot*, mostrada en la figura A.15c. La ventana siguiente confirma la selección, y se presiona *Aceptar* (figura A.15d). Finalmente, se elige *Finish* (figura A.15e), y se confirma el reinicio seleccionando la opción *Sí*, como se aprecia en la figura A.15f.

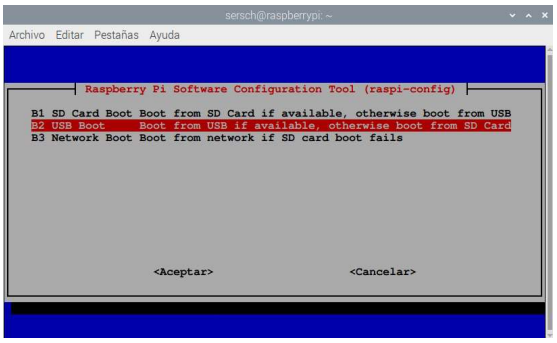
Una vez reiniciado el sistema, el sistema operativo se ejecuta desde el disco de estado sólido, permitiendo retirar la memoria micro *SD*.



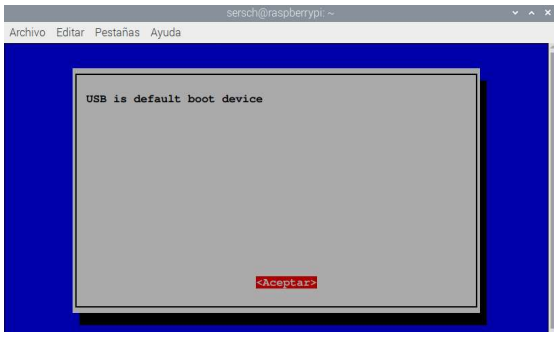
(a) 6 *Advanced Options.*



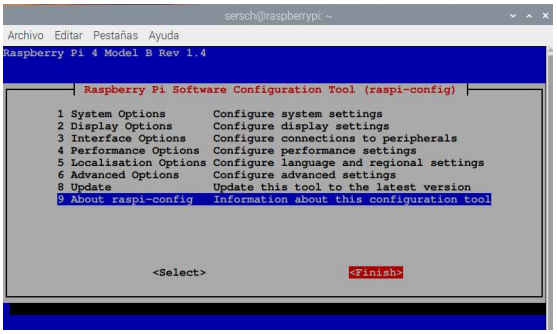
(b) A6 *Boot Order.*



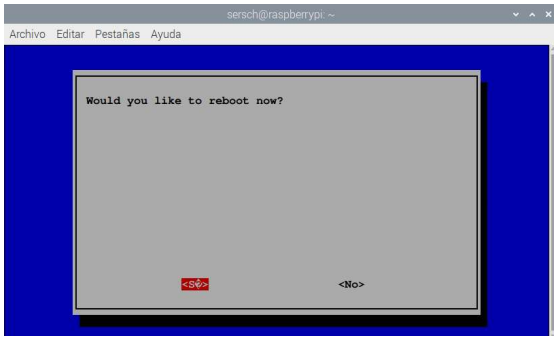
(c) B2 *USB Boot.*



(d) Aceptar configuración.



(e) Terminar con la configuración general.



(f) Reiniciar el sistema.

Figura A.15: Proceso para cambiar el orden de dispositivo de arranque.

Con esto concluye el proceso de instalación del sistema operativo en una nueva unidad de almacenamiento de tipo *SSD*.

Durante el arranque mediante el *SSD*, se percibe una mejora notable en el tiempo de encendido, así como una mayor fluidez al interactuar con el entorno gráfico, navegar entre aplicaciones o gestionar archivos.

Para cuantificar esta mejora en el desempeño del sistema operativo *Raspberry Pi OS*, se utilizó nuevamente la herramienta *Raspberry Pi Diagnostics*. Al ejecutar la prueba sobre el *SSD*, la figura A.16 muestra valores significativamente superiores en *IOPS* y velocidad

de escritura secuencial, en comparación con los obtenidos previamente con la tarjeta *micro SD*.

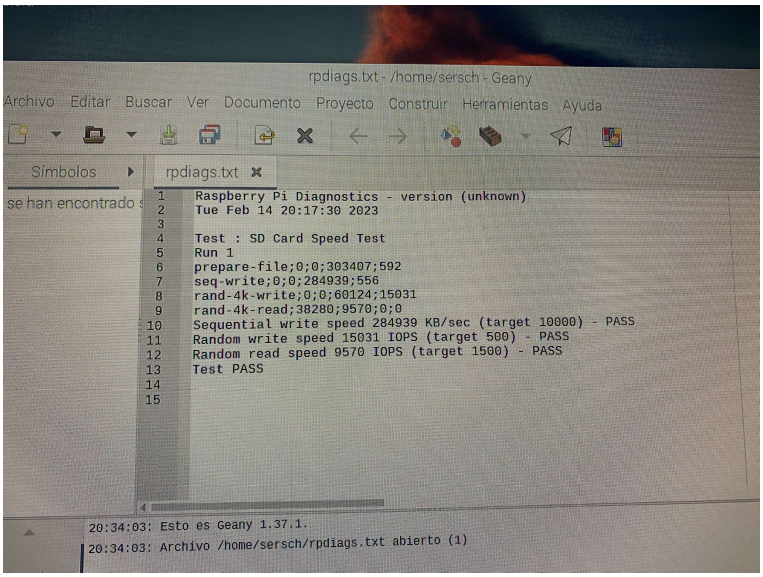


Figura A.16: Desempeño de escritura y lectura de la tarjeta de disco de estado sólido *SSD*.

En la tabla A.1 se presenta una comparación entre la memoria *micro SD* y el disco de estado sólido en cuanto a velocidad de escritura secuencial y operaciones aleatorias por segundo. La velocidad de escritura secuencial del *SSD* es 17.8 veces superior a la de la *micro SD*. En cuanto a las operaciones aleatorias por segundo, el *SSD* alcanza un valor 16.39 veces mayor en escritura y 3.57 veces mayor en lectura, lo que evidencia una mejora sustancial en el rendimiento del sistema.

Tabla A.1: Comparación de valores de velocidad ente *SSD* y *MicroSD*.

Dispositivos	Velocidad de Escritura Secuencial KB/s	Velocidad de escritura aleatoria IOPS	Velocidad de lectura aleatoria *IOPS*
SSD	284,393	15,031	9570
Micro SD	15,976	917	2679

Esta comparación permitió evidenciar que el cambio de unidad de almacenamiento mejora significativamente el rendimiento del sistema operativo, siempre que el nuevo dispositivo presente velocidades de transferencia de datos considerablemente superiores al anterior.

Bibliografía

[1] W. H. Organization, *Road traffic injuries*, 2023. dirección: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[2] Team-BHP, *Drowsiness and Sleepy Driving: The silent killer on Indian roads*, 2017. dirección: https://www.team-bhp.com/forum/road-safety/182716-drowsiness-sleepy-driving-silent-killer-indian-roads.html.

[3] B. Verma y A. Choudhary, «Framework for dynamic hand gesture recognition using Grassmann manifold for intelligent vehicles,» *IET Intelligent Transport Systems*, vol. 12, n.º 7, págs. 721-729, 2018.

[4] M. Matsugu, K. Mori, Y. Mitari e Y. Kaneda, «Subject independent facial expression recognition with robust face detection using a convolutional neural network,» *Neural Networks*, vol. 16, n.º 5, págs. 555-559, 2003, Advances in Neural Networks Research: IJCNN '03, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(03)00115-1. dirección: https://www.sciencedirect.com/science/article/pii/S0893608003001151.

[5] P. Lucey, J. Cohn, S. Lucey, I. Matthews, S. Sridharan y K. M. Prkachin, *Automatically detecting pain using facial actions*, 2009. DOI: 10.1109/ACII.2009.5349321.

[6] L. Liu, D. Preotiuc-Pietro, Z. R. Samani, M. E. Moghaddam y L. Ungar, *Analyzing Personality through Social Media Profile Picture Choice*, 2016. dirección: https://www.aaai.org/ocs/index.php/ICWSM/ICWSM16/paper/view/13102/12741.

[7] S. Dey, S. A. Chowdhury, S. Sultana, M. A. Hossain, M. Dey y S. K. Das, *Real Time Driver Fatigue Detection Based on Facial Behaviour along with Machine Learning Approaches*, 2019, págs. 135-140. DOI: 10.1109/SPICSCON48833.2019.9065120.

[8] STCONAPRA, «Informe sobre la situación de la seguridad vial, México 2020,» *Documentos de Secretaría de Salud*, vol. 1, pág. 191, 2022.

[9] S. de Salud/STCONAPRA, *Informe sobre la situación de la seguridad vial de México 2021*, 2021.

[10] S. D. Lin, J. J. Lin y C. Y. Chung, «Sleepy eye's recognition for drowsiness detection,» *Proceedings - 2013 International Symposium on Biometrics and Security Technologies, ISBAST 2013*, págs. 176-179, 2013. DOI: 10.1109/ISBAST.2013.31.

[11] J. W. Baek, B.-G. Han, K.-J. Kim, Y.-S. Chung y S.-I. Lee, «Real-Time Drowsiness Detection Algorithm for Driver State Monitoring Systems,» en *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018, págs. 73-75. DOI: 10.1109/ICUFN.2018.8436988.

[12] S. Dey, S. A. Chowdhury, S. Sultana, M. A. Hossain, M. Dey y S. K. Das, «Real Time Driver Fatigue Detection Based on Facial Behaviour along with Machine Learning Approaches,» en *2019 IEEE International Conference on Signal Processing, Information, Communication AND Systems (SPICSCON)*, 2019, págs. 135-140. DOI: 10.1109/SPICSCON48833.2019.9065120.

[13] A. Sinha, R. P. Aneesh y S. K. Gopal, «Drowsiness Detection System Using Deep Learning,» *Proceedings of 2021 IEEE 7th International Conference on Bio Signals, Images and Instrumentation, ICBSII 2021*, págs. 1-6, 2021. DOI: 10.1109/ICBSII51839.2021.9445132.

[14] S. Mohanty, S. V. Hegde, S. Prasad y J. Manikandan, «Design of real-time drowsiness detection system using dlib,» *2019 5th IEEE International WIE Conference on Electrical and Computer Engineering, WIECON-ECE 2019 - Proceedings*, págs. 1-4, 2019. DOI: 10.1109/WIECON-ECE48653.2019.9019910.

[15] J. W. Valvano, *Embedded systems : real-time interfacing to ARM® Cortex(TM)-M microcontrollers. Vol. 2*. Jonathan W. Valvano, 2012, pág. 579, ISBN: 9781463590154.

[16] J. W. Valvano, *Embedded Systems : real-time operating systems for the arm® cortex(TM)-M3. Volume 3*. CreateSpace, 2012, pág. 366, ISBN: 9781466468863.

[17] J. Peckol, *Embedded Systems: A Contemporary Design Tool*. Wiley, 2019, ISBN: 9781119457497. dirección: https://books.google.com.mx/books?id=mwGQDwAAQBAJ.

[18] A. R. Gallardo, J. R. H. Morales, S. S. Carrillo, M. Andrade-Aréchiga y E. M. R. Michel, *Internet de las cosas*. 31 de ene. de 2023. DOI: 10.53897/li.2023.0001.ucol. dirección: https://doi.org/10.53897/li.2023.0001.ucol.

[19] I.-R. Adochiei, O.-I. Știrbu, N. .-. I. Adochiei et al., «Drivers' Drowsiness Detection and Warning Systems for Critical Infrastructures,» en *2020 International Conference on e-Health and Bioengineering (EHB)*, 2020, págs. 1-4. DOI: 10.1109/EHB50910.2020.9280165.

[20] T. T. Hien, Q. Liang y N. T. D. Linh, «Design Driver Sleep Warning System Through Image Recognition and Processing in Python, Dlib, and OpenCV,» en *Intelligent Systems and Networks*, D.-T. Tran, G. Jeon, T. D. L. Nguyen, J. Lu y T.-D. Xuan, eds., Singapore: Springer Singapore, 2021, págs. 386-393, ISBN: 978-981-16-2094-2.

[21] P. Inthanon y S. Mungsing, «Detection of Drowsiness from Facial Images in Real-Time Video Media using Nvidia Jetson Nano,» *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, págs. 246-249, 2020.

[22] D. Athow, *ARM Cortex-A57 and A53 vs Cortex A8, A9, A15 and A7: a performance analysis*, 5 de oct. de 2022. dirección: https://www.itpro.com/hardware/367835/arm-cortex-a57-and-a53-vs-cortex-a8-a9-a15-and-a7-a-performance-analysis.

[23] E. Rosales Mayor y J. Rey De Castro Mujica, «Somnolencia: Qué es, qué la causa y cómo se mide,» es, *Acta Médica Peruana*, vol. 27, págs. 137-143, abr. de 2010,

ISSN: 1728-5917. dirección: http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S1728-59172010000200010&nrm=iso.

[24] M. E. Machado-Duque, J. E. Echeverri Chabur y J. E. Machado-Alba, «Somnolencia diurna excesiva, mala calidad del sueño y bajo rendimiento académico en estudiantes de Medicina,» *Revista Colombiana de Psiquiatría*, vol. 44, n.º 3, págs. 137-142, 2015, ISSN: 0034-7450. DOI: https://doi.org/10.1016/j.rcp.2015.04.002. dirección: https://www.sciencedirect.com/science/article/pii/S0034745015000426.

[25] M. Đokić, S. Nićetin, G. Velikić y T. Anđelić, «Driver Monitoring implementation in Adaptive AUTOSAR environment,» en *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2018, págs. 1-4. DOI: 10.1109/ICCE-Berlin.2018.8576255.

[26] A. Mohanty y S. Bilgaiyan, «Drowsiness Detection System Using KNN and OpenCV,» en *Machine Learning and Information Processing*, D. Swain, P. K. Pattnaik y T. Athawale, eds., Singapore: Springer Singapore, 2021, págs. 383-390, ISBN: 978-981-33-4859-2.

[27] E. Vural, M. Bartlett, G. Littlewort, M. Cetin, A. Ercil y J. Movellan, «Discrimination of Moderate and Acute Drowsiness Based on Spontaneous Facial Expressions,» en *2010 20th International Conference on Pattern Recognition*, 2010, págs. 3874-3877. DOI: 10.1109/ICPR.2010.943.

[28] M. Sunagawa, S.-i. Shikii, W. Nakai, M. Mochizuki, K. Kusukame y H. Kitajima, «Comprehensive Drowsiness Level Detection Model Combining Multimodal Information,» *IEEE Sensors Journal*, vol. 20, n.º 7, págs. 3709-3717, 2020. DOI: 10.1109/JSEN.2019.2960158.

[29] A. G. Serrano, *Inteligencia artificial, fundamentos práctica y aplicaciones*. Alfaomega, 2017, pág. 285, ISBN: 9786076227251.

[30] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd. Upper Saddle River, NJ: Prentice Hall, 2010.

[31] S. Aghabozorgi, *Machine Learning with Python: A Practical Introduction*, IBM, sep. de 2022. dirección: https://www.edx.org/es/course/machine-learning-with-python-a-practical-introduct.

[32] M. González y M. Penedo, «Visión artificial y aprendizaje profundo en el diagnóstico por imagen,» en *XVIII Reunión de Trabajo en Procesamiento de la Información y Control (RPIC)*, 2019.

[33] D. la Escalera A. Al-Kaff A., *Introducción a la visión por computador: desarrollo de aplicaciones con OpenCV*, Universidad Carlos III de Madrid, abr. de 2023. dirección: https://www.edx.org/es/course/introduccion-a-la-vision-por-computador-desarrollo.

[34] A. Duque, «Sistema de Atención Visual para la Detección de Puntos Topológicos de Referencia,» *Universidad Carlos III de Madrid*, 2009.

[35] A. Rosebrock, *Deep Learning for Computer Vision*. PyImageSearch.com, 2018.

[36] P. Viola y M. Jones, «Rapid Object Detection using a Boosted Cascade of Simple Features,» 2001.

[37] OpenCV, *Cascade Classifier*, 2023. dirección: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[38] M. Tyagi. «HOG (Histogram of Oriented Gradients): An Overview.» (2020), dirección: https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f.

[39] A. S. edX, *Machine Learning with Python: A Practical Introduction*, 2022. dirección: https://www.edx.org/es/course/machine-learning-with-python-a-practical-introduct.

[40] Arijit1080, *Drowsiness and Yawn Detection with voice alert using Dlib*, 2023. dirección: https://github.com/Arijit1080/Drowsiness-and-Yawn-Detection-with-voice-alert-using-Dlib.

[41] V. Kazemi y J. S. Kth, «One Millisecond Face Alignment with an Ensemble of Regression Trees.»

[42] D. developers, *DLib: C++ Toolkit for Making Real World Machine Learning Applications*, DLib developers, 2023. dirección: http://dlib.net/.

[43] Twilio, *Twilio*, 2023. dirección: https://www.twilio.com/en-us.

[44] L. L., *Android Studio y Arduino – Comunicación Bluetooth – Kotlin – BASICO*, 2019. dirección: https://cursos.innovadomotics.com/instructor/.

[45] U. I. Forum, *USB 3.0 Specification*, 2008. dirección: https://www.usb.org/document-library/usb-30-specification (visitado 11-05-2022).