ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.645

Drowsiness Detection System Using Computer Vision for Performance Characterization on a Raspberry Pi

Ricardo Yauri[1]; Antero Castro[1]; Rafael Espino[1]
[1]Universidad Tecnológica del Perú, Perú, *c24068@utp.edu.pe, C19828@utp.edu.pe, C22165@utp.edu.pe*

Abstract*– There is a need to monitor individuals performing critical tasks who are affected by drowsiness or fatigue using computer vision detection processes on embedded devices, but this presents a challenge due to the use of complex processing hardware. Previous research describes how embedded devices and microcontrollers perform drowsiness detection tasks using algorithms optimized for limited resources. Additionally, facial recognition methods, such as Haar Cascades or convolutional neural networks, are used to identify fatigue based on facial features. This research implements a computer vision-based drowsiness detection system that calculates facial metrics on a Raspberry Pi 4, using MediaPipe and OpenCV tools, integrating an ESP32 module for MQTT alerts, and a remote monitoring platform with Node-RED. The system recorded an average processing time per frame of 129.08 ms, with increases during drowsiness events (eye closure at 270 ms and yawning at 230 ms). Furthermore, the analysis on the Raspberry Pi showed a maximum CPU usage value of 52% without saturating this computational resource. The results indicate that it is possible to detect drowsiness and yawning states, characterizing their computational performance, although it has limitations due to the camera resolution and processing time.

Keywords*-- Computer vision, Raspberry Pi, Drowsiness detection, Edge computing, EAR.

Sistema de Detección de Somnolencia usando Visión Computacional para la Caracterización del Rendimiento en un Raspberry Pi

Ricardo Yauri[1]; Antero Castro[1]; Rafael Espino[1]
[1]Universidad Tecnológica del Perú, Perú, *c24068@utp.edu.pe, C19828@utp.edu.pe, C22165@utp.edu.pe*

Resumen*– Existe la necesidad de monitorear a personas que hacen tareas críticas que son afectadas por la somnolencia o fatiga usando sistemas de detección basados en visión computacional en dispositivos embebidos, pero hay un problema relacionado al uso de hardware de procesamiento complejo. Las investigaciones previas describen que los dispositivos embebidos y microcontroladores ejecutan tareas de detección de somnolencia usando algoritmos optimizados para recursos limitados. Además, los métodos de detección facial, como Haar Cascades o redes convolucionales son usados para identificar la fatiga mediante el rostro. En esta investigación se implementa un sistema de detección de somnolencia basado en visión computacional que calcula las métricas faciales en un Raspberry Pi 4, usando las herramientas MediaPipe, OpenCV, integrando un módulo ESP32 para alertas por MQTT y una plataforma de monitoreo remoto con Node-RED. El sistema registró un tiempo promedio de procesamiento por fotograma de 129.08 ms, con incrementos en eventos de somnolencia (cierre de ojos con 270 ms y bostezo con 230 ms). Además, el análisis en el Raspberry Pi mostró un valor máximo de uso CPU del 52% sin saturar este recurso computacional. Los resultados indican que es posible detectar los estados somnolencia y bostezo, caracterizando su rendimiento computacional, aunque presenta limitaciones por la resolución de la cámara y al tiempo de procesamiento.

Palabras clave—Visión computacional, Raspberry Pi, Detección de somnolencia, computación de borde, EAR.

I. Introducción

Actualmente existe la necesidad de realizar la supervisión de las actividades en personas que realizan trabajos críticos, como manejo de vehículos [1], actividades de cuidado de la salud [2], manipulación de maquinaria o actividades de larga duración [3], considerando el ámbito de la seguridad y la salud en el trabajo . En el caso específico de la somnolencia, este es un factor genera errores en las personas [4], reduce su rendimiento generando accidentes en el trabajo [5]. En este caso se hace necesario, que los sistemas de detección de eventos utilicen técnicas de visión computacional [6] para identificar signos de fatiga [7]. Es por ello que el uso de soluciones embebidas, que permitan procesar información en el borde sin depender de la computación en la nube [8], es una tendencia actual para aplicaciones portátiles y autónomas [9].

El problema de esta investigación se orienta a la necesidad de implementar una solución que no solo detecte estados de somnolencia, sino que también caracterice y analice su impacto relacionado al rendimiento computacional del hardware cuando se ejecuta directamente en un dispositivo de borde como el Raspberry Pi 4. Aunque existen sistemas de detección de fatiga en literatura previa, muchos de ellos requieren hardware de alto rendimiento o procesamiento en servidores en la nube [10]. Además, no se suele analizar la carga computacional durante la ejecución del algoritmo de detección [11], lo cual es importante para determinar la estabilidad del sistema [12].

En investigaciones previas se ha utilizado técnicas de visión computacional basadas en el Eye Aspect Ratio (EAR) y Mouth Aspect Ratio (MAR) [5], usando técnicas de aprendizaje profundo orientados para reconocimiento facial [13]. Otras soluciones han integrado alertas de eventos anómalos de somnolencia con microcontroladores [14] o tecnologías del Internet de las Cosas para el envío de notificaciones remotas [15].

En base a lo anterior, se plantea el diseño e implementación de un sistema de detección de somnolencia con visión computacional que procese valores de EAR y MAR en un Raspberry Pi 4, complementado con un módulo basado en ESP32 y visualización remota con Node-Red. Para ellos se tiene como objetivos específicos integrar las herramientas de MediaPipe y OpenCV para el algoritmo de detección; comunicación con un bróker MQTT para activar alertas en un dispositivo externo ESP32; implementar una aplicación de monitoreo mediante Node-RED y notificaciones por Telegram.

La motivación principal se justifica por la necesidad de demostrar que es posible implementar un sistema que combine algoritmos de visión computacional con tecnologías IoT sin afectar negativamente la eficiencia del hardware.

En este contexto, la investigación evalúa la viabilidad de implementar un sistema de detección en un dispositivo de borde, evaluando y caracterizando su desempeño computacional bajo condiciones reales de operación.

II. Trabajos relacionados

La revisión del estado de arte permitió identificar avances en sistemas de monitoreo para la detección de somnolencia en dispositivos embebidos como el Raspberry PI. Para ello, se usó una estrategia de búsqueda con la metodología Kitchenham complementada con la formulación de preguntas usando un enfoque PICO (Población, Intervención, Comparación y Resultados), y la guía PRISMA.

En artículos de investigación se han empleado herramientas de visión computacional y visión artificial para detectar de somnolencia, con el análisis del comportamiento ocular y facial. En el caso del Eye Aspect Ratio (EAR), se ha utilizado para el seguimiento del parpadeo y el cálculo de cierres prolongados [16] los cuales son adecuados para la identificación de eventos del sueño [17]. Otros métodos se orientan a la detección de bostezos mediante CNNs [18] y análisis mediante redes recurrentes [19].

En el caso del hardware, investigaciones muestran el uso de Raspberry Pi, Arduino y microcontroladores para la detección de alertas de sueño en entornos laborales de alto riesgo. Estos componentes pueden integrar cámaras, sensores ambientales [20] y algoritmos de inferencia [21]. Los artículos muestran que su uso es posible a pesar de los pocos recursos de hardware [22] considerando que se ejecuten tareas ligeras [23].

La Raspberry Pi ha sido ampliamente utilizada en visión computacional, debido a que es compatible con librerías como OpenCV, TensorFlow Lite o modelos como MobileNet [24] o YOLO-Tiny [25]. Por otro lado, en el caso de microcontroladores, como el ESP32-CAM, este realiza la captura de imágenes y la ejecución de procesos de inferencias con imágenes de baja resolución [26], orientado a aplicaciones de monitoreo en sistemas con tecnologías del IoT [27].

En el caso de los métodos de detección de características faciales, se utilizan enfoques clásicos como Haar Cascades [28], y también el uso de redes neuronales convolucionales [29]. Los estudios encontrados también pueden emplear arquitecturas como MediaPipe FaceMesh para encontrar puntos de características del movimiento de los ojos, labios o postura de cabeza. De esta forma se pueden encontrar signos de fatiga, como la reducción del EAR, o inclinación continúa de cabeza [30].

En el caso del análisis del rendimiento de hardware embebido, la literatura indica el uso de métricas como consumo de energía [31] y tiempo de inferencia durante la ejecución del algoritmo de detección de fatiga [32]. Además, se han encontrado el uso de tecnologías para la detección de fatiga y eventos asociados a la seguridad laboral que utilizan componentes como sensores portátiles [33], sistemas de monitoreo con análisis de señales fisiológicas (EEG o PPG) [34] y técnicas que combinan visión computacional con sensores adicionales en modelos multimodales.

La mayoría de los trabajos se enfocan en mejorar la precisión de los modelos de detección de somnolencia, empleando técnicas de aprendizaje profundo o clasificadores avanzados, sin embargo, se limitan al análisis del desempeño del modelo, sin considerar el comportamiento computacional del dispositivo de borde. Por otro lado, en esta investigación no se propone un nuevo modelo de detección, sino que se integración componentes de un sistema basado en visión computacional utilizando métricas EAR y MAR. Además, en comparación a otras investigaciones, se analizan métricas como el uso de CPU, memoria RAM, temperatura y tiempo de procesamiento por fotograma.

III. Implementación del Sistema

El sistema implementado este compuesto por el dispositivo de computación de borde basado en una computadora de placa única Raspberry PI versión 4. En este dispositivo se integran herramientas de software basadas de software utilizando Python, OpenCV y cvzone para el reconocimiento de cierre de ojos prolongado (Usando la métrica del EAR) y bostezos (usando la métrica del MAR). Además se integra una aplicación de monitoreo de local basado en Node-Red y comunicación con un bróker MQTT para envío de información hacia un dispositivo ESP32.

A. Dispositivo de Borde Raspberry PI

La Fig. 1 muestra la arquitectura del sistema con el Raspberry Pi 4 representando la conexión con la cámara web y pantalla táctil. Además se incluyen bloques que muestran los procesos de detección en Python para la detección de ojos cerrados mediante el cálculo del EAR, la identificación de episodios de bostezo mediante el MAR, y el módulo de grabación automática de video ante la ocurrencia de eventos de somnolencia.

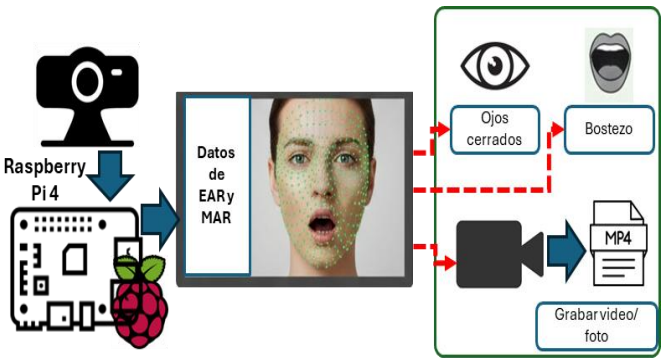


Fig. 1. Diseño de la integración de componentes del módulo Raspberry PI

El dispositivo fue colocado es una estructura de protección para su facilitar su movilidad por lo que en la Fig. 2 se muestra el diseño físico del módulo Raspberry Pi 4 y sus componentes dentro de una caja de protección IP65. En el centro se representa la ubicación del Raspberry Pi 4 fijado sobre la base interna de la caja, mientras que en la parte lateral derecha se señala la posición de la cámara USB. Además, en el mismo lado lateral se muestran las salidas de los cables HDMI y USB que alimentan y transmiten la señal hacia la pantalla táctil para la visualización del resultado de detección.

Para la implementación del prototipo se usa el sistema operativo Debian GNU/Linux versión 12 (bookworm), compatible con las librerías de Mediapipe y OpenCV. Un resumen de las características del sistema operativo compatible se observa en la Tabla I.

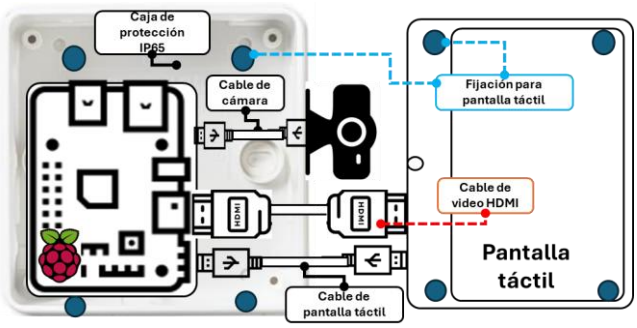


Fig. 2. Disposición de componentes en la estructura de protección IP65

TABLA I
PARÁMETROS DEL RASPBERRY PI 4

Parámetro	Valor Obtenido	Fuente (Comando)
Modelo del Dispositivo	Raspberry Pi 4 Model	cat /proc/device-tree/model
Nombre del SO	Debian GNU/Linux	cat /etc/os-release
ID de Versión del SO	12	cat /etc/os-release
Versión del SO	12 (bookworm)	cat /etc/os-release
Distribuidor (lsb_release)	Debian	lsb_release -a

Para la instalación de librerías de detección sobre el estado de somnolencia se crea un entorno virtual en el sistema operativo, el cual toma como base el directorio del Escritorio (~/Desktop). El entorno virtual de Python se denomina projects-env, y es activado para instalar la biblioteca MediaPipe con el comando pip, agregando las librerías OpenCV, NumPy y Matplotlib (Tabla II).

Se validó que la versión compatible para el programa de detección de somnolencia es MediaPipe 0.10.18, mientras que para el procesamiento de imágenes y la visión por computadora se utilizó opencv-python versión 4.12.0.88, que se complementa con el paquete opencv-contrib-python. Además se utiliza la versión cvzone (versión 1.6.1), para simplificar de OpenCV y MediaPipe. Finalmente, la biblioteca paho-mqtt (versión 2.1.0), permite la comunicación a través del protocolo MQTT para el envío de alertas de somnolencia.

TABLA II
COMANDOS UTILIZADOS

Comando	Comentario
cd ~/Desktop	Navega al directorio del Escritorio para empezar la organización del proyecto.
mkdir projects	Crea la carpeta principal del proyecto, llamada projects.
cd projects	Entra al directorio projects donde se creará el entorno virtual.
python3 -m venv projects-env	Crea el entorno virtual de Python llamado projects-env.
ls -l	Verifica la creación exitosa del directorio del entorno virtual.
source projects-env/bin/activate	Activa el entorno virtual. Las instalaciones posteriores se realizarán solo dentro de este ambiente aislado.
pip install mediapipe	Instala la biblioteca MediaPipe y todas sus dependencias necesarias (OpenCV, NumPy, etc.) dentro del entorno activo.

B. Diseño del dispositivo ESP32

El sistema de detección también considera el uso de un dispositivo embebido en base al microcontrolador ESP32, que activa señales físicas como un zumbador, luces LED y una pantalla OLED (para mostrar el tipo de evento detectado). Este dispositivo se conecta de manera inalámbrica con un bróker MQTT, para intercambio de mensajes con el Rapsberry PI procesando los mensajes recibidos para activar acciones en tiempo real (Fig. 3).

El algoritmo del módulo ESP32 integra bibliotecas de comunicación PubSubClient para suscripción MQTT a los tópicos "sensor" y "data", que se procesan en un hilo principal independiente para actualizar la información en una pantalla OLED. Este algoritmo genera una respuesta inmediata y visible en el entorno de la persona monitoreada, incluso en ausencia de una interfaz gráfica (Fig. 4).

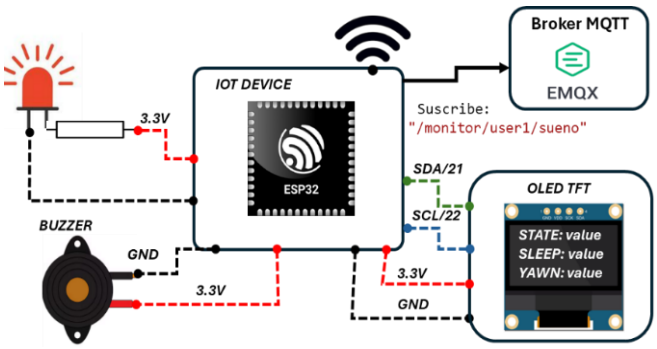


Fig. 3. Esquema ESP32 del dispositivo integrado basado en el microcontrolador ESP32

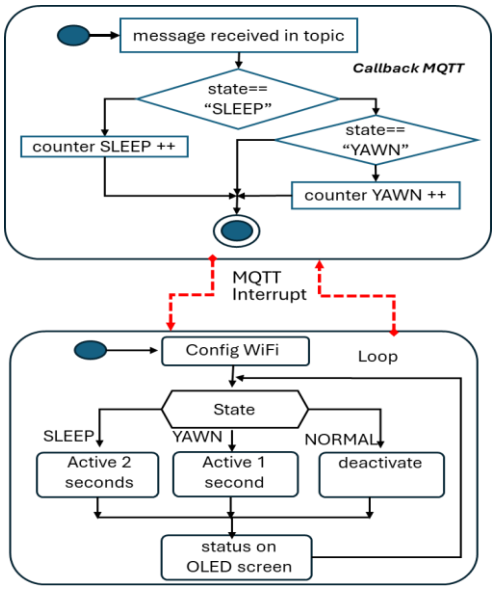


Fig. 4. Diagrama de flujo del firmware del ESP32

C. Configuración del bróker MQTT para monitoreo remoto

La Tabla III muestra los parámetros de configuración del broker MQTT para establecer el intercambio de información con el dispositivo ESP32. En este caso se define el tópico de

publicación y suscripción, el tipo de mensajes transmitidos y la estructura en texto plano utilizada para la recepción (/monitor/user1/sueno). Además, se especifican los parámetros de conexión, la dirección del broker, el puerto de comunicación y el protocolo de transporte.

TABLA III
PARÁMETROS DE CONFIGURACIÓN

Parámetro	Descripción	Valor en el sistema
Protocolo	Protocolo de mensajería ligero usado para comunicación IoT	MQTT v3.1.1
Broker	Servidor encargado de recibir y distribuir los mensajes	broker.emqx.io
Puerto	Puerto estándar para conexión no segura	1883
Cliente MQTT	ID generado por el ESP32 para conectarse	"clientId-[random]"
Tópico suscrito (entrada)	Tópico mediante el cual el ESP32 recibe el estado enviado por la Raspberry Pi	/monitor/user1/sueno
Formato de datos	Estructura de los datos enviados desde la Raspberry al ESP32	Texto plano
QoS (por defecto en PubSubClient)	Nivel de calidad del mensaje	0 (at most once)
Reconexión automática	Intento automático de reconectar cada 5s si se pierde la conexión	Sí
Callback MQTT	Función que procesa mensajes entrantes	callback()

En la Fig. 5 se muestra el diagrama de funcionamiento que describe el proceso para la comunicación con el Broker MQTT. Para ello se verifica continuamente el estado de la conexión, y luego el ESP32 inicializa el cliente MQTT. Después se realiza la suscripción al tópico de recepción de estado de somnolencia. Este flujo permite que el ESP32 active la alarma sonora y actualice la pantalla OLED.

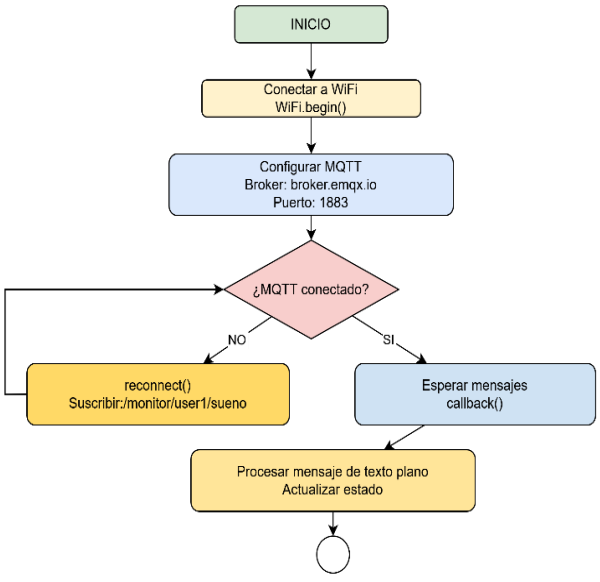


Fig. 5. Funcionamiento de la etapa de comunicación MQTT

D. Implementación del dispositivo de borde Raspberry PI

El módulo de detección Raspberry fue integrado con componentes externos de visualización para validar el funcionamiento del programa de detección en Python. Además, en el desempeño del sistema no se evaluó el impacto de diferentes condiciones de iluminación, lo cual constituye una limitación del sistema. En la Fig. 6 se observa la ejecución del programa y la visualización de datos en la interfaz de Node-Red (Fig. 6(a) y Fig. 6(b)) instalada en el mismo dispositivo. Los valores de detección de sueño y bostezo determinaron que se tuvo que ajustar los umbrales de detección de estado normal (Fig. 6(c)) y bostezo (Fig. 6 (d)) que se ven afectada por la resolución de la cámara.

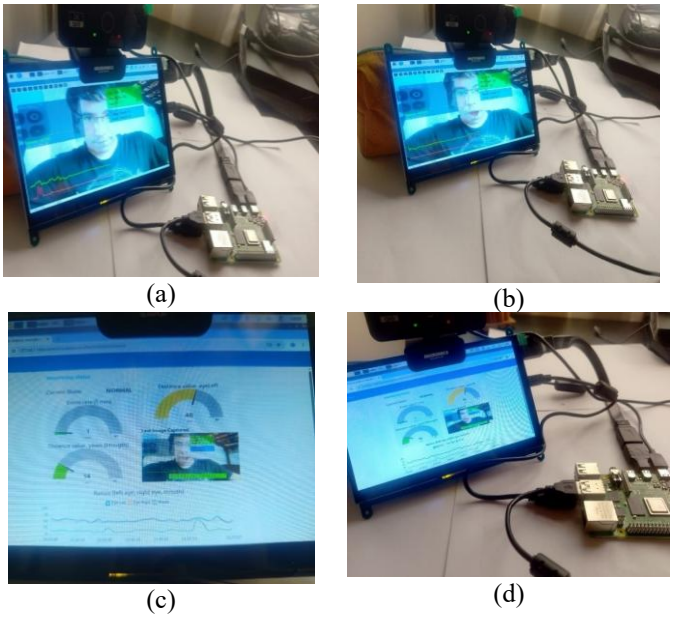


Fig. 6. Prueba de detección en (a) estado normal, (b) bostezo, (c) visualización en Node-RED para bostezo y (d) estado normal

Para la detección del estado de somnolencia se realiza la ejecución del script en Python desde una terminal, considerando que todos los componentes ya estaban integrados en una estructura IP65, fijando la pantalla táctil, la cámara y el módulo Raspberry Pi 4 (Fig. 7).

E. Integración de componentes del dispositivo ESP32

En la Fig. 8 se muestra la conexión electrónica del dispositivo de alerta basado en el ESP32, con los componentes buzzer, la pantalla OLED I2C, el LED indicador y componentes de alimentación y señal. En este caso, las pruebas de medición de voltajes confirmaron la energización eléctrica adecuada del sistema.

La Fig. 9 muestra la integración final de los componentes electrónicos dentro de una estructura de protección IP65 (Fig. 9 (a)), donde el ESP32, el buzzer, el LED y la pantalla OLED quedan fijados en la estructura. Además el funcionamiento del prototipo se verifica con los resultados mostrados en la pantalla OLED visualizando la cantidad de eventos detectados

(Bostezo o cierre de ojos prolongado) (Fig. 9(b)) complementándose con la alarma sonora del buzzer.

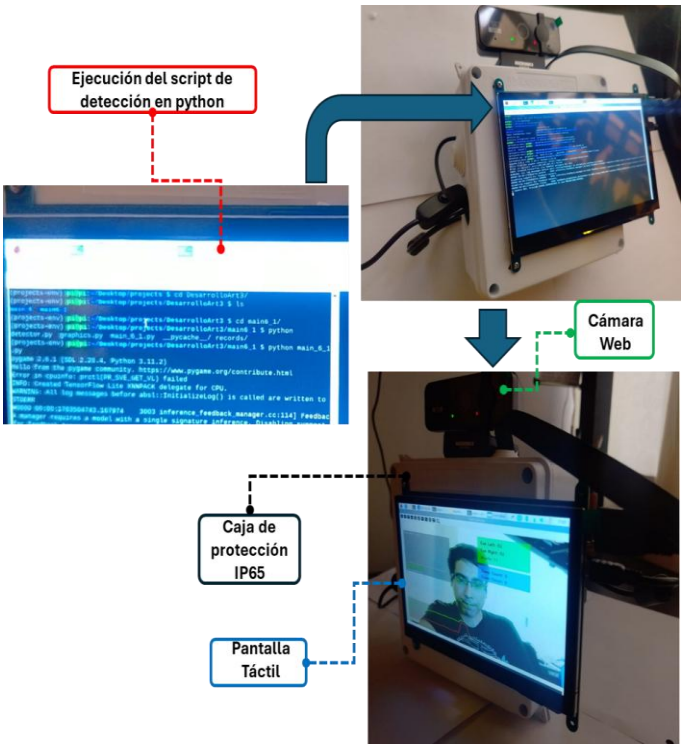


Fig. 7. Dispositivo de alerta Raspberry PI integrado

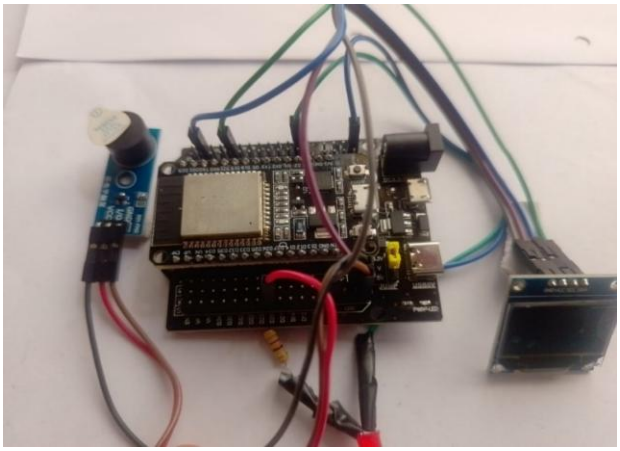
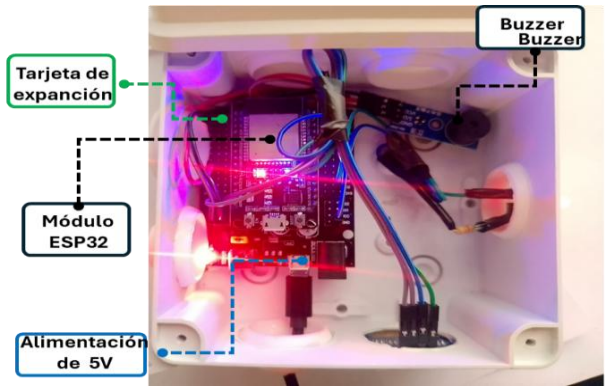


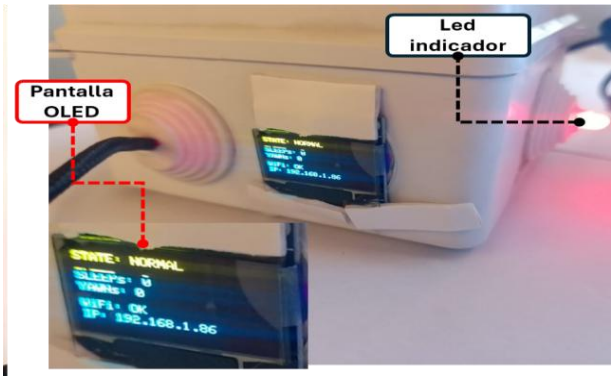
Fig. 8. Prototipo del dispositivo de alerta basado en ESP32

F. NodeRed y comunicación con Telegram

Esta sección valida los mecanismos de comunicación y visualización remota utilizando Node-RED instalada en Raspberry Pi y transmisión de información de alerta a una aplicación de Telegram. La Fig. 10 (a) muestra el panel de control, que incluye un indicador de nivel de somnolencia, gráficos de tendencias EAR y MAR, y un registro de eventos con marca de tiempo. También se muestra la visualización de las notificaciones automáticas enviadas por Telegram, incluyendo evidencia de la detección de somnolencia o bostezos (Fig. 10 (b)) y los comandos utilizados para acceder a la información.

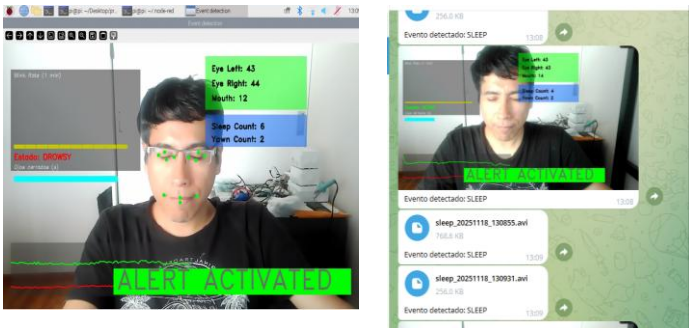


(a)



(b)

Fig. 9. Integración de componentes electrónicos del dispositivo ESP32 (a) y resultados en pantalla Oled (b)



(a) (b)

Fig. 10. Detección de (a) eventos en Raspberry y (b) envío de alerta a Telegram

La aplicación Node-Red fue correctamente instalada en el Raspberry PI 4, considerando la utilización de dos etapas especializadas para la comunicación con el bróker MQTT, almacenamiento de información en dispositivo local. Por otro lado se tiene la configuración de una interfaz gráfica con los

datos de EAR, MAR, imágenes de detección y evolución temporal de los datos (Fig. 11).

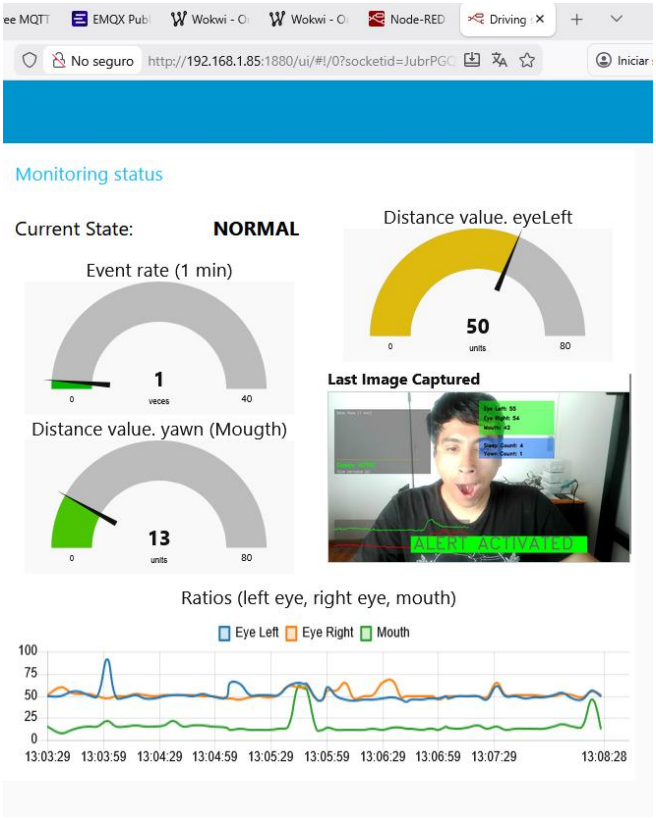


Fig. 11. Visualización de resultados en Node-RED desde Raspberry PI

IV. Resultados y Discusión

Sobre el protocolo de adquisición de video se consideró 4 sujetos de prueba capturando la información por medio del Raspberry Pi, donde se grabaron secuencias de video con una duración promedio de entre 1 y 2 minutos. Además, los usuarios realizaron la simulación de eventos para luego realizar el etiquetado manual, mediante la revisión del video. Las pruebas se realizaron en un entorno interior considerando una distancia con la cámara de 50 cm.

A. Métricas de evaluación para Características de EAR y MAR

Se realizaron procesos de adquisición de datos para los índices de EAR (Eye Aspect Ratio) y MAR (Mouth Aspect Ratio), en secuencias repetitivas de cierre de ojos prolongados y bostezos. La Fig. 12 muestra el comportamiento de ambas métricas en el tiempo, evidenciando un comportamiento diferenciado entre ambas señales que permite discriminar de manera las actividades y permite el cálculo de métricas estadísticas para la evaluación del comportamiento del Raspberry PI.

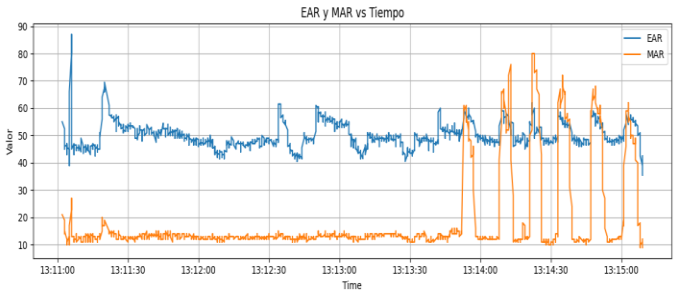


Fig. 12. Evolución de EAR y MAR en el tiempo

La Tabla IV muestra datos estadísticos descriptivos de las métricas del tiempo por fotograma (Frame Processing Time, FPT_ms), el Eye Aspect Ratio (EAR) y el Mouth Aspect Ratio (MAR). En el caso de las columnas se tiene count para la cantidad total de valores válidos; valores mínimos y máximos; mean como promedio general; y std es la variabilidad mediante la desviación estándar. Los resultados muestran que FPT_ms tiene un promedio de 129.08 ms. Por otro lado EAR tiene 1761 y MAR tienen 1761 mediciones. El FPT_ms tiene un rango amplio de valores (36.7–1908.8 ms) y una desviación estándar alta (87.77 ms), debido a los cambios en la carga de procesamiento por variaciones en la detección de eventos complejos. Por otro lado EAR tiene baja variabilidad (std = 4.60), por la estabilidad de la apertura ocular durante periodos sin somnolencia. MAR, sin embargo, muestra una desviación estándar considerable (15.59) y un rango amplio (9–80).

Tabla IV
Resumen de datos de FPT, EAR y MAR

metric	count	min	max	mean	std
FPT_ms	1979	36.718607	1908.819199	129.083409	87.770484
EAR	1761	35.500000	87.000000	50.056502	4.603570
MAR	1761	9.000000	80.000000	18.909710	15.593586

El análisis del FPT, según el estado detectado, muestra diferencias sobre los recursos computacionales por cada actividad fisiológica (Fig. 13). Los estados NORMAL y NO_FACE presentan valores promedio cercanos a 130 ms, lo que indica un comportamiento estable del sistema cuando el rostro está presente sin eventos de interés. Por otro lado, los estados SLEEP y YAWN tienen valores altos de aproximadamente 270 ms y 230 ms. Se infiere que los eventos de cierre prolongado de ojos y de apertura de boca requieren más procesamiento asociado a la activación de procesos de cálculo para el algoritmo de detección de somnolencia.

La comparación de las métricas fisiológicas EAR y MAR entre los estados NORMAL, SLEEP y YAWN muestra algunas características identificables entre los patrones que permiten caracterizar las actividades. La Tabla V muestra que el estado SLEEP presenta los valores más bajos y menos variables de EAR, lo cual es coherente con la disminución de la apertura de los ojos durante el cierre prolongado de ojos. Por otro lado, el estado YAWN tiene valores más altos de

MAR (media = 57.50; máx. = 72.0) durante los episodios de bostezo. Estas relaciones se observan en el gráfico radial de la Fig. 14, observando que el estado SLEEP se concentra en un área reducida indicando pocas variaciones mientras que YAWN se amplía en componentes de MAR y NORMAL

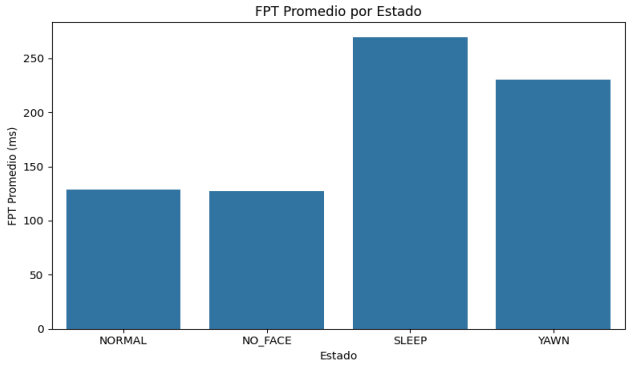


Fig. 13. FTP promedio por estado

Tabla V
MÉTRICAS OBTENIDAS POR CADA ESTADO DETECTADO

STATE	EAR mean	EAR min	EAR max	EAR std	MAR mean	MAR min	MAR max	MAR std
NORMAL	50.06	35.5	87.0	4.59	18.801	9.0	80.0	15.47
SLEEP	43.83	41.5	46.5	1.63	11.833	11.0	13.0	0.752
YAWN	54.66	53.0	55.5	0.98	57.500	42.0	72.0	10.94

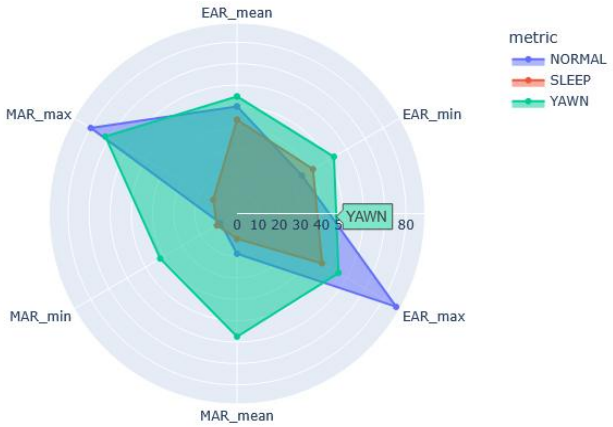


Fig. 14. Gráfico radial por estado y métricas de comportamiento

Para complementar la validación del sistema se realizó una evaluación basada en métricas de clasificación generando datos de referencia mediante el etiquetado manual de eventos. Posteriormente, se compararon las predicciones del sistema basado en umbrales de EAR y MAR con las etiquetas reales. Los resultados indican que se obtuvo una exactitud general de 91.8%, para el estado SLEEP una exactitud de 90.7% y YAWN con 91.5%. Además se evidencia que los errores corresponden a confusiones entre estados NORMAL y SLEEP durante los cambios de estado (Tabla VI).

B. Métricas de evaluación del comportamiento en el dispositivo de borde Raspberry PI

La Tabla VII presenta una muestra de los datos recopilados en el Raspberry PI, los cuales son usados para el análisis del comportamiento del hardware durante la ejecución del programa de detección de somnolencia. En este caso se consideran parámetros como el uso de CPU y RAM, la temperatura del dispositivo, tiempo de procesamiento por fotograma (FrameTime), y los valores de EAR y MAR asociados a cada variable del hardware. De esta manera se puede caracterizar las variaciones de la carga computacional del hardware en diferentes condiciones registradas por el sistema.

Tabla VI
EVALUACIÓN DEL SISTEMA

Clase	Exactitud	Recall
Normal	90%	90%
Sleep	90.7%	89%
Yawn	91.5%	91%

Tabla VII
MUESTRA DE MÉTRICAS OBTENIDAS

Sample	CPU (%)	RAM (%)	Temp (°C)	Date/Time	FrameTime (ms)	EAR	MAR	State
1	14.1	12.9	51.6	01-12-2025 15:27:07	886.3	44.5	13.0	NORMAL
2	46.7	13.1	53.0	01-12-2025 15:27:07	160.9	43.5	14.0	NORMAL
3	67.2	13.2	53.5	01-12-2025 15:27:07	116.26	47.5	16.0	NORMAL

Se usaron 2600 muestras durante la ejecución del programa para caracterizar el comportamiento del hardware en. Los resultados muestran un uso promedio de CPU del 51.25%, con desviación estándar de 10.19%, lo que indica variaciones asociadas a los procesos de cálculo de métricas EAR y MAR. Por otro lado, la memoria RAM es más estable, con promedio de 13.35%. Los valores en la Tabla VIII evidencian que la estabilidad del uso de la memoria RAM con las herramientas como Mediapipe gestionan adecuadamente sus procesos de cálculo sin incrementos en el consumo de memoria.

La señal correspondiente al CPU presenta variaciones pronunciadas, con picos y descensos que reflejan los momentos de mayor carga computacional asociados al procesamiento facial. En contraste, la RAM se mantiene estable durante la ejecución, evidenciando que el sistema no requiere asignaciones dinámicas importantes. La temperatura muestra una tendencia ascendente y luego estable, lo cual es consistente con el calentamiento progresivo del procesador ante cargas permanentes (Fig. 15).

Tabla VIII
ESTADÍSTICAS GENERALES DE CPU, RAM Y TEMP

	count	mean	std	min	25%	50%	75%	max
CPU (%)	2600.0	51.257923	10.193024	14.1	50.0	54.3	57.4	72.7
RAM (%)	2600.0	13.357462	0.107322	12.9	13.3	13.3	13.4	13.7
Temp (°C)	2600.0	59.408731	1.714462	51.6	58.4	59.9	60.8	62.8

La Tabla IX muestra las estadísticas promedio de uso de CPU, RAM y temperatura para cada uno de los estados (NORMAL, NO_FACE, SLEEP y YAWN), permitiendo caracterizar cómo varía la carga computacional en función del tipo de actividad registrada. Se observa que el estado NORMAL presenta el mayor uso promedio de CPU (52.85%). El estado NO_FACE, muestra el menor uso computacional (36.09%), debido a que el sistema no ejecuta los módulos de cálculo cuando no se detectan puntos faciales. El comportamiento del hardware muestra que el mayor uso del CPU en el estado NORMAL se explica por la operación continua de detección y seguimiento, que procesa cada fotograma en tiempo real. Los estados SLEEP y YAWN requieren un esfuerzo computacional adicional, pero menor al estado NORMAL, ya que el procesamiento se centra en identificar patrones.

La representación radial permite visualizar patrones que no son evidentes únicamente en la Tabla IX. En la Fig. 16 se aprecia cómo el estado NORMAL tiene la mayor superficie en el eje del CPU, verificando la carga computacional más alta. Por otro lado, el estado NO_FACE presenta la menor extensión en este eje, debido a la reducción de procesamiento cuando no se identifica un rostro. Además, la memoria RAM, muestra valores estables en todos los estados, indicando que el consumo de memoria es independiente del evento detectado.

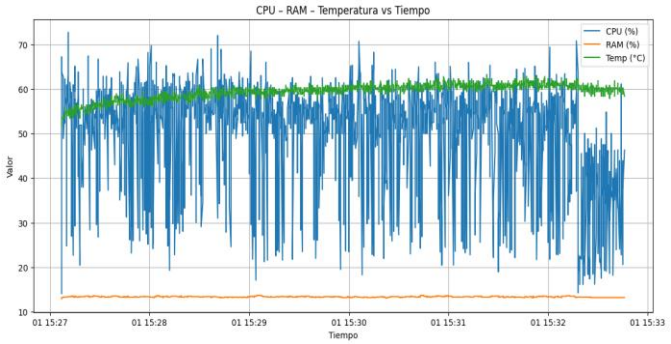


Fig. 15. Variación de métricas del Raspberry PI en el tiempo

Tabla IX
ESTADÍSTICAS POR ESTADO

state	CPU (%)	RAM (%)	Temp (°C)
NORMAL	52.849552	13.370789	59.351258
NO_FACE	36.094142	13.228033	59.985774
SLEEP	44.060000	13.320000	58.360000
YAWN	45.216667	13.366667	60.633333

IV. Conclusiones

Los resultados obtenidos indican que se cumplió con el objetivo de la implementación y evaluación de un sistema de detección de somnolencia basado en visión computacional, integrado en un dispositivo de borde Raspberry PI 4 junto con un módulo de alerta independiente basado en el microcontrolador ESP32 y una plataforma de monitoreo remoto. Además se implementó procesos de comunicación mediante Node-RED y alerta por Telegram. Por otro lado se cumplió con la caracterización del comportamiento de programa de detección de somnolencia y comportamiento del Raspberry PI por el uso de recursos y comportamiento térmico.

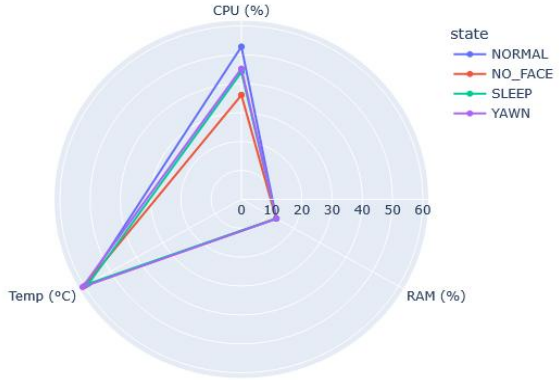


Fig. 16. Gráfico de radar con datos CPU / RAM / TEMP por Estado

El análisis de datos obtenidos demuestra que se puede detectar de manera adecuada los estados de normalidad, somnolencia y bostezo, mostrando consistencia entre las mediciones de EAR y MAR y el comportamiento del tiempo de procesamiento. El análisis estadístico del hardware muestra que la CPU varía de manera significativa según el estado detectado, siendo mayor en estado NORMAL, mientras que la memoria RAM es estable en todos los estados. Además los gráficos radiales permitieron identificar patrones del sistema, confirmando el comportamiento del CPU y memoria RAM del hardware Raspberry PI ante diferentes estados. Por otro lado, durante la evaluación con métricas de clasificación se validó el comportamiento del sistema con una exactitud cerca al 90%.

A diferencia de otras investigaciones centradas en la optimización de modelos, en este trabajo se aporta una caracterización del comportamiento de un sistema embebido en condiciones, demostrando su aplicación real.

El sistema presenta algunas limitaciones relacionadas a la resolución de la cámara que influye en la precisión de los umbrales de detección, por lo que es importante considerar usar cámaras de mayor calidad o técnicas de preprocesamiento en futuros trabajos. Además, el tiempo de procesamiento por fotograma aumenta durante eventos de somnolencia como el bostezo o el cierre prolongado de ojos, lo cual sugiere la necesidad de optimizar el algoritmo o usar modelos de detección más ligeros.

Agradecimiento/Reconocimiento

Un agradecimiento a la Dirección de Investigación de la Universidad Tecnológica del Perú (UTP-Lima Sur), por el apoyo en el proyecto de Investigación financiado 2025, lo cual ha permitido el logro del presente estudio.

REFERENCIAS

[1] R. Vijeikis, I. D. Dada, A. A. Abayomi-Alli, and V. Raudonis, "Enhancing Driver Monitoring Systems Based on Novel Multi-Task Fusion Algorithm," *Sensors*, vol. 25, no. 21, p. 6799, Nov. 2025, doi: 10.3390/s25216799.

[2] O. J. Ouma, D. Omondi, E. Museve, J. Owenga, J. Bogers, S. Abrams, and J. van Olmen, "Assessing the capacity of primary healthcare facilities and healthcare workers in managing diabetes and hypertension in Kisumu county, Kenya," *BMC Health Serv. Res.*, vol. 25, no. 1, p. 1221, Sep. 2025, doi: 10.1186/s12913-025-13390-5.

[3] Y. Horesh, R. Oz Rokach, Y. Kolben, and D. Nachman, "Real-Time Monitoring of Personal Protective Equipment Adherence Using On-Device Artificial Intelligence Models," *Sensors*, vol. 25, no. 7, p. 2003, Mar. 2025, doi: 10.3390/s25072003.

[4] S. Snehalakshmi, R. M. Sridar, B. Devanathan, and S. A. Lakshmanan, "Smart Safety Surveillance System: Personal Protective Equipment and Drowsiness Detection in Industrial Environments," in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, Jun. 2024, pp. 1–6. doi: 10.1109/ICCCNT61001.2024.10724032.

[5] C. Xu, W. Huang, J. Liu, and L. Li, "Detecting Driver Drowsiness Using Hybrid Facial Features and Ensemble Learning," *Information*, vol. 16, no. 4, p. 294, Apr. 2025, doi: 10.3390/info16040294.

[6] A. Debsi, G. Ling, M. Al-Mahbashi, M. Al-Soswa, and A. Abdullah, "Driver distraction and fatigue detection in images using ME-YOLOv8 algorithm," *IET Intell. Transp. Syst.*, vol. 18, no. 10, pp. 1910–1930, Oct. 2024, doi: 10.1049/itr2.12560.

[7] A. Alvarez Oviedo, J. F. Mamani Villanueva, G. A. Echaiz Espinoza, J. M. M. Villanueva, A. O. Salazar, and E. R. L. Villarreal, "Design of a System for Driver Drowsiness Detection and Seat Belt Monitoring Using Raspberry Pi 4 and Arduino Nano," *Designs*, vol. 9, no. 1, p. 11, Jan. 2025, doi: 10.3390/designs9010011.

[8] S. Liawatimena and N. Isworo, "Annotated drowsiness detection dataset captured using Raspberry Pi 5," *Data Br.*, vol. 63, no. 1, p. 112211, Dec. 2025, doi: 10.1016/j.dib.2025.112211.

[9] N. Reza Nazar Zadeh, R. Abais, M. Joie Mananqui, J. P. Cabarles, E. Patricia Estudillo, and M. Donnabell Rivera, "DEVELOPMENT OF ANTI DROWSINESS AND ALERT SYSTEM FOR AUTOMOBILE DRIVERS," *Proc. Eng. Sci.*, vol. 6, no. 4, pp. 1663–1672, Dec. 2024, doi: 10.24874/PES06.04.025.

[10] M. A. Khan, T. Nawaz, U. S. Khan, A. Hamza, and N. Rashid, "IoT-Based Non-Intrusive Automated Driver Drowsiness Monitoring Framework for Logistics and Public Transport Applications to Enhance Road Safety," *IEEE Access*, vol. 11, no. 1, pp. 14385–14397, 2023, doi: 10.1109/ACCESS.2023.3244008.

[11] F. Safarov, F. Akhmedov, A. B. Abdusalomov, R. Nasimov, and Y. I. Cho, "Real-Time Deep Learning-Based Drowsiness Detection: Leveraging Computer-Vision and Eye-Blink Analyses for Enhanced Road Safety," *Sensors*, vol. 23, no. 14, p. 6459, Jul. 2023, doi: 10.3390/s23146459.

[12] S. J. N. Aprilia and D. Fitrianah, "Automatic drowsiness detection system to reduce road accident risks," *Bull. Electr. Eng. Informatics*, vol. 14, no. 4, pp. 2674–2683, Aug. 2025, doi: 10.11591/eei.v14i4.8902.

[13] S. C. Kai Yuen, N. H. B. Zakaria, G. Eg Su, R. Hassan, S. Kasim, and T. Sutikno, "Real-time smart driver sleepiness detection by eye aspect ratio using computer vision," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 34, no. 1, p. 677, Apr. 2024, doi: 10.11591/ijeecs.v34.i1.pp677-686.

[14] V. R. Vijaykumar and K. Vijayagopal, "Performance Evaluation of Wrist Pulse Signals by using Hybrid Artificial Bee Colony with Feed Forward Neural Networks," *Teh. Vjesn. - Tech. Gaz.*, vol. 32, no. 5, pp. 1624–1630, Oct. 2025, doi: 10.17559/TV-20240828001950.

[15] K. Rezaee, A. Nazerian, H. Ghayoumi Zadeh, H. Attar, M. Khosravi, and M. Kanan, "Smart IoT-driven biosensors for EEG-based driving fatigue detection: A CNN-XGBoost model enhancing healthcare quality," *BioImpacts*, vol. 15, no. 1, p. 10, Nov. 2024, doi: 10.34172/bi.30586.

[16] R. Florez, F. Palomino-Quispe, A. B. Alvarez, R. J. Coaquira-Castillo, and J. C. Herrera-Levano, "A Real-Time Embedded System for Driver Drowsiness Detection Based on Visual Analysis of the Eyes and Mouth Using Convolutional Neural Network and Mouth Aspect Ratio," *Sensors*, vol. 24, no. 19, p. 6261, Sep. 2024, doi: 10.3390/s24196261.

[17] S. A. El-Nabi, K. F. Ramadan, E.-S. M. El-Rabaie, A. Emam, and W. El-Shafai, "A real-time design and implementation of intelligent drowsiness and fatigue recognition system for enhancing driver safety," *Eng. Appl. Artif. Intell.*, vol. 162, no. 1, p. 112665, Dec. 2025, doi: 10.1016/j.engappai.2025.112665.

[18] A. Sohail, A. A. Shah, S. Ilyas, and N. Alshammry, "A CNN-based Deep Learning Framework for Driver's Drowsiness Detection," *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 3, pp. 169–178, 2024, doi: 10.14569/IJACSA.2024.0150317.

[19] S. Vidyanathan Dixith, S. Jadhav, Y. Kim, and N. Jayakumar, "Deep Learning-Based Drowsiness Detection System for Driver's Safety," *IEEE Access*, vol. 13, no. 1, pp. 154080–154102, 2025, doi: 10.1109/ACCESS.2025.3604728.

[20] M. J. R. Moredo, J. D. S. Celino, and J. B. G. Ibarra, "Multi-Feature Long Short-Term Memory Facial Recognition for Real-Time Automated Drowsiness Observation of Automobile Drivers with Raspberry Pi 4," in *2024 IEEE 6th Eurasia Conference on IoT, Communication and Engineering*, Basel Switzerland: MDPI, May 2025, p. 52. doi: 10.3390/engproc2025092052.

[21] M. A. Noaman Al-hayanni, M. Sadik Croock, and H. Ali Ahmed, "Intelligent and secure real-time auto-stop car system using deep-learning models," *Int. J. Electr. Comput. Eng. Syst.*, vol. 15, no. 1, pp. 31–39, Jan. 2024, doi: 10.32985/ijeces.15.1.4.

[22] S. Essahraui, I. Lamaakal, Y. Maleh, K. El Makkaoui, M. F. Bouami, I. Ouahbi, H. Elmannai, and A. A. Abd El-Latif, "FastKAN-DDD: A novel fast Kolmogorov-Arnold network-based approach for driver drowsiness detection optimized for TinyML deployment," *PLoS One*, vol. 20, no. 11, p. e0332577, Nov. 2025, doi: 10.1371/journal.pone.0332577.

[23] N. N. Alajlan and D. M. Ibrahim, "DDD TinyML: A TinyML-Based Driver Drowsiness Detection Model Using Deep Learning," *Sensors*, vol. 23, no. 12, p. 5696, Jun. 2023, doi: 10.3390/s23125696.

[24] U. P. Lakshmi, P. V. S. Srinivas, S. Shyam, M. R. Muchintala, V. R. Palugulla, and H. Y. Mandra, "Driver Drowsiness Detection using Hybrid Algorithm," *Indones. J. Electr. Eng. Informatics*, vol. 12, no. 1, pp. 171–178, Mar. 2024, doi: 10.52549/ijeei.v12i1.4738.

[25] M. M. Abo-Zahhad, S. Elghamrawy, A. A. Hefny, and Y. H. Elawady, "Early drowsiness detection model in autonomous vehicles using GAN and YOLO integration," *Neural Comput. Appl.*, vol. 37, no. 33, pp. 28353–28378, Nov. 2025, doi: 10.1007/s00521-025-11676-7.

[26] P. V. Kumar, A. Ali, A. Z. Sha, and S. Rajesh, "IoT based Intelligent Systems for Vehicle," in *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, IEEE, Jan. 2024, pp. 138–143. doi: 10.1109/IDCIoT59759.2024.10467841.

[27] Y. Wang, Y. Sun, Z. Lu, F. Liu, and W. Wang, "Research on Design of Intelligent Wearable Warning System for Work at Heights," in *2024 5th International Conference on Intelligent Design (ICID)*, IEEE, Oct. 2024, pp. 380–387. doi: 10.1109/ICID64166.2024.11025098.

[28] J.-F. Yeh, K.-M. Lin, C.-C. Chang, and T.-H. Wang, "Expression Recognition of Multiple Faces Using a Convolution Neural Network Combining the Haar Cascade Classifier," *Appl. Sci.*, vol. 13, no. 23, p. 12737, Nov. 2023, doi: 10.3390/app132312737.

[29] G. Mishra, S. Bajpai, D. Saini, R. Jain, D. K. Jain, and V. Štruc, "EmoVisioNet: A hybrid network unifying lightweight CNN and attention-based vision model for facial emotion detection," *Neurocomputing*, vol. 665, no. 1, p. 132224, Feb. 2026, doi: 10.1016/j.neucom.2025.132224.

[30] J. E. Delgado Gómez, E. M. España, C. F. R. Rodas, and D. E. G. Villamarín, "Proposal for a Eye Blink Detection Using Mediapipe, Eye Aspect Ratio and Peak Identification," in *Biosystems and Biorobotics*, vol. 32, Springer Science and Business Media Deutschland GmbH, 2024, pp. 345–349. doi: 10.1007/978-3-031-77584-0_67.

[31] S. S. A. Challapalli, "Facial Feature Recognition Model for the Sleepiness Detection of the Drivers," *Int. J. Marit. Eng.*, vol. 167, no. A2(S), p. 10, Aug. 2025, doi: 10.5750/ijme.v167iA2(S).1633.

[32] Z. Xie and H. Mi, "Point-yolo: A fatigue driving detection method for

edge detection devices," in *Proceedings of the 2025 6th International Conference on Computer Information and Big Data Applications*, New York, NY, USA: ACM, Mar. 2025, pp. 111–116. doi: 10.1145/3746709.3746730.

[33] Z. AlArnaout, C. Zaki, Y. Kotb, M. AlAkkoumi, and N. Mostafa, "Exploiting heart rate variability for driver drowsiness detection using wearable sensors and machine learning," *Sci. Rep.*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-08582-2.

[34] S. Chowdhury, A. Munis Alanazi, and E. Talal Attar, "Caffeine on the mind: EEG and cardiovascular signatures of cortical arousal revealed by wearable sensors and machine learning—a pilot study on a male group," *Front. Syst. Neurosci.*, vol. 19, no. 1, p. 10, Sep. 2025, doi: 10.3389/fnsys.2025.1611293.