UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE ELECTROTECNIA Y COMPUTACIÓN

TRABAJO MONOGRÁFICO PARA OPTAR AL TÍTULO DE INGENIERO ELECTRÓNICO

“PROTOTIPO DE UN SISTEMA DETECTOR DE SOMNOLENCIA EN TIEMPO REAL BASADO EN VISION POR COMPUTADOR PARA MONITOREO DE PATRONES DE COMPORTAMIENTO EN CONDUCTORES VEHICULARES”

AUTORES

Br. María Alejandra Espinoza Solórzano 2017-1331U

Br. Ulises Adolfo Rizo Ugarte 2017-0707U

TUTOR

MSc. Cedrick Elksnherr Dalla Torre Parrales

Managua, Nicaragua

Mayo 2023

DEDICATORIA

“Querido Dios, gracias por darme la vida, la salud y la fuerza por permitirme llegar hasta este momento especial en mi vida, gracias por bendecirme con dos ángeles en mi vida: Mi abuela Reyneri Ortega y mi mamá Brenda Solórzano. Con todo mi cariño dedico este trabajo monográfico a ellas dos quienes siempre han sido mi motor, mi apoyo incondicional y siempre me motivan a ser mejor persona, me han brindado cuido, amor y guía. Las amo con todo mi corazón y estoy inmensamente agradecida por su amor y bondad”.

María Alejandra Espinoza Solórzano

Dedico este trabajo a mis padres: Dalia Zelaya y Ulises Rizo, les agradezco por todo el esfuerzo y el apoyo para culminar con toda mi educación a lo largo de mi vida. A mi abuela: María Amanda Zelaya que fue la que confió en mí y deposito todo su apoyo para iniciar y culminar esta carrera, le dedico este éxito. A mis hermanos: Deyver Rizo y Isabela Rizo, por motivarme a alcanzar mis sueños y metas. A mamá Rey por haberme brindado otro hogar en el cual pude sentirme en casa y fue mi lugar de trabajo a lo largo de este periodo de estudio. Y le dedico este trabajo a mis amigos: Alejandra Espinoza, Gerald Aburto, Jehú Guerrero, Yusara Castillo, lo cuales me apoyaron con los recursos necesarios para poder culminar este proyecto.

Ulises Adolfo Rizo Ugarte

AGRADECIMIENTOS

Queremos comenzar agradeciendo a Dios por permitirnos llegar hasta aquí y por guiarnos en cada paso del camino. También queremos extender nuestro agradecimiento a nuestras familias, quienes han sido un pilar fundamental en nuestra formación y desarrollo personal, y cuyo apoyo incondicional nos ha impulsado a superar cada obstáculo.

Asimismo, deseamos expresar nuestro profundo agradecimiento a nuestro tutor MSc. Cedrick Elksnherr Dalla Torre Parrales, por su constante asesoría, seguimiento y valiosos consejos, los cuales fueron cruciales para la culminación de este trabajo monográfico.

A nuestros amigos y compañeros de clases, les agradecemos de corazón por compartir grandes experiencias y dificultades a lo largo de la carrera, y por brindarnos su incondicional apoyo. Juntos hemos sido capaces de superar los retos que se nos han presentado y les tenemos un gran cariño.

Por último, queremos agradecer a todas las personas que han contribuido de alguna manera en nuestro proyecto. Su apoyo ha sido fundamental para alcanzar nuestros objetivos y estamos profundamente agradecidos por su colaboración.

RESUMEN

La detección de somnolencia en vehículos es una tecnología que tiene como objetivo la prevención de accidentes que puede ser causado por diversos factores que conllevan a la fatiga del conductor disminuyendo las habilidades y el rendimiento a la hora de realizar un trabajo que puede atentar contra su vida y los que le rodean.

El presente trabajo monográfico tiene como objetivo el desarrollo de un prototipo de sistema de detección de somnolencia en tiempo real. El enfoque se centra en la detección del rostro del conductor y posteriormente se analiza el comportamiento de los parpados mediante técnicas de visión por computadora para detectar microsueños. Para ello, se emplea un algoritmo que permite analizar los parpadeos del conductor utilizando la relación de aspecto del ojo (EAR) y la duración de los parpadeos. Esto con el fin de emitir una alerta visual y sonora para prevenir situaciones de somnolencia al volante.

Entre las librerías que se implementan en el programa se destacan Dlib y OpenCV en Python, las cuales nos permiten realizar la detección del rostro, predecir los puntos de referencia sobre el ojo y procesar el video de entrada. Se realizaron pruebas en 20 voluntarios, exponiendo el sistema a 3 subrutinas con el objetivo de evaluar la eficacia del prototipo en condiciones de somnolencia diurna.

INDICE

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

I. INTRODUCCIÓN

Los accidentes de tránsito son un problema recurrente en todo el mundo, cuya magnitud aumenta cuando involucran la pérdida de vidas humanas. Es necesario elevar la conciencia sobre este problema y estudiar formas de evitar y reducir estos eventos.

Factores como la fatiga, el consumo de alcohol y ciertos fármacos pueden llevar a la somnolencia, por lo que se consideran factores de riesgos para la conducción segura. En particular, conducir durante trayectos extensos y monótonos como autopistas puede requerir un esfuerzo mental para mantener la atención, cuando el conductor de un vehículo entra en estado de somnolencia tiende a dormirse mientras está conduciendo, siendo perjudicial para su entorno y el mismo.

El conductor que entra en estado de somnolencia durante la conducción puede experimentar microsueños, lo que aumenta significativamente la probabilidad de sufrir un accidente de tránsito. En este contexto, el desarrollo tecnológico ha cambiado nuestras vidas y ofrece herramientas avanzadas, como computadoras y software, que pueden mejorar la calidad de vida de las personas.

Los sistemas de visión por computadora suelen utilizar software de inteligencia artificial, que utilizan algoritmos matemáticos para interpretar datos y tomar decisiones en consecuencia. Estas herramientas buscan mejorar la seguridad de los conductores durante sus viajes.

Uno de los indicadores de somnolencia es el comportamiento ocular, que puede cuantificarse utilizando diversas variables, como la frecuencia, duración y velocidad del parpadeo. En este prototipo, se presentará la implementación de un sistema que consta de una cámara que enfoca el rostro del conductor y un ordenador (Raspberry Pi) que utiliza un algoritmo para procesar imágenes y detectar la somnolencia a partir de los microsueños del conductor. El objetivo es brindar una advertencia visual a través de una interfaz gráfica en la pantalla y, a su vez, emitir una alarma sonora, con el fin de prevenir los accidentes de tránsito.

La estructura de la tesis monográfica consta de nueve apartados. En el primer, segundo y tercer capítulo se aborda la introducción, los antecedentes investigativos y la justificación, respectivamente. En el cuarto capítulo se establecen los objetivos a seguir para la elaboración del sistema. El quinto capítulo se enfoca en el marco teórico, donde se incluyen los fundamentos teóricos, los algoritmos a emplear y la tecnología a implementar. En el sexto capítulo se detallará el diseño metodológico donde se incluye la fase de diseño, implementación, experimentación y evaluación del sistema, para luego, en el séptimo capítulo analizar y presentar los resultados obtenidos. En el octavo capítulo se presentarán las conclusiones y recomendaciones, y finalmente, en el noveno capítulo se detallará la bibliografía utilizada como base teórica para el desarrollo del sistema.

II. ANTECEDENTES

Con el objetivo de complementar la información relacionada con la propuesta del trabajo monográfico, se realizó una investigación a nivel internacional. Se encontraron diversas tesis y artículos que hacen énfasis en el desarrollo de sistemas de detección de somnolencia. Cabe destacar que también se buscó documentación a nivel nacional, sin embargo, no se encontraron trabajos relacionados con el tema.

Arun Sahayasahas, Kenneth Sundaraj y Murugappan Murugappan desarrollan un escrito titulado ***"Detecting Driver Drowsiness Base don Sensors: A review"*** [1] en el cual destacan que en los últimos años se están realizando experimentos para validar sistemas no intrusivos a diferencia de sistemas que requieren de la utilización de electrodos en la cabeza del usuario, entre ellas se destacan las medidas de comportamiento que incluyen la apertura y parpadeo de los ojos, boca y posición de la cabeza e incluso sistemas híbridos donde se incluye análisis facial, SDLP (Standard deviation of lane position) para corrección de desvíos vehiculares, ECG y EEG (Electrocardiogramas y electroencefalogramas respectivamente).

Marcos Javier Flores Calero en su tesis doctoral titulada ***"Sistema avanzado de asistencia a la conducción mediante visión por computador para la detección de la somnolencia"*** [2] desarrolla un sistema que detecta automáticamente la somnolencia y la distracción del conductor basado en información visual y técnicas de inteligencia artificial, su trabajo se dividió en dos partes donde analizó la condición de la conducción diurna (espectro visible) y nocturna (infrarrojo cercano), de esta manera permite analizar al conductor en situaciones reales de conducción sobre un vehículo real. De igual forma, otra aportación importante es la versatilidad donde en el diseño se analizan factores subjetivos como el color de ojos o la piel, dándole más robustez al sistema ante diversos escenarios.

Otro trabajo relacionado con la temática fue desarrollado Marcillo Plúa Francisco Guillermo ***"Prototipo de un sistema detector de somnolencia con alerta vía tuits para conductores vehiculares"*** [3] donde emplea el uso de una cámara infrarroja y lenguaje de programación Python donde la detección de los ojos en un ambiente normal trabajó a un 99% y la detección del factor EAR funcionó a un 98% mientras que para ambientes con poca luz el sistema trabajó a 80% y su factor EAR a un 98%, del mismo modo se realizaron pruebas con rostros de personas con contornos de ojos pequeños, donde el sistema trabajó un 60% y su factor EAR en un 60%. Finalmente, al detectar somnolencia alerta al usuario vía Twitter donde se paga alrededor de $0.20 centavos de dólar por cada mensaje emitido en dependencia de la disponibilidad de datos que el usuario tenga en su móvil.

Por otro lado, Tereza Soukupová y Jan Cech desarrollaron el paper ***titulado "Real Time Eye Blink Detection using Facial Landmarks"*** [4] demuestran cuantitativamente que los puntos de referencias faciales basados en regresión son los suficientemente precisos como para estimar de forma fiable el nivel de apertura de los ojos, el método SVM propuesto utiliza una ventana temporal de la relación y aspecto del contorno de los ojos (EAR), igualmente resalta que el algoritmo se ejecuta en tiempo real ya que los costos computacionales adicionales para la detección son insignificantes. También asumió una duración de parpadeo fija para todos los sujetos ya que el parpadeo de todos dura de manera diferente, por último, indica que, si bien EAR se estima a partir de una imagen 2D, es bastante sensible a la orientación de la cabeza por lo que también existe un modelo de detectores de puntos de referencia que estiman pose 3D.

III. JUSTIFICACIÓN

Los accidentes de tránsito forman parte de una situación global, la Organización Mundial de la Salud (OMS) en su Informe sobre el Estado Mundial de la Seguridad Vial estima que 1.35 millones de personas mueren anualmente por esta causa y va en incremento [5]. En el 2021, la Dirección de Seguridad de Tránsito Nacional de Nicaragua contabilizó 45,927 accidentes de tránsito, de las cuales hubo 904 personas fallecidas en colisiones y 3,332 lesionados [6]. Por lo que se estima que en las vías nicaragüenses a diario ocurrieron 125.8 colisiones, 9.6 lesionados y 2.5 fallecidos.

La Policía Nacional de Nicaragua trabaja en coordinación con otras instituciones nacionales, recopilando un anuario estadístico con los accidentes ocurridos (entre otras incidencias). El último registro que se tiene fue de 2021 y no se evidencian accidentes relacionados con la fatiga o somnolencia a la hora de conducir, sin embargo, esto no quiere decir que esta problemática sea inexistente ya que otros medios de comunicación como periódicos o canales de televisión evidencian que este problema existe, pero no se registra estadísticamente.

De las estadísticas antes mencionada, los accidentes de tránsito ocasionados por conductores que se encuentran bajo el efecto del alcohol constituyen una de las principales causas de accidentalidad y mortalidad en el país.

Por lo cual cabe destacar, que la ingesta de alcohol al conducir se ve completamente ligada a la somnolencia del individuo, por ejemplo, en el estudio de Carlos Gómez y acompañantes “**Blood Alcohol Concentration and Somnolence among Drivers Studied in Simulators: A Meta-Analysis**” [7] concluyen “Cualquier nivel de alcohol en sangre se presenta un compromiso importante en el estado de vigilia, pues el efecto de somnolencia causado por el consumo de alcohol afecta la habilidad en la conducción, comparado con niveles de alcoholemia de 0.”

El sistema de detección de somnolencia adaptado para vehículos es una herramienta importante para diversas aplicaciones, incluyendo el transporte de carga, la industria y para personas que trabajan largas jornadas al volante y pueden experimentar fatiga, lo que puede resultar en fases de sueño sin control y graves consecuencias, como la muerte.

Por lo tanto, la implementación de un sistema de detección de somnolencia compacto puede ser una solución viable para abordar la problemática de los accidentes de tránsito relacionados con la fatiga.

Existen diversos métodos para la somnolencia basados en visión por computador, según la conclusión presentada en la investigación [8] los métodos basados en señales fisiológicas son exactos pero caros, incómodos e intrusivos, sin embargo, se propone una alternativa no intrusiva que no causa molestias al conductor.

Este proyecto puede ser de utilidad para futuras investigaciones debido a la importancia que tiene la implementación de sistemas de inteligencia artificial o en este caso, de visión artificial, ya que estos algoritmos pueden brindar innumerables beneficios a todos los sectores productivos, en la salud, manufactura, ciudades inteligentes, etc.

IV. OBJETIVOS

4.1. Objetivo general

Diseñar el prototipo de un sistema detector de somnolencia en tiempo real basado en visión por computador para monitoreo de patrones de comportamiento en conductores vehiculares.

4.2. Objetivos específicos:

- Analizar los fundamentos teóricos relacionados con la somnolencia y la visión por computadora para la elección del método más adecuado en el desarrollo del prototipo.
- Seleccionar los parámetros y algoritmos que permitan la identificación de la condición de somnolencia a partir del comportamiento de los parpadeos de los ojos del conductor.
- Desarrollar una interfaz de usuario para la visualización del sistema y alerta del estado de somnolencia del conductor.
- Calcular la eficacia del prototipo en la detección de somnolencia mediante la evaluación de los resultados obtenidos a partir de pruebas realizadas.

V. MARCO TEÓRICO

En este capítulo se desarrolló una serie de bases conceptuales y teorías como reseña de material bibliográfico en la cual se fundamenta y da sustento al diseño del prototipo de sistema detector de somnolencia en tiempo real basado en visión por computador para monitoreo de patrones de comportamiento en conductores vehiculares.

En el presente de describirán los fundamentos conceptuales de la somnolencia, así como los métodos y parámetros utilizados para la detección de esta, se detallarán generalidades del sistema de visión por computador de las cuales se derivan técnicas, elementos, librerías y algoritmos que nos permitirán realizar un sistema óptimo y sin los cuales sería difícil de modelar.

5.1. Somnolencia

5.1.1. Definición

La somnolencia es un trastorno del sueño, se considera como ***"Pesadez y torpeza de los sentidos motivadas por el sueño"*** [9], denota ganas de dormir, pereza y falta de actividad de algo considerado como una necesidad fisiológica básica. La somnolencia o tendencia para quedarse dormido puede ser el principal síntoma de diversas patologías, la causa más común es la privación de sueño, su medición es compleja debido a sus diferentes conceptos operacionales, sin embargo, se pueden separar en dos tipos [10]:

5.1.2. Tipos de somnolencia

Se han desarrollado dos conceptos importantes para definir los tipos de somnolencia ya que a partir de ellos se crean los diferentes instrumentos para cuantificar la somnolencia:

- **Somnolencia objetiva:** Se refiere a la tendencia de una persona a quedar dormida, también referida como la propensión del sueño y puede medirse a través de instrumentos destinados para eso. Un encefalograma lo puede detectar, así como un oculograma. Son mediciones de la actividad eléctrica de los tejidos cuándo el cuerpo solicita descanso [11].

- **Somnolencia subjetiva:** Es considerada como la percepción subjetiva de la necesidad de dormir o el estado de transición entre la vigilia y el sueño asociado a un número de sensaciones y síntomas subjetivos, los signos (manifestaciones objetivas) de la somnolencia subjetiva incluyen el bostezo, perdida del tono de músculos extensores del cuello, la constricción pupilar, la ptosis (el párpado superior cae sobre el ojo) y la disminución de la atención, desempeño psicomotor y cognitivo [12].

5.1.3. Síntomas y alteraciones de los conductores producidos por la somnolencia

Los principales síntomas que se apoderan del conductor y son producidos por la somnolencia son [13]:

- Picor de los ojos, pesadez y agotamiento
- Dolor de cabeza y sensación de brazos dormidos
- Fatiga
- Bostezo
- Adormecimiento por una elevada temperatura en el vehículo.

Las alteraciones o consecuencias más relevantes ocasionadas por la somnolencia son:

- Incremento del tiempo de reacción
- Menos concentración y más distracción
- Toma de decisiones más lentas y más errores de conducción.
- Movimientos más automatizados
- Aparición de microsueños

- Alteración de las funciones sensoriales
- Alteración en la percepción
- Cambios en el comportamiento

5.1.4. Microsueños

Los microsueños son un breve período de inconsciencia en el que una persona se queda dormida y se desconecta del entorno y puede durar de un segundo a dos minutos [14]. Esta experiencia puede ocurrir en cualquier momento y lugar durante el día, en el trabajo, en el hogar, mientras se conduce o se realizan actividades que requieren atención, por lo que se clasifican como graves. Las noches de insomnio a menudo causan microsueños, pero también se puede experimentar al realizar actividades monótonas y rutinarias durante largos períodos de tiempo. Ciertos medicamentos y/o trastornos del sueño también pueden causar este efecto.

5.1.5. Métodos utilizados para sistemas detectores de somnolencia

Así como en los últimos años se ha observado el incremento de los accidentes de tránsito, también la tecnología ha tenido un avance vertiginoso, por lo que existen numerosos métodos para poder detectar la somnolencia de un conductor, entre ellos se destacan los siguientes métodos [15]:

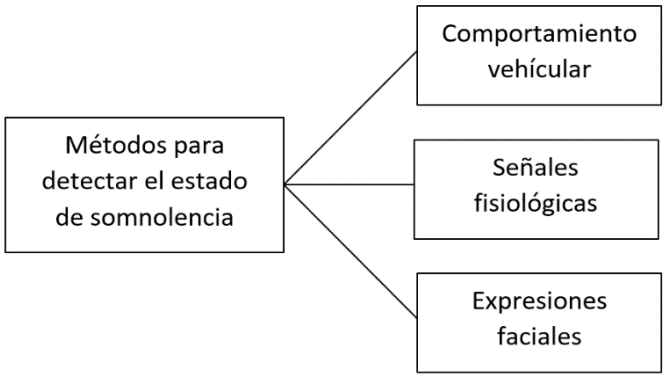


Fig. 1 Métodos para la detección de somnolencia. Elaboración propia.

Comportamiento vehicular

Se toma en cuenta los factores medibles y observables del comportamiento del vehículo con el entorno, unos buenos indicadores son la velocidad de conducción, la distancia lateral con relación a las líneas de señalización, el ángulo de desviación, el tiempo de reacción, el uso de pedales del acelerador y el freno.

La empresa AssitWare Technology utiliza esta técnica en la que se construye un patrón basado en la posición del vehículo sobre la carretera [16] y emplea un sistema de alarma en caso de que se desvíe basado en el estudio de la posición promedio de los vehículos a como se observa en la figura 2:

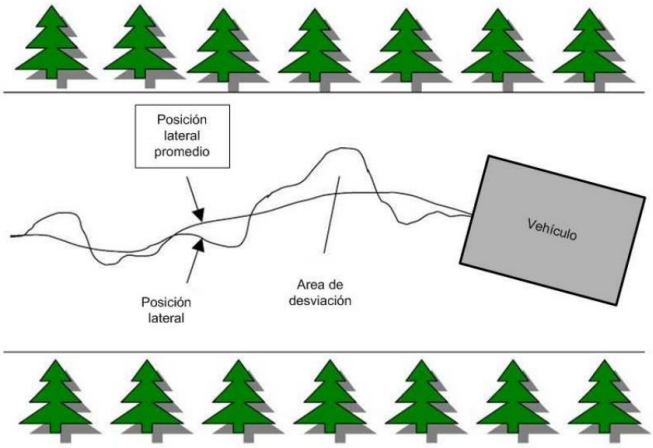


Fig. 2 Patrón de conducción a partir de la posición del vehículo en la carretera [16].

Señales fisiológicas

Se analizan las variaciones fisiológicas para detectar el estado cognitivo del conductor, se estudia el ritmo cardiaco, la respiración y las ondas cerebrales por medio de ondas de electroencefalografía (EEG) figura 3, por lo que estudia la somnolencia subjetiva siendo altamente predictiva y confiable.

Este método tiene múltiples ventajas, sin embargo, es intrusivo al utilizar electrodos sobre la cabeza a como observamos en la fig.3 por lo que provoca molestia e incomodidad e incluso estrés sobre los conductores, [17].

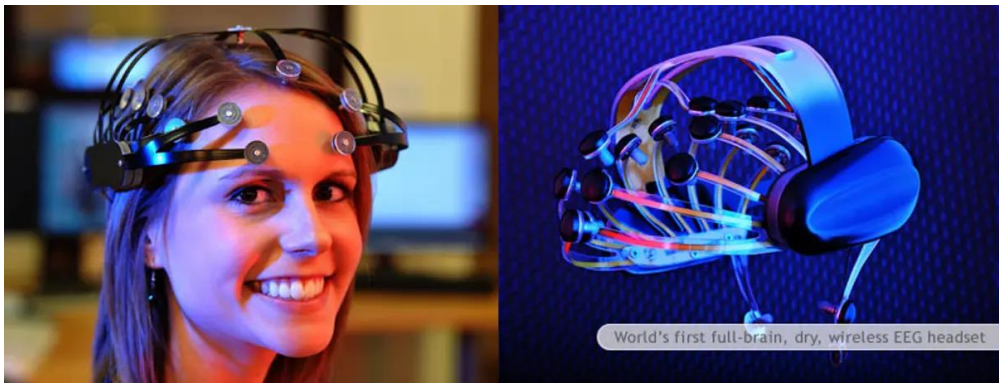


Fig. 3 Aparatos para el estudio EEG [17].

Expresiones faciales

Como tercer método se utilizan técnicas de procesamiento de imágenes para detectar aquellos cambios físicos durante la conducción somnolienta sobre los ojos (Movimiento de los párpados, el promedio de la velocidad del cierre de los ojos) y el rostro del conductor (movimientos de cabeza del conductor), entre otros, ver figura. 4.

Fig. 4 Ejemplo de conducción inteligencia de Quectel [18].

El método tiene alta confiabilidad y no es intrusivo, sin embargo, al utilizar una cámara se tiene el problema de que el modelo no reconoce correctamente al conductor en casos de poca iluminación o cuando utiliza anteojos o gafas de sol por lo que los sistemas deben adecuarse para contrarrestar estas situaciones.

5.1.6. Sistemas de detección de somnolencia de visión por computador implementados a nivel industrial.

Las compañías automovilísticas y agencias que vigilan las carreteras y los derechos y deberes de trabajadores concentran sus esfuerzos en salvar el mayor número de vidas posible bajando al mínimo el número de accidentes que se puedan presentar. Es por esto por lo que se desarrollan técnicas, controles, reglamentaciones laborales, entre otras, para proteger la vida no solo del que conduce sino de los que están en su entorno [19]. Según la tabla 1 podemos observar algunos de los sistemas basados en visión por computador que actualmente son comerciales:

Tabla 1 - Sistemas de detección de somnolencia de visión por computador existentes en el mercado *[8]*

Fabricante	Comportamiento para estudiar	Alarmas
Ford	Movimiento del vehículo en la carretera	Auditivas y visual
Lexus	Ojos del conductor	NoREM 1: Visuales y auditivas. NoREM 2: Frenado del vehículo
Mercedes	Movimiento del vehículo en la carretera	Auditivas y visual
VW	Ojos del conductor	Auditivas y visual
Volvo	Movimiento del vehículo en la carretera	Auditivas y visual

5.2. Generalidades de los sistemas de visión por computador

La visión por computadora es un campo de la inteligencia artificial que se enfoca en el análisis de imágenes y vídeos mediante una variedad de herramientas y métodos que permiten obtener, procesar y analizar imágenes del mundo real. Esta tecnología permite automatizar una amplia gama de tareas al proporcionar a las máquinas la información necesaria para tomar decisiones precisas en cada tarea asignada. Todo cálculo que involucre contenido visual, como imágenes, videos, iconos y cualquier elemento con píxeles, se considera como parte de la visión por computadora (CV), [20].

Una limitación importante de este campo es la falta de formulaciones o métodos universales para resolver problemas, ya que a menudo no se adaptan a situaciones diferentes de aquellas para las cuales fueron originalmente diseñados.

5.2.1. Técnicas de visión por computadora

En la visión por computadora se utilizan varios métodos para evaluar las entradas y obtener las salidas. A continuación, se detallan alguna de las técnicas comúnmente utilizadas por visión por computadora.

5.2.1.1 Clasificación de imágenes

El proceso de identificación de imágenes por parte de una máquina es complejo y requiere entrenamiento con imágenes y modelos de clasificación. Las redes neuronales convolucionales (CNN) son la técnica de aprendizaje profundo más utilizada para este fin, figura 5. Con CNN, las imágenes preetiquetadas se utilizan para crear conjuntos de datos de entrenamiento y se entrena a la red para identificar propiedades independientes de cada clase de imágenes representadas por vectores. El clasificador puede mejorarse con nuevos conjuntos de datos y, si es necesario, agregar más conjuntos de prueba o entrenamiento, [21].

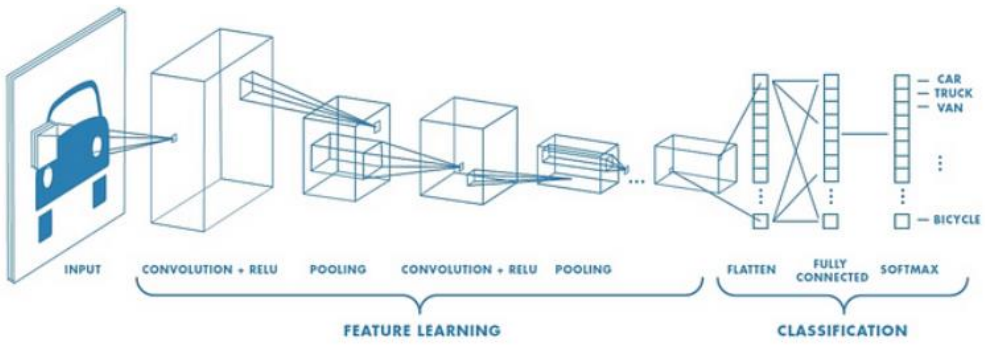


Fig. 5 Descripción general de una CNN *[22]*

5.2.1.2. Detección de objetos

La detección de objetos se basa en el procesamiento de imágenes y el aprendizaje profundo que se ocupa de detectar instancias de objetos en imágenes y videos.

Uno de los métodos más conocidos es Haar Cascade el cual utiliza técnicas de aprendizaje automática. Este método fue propuesto por Paul Viola y Michael Jones en el artículo *"Rapid Object Detection using a Boosted Cascade of Simple Features",* es un enfoque basado en el aprendizaje automático en el que se utilizan muchas imágenes positivas (las que queremos que el clasificador identifique) y negativas (todo lo demás que no contienen el objeto que se desea detectar) para entrenar un clasificador, por ejemplo, un rostro, a como se observa en la figura 6 [23].

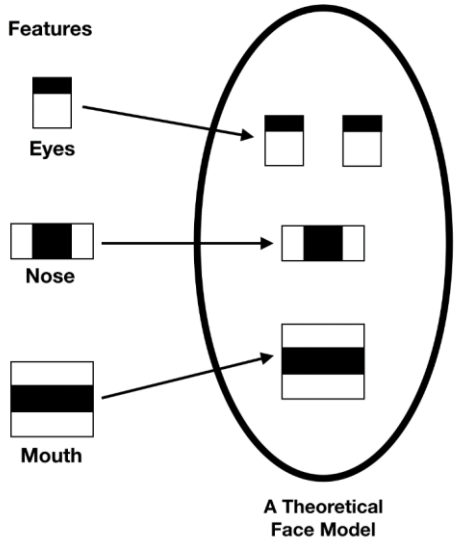


Fig. 6 Haar Cascade datos de entrenamiento para identificar las características que puede considerar una cara [23]

La idea de la detección de objetos es determinar un conjunto de categorías fijas que nos interesan. Siempre que aparezca alguna de estas categorías en la imagen de entrada, se dibujará un cuadro delimitador [24] alrededor de la imagen y se predecirá su etiqueta de clase a como se muestra en la Fig. 7.

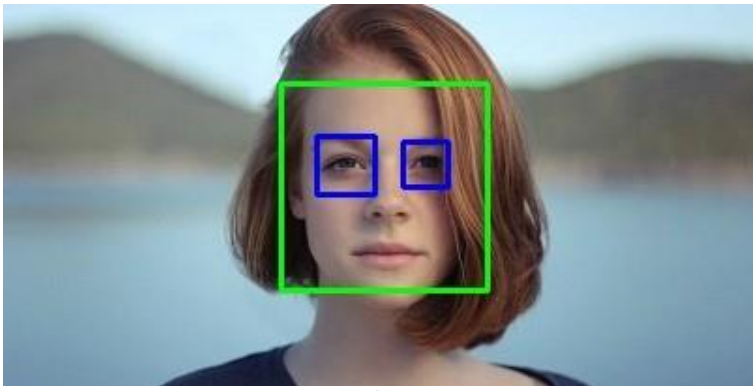


Fig. 7 Detección de rostro [24]

Se tiene en cuenta la baja eficiencia computacional y pocos algoritmos pueden resolver eficazmente este problema como Haar Cascade. Se han desarrollado otros algoritmos como R-CNN, Fast R-CNN, YOLO, Single Shot MultiBox Detector (SSD) y redes totalmente convolucionales basadas en regiones para encontrar rápidamente estas ocurrencias, pero requieren un alto costo computacional [21].

5.1.2.3. Seguimiento de objetos

El seguimiento de objetos es el método que rastrea el movimiento del objeto en una imagen al encontrar el mismo objeto en la siguiente imagen, se pueden dividir en tres categorías según los métodos de observación:

Técnicas generativas: se formula como la búsqueda de las regiones de la imagen que son más similares al modelo objetivo. El análisis de componentes principales (PCA), el análisis de componentes independientes (ICA), la factorización matricial no negativa (NMF) son ejemplos de modelos generativos que intentan encontrar una representación adecuada de los datos originales.

Técnicas discriminatorias: el seguimiento se considera un problema de clasificación binaria, cuyo objetivo es encontrar un límite de decisión que separe mejor al objetivo del fondo. A diferencia de los métodos generativos, tanto la información de fondo como la de destino se utilizan simultáneamente. Ejemplos de métodos discriminativos son los codificadores automáticos apilados (SAE), las redes neuronales convolucionales y las máquinas de vectores de soporte (SVM).

Técnicas híbridas: estas dos técnicas se utilizan de forma conjunta y se adaptan diferentes técnicas según el problema.

5.2.1.4. Segmentación de imagen

El propósito de la segmentación de imágenes es simplificar la representación de una imagen y facilitar el análisis como se observa en la figura 8. Dado que existen muchos enfoques diferentes para la segmentación de imágenes, Mask R-CNN y Fully Convolutional Networks (FCN) pueden usarse para predicciones densas sin capas completamente conectadas [21].

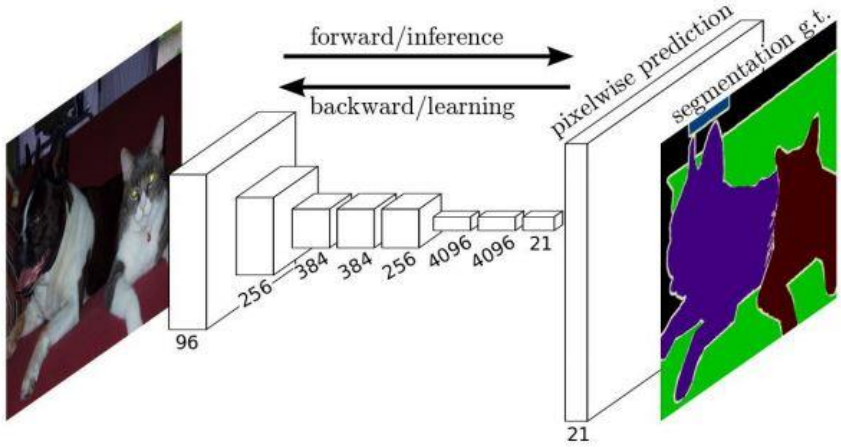


Fig. 8 Segmentación de imagen [21].

5.2.2. Aplicaciones de la visión por computador

La visión por computadora tiene muchas aplicaciones en la actualidad, incluyendo el reconocimiento de objetos, detección de rostros, detección de sucesos, reconstrucción de escenas y restauración de imágenes.

La robótica se ha beneficiado significativamente del uso de la visión por computadora, permitiendo a los robots reconocer objetos y evitar colisiones. En el campo de la medicina, los sistemas de visión por computadora pueden diagnosticar enfermedades de forma automatizada.

Existen otras áreas donde se emplea la visión por computadora, como sistemas de seguridad, seguimiento de objetos, o detección de anomalías en piezas fabricadas en una cadena de producción. En torno a esto, la visión por computadora resulta beneficiosa en cualquier área de aplicación, puesto que, pueden lograrse avances que hasta ahora no se tenían imaginados [25].

5.2.3. Elementos de un sistema de visión por computador

A continuación, se detallan los elementos de un sistema de visión por computador:

5.2.3.1. Sistema de iluminación

Una de las partes fundamentales de un sistema de visión por computador es la correcta elección del sistema de iluminación ya que de esto depende gran parte del éxito de la aplicación, ***"No puede analizarse aquello que no puede verse"*** [26].

Las cámaras capturan la luz reflejada de objetos, obteniendo información a través de la captura lumínica para posteriormente ser analizada. El propósito de la iluminación es controlar la forma en que la cámara ve el objeto, existen diferentes técnicas de iluminación para las aplicaciones de visión artificial, las más comunes son: Luz frontal, luz lateral, iluminación por campo oscuro (Darkfield), iluminación por contraste (Backlight) ejemplo en la figura 9, iluminación por láser, fluorescente, LED, polarizada, entre otros.

Fig. 9 Iluminación por contraste (Backlight), la luz es emitida desde la parte posterior del objeto quedando este entre la fuente de iluminación y la cámara [26].

5.2.3.2. Sensor o cámara de captura de imagen

La adquisición de información es una de las primeras etapas definidas en todos los sistemas, ya que sin datos de entrada no se avanzaría con el procedimiento. Las cámaras son los encargados de la obtención de la información, sin embargo, se tendrá mayor índice de éxito en el reconocimiento en dependencia de la resolución, posicionamiento y estabilidad de la cámara, por lo que necesitan manipular las características de la imagen para obtener mejores resultados [3].

5.2.3.3. Módulo de procesamiento y software

Puede ser una computadora o un sistema integrado, el sistema que recibe, almacena las imágenes y las procesa a través de algoritmos adecuados para extraer la información necesaria y luego tomar decisiones según la aplicación del sistema de visión por computador. Los sistemas integrados son aquellos que incorporan el software y todo el hardware necesario en un mismo sistema, disponen de un procesador integrado con capacidad de tomar decisiones [27].

5.2.3.4. Procesamiento de imagen

Se puede llevar a cabo a través de algoritmos de pretratamiento, filtrado de la imagen, de segmentación, reconocimiento de formas, extracción de descriptores y de clasificación) [28].

5.2.3.5. Comunicación

La comunicación se suele realizar mediante una señal de E/S discreta o información enviada mediante una conexión serial a un dispositivo que registra o usa esta información.

5.2.4. Librerías utilizadas en visión por computadora

5.2.4.1. Librería OpenCV

OpenCV (Open Source Computer Vision Library) está escrito en C/C++, para visión por ordenador en tiempo real. Aprovecha el procesamiento multinúcleo y la aceleración de hardware. Las aplicaciones de OpenCV incluyen estimación de emociones, reconocimiento de gestos, sistema de reconocimiento facial y redes neuronales artificiales [29]. Se publicó bajo la licencia BSD (Berkeley Software Distribution), por lo que se utiliza tanto en proyectos académicos como en productos comerciales.

Diversos factores hacen que la visión artificial sea difícil de manejar a diferencia del sistema de visión humano, es importante adaptar los fotogramas que se obtengan para evitar la pérdida de información.

5.2.4.2. Librería Numpy

El paquete o librería Numpy es conocido porque permite a los desarrolladores de Python realizar en forma rápida y eficiente una amplia variedad de cálculos numéricos. Numpy también está licenciado bajo el formato BSD lo que permite la reutilización con pocas restricciones. Este significa “Python numérico” y es el paquete fundamental para la computación científica con Python, el principal beneficio es que permite una generación y manejo de datos rápidos basado en su propia estructura de datos llamado “arrays” donde opera de manera eficiente [30], este contiene:

- Un poderoso objeto de arreglo N-dimensional
- Funciones sofisticadas
- Herramientas para integrar código en C/C++ y Fortran
- Útiles capacidades de álgebra lineal, transformación de Fourier, números aleatorios entre otros.

5.2.4.3. Librería Dlib

Dlib es una colección de herramientas y algoritmos de aprendizaje automático en C++, diseñados para resolver problemas del mundo real en una variedad de campos, desde la robótica hasta los grandes entornos informáticos. Dlib se utiliza ampliamente en la industria y la academia, incluyendo dispositivos integrados, teléfonos móviles y aplicaciones de alto rendimiento. Además, su licencia de código abierto permite a los usuarios utilizarlo en cualquier aplicación sin costo alguno [31].

5.2.5. Preprocesamiento de imagen

5.2.5.1. Redimensionamiento

El redimensionamiento de una imagen implica ajustar sus dimensiones a través de un proceso de cambio de tamaño, ya sea aumentando o reduciendo sus proporciones, tal como se ilustra en la figura 10.

Consiste en modificar las medidas de una imagen con el propósito de adaptarla a requisitos específicos, de esta manera administramos la capacidad del dispositivo haciendo más eficiente la utilización de otras funciones [32].

Fig. 10 Redimensionamiento de una imagen [13].

5.2.5.2. Conversión de una imagen RGB a escala de grises

Esto se realiza para simplificar la información de la imagen, reducir la carga computacional, eliminar la variabilidad del color y adaptarse a aplicaciones específicas que solo requieren información de intensidad de los píxeles.[33].

5.2.6. Algoritmos de detección y predicción

5.2.6.1. Detector DLIB (HOG)

El detector DLIB HOG es un algoritmo utilizado en la detección de objetos en imágenes. Fue desarrollado por Davis E. King en la biblioteca de software DLIB, una plataforma de aprendizaje automático de código abierto escrita en C++. Se basa en el método de histograma de gradientes orientados (HOG), que identifica patrones y características de los objetos presentes en una imagen a través de la distribución de gradientes de la intensidad de la imagen. HOG es un enfoque popular para la detección de objetos en imágenes. [34].

A detalle, el método HOG (Histogram of Oriented Gradients) es una técnica de extracción de características utilizada en la detección de objetos en imágenes. El método de manera detallada se basa en 5 pasos [35], los cuales son:

1. Cálculo de gradientes: el primer paso en el método HOG es calcular los gradientes de la imagen en escala de grises. Esto se hace mediante el cálculo de la magnitud del gradiente y la orientación del gradiente en cada píxel. Para calcular la magnitud del gradiente, se utilizan las derivadas parciales de la imagen en x e y. La orientación del gradiente se calcula mediante el arco tangente de la derivada en y dividida por la derivada en x.

2. División de la imagen en celdas: una vez que se han calculado los gradientes, se divide la imagen en celdas de tamaño fijo (ejemplo: 8x8 píxeles). La idea detrás de esto es que la distribución de gradientes en pequeñas regiones de la imagen puede proporcionar información valiosa sobre los objetos presentes en ella.

3. Creación de histogramas de orientación: para cada celda, se crea un histograma de orientación. Este histograma cuenta la cantidad de píxeles con una determinada orientación del gradiente. En general, se utilizan 9 bin para el histograma de orientación (es decir, se divide el rango de 0 a 180 grados en 9 intervalos de 20 grados cada uno).

4. Normalización de bloques: los histogramas de orientación se normalizan mediante la normalización de bloques. Los bloques son conjuntos de celdas adyacentes, y la normalización de bloques se realiza para mejorar la invariancia a cambios locales en la iluminación. La normalización de bloques se realiza dividiendo los histogramas de orientación de las celdas en el bloque por la norma L2 del bloque completo.

5. Concatenación de los histogramas: Finalmente, se concatenan los histogramas normalizados para todas las celdas para formar un vector de características. Este vector de características se puede utilizar para entrenar un clasificador de aprendizaje automático, como una máquina de vectores de soporte (SVM), que se utiliza para detectar objetos en nuevas imágenes.

El detector DLIB HOG utiliza una ventana deslizante para la detección de objetos en imágenes. En cada ubicación de la ventana, se calcula un histograma de gradientes orientados para identificar los patrones de los objetos presentes en la imagen. Luego, se utiliza un clasificador, como una máquina de vectores de soporte (SVM), para determinar si esa ubicación contiene un objeto o no. De esta manera, el detector DLIB HOG logra una alta precisión y eficiencia en la detección de objetos en imágenes.

5.2.6.2. Predictor 68 puntos de referencias de Dlib

El predictor de DLIB es una herramienta de aprendizaje automático que detecta y localiza puntos de referencia faciales en imágenes de caras humanas. Utiliza el modelo de regresión de bosques aleatorios para encontrar características en la imagen para encontrar patrones como bordes, esquinas y cambios de intensidad, y se entrena con datos obtenidos de la extracción manual de puntos de referencia faciales [36]. Este predictor puede estimar las coordenadas faciales y utiliza los puntos 36-41 para el ojo derecho y los puntos 42-47 para el ojo izquierdo y se detallan en la figura 11.

La identificación de puntos de referencia faciales a través del modelo 68 de Dlib se realiza en dos pasos:

- En primer lugar, se utiliza la detección de rostros para localizar un rostro humano y obtener un rectángulo con valores en x, y, w, h.
- En segundo lugar, se atraviesan puntos dentro de este rectángulo para obtener la ubicación de la cara y anotar los 68 puntos de referencia, los cuales son parte del conjunto de datos utilizado para entrenar el predictor de punto de referencia facial [37].

La base de datos de Dlib iBUG 300-W es un conjunto de datos faciales que consta de 300 imágenes en interiores y 300 en exteriores en estado salvaje. Esta base de datos cubre una gran variación de identidad, expresión, condiciones de iluminación, pose, oclusión y tamaño de la cara.

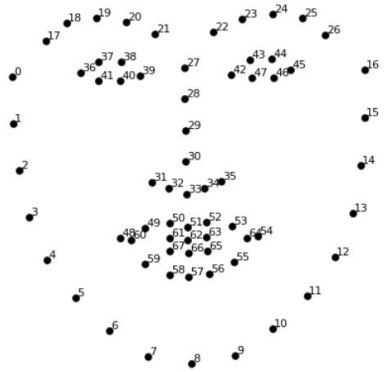


Fig. 11 Puntos de referencias con librería Dlib [35]

Las imágenes dentro de la base de datos fueron descargadas de internet realizando consultas como: fiesta, conferencia, protestas, futbol y celebridades. En comparación con el resto de los conjuntos de datos en estado salvaje esta continente imágenes ocluidas y cubre más expresiones que el común neutro o sonrisa, como sorpresa o grito [38].

5.2.7. EAR (Eye Aspect Ratio)

Facial Landmark enmarca los ojos usando 6 puntos referenciales como se indica en la figura 12, con la ayuda de estos puntos se puede obtener la abertura de los ojos utilizando una relación de aspecto del ojo también conocido como (EAR), por sus siglas en inglés, Eye Aspect Ratio.

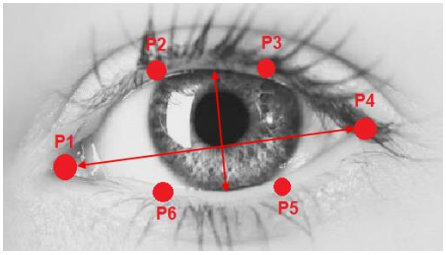


Fig. 12 Puntos de referencia en los ojos para cálculo de EAR. Elaboración propia.

El sistema toma un video en tiempo real como entrada y devuelve una secuencia de eventos de parpadeos {$blink_i$,…,$blink_k$}. Cada $blink_i$ es un vector de cuatro dimensiones que define un parpadeo mediante su duración, amplitud, velocidad de apertura del ojo y frecuencia. Además, cada evento de parpadeo se divide en tres etapas: $start_i$, $bottom_i$ y end_i que corresponden al inicio, fondo y final de un parpadeo respectivamente [39]. Estas etapas se ilustran en la siguiente figura:

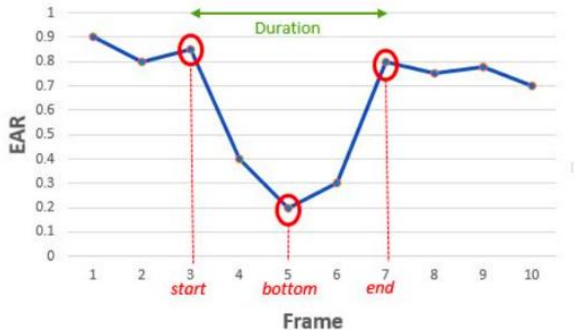


Fig. 13 EAR durante un parpadeo completo, puntos inicial, inferior y final [39].

De esta manera, para cada fotograma *k* se denota:

$$EAR[k] = \frac{\left||P2 - P6|\right| + ||P3 - P5||}{2||P1 - P4||} \quad (1)$$

Ahora bien, necesitamos calcular el valor de EAR en cada ojo, debido a que el valor en ellos es distinto, para ello se tiene:

$$EAR[k]left = \frac{\left||P2left - P6left|\right| + ||P3left - P5left||}{2||P1left - P4left||}$$

$$EAR[k]right = \frac{\left||P2right - P6right|\right| + ||P3right - P5right||}{2||P1right - P4right||}$$

Ya que se analizan los 2 ojos se procede con la siguiente ecuación para calcular el promedio de EAR de ambos ojos:

$$EAR[k]prom = \frac{EAR[k]left + EAR[k]right}{2} \quad (2)$$

5.2.7.1. Parámetros que evalúa EAR para detectar la somnolencia

EAR por sí solo no define somnolencia, según el documento ***"A realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection"*** [39] se pueden definir cuatro características que capturan patrones temporales que aparecen de forma natural en los ojos humanos y que podrían pasar desapercibidos por los detectores de características espaciales como las CNN (como es el caso de la visión humana), a continuación, se describen estos parámetros:

Duración del parpadeo

La duración del párpado varía en dependencia a lo que la persona esté realizando en una determinada situación, se determinará la duración del tiempo de parpadeo mientras conduce por medio de la siguiente ecuación:

$$Duracion_i = end_i - start_i + 1 \tag{3}$$

Amplitud de apertura del ojo

La amplitud de apertura del ojo según la ecuación (4) se calculará en dependencia de la persona, puesto que cada individuo posee formas de ojos distintas, algunos más rasgados y otros más redondos.

$$Amplitud_i = \frac{EAR[start_i] - 2EAR[bottom_i] + EAR[end_i]}{2} \tag{4}$$

Velocidad de apertura del ojo

En sujetos alertas, la velocidad máxima de cierre de los parpadeos está altamente correlacionada con su amplitud [40]. Los parpadeos no son todos iguales, y cuanto mayor sea el parpadeo mayor será su velocidad.

$$Velocidad\ de\ apertura\ del\ ojo_i = \frac{EAR[end_i] - EAR[bottom_i]}{end_i - bottom_i} \tag{5}$$

Frecuencia de parpadeo

Una persona normalmente en un minuto parpadea de 17 a 20 veces, esta frecuencia se reduce cuando está leyendo, siendo la frecuencia de 12 a 16 veces en un minuto y si está delante de un ordenador esta disminuye drásticamente a 3 o 4 veces [41]. Mediante la realización de pruebas se establecerá un valor umbral mediante las ecuaciones EAR.

$$Frecuencia_i = 100 * \frac{Numero\ de\ parpadeos\ hasta\ blink_i}{Numero\ de\ fotogramas\ hasta\ end_i} \tag{6}$$

VI. DISEÑO METODOLÓGICO

El proyecto es de tipo aplicado, ya que se diseñó un prototipo de un sistema detector de somnolencia en tiempo real basado en visión por computador para monitoreo de patrones de comportamiento visual de conductores vehiculares con el objetivo de prevenir accidentes de tránsito brindando una alerta, de esta manera se podrá indicar al conductor para tomar medidas de precaución.

El método por utilizar es experimental, debido a que el desarrollo del prototipo fue sometido a pruebas, ya sea para el entrenamiento del sistema, el ajuste de diferentes parámetros para la detección de los ojos y corrección de errores que se presentaron al momento de unir todas las partes.

6.1. Fase de diseño

En el diseño del sistema se implementó el uso de los algoritmos de detección y predicción de rostros de Dlib junto con OpenCV para el procesamiento de un vídeo de entrada. Para ello se propuso utilizar Facial Landmark para la ubicación de puntos de referencia en ojos y la extracción de un rostro alineado y luego por medio del indicador EAR se procedió al cálculo de duración del tiempo del parpadeo en bajo, lo cual permite registrar los microsueños, con ello se determinó si la persona está alerta o en estado de somnolencia. A modo de resumen describimos el proceso de la siguiente manera:

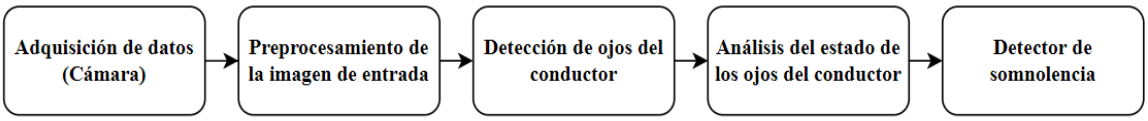


Fig. 14 Proceso general para detección de somnolencia. Elaboración propia.

6.1.1 Descripción de la arquitectura del sistema

El sistema está basado en el análisis de imágenes en tiempo real, la adquisición de datos de entrada se realizó por medio de una cámara CSI que estará conectada a una tarjeta Raspberry Pi 4 en la cual se desarrolló el algoritmo de reconocimiento y detección de somnolencia.

Cuando el sistema se encuentre activo se visualizará la GUI en pantalla y se tendrá una alerta visual y sonora en caso de somnolencia. Cabe destacar que no se especifica un sistema de audio ya que con el prototipo se podrían conectar los parlantes del vehículo con el sistema en general.

Se instaló Raspbian en la tarjeta Rapsberry Pi al ser un software gratuito basado en Linux, en la cual se optimizan recursos de memoria, cabe notar que la instalación de OpenCV no es precisamente rápida pero muchos proyectos de visión por computadora se tiene ventaja de esta biblioteca pues puede analizar un rostro hasta cinco veces por segundo [43]. Los componentes tecnológicos tanto de hardware como de software se describen en la tabla 2.

Tabla 2 - Componentes tecnológicos utilizados.

SOTWARE/HARDWARE	FUNCIÓN	CARACTERÍSTICAS
RASPBERRY PI 4	Procesamiento del programa	Procesador a un ARM Cortex-172 con cuatro núcleos a 1,5 GHz, 4 GB LPDDR4 SDRAM
RASPBIAN	Sistema operativo de la Raspberry Pi 4	Raspbian GNU/Linux 11
PYTHON	Lenguaje de programación	Versión 3.9.2
OPENCV	Librería para el tratamiento de imágenes	Versión 4.5.5
DLIB	Librería para la extracción de rostro	Versión 19.24.0
CÁMARA CSI	Adquisición de vídeo	Módulo de cámara V2 – Sony IMX219 8-megapixel sensor 1080p
PANTALLA TOUCH SCREEN DSI	Mostrar la interfaz de usuario	Pantalla de 7 pulgadas, resolución de 800x400 pixeles, 60Hz

6.1.2 Diagrama de flujo de funcionamiento del sistema

Previo a proporcionar una explicación detallada del algoritmo utilizado, se presenta una vista general del funcionamiento del código mediante la representación gráfica de un diagrama de flujo como se muestra en la Fig. 15:

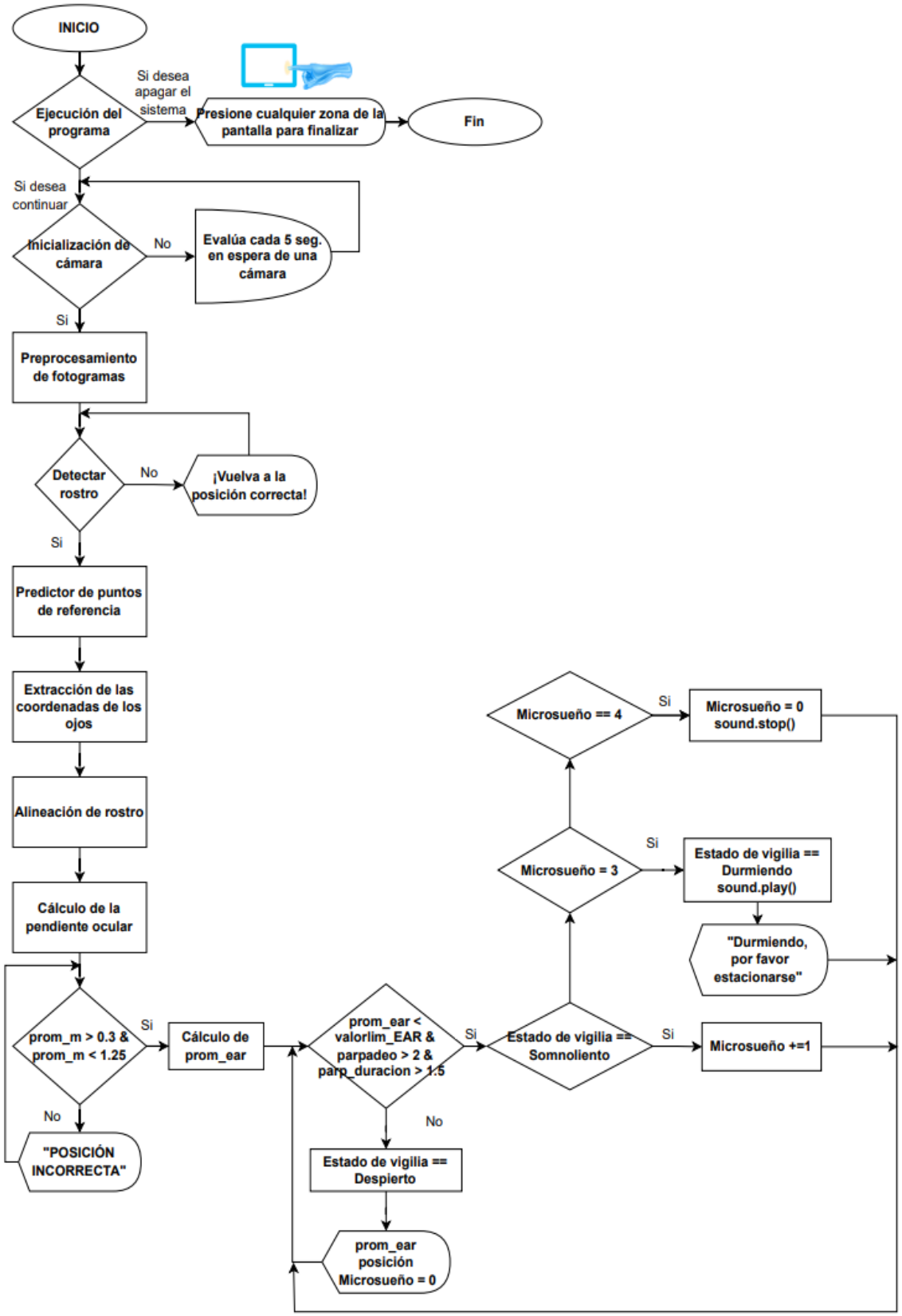


Fig. 15 Diagrama de flujo del sistema. Elaboración propia.

6.1.3 Análisis del algoritmo utilizado

El programa inicia llamando a las librerías cv2, dlib, numpy, imutils, entre otras. Cargamos el detector HOG y predictor de Dlib que permitirá encontrar rostros en el video de entrada y colocar los puntos de referencia faciales sobre el mismo a como se muestra en la Fig.16. Seguido se declaran una serie de variables que se pueden apreciar con más detalle en anexo 1 “Variables globales del código”, las cuales conforme continue el análisis se hablará sobre ellas con más detalle.

```
1   import cv2
2   import dlib
3   from scipy.spatial import distance
4   from imutils import face_utils
5   import numpy as np
6   import time
7   from pygame import mixer
8
9   #------------------------------PREDICTOR Y DETECTOR-----------------------------
10  # Inicializa el detector de rostros dlib y el predictor de puntos de referencia faciales
11  detector_HOG = dlib.get_frontal_face_detector()
12  predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

Fig. 16 Librerías, detector y predictor en código. Elaboración propia.

En la figura 17 se inicializa el mezclador de audio donde se carga una alerta de audio en formato *.wav* con nombre “*sonido*” dentro de la misma carpeta del ejecutable el cual se mandará a llamar siempre que se detecte el estado de somnolencia.

```
56   mixer.init()
57   sound = mixer.Sound('sonido.wav')
```

Fig. 17 Inicialización del mezclador de audio. Elaboración propia.

A continuación, se declaran una serie de funciones que serán las responsables de dar la correcta ejecución del código. Primero se definió una función en la que se calculó el promedio de la pendiente ocular (Fig.18) con el cual se limita la rotación del rostro en cierto rango para que el sistema sea funcional siempre y cuando la persona se encuentre frente a la cámara.

```python
#------------------------------FUNCIONES-------------------------------
def calcular_promedio_pendiente_ocular(valores_mo, num_frames):
    ojoder_x, ojoder_y = forma.part(36).x, forma.part(36).y
    nariz_x, nariz_y = forma.part(30).x, forma.part(30).y

    if (nariz_x - ojoder_x) <= 0:
        return None
    else:
        m = round((nariz_y-ojoder_y)/(nariz_x-ojoder_x),2)

    valores_mo.append(m)

    if len(valores_mo) > num_frames:
        valores_mo.pop(0)
    prom_mo = round(sum(valores_mo) / len(valores_mo),2)
    return prom_mo
```

Fig. 18 Función para calcular el promedio de la pendiente ocular. Elaboración propia.

Esto se logra calculando la pendiente promedio de la línea entre el ojo derecho y la nariz. La función toma dos parámetros: *valores_mo* y *num_frames*, y utiliza coordenadas *x* e *y* de la *parte 36 (ojo derecho) y parte 30 (nariz)* de un objeto *forma,* se puede apreciar de mejor forma en la Fig. 19.

Si la coordenada *x* de la nariz es menor o igual que la coordenada *x* del ojo derecho, la función devuelve *None,* de lo contrario, se calcula *(nariz_y-ojoder_y) / (nariz_x-ojoder_x)* perteneciente a la ecuación de la pendiente *y* se redondea a 2 decimales. La pendiente calculada se agrega a la lista *valores_mo* y se verifica si la lista tiene más elementos que *num_frames* el cual equivale a 15*.* Si es así, se elimina el primer elemento. Finalmente, se calcula la pendiente promedio *de valores_mo* y se devuelve como resultado de la función.

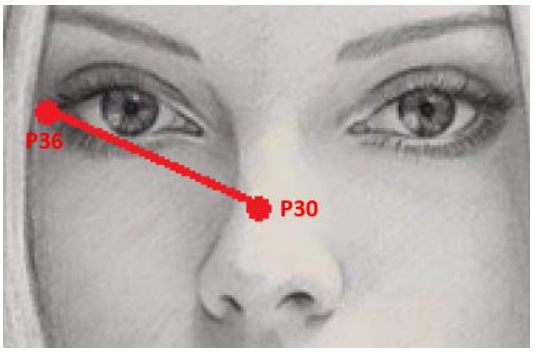


Fig. 19 Puntos faciales para el cálculo de la pendiente ocular. Elaboración propia.

Se utiliza EAR para identificar y manipular los parpadeos en imágenes de rostros. Primero se utiliza un predictor y detector para obtener la ubicación de los ojos en el rostro. Luego se utiliza la función "*calculo_EAR*" para calcular la relación de aspecto para un ojo específico en el vídeo de entrada. Se utiliza la función *distance.euclidean* de la biblioteca *scipy.spatial* para medir las longitudes de las líneas entre los puntos clave del ojo. El valor de EAR se calcula sumando la longitud de las líneas diagonales y dividiéndolo por dos veces la longitud de la línea horizontal a como se puede observar en la Fig. 20. El valor resultante se devuelve como el valor de EAR calculado.

```
76  def calculo_EAR(eye):
77      return (distance.euclidean(eye[1], eye[5]) + distance.euclidean(eye[2],
78              eye[4])) / (2.0 * distance.euclidean(eye[0], eye[3]))
```

Fig. 20 Función para calcular EAR. Elaboración propia.

La función "*final_EAR*" mostrada en la figura 21 recibe como entrada una matriz de 68 puntos faciales detectados por Dlib y calcula la relación de aspecto del ojo (EAR) para ambos ojos. Primero, se extraen los puntos faciales para cada ojo a partir de los índices correspondientes. Luego, se llama a la función "*calculo_EAR*" para calcular la relación de aspecto del ojo para cada ojo. Después, se calcula la relación de aspecto del ojo promedio para ambos ojos y se redondea a 2 decimales. La función devuelve la relación de aspecto del ojo promedio y los puntos faciales correspondientes para cada ojo.

```
79  def final_EAR(forma):
80      (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
81      (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
82
83      ojoIzq = forma[lStart:lEnd]
84      ojoDer = forma[rStart:rEnd]
85
86      ear_izq = calculo_EAR(ojoIzq)
87      ear_der = calculo_EAR(ojoDer)
88
89      EAR = (ear_izq+ear_der)/2
90      EAR = round(EAR, 2)
91      return (EAR, ojoIzq, ojoDer)
```

Fig. 21 Función para calcular EAR en ambos ojos. Elaboración propia.

La función "*calcular_promedio_EAR*" (ver Fig. 22) toma una lista de valores de EAR como entrada y devuelve el valor promedio (filtro medio) de los valores de EAR en la lista. Si la lista de valores de EAR está vacía, la función devuelve 0.

```
93   def calcular_promedio_EAR(valores_EAR):
94       if len(valores_EAR) == 0:
95           return 0
96       return sum(valores_EAR) / len(valores_EAR)
```

Fig. 22 Función para calcular el promedio de EAR. Elaboración propia.

La función "*calcular_duración_parpadeo*" (ver Fig. 23) detecta parpadeos en función del valor promedio de EAR y calcula la duración de los parpadeos. Si se detecta un parpadeo, registra el tiempo de inicio y finalización del mismo, calcula su duración y la almacena en una variable, también se dibuja la duración del último parpadeo en la interfaz. Si se detectan un parpadeo (*parpadeo >=2*), almacena la duración del último parpadeo en una lista y reinicia los tiempos de inicio, finalización y el contador de parpadeos.

```
98   def calcular_duración_parpadeo(prom_ear, valorlim_EAR, parpadeo, parp_inicio,
99                                  parp_final, parp_duracion, parp_tiempo, tablero):
100      if prom_ear < valorlim_EAR:
101          parpadeo += 1
102          if parpadeo == 1:
103              parp_inicio = time.time()
104          elif parpadeo >= 2:
105              parp_final = time.time()
106              parp_duracion = round(parp_final - parp_inicio, 2)
107              cv2.putText(tablero, "Duracion: {:.2f}".format(parp_duracion),
108                          (50, 120), fuente, 0.6, (128, 128, 128), 2)
109      else:
110          if parpadeo >= 2:
111              parp_inicio, parp_final = None, None
112              parp_tiempo.append(parp_duracion)
113              parpadeo = 0
114      return parpadeo, parp_inicio, parp_final, parp_duracion, parp_tiempo, tablero
```

Fig. 23 Función para calcular los parpadeos y su duración. Elaboración propia.

La función "*determinar_estado_vigilia*" verifica si el valor promedio de EAR es menor que el valor límite de EAR de 0.15, si el número de parpadeos detectados es mayor que 2 y si la duración del último parpadeo es mayor que 1.5 segundos, si se cumplen estas tres condiciones, la función devuelve la cadena "*Somnoliento*". De lo contrario, devuelve la cadena "*Despierto*" (ver Fig. 24).

En otras palabras, si el usuario parpadea y la duración es anormal, se considera que está somnoliento, se decidió la duración del tiempo de parpadeo en 1.5 seg. en base al apartado 6.1.5 "Discretización".

```
116  def determinar_estado_vigilia(prom_ear):
117      if prom_ear < valorlim_EAR and parpadeo > 2 and parp_duracion > 1.5:
118          return "Somnoliento"
119      else:
120          return "Despierto"
```

Fig. 24 Función para determinar el estado de vigilia. Elaboración propia.

La función "actualizar_microsueno" actualiza el conteo de microsueños basado en el estado de vigilia detectado. Según la Fig. 25, si el estado de vigilia es "*Somnoliento*" y aún no se ha contado un microsueño, se incrementa el conteo de microsueño y se establece "*ya_contado*" en Verdadero. Si el estado de vigilia no es "*Somnoliento*", se restablece "*ya_contado*" en Falso. Si el conteo de microsueños alcanza el valor de 4, se reinicia a 0.

```
122  def actualizar_microsueno(estado_vigilia, microsueno, ya_contado, tablero):
123      if estado_vigilia == "Somnoliento" and not ya_contado:
124          microsueno += 1
125          ya_contado = True
126      elif estado_vigilia != "Somnoliento":
127          ya_contado = False
128      cv2.putText(tablero, f"Microsueno: {microsueno}", (50, 95),
129                  fuente, 0.6, (128, 128, 128), 2)
130      if microsueno == 4:
131          microsueno = 0
132      return microsueno, ya_contado
```

Fig. 25 Función para establecer microsueños. Elaboración propia.

La función "*mostrar_estado_vigilia*" (ver Fig. 26) muestra el estado de vigilancia del sujeto en la imagen "*tablero*". Si el estado es "*Somnoliento*", se muestra la etiqueta "Somnoliento" en la esquina superior derecha de la imagen y se coloca una imagen de alerta en la esquina inferior derecha de la imagen. Si el estado es "*Despierto*" pero el sujeto ha tenido tres microsueños consecutivos, se muestra la etiqueta "*Durmiendo*" en la esquina superior derecha de la imagen y se coloca una imagen de alerta diferente en la esquina inferior derecha de la imagen. Si el estado es "*Despierto*" y el sujeto no ha tenido tres microsueños consecutivos, se muestra la etiqueta "*Despierto*" en la esquina superior derecha de la imagen y se detiene cualquier sonido de alerta que esté reproduciéndose.

```
134  def mostrar_estado_vigilia(tablero, estado_vigilia, microsueno):
135      if estado_vigilia == "Somnoliento":
136          cv2.putText(tablero, f"{estado_vigilia}", (60, 20), fuente, 0.6, (128, 128, 255), 2)
137          alerta = cv2.rectangle(img=cv2.imread("alerta.png"), pt1=(200, 200),
138                                 pt2=(200, 200), color=(0, 255, 0),thickness=-1)
139          salida[250:390, 550:780] = alerta
140      elif microsueno == 3:
141          cv2.putText(tablero, f"Durmiendo", (60, 20), fuente, 0.6, (128, 128, 255), 2)
142          sound.play()
143          alerta1 = cv2.rectangle(img=cv2.imread("alerta1.png"), pt1=(200, 200),
144                                  pt2=(200, 200), color=(0, 255, 0),thickness=-1)
145          salida[250:390, 550:780] = alerta1
146      else:
147          cv2.putText(tablero, f"{estado_vigilia}", (70, 20), fuente, 0.6, (128, 128, 128), 2)
148          sound.stop()
```

Fig. 26 Función para mostrar la alerta según el estado de vigilia. Elaboración propia.

Como última función, se define un controlador de eventos de mouse para la ventana de OpenCV la cual actúa como GUI (ver Fig. 27). La función se ejecutará cada vez que el usuario presione cualquier parte de la pantalla al querer apagar el sistema. Cuando esto sucede, la variable global "*running*" se establece en "*False*", lo que detiene la ejecución del programa.

```
151  def mouse_callback(event, x, y, flags, param):
152      global running
153      if event == cv2.EVENT_LBUTTONDOWN:
154          running = False
```

Fig. 27 Función de devolución de llamada táctil. Elaboración propia.

En la figura 28, se inicializa la captura del vídeo utilizando la cámara predeterminada del sistema operativo (en este caso, la cámara con índice 0). También establece la resolución del marco de video en 320x240 píxeles utilizando los métodos set() del objeto cap.

Luego, se verifica si la cámara se ha abierto correctamente usando el método isOpened(). Si la cámara no se ha abierto correctamente, se muestra un mensaje de error en la consola mientras espera 5 segundos en espera de una cámara, esto se hace con el fin de verificar si hubo un problema en la conexión con la misma. Este ciclo se repite hasta que la cámara se abre correctamente.

```
156  #-------------------------------CÁMARA-------------------------------
157  cap = cv2.VideoCapture(0)
158  cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
159  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
160
161  while not cap.isOpened():
162      print("No se puede abrir la cámara. Esperando...")
163      time.sleep(5)
164      cap = cv2.VideoCapture(0)
165      cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
166      cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
```

Fig. 28 Inicialización de la cámara y redimensionamiento del vídeo de entrada. Elaboración propia.

En la figura 29 se muestra cómo se crea una ventana de GUI con el nombre "*ESTADO DEL CONDUCTOR*" utilizando la función *namedWindow* de OpenCV. Este nombre se utilizará más adelante para referirse a la ventana. Y se establece una función de devolución de llamada de mouse para esa ventana, lo que permite a los usuarios interactuar con la ventana al presionar la pantalla.

```
167  cv2.namedWindow("ESTADO DEL CONDUCTOR")
168  cv2.setMouseCallback("ESTADO DEL CONDUCTOR", mouse_callback)
```

Fig. 29 Creación de la ventana de GUI. Elaboración propia.

Inicio del bucle

Se destaca que el código está dentro de un bucle "while" que se ejecuta mientras la variable del controlador de evento sea "*running*" verdadera.

Según la figura 30, cada vez que se ejecuta el bucle, se captura un cuadro de video de una cámara (u otro dispositivo de entrada de video), se procesa y se muestra en pantalla. Cada cuadro se lee mediante la función cap.read() y se almacena en la variable cuadro. Luego, se crea una copia del cuadro original para otros procesos.

Cuadro se convierte a escala de grises mediante la función "*cv2.cvtColor(cuadro, cv2.COLOR_BGR2GRAY)",* lo que facilita su análisis (debido a que el predictor fue entrenado bajo escalas de grises). Luego, el detector de HOG se utiliza para encontrar los cuadros delimitadores de cada rostro en la imagen en escala de grises.

Para asegurarse de que solo se haya detectado un rostro en la imagen se utiliza *if len(rostros) ==1*, lo que evita que el código tenga dificultades para identificar el rostro correcto o genere errores al intentar analizar un rostro que no existe.

```
169  #-------------------------------BUCLE--------------------
170  while running:
171      ret, cuadro = cap.read()
172
173      copia_cuadro = cuadro.copy()
174
175      gris = cv2.cvtColor(cuadro, cv2.COLOR_BGR2GRAY)
176
177      rostros = detector_HOG(gris)
178
179      # Evaluamos que solo se haya detectado un rostro
180      if len(rostros) ==1:
```

Fig. 30 Captura de cuadro, implementación del detector de rostros y evaluación de un rostro. Elaboración propia.

En la figura 31 se recorren todos los *rostros* detectados en la imagen. En cada iteración del bucle, se calculan los puntos de referencia facial (forma) para cada rostro utilizando el predictor. Después, se utiliza la función para calcular la pendiente ocular *prom_m* (observar fig. 19), obtenido a partir de los puntos correspondientes al ojo derecho y nariz. Luego, se convierten las coordenadas (x,y) de los puntos de referencia facial a un arreglo NumPy para poder manipularlas con mayor facilidad. Posteriormente, se extraen los datos EAR utilizando la función *final_EAR*.

```python
    # Recorremos cada uno de los rostros detectados en la imagen
    for (i, rostro_det) in enumerate(rostros):

        # Determinacion de puntos de referencia facial
        forma = predictor(gris, rostro_det)

        # Extraemos los datos de la  pendiente ocular
        prom_m = calcular_promedio_pendiente_ocular(valores_mo, num_frames)

        # Conversion de coordenadas (x,y) de facial landmarks a arreglo tipo numpy
        forma = face_utils.shape_to_np(forma)

        # Extraemos los datos de EAR
        ojo = final_EAR(forma)

        # Asignamos las coordenadas de los ojos izquierdo y derecho
        ojoIzq = ojo[1]
        ojoDer = ojo[2]
```

Fig. 31 Extracción de las coordenadas de los ojos izquierdo y derecho. Elaboración propia.

Finalmente, se asignan las coordenadas de los puntos de referencia faciales del ojo izquierdo y derecho a las variables *ojoIzq* y *ojoDer*, respectivamente. Estas variables se utilizan para determinar si el conductor está somnoliento o no. Seguido de ello, se expone el método con el cual se obtendrá la **alineación de ojos** a partir de la figura 32.

Convertimos un rectángulo de Dlib en un cuadro delimitador, la función *rect_to_bb* devuelve cuatro valores, toma como entrada el rectángulo *x* e *y* del punto superior izquierdo del rectángulo y brinda las dimensiones *w* y *h* como ancho y altura del rectángulo.

```python
        # Convertir el rectángulo de dlib en un cuadro delimitador de estilo OpenCV
        (x, y, w, h) = face_utils.rect_to_bb(rostro_det)

        # Extraemos una subimagen de la imagen original "cuadro"
        ext_subcuadro = cuadro[y:y + h,  x:x + w]

        #Redimensionamiento de imagen incial
        if not ext_subcuadro.size:
            print("La imagen está vacía, no se puede redimensionar")
        else:
            ext_subcuadro = cv2.resize(ext_subcuadro, (120, 120),
                                       interpolation=cv2.INTER_AREA)

        # Dibujo del rectangulo delimitador del rostro
        cv2.rectangle(cuadro, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(cuadro, "Rostro #{}".format(i + 1), (x - 10, y - 10),
                    fuente, 0.5, (0, 255, 0), 2)

        # Determinación del centro de masa de cada ojo
        ojoIzqCentro = ojoIzq.mean(axis=0).astype("int")
        ojoDerCentro = ojoDer.mean(axis=0).astype("int")

        # Determina el ángulo entre los centroides
        dY = ojoDerCentro[1] - ojoIzqCentro[1]
        dX = ojoDerCentro[0] - ojoIzqCentro[0]
```

Fig. 32 Obtención de los centroides de los ojos. Elaboración propia.

Luego con la línea *ext_subcuadro = cuadro[y:y + h, x:x + w]* extraemos un *subcuadro* de la imagen original que corresponde a la posición y dimensiones del rostro detectado y dibujamos un rectángulo verde alrededor del rostro en la imagen original utilizando la función *cv2.rectangle()*, tal como se muestra en la Fig. 33.

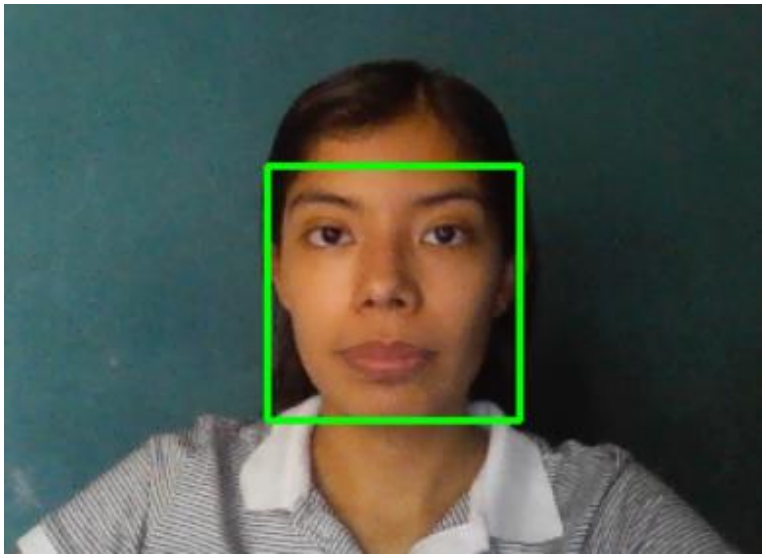


Fig. 33 Cuadro delimitador del rostro detectado. Elaboración propia.

Seguido calculamos el centro de masa del ojo izquierdo y derecho utilizando el método *mean()* de NumPy para calcular la media de las coordenadas de todos los puntos del ojo, luego según la Fig. 34 se calcula la diferencia en coordenadas (dY y dX) entre los centroides de los ojos para finalmente, con la función arctan2 de NumPy calcular el ángulo entre los centroides y convertir de radianes a grados

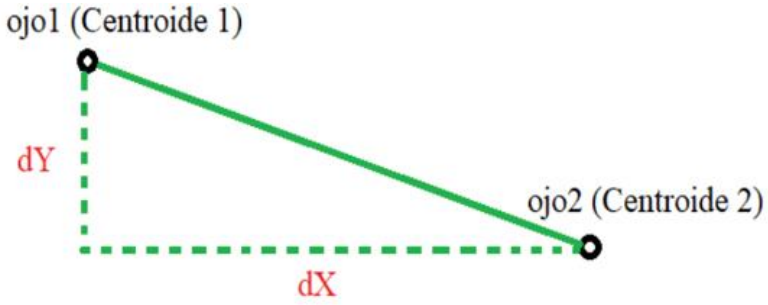


Fig. 34 Calculo de los centroides de los ojos. Elaboración propia.

De acuerdo con la figura. 35, el ángulo calculado se utiliza para corregir la orientación de la cara en la imagen. Se asignan las coordenadas (x,y) de los puntos centrales de los ojos derecho e izquierdo en variables separadas: Rx, Ry, Lx y Ly, se dibuja una línea que conecta los puntos centrales de los ojos derecho e izquierdo utilizando la función cv2.line(), (es decir, la cara gira de tal manera que los ojos se encuentran a lo largo de las mismas coordenadas y). También se dibuja un círculo en cada punto central del ojo derecho e izquierdo utilizando la función cv2.circle(), ver Fig. 36.

```
226            # Ángulo de rotación de la cara
227            angulo = np.degrees(np.arctan2(dY, dX)) - 180
228
229            #Almacenar las coordenadas (x,y) el punto central de
230            #cada ojo detectado en variables separadas
231            Rx = ojoDerCentro[0]
232            Ry = ojoDerCentro[1]
233            Lx = ojoIzqCentro[0]
234            Ly = ojoIzqCentro[1]
235
236            # Dibujamos una línea entre los puntos centrales de los ojos izquierdo y derecho
237            cv2.line(cuadro, (Rx, Ry), (Lx, Ly), (0, 255, 255), 3, cv2.LINE_AA)
238            # Dibujamos un círculo en el punto central del ojo derecho e izquierdo
239            cv2.circle(cuadro, (Rx, Ry), 3, (255, 0, 0), -1)
240            cv2.circle(cuadro, (Lx, Ly), 3, (0, 0, 255), -1)
241
242            pos_ojoder = 1.0 - pos_ojoizq[0]
243            dist_ojo = np.sqrt((dX ** 2) + (dY ** 2))
244            dist_des = (pos_ojoder - pos_ojoizq[0])
245            dist_des = dist_des*ancho_rostro
246            escala = dist_des / dist_ojo
```

Fig. 35 Angulo de rotación de la cara y centroide de ojos. Elaboración propia.

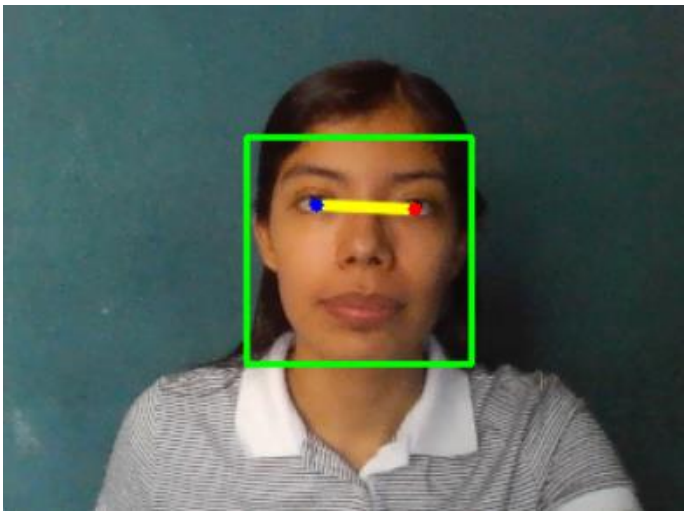


Fig. 36 Centroide y alineación de ojos. Elaboración propia.

La variable *pos_ojoizq[0]* representa la posición del ojo izquierdo en la imagen, y se utiliza para calcular la posición del ojo derecho *(pos_ojoder)* mediante la resta de 1.0 menos la posición del ojo izquierdo. Luego, se calcula la distancia entre los ojos *(dist_ojo)* utilizando el teorema de Pitágoras con las coordenadas del centro de cada ojo *(ojoIzqCentro y ojoDerCentro).* Posteriormente, se calcula la distancia deseada entre los ojos *(dist_des)* como la diferencia entre la posición del ojo derecho y la del ojo izquierdo multiplicada por el ancho del rostro. La escala necesaria para ajustar la imagen se calcula como la relación entre la distancia deseada y la distancia real entre los ojos (escala = dist_des / dist_ojo). Estás variables se encontraron con el objetivo de calcular la matriz de rotación.

Procedemos a encontrar el punto medio entre los centroides de los ojos en el rostro y graficamos un círculo en el centro calculado, ver Fig. 37.

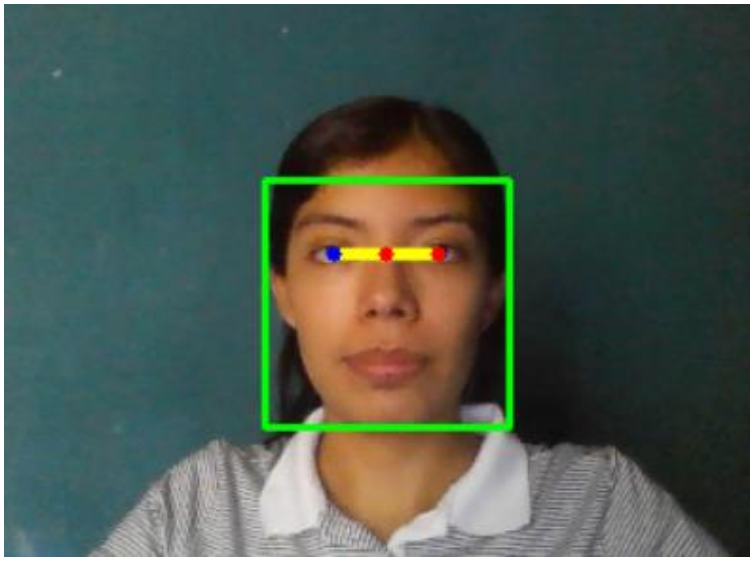


Fig. 37 Punto medio entre los centroides. Elaboración propia.

Seguido, en la figura 38 calculamos la matriz de rotación *M* según la ecuación 7 que se utilizará para rotar y escalar la imagen. Calculamos el desplazamiento en el eje x e y que se aplicará a la imagen; en X lo calculamos multiplicando la mitad del ancho del rostro por 0.5, lo que resulta en un desplazamiento de la mitad de la imagen en la mitad del ancho del rostro, en Y se multiplica la altura del rostro por la posición del ojo izquierdo, lo que resulta en un desplazamiento vertical de la imagen.

Matriz de rotación $$M = \begin{bmatrix} \alpha & \beta & (1-\alpha).center.x - \beta.center.y \\ -\beta & \alpha & \beta.center.x + (1-\alpha).center.y \end{bmatrix} \quad (7)$$

```
248 #------------------------------MATRIZ DE ROTACIÓN M----------------------------
249             # Calculamos el punto medio entre los ojos
250             centro_ojo = (int((ojoIzqCentro[0] + ojoDerCentro[0]) // 2),
251                           int((ojoIzqCentro[1] + ojoDerCentro[1]) // 2))
252
253             # Asignamos las coordenadas x e y del centro de los ojos a dos
254             # variables distintas y graficamos
255             centro_ojox = centro_ojo[0]
256             centro_ojoy = centro_ojo[1]
257             cv2.circle(cuadro, (centro_ojox, centro_ojoy), 3, (0, 0, 255), -1)
258
259             # Rotamos y escalamos la imagen de la cara para
260             # que los ojos queden en una posición deseada.
261             M = cv2.getRotationMatrix2D(centro_ojo, angulo, escala)
262
263             # Calculamos el valor de translación en el eje X e Y.
264             tX = ancho_rostro * 0.5
265             tY = alto_rostro * pos_ojoizq[1]
266
267             # Ajustamos el desplazamiento en el eje x e y para desplazar
268             # el rostro hacia el centro de la imagen.
269             M[0, 2] += (tX - centro_ojo[0])
270             M[1, 2] += (tY - centro_ojo[1])
```

Fig. 38 Calculo de la matriz de rotación. Elaboración propia.

Posteriormente se ajusta la matriz de transformación M para desplazar la imagen hacia el centro de la imagen en los ejes *x e y*. Para el desplazamiento en el *eje x*, se agrega el desplazamiento calculado en el *eje x* a la posición *x* de la matriz de transformación *M*, y se resta el centro del *ojo x* para asegurarse de que la imagen se desplace hacia el centro.

Para el desplazamiento en el *eje y*, se agrega el desplazamiento calculado en el *eje y* a la posición y de la matriz de transformación *M*, y se resta el centro del *ojo y* para asegurarse de que la imagen se desplace hacia el centro.

En la figura 39 se establecen las dimensiones de ancho y alto de la imagen de la cara en *w y h*, respectivamente, utilizando las medidas de ancho y alto del rostro previamente obtenidas. Luego, se aplica una transformación afín a la imagen *copia_cuadro* utilizando la matriz de transformación *M* previamente calculada y las dimensiones *w y h.*

```
272         # Establecemos las dimensiones de ancho y alto de la imagen de la cara.
273         (w, h) = (ancho_rostro, alto_rostro)
274
275         # Aplicamos una transformación afín a la imagen en (w, h)
276         rostro_alin = cv2.warpAffine(copia_cuadro, M, (w, h), flags=cv2.INTER_CUBIC)
277
278         # Comprueba una curva de los defectos de convexidad y la corrige.
279         ojoIzqHull = cv2.convexHull(ojoIzq)
280         ojoDerHull = cv2.convexHull(ojoDer)
281         cv2.drawContours(cuadro, [ojoIzqHull], -1, (0, 255, 0), 1)
282         cv2.drawContours(cuadro, [ojoDerHull], -1, (0, 255, 0), 1)
```

Fig. 39 Obtención del rostro alineado y convexidad. Elaboración propia.

La transformación afín es una transformación lineal que mantiene las líneas rectas y paralelas en la imagen y se aplica utilizando la matriz de transformación M que se define con la rotación, escala y traslación necesarias para alinear la imagen de la cara. Como resultado, se obtiene una nueva imagen llamada *rostro_alin* que se utiliza posteriormente para dibujar los contornos y mostrar la imagen final.Para encontrar y dibujar los contornos convexos de los ojos en una imagen de la cara, se utiliza la función cv2.convexHull() para encontrar los contornos convexos de los ojos y la función cv2.drawContours() se utiliza para dibujarlos a como se muestra en la Fig. 40.

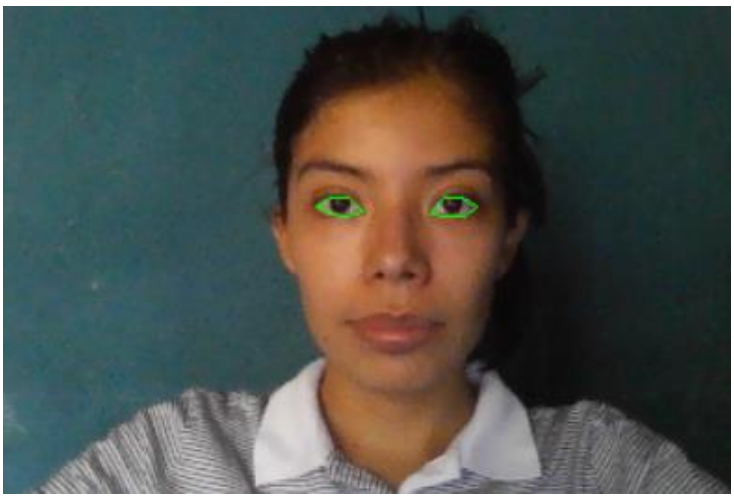


Fig. 40 Polígono convexo del contorno de ojos. Elaboración propia.

En la figura 41 tenemos las líneas de código las cuales son la base de la creación de la interfaz gráfica de usuario (GUI) para el proyecto en Python utilizando la biblioteca OpenCV.

```
284         salida = np.zeros((480, 800, 3), np.uint8)
285         salida[:] = cv2.imread("GUI.jpg")
286
287         # Inicializamos el tablero de la GUI
288         tablero = cv2.rectangle(img=cv2.imread("tablero.png"), pt1=(200, 200),
289                                 pt2=(200, 200), color=(0, 255, 0),thickness=-1)
```

Fig. 41 Creación de la ventana e injertación de la GUI. Elaboración propia.

Primero se crea una nueva matriz de NumPy llamada "*salida*" con una forma de (480, 800, 3). Esta matriz será utilizada para mostrar la GUI en una ventana separada. Segundo, se carga una imagen "*GUI.png*" (ver Fig. 42) utilizando la función *cv2.imread()* y se asigna a la matriz "*salida*" utilizando la sintaxis de "*slicing*" en NumPy *(salida[:]).* Esto se hace para superponer la imagen "*GUI.png"* en la matriz de salida. Y luego se inicializa un tablero para la GUI utilizando la función cv2.rectangle(). Esta función dibuja un rectángulo en una imagen dada, en este caso, en la imagen "tablero.png" (Fig. 45).

Fig. 42 GUI.png. Elaboración propia.

En la figura 43 primero se verifica si “prom_m” se encuentra dentro de un rango específico. Si es así, asigna el valor del “EAR” de la primera tupla en "ojo" a la variable "ear" y lo agrega a una lista llamada "valores_EAR". Luego, si la longitud de la lista "valores_EAR" es mayor que un número de fotogramas específico "num_frames", elimina el primer valor de la lista. De esta forma, se calcula el promedio ear de los valores en la lista "valores_EAR" y se muestra en un tablero. Después, se llama a una función para calcular la duración del parpadeo y actualizar variables relacionadas.

```
291            # Discretizamos el promedio de la pendiente ocular
292            if prom_m is not None and prom_m > 0.30 and prom_m < 1.25:
293
294            # Utiliza para asignar el valor de EAR que se encuentra
295            # en la 1era tupla de funcion "final_ear"
296                ear = ojo[0]
297
298                # Agregamos el valor de EAR a la lista
299                valores_EAR.append(ear)
300
301                # Si la lista de valores_EAR es mayor que num_frames, entonces eliminamos el primer valor
302                if len(valores_EAR) > num_frames:
303                    valores_EAR.pop(0)
304
305                # Extraemos el promedio de los valores de EAR y gráficamos en tablero
306                prom_ear = calcular_promedio_EAR(valores_EAR)
307                cv2.putText(tablero, f"EAR: {prom_ear:.2f}", (70, 45), fuente, 0.6, (128, 128, 128), 2)
308
309                # calcular la duración del parpadeo y actualizar variables relacionadas
310                parpadeo, parp_inicio, parp_final, parp_duracion, parp_tiempo, tablero = calcular_duración_parpadeo(
311                    prom_ear, valorlim_EAR, parpadeo, parp_inicio, parp_final, parp_duracion, parp_tiempo, tablero)
```

Fig. 43 Condicional para ejecución de funciones a utilizar. Elaboración propia.

En la figura 44 se llama a la función “determinar el estado de vigilia de la persona” en la cual determinar los dos estados de vigilia y los almacena en dicha variable para después actualizar los valores relacionados con el sueño y vigilia que nos brinda los microsueños. Luego, se muestran estos valores en el tablero usando la función cv2.putText. Finalmente, si la posición de la persona es incorrecta, se muestra un mensaje de error en el tablero.

```
312                # Determinamos si la persona está durmiendo o despierta
313                estado_vigilia = determinar_estado_vigilia(prom_ear)
314
315                microsueno, ya_contado = actualizar_microsueno(estado_vigilia, microsueno, ya_contado, tablero)
316
317                # Mostrar estado de vigilia y EAR
318                mostrar_estado_vigilia(tablero, estado_vigilia, microsueno)
319
320                cv2.putText(tablero, f"Posicion: {prom_m}", (50, 70), fuente, 0.6, (128, 128, 128), 2)
321
322            else:
323                cv2.putText(tablero, f"POSICION INCORRECTA", (30, 70), fuente, 0.5, (0, 0, 255), 2)
```

Fig. 44 Continuación del condicional de funciones utilizar. Elaboración propia.

En la figura 45 se muestra en la (GUI) información de diferentes elementos en una ventana. Primeramente, se ubican los elementos de video en posiciones específicas dentro de la GUI para luego, mediante la función "cv2.imshow" de OpenCV mostrar la GUI en una ventana titulada "ESTADO DEL CONDUCTOR". Si no hay información para mostrar, se crea una ventana vacía y se carga una imagen llamada "GUI2.png" en ella, y luego se muestra la ventana.

```
327             # Ubicamos los elementos en la GUI
328             salida[125:365, 30:350] = cuadro
329             if ext_subcuadro.shape[0] > 0 and ext_subcuadro.shape[1] > 0:
330                 salida[110:230, 400:520] = ext_subcuadro
331             salida[260:380, 400:520] = rostro_alin
332             salida[100:240, 550:780] = tablero
333 
334             # Muestra el cuadro en la ventana
335             cv2.imshow("ESTADO DEL CONDUCTOR", salida)
336     else:
337         salida1 = np.zeros((480, 800, 3), np.uint8)
338         salida1[:] = cv2.imread("GUI2.png")
339         salida1[125:365, 30:350] = cuadro
340         cv2.imshow("ESTADO DEL CONDUCTOR", salida1)
```

Fig. 45 Ubicación de elementos en la GUI. Elaboración propia.

En este último segmento del programa (Fig. 46), se logra la finalización de la ejecución del programa, ya que con “cap.release()” se detiene la captura de la cámara y mediante “cv2.destroyAllWindows()” se cierran las ventanas de cv2.

```
347 # Libera el uso de la camara y cierra las ventanas
348 cap.release()
349 cv2.destroyAllWindows()
```

Fig. 46 Finalización del programa. Elaboración propia.

6.1.4 Interfaz de usuario

La GUI del programa utiliza imágenes y alertas para que el usuario pueda monitorear su estado. En lugar de utilizar bibliotecas como Tkinter o PyQt, se decidió utilizar la ventana de terminal integrada para mejorar la velocidad de la interfaz y presentar una apariencia diferente. Las imágenes fueron creadas y ajustadas de tamaño en Canvas y se mandan a llamar en las partes del código donde sean requeridas de forma específica.

Cabe destacar que se desarrolló un ejecutable el cual se abre cada vez que se enciende el ordenador Rapsberry Pi, este ejecutable puede volver a abrirse mediante un icono en el escritorio cuando se desee utilizar el detector.

Fig. 47 tablero.png. Elaboración propia.

Fig. 48 GUI2.png. Elaboración propia

Fig. 49 Posición normal frente a la cámara en estado "Despierto". Elaboración propia.

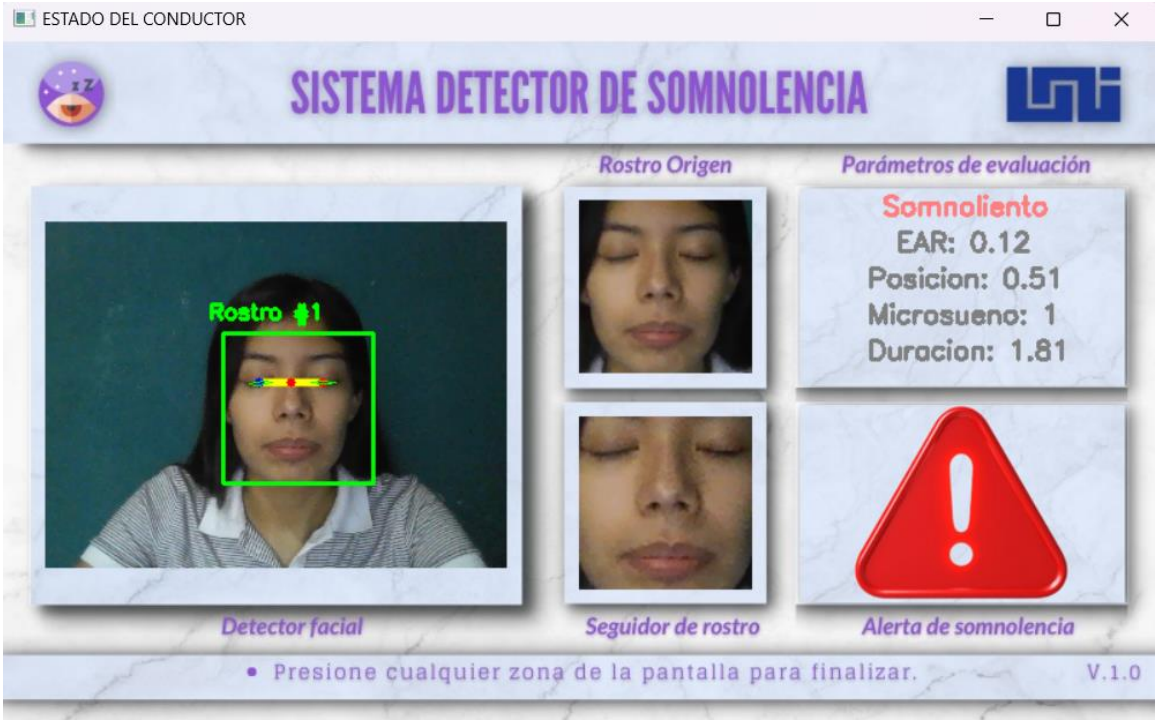


Fig. 50 Usuario presenta un parpadeo mayor a 1.5 segundos el cual de denomina como "Somnoliento". Elaboración propia.

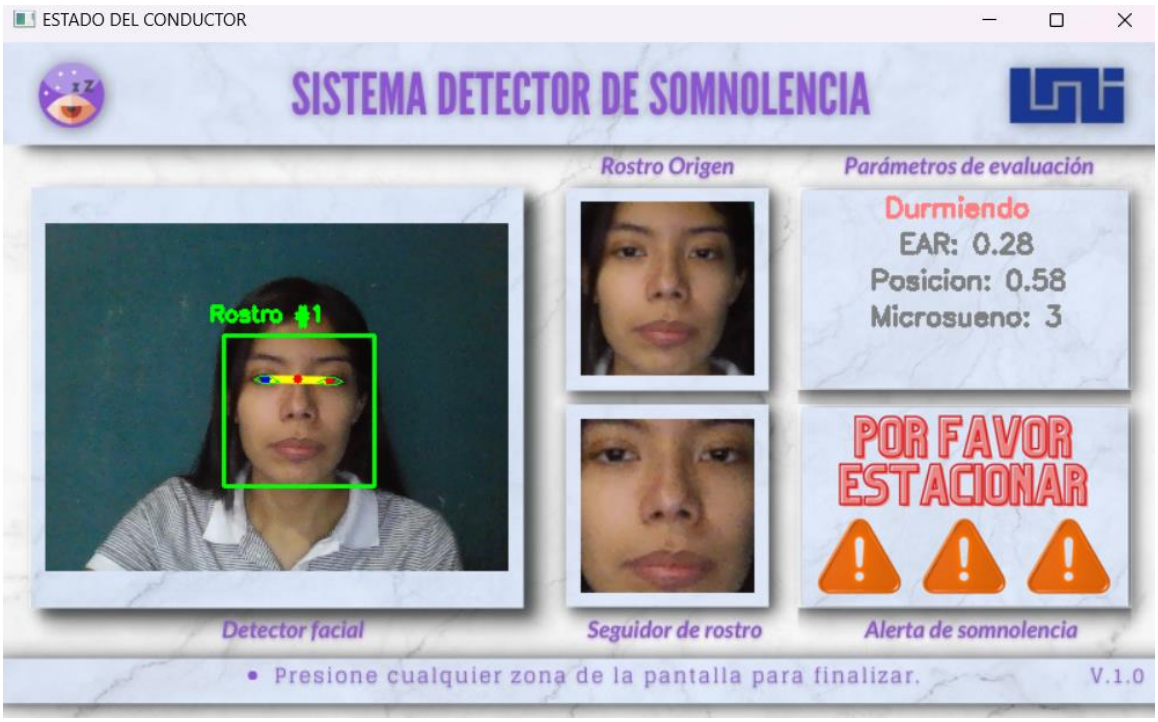


Fig. 51 Detección de estado "Durmiendo" al tener 3 microsueños, alarma sonora activada. Elaboración propia.

Fig. 52 Alarma de posición incorrecta al girar el rostro hacia la derecha. Elaboración propia.

Fig. 53 Alarma de posición incorrecta al girar el rostro hacia la izquierda. Elaboración propia.

6.1.5. Discretización

La duración promedio del parpadeo humano de 100 a 150 milisegundos es una estimación generalmente aceptada en la literatura científica. Esta cifra se basa en varios estudios que han medido la duración del parpadeo utilizando diversas técnicas, como el registro electromiográfico de la actividad muscular de los párpados o el análisis de imágenes de alta velocidad de los ojos [40]. Ejemplo de ello, se tienen algunos estudios consultados para poder establecer una referencia de un valor guía:

En Asano et al. (2017), los autores midieron la duración del parpadeo en 12 participantes utilizando un electroculograma y la frecuencia de parpadeo mediante un sensor de infrarrojos. Los resultados mostraron que la duración del parpadeo disminuyó significativamente después de realizar una tarea que requería esfuerzo visual sostenido. Los autores reportaron que la duración promedio del parpadeo fue de 0.15 segundos (150 milisegundos) antes de la tarea visual y de 0.12 segundos (120 milisegundos) después de la tarea visual [42].

En el estudio de Aoyama et al. se midieron la duración y frecuencia del parpadeo en 16 participantes después de instilar gotas para los ojos. Descubrieron que la duración del parpadeo aumentó significativamente después de la instilación de las gotas y que la frecuencia de parpadeo disminuyó. Los autores concluyeron que la instilación de gotas para los ojos puede afectar la función del parpadeo y la salud ocular. Además, los autores informaron que la duración promedio del parpadeo fue de 0.11 segundos (110 milisegundos) antes de la instilación de las gotas y de 0.14 segundos (140 milisegundos) después de la instilación de las gotas [43].

A partir de estos y otros estudios se puede decir que la duración promedio del parpadeo humano es de aproximadamente 100 a 150 milisegundos. Existen varios estudios que han medido la duración del parpadeo utilizando diferentes técnicas y han encontrado resultados similares en términos de duración promedio, pero también han destacado la variabilidad en la duración del parpadeo dependiendo de la actividad visual y otros factores.

Al tener claro estos valores, se establece la relación entre la duración de un parpadeo y la somnolencia. Por ejemplo, un estudio publicado en la revista Sleep en 2016 encontró que la duración promedio de los parpadeos aumentaba en personas que se encontraban en un estado de somnolencia diurna excesiva, en comparación con personas sin somnolencia. En particular, los participantes somnolientos tenían una duración promedio de parpadeo de 174 milisegundos, mientras que los participantes sin somnolencia tenían una duración promedio de 146 milisegundos [44].

Otro estudio publicado en la revista Sleep en 2018 encontró resultados similares, donde la duración de los parpadeos se asoció con la somnolencia diurna excesiva en un grupo de pacientes con apnea del sueño [45].

En general, estos estudios sugieren que una duración de parpadeo más larga que el promedio puede ser un indicador de somnolencia diurna excesiva. Sin embargo, es importante destacar que la duración de los parpadeos es solo uno de varios factores que pueden indicar somnolencia, y que un diagnóstico preciso requiere una evaluación integral de los síntomas y la historia clínica del individuo.

A su vez, se tiene una tercera relación para discretizar el sistema. Un estudio publicado en la revista Sleep en 2020 encontró que la duración promedio de los microsueños en una prueba de conducción simulada fue de 5.7 segundos. Los participantes que informaron una mayor somnolencia diurna tenían microsueños más largos y frecuentes [46].

Otro estudio publicado en la revista Sleep en 2019 encontró que los microsueños durante la conducción se asociaron con un mayor riesgo de accidentes de tránsito. Los participantes que tuvieron microsueños durante una prueba de conducción tenían un mayor riesgo de sufrir accidentes simulados en comparación con los participantes que no tuvieron microsueños [47].

Un estudio publicado en la revista Sleep Medicine en 2018 encontró que la duración promedio de los microsueños en pacientes con trastorno de sueño-vigilia del ritmo circadiano era de 8.7 segundos.

Los pacientes con este trastorno experimentan somnolencia diurna excesiva y tienen un mayor riesgo de accidentes de tránsito y laborales [48].

En resumen, la duración promedio de los microsueños parece estar en el rango de 2 a 10 segundos, pero puede variar según la persona y las circunstancias en las que ocurren. Los microsueños pueden ser peligrosos si ocurren durante actividades que requieren atención, como conducir, y pueden ser un signo de un trastorno del sueño subyacente.

6.2. Fase de implementación

6.2.1. Montaje del sistema

La idea del prototipo es que fuera portable, fácil de armar y funcionara como un dispositivo todo en uno. Se encapsuló el sistema en una carcasa protectora donde se incluye la Raspberry Pi 4, cámara, pantalla, fan, etc. (ver Fig. 50). Primeramente, detallamos las conexiones del sistema:

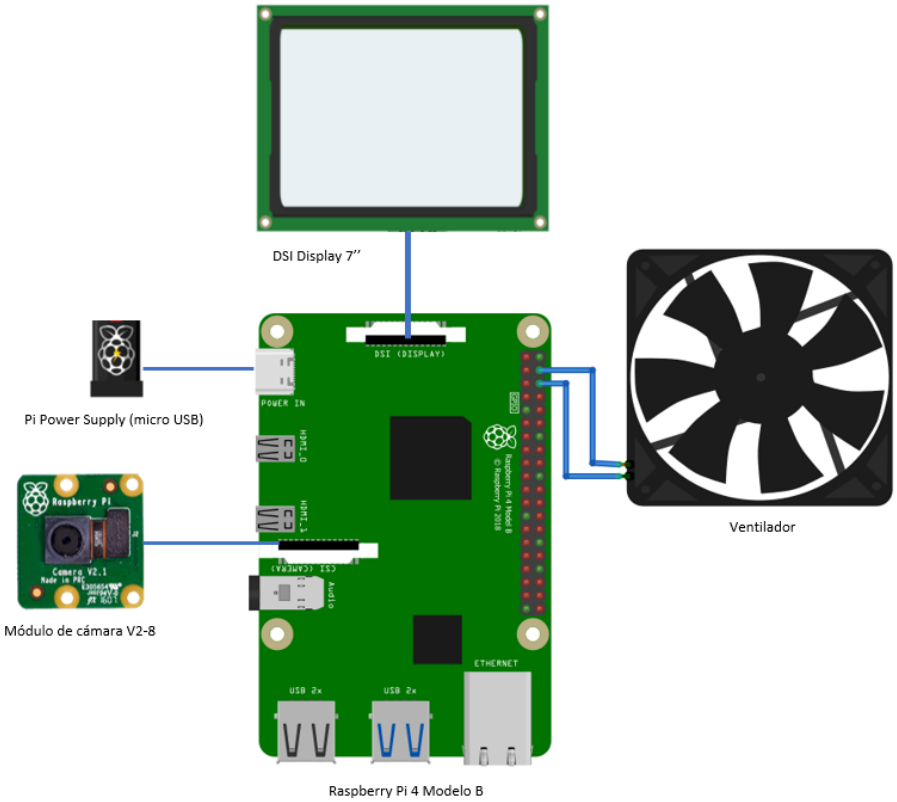


Fig. 54 Diagrama de conexiones. Elaboración propia.

El procedimiento del armado de la carcasa se puede apreciar en anexo 2, a continuación, se muestra el resultado del montaje del sistema:

Fig. 55 Prototipo vista frontal, cámara frontal. Elaboración propia.

Fig. 56 Prototipo vista trasera. Elaboración propia.

Fig. 57 Prototipo vista trasera - conexiones internas. Elaboración propia.

6.2.2. Instalación del sistema operativo

Para instalar el sistema operativo en la memoria SD de 64GB se debe descargar el programa Raspberry Pi Imager. Para acceder a la Raspberry de manera remota desde otra computadora, se utiliza el protocolo SSH, el cual se activa creando un archivo vacío llamado ssh en la partición /boot de la tarjeta SD. Luego se accede al escritorio de la Raspberry mediante VNC para instalar las librerías necesarias. En anexo 3 y 4 se detalla la instalación del sistema operativo y las librerías utilizadas.

Fig. 58 Prototipo con sistema operativo Raspbian. Elaboración propia.

Fig. 59 Sistema detector de somnolencia. Elaboración propia.

6.2.3. Costo unitario del prototipo

A continuación, se describen los costos unitarios de los elementos del sistema. Se incluye el costo de envío ya que en Nicaragua no se tuvo facilidad de encontrar algunos de los componentes en tiempo y forma. Se optó por comprarlos por medio de Amazon y se tuvo un tiempo de entrega de 2 semanas aproximadamente.

Tabla 3 - Costo del proyecto.

Costo del sistema	
Producto	**Precio**
Raspberry Pi 4 Modelo B 2019	$170
Enokay Raspberry Pi 4B Power Supply	$10
SanDisk 64Gb Ultra microSDHC UHS-I Memory Card	$9
SmartiPi Touch 2 – Case for the Official Raspberry Pi 7” Touchscreen Display – with Cooling Fan	$30
Osoyoo 7 Inch DSI Touch Screen LCD Display 800x480 for Raspberry Pi	$50
Raspberry Pi Camera Module V2-8 Megapixel, 1080p	$35
Costo de envío	$11
Mano de obra	$150
Precio total	**$465**

6.3. Fase experimental

Mediante pruebas experimentales nos percatamos de que ciertos parámetros necesitan limitarse para un mejor funcionamiento del sistema.

- La variable que nos permite analizar el comportamiento de los parpadeos es EAR, por medio de pruebas se observó que el valor de apertura cambia en dependencia del usuario. El valor promedio de apertura que se encontró en ellos fue de 0.30 y el valor de cierre aproximado fue de 0.15, determinando así que el ojo se encuentra abierto cuando EAR se encuentra por encima de 0.15 y está cerrado cuando está por debajo de este.
- El valor de EAR es variable en dependencia de la posición del rostro del usuario ya que se mide la relación entre los puntos y varía según la posición y la orientación de la cabeza. Por ejemplo, si el rostro está inclinado hacia la izquierda, la distancia entre el párpado inferior y superior es menor que cuando el rostro está en posición frontal.
- El cálculo del promedio ocular fue la manera con la que se ubicó al pasajero viendo hacia el frente encontrando la pendiente que hay entre el dorso de la nariz hasta el extremo del ojo izquierdo, de tal forma que, cuando se esté viendo frente a la cámara el valor obtenido de posición será de 0.5.
- Si el usuario mueve el rostro hacia la izquierda tendrá un alcance hasta 0.30 y hacia la derecha de 1.25, dentro de este rango el valor de EAR no cambia drásticamente y permite un mejor análisis, en caso de encontrarse fuera de esos valores se emite el mensaje “Posición incorrecta”.
- En base al apartado 6.1.5 “Discretización”, observamos que un parpadeo normal dura en promedio de 100 a 150 milisegundos, sin embargo, en estado de somnolencia estos valores se ven afectado en promedio a 175 a 250 ms y a su vez los microsueños oscilan entre 2 a 10 segundos, por esta razón se estableció que el sistema captará microsueños cuando los ojos se encuentren cerrados en un tiempo de 1.5 segundos, ya que sería el tiempo que englobaría un estado de somnolencia y entrará en un microsueño.

6.4. Fase de evaluación

Se utiliza el muestreo no probabilístico por conveniencia ya que se escogen los participantes de la muestra según la facilidad de acceso. Se tomaron las siguientes consideraciones para analizar los resultados obtenidos en las pruebas:

6.4.1. Especificaciones de diseño

- El sistema está diseñado para condiciones de iluminación continua.
- Se debe evitar obstruir la vista con cualquier objeto (lente, máscara y/o material opaco) que dificulte la captura del rostro.
- La cámara debe colocarse a la altura de los ojos lo más recto posible donde EAR alcance un valor de 0.30 con los ojos abiertos.
- Se realizaron las pruebas con 20 voluntarios en edades que oscilan entre los 18 a 80 años con experiencia en conducción.
- Los voluntarios se encontraban sentados entre 50 cm a 60 cm de la cámara, lo más centrado posible a ella.
- La ubicación de la cámara desde el piso: 100 cm a 115 cm aprox. y la ubicación de sus ojos desde el piso: 110 cm a 120cm aprox.
- Las pruebas se realizaron entre las 8:00 am y las 5:00 pm, no se realizaron pruebas de noches ya que el sistema no soporta este tipo de condición.
- Se analizaron 2 tipos de pruebas: rostro sin accesorios y rostro con accesorio (utilizando gorra y/o sombrero). En cada prueba se realizaron 3 subrutinas: rostro de frente, rostro con posición hacia la izquierda y posición hacia la derecha.
- Cada subrutina tuvo una duración de 1 minuto aproximadamente, se les solicitó a los voluntarios que generaran un microsueño aleatorio durante ese tiempo, el cual sería registrado por el sistema marcando "✓". Se consideró un falso positivo cuando se registró una acción que no correspondía y un falso negativo cuando se quiso registrar un evento y no se contabilizó siendo marcado con una "**X**".
- La eficacia se expresa como un porcentaje y se calcula dividiendo el número de aciertos (p) entre el número total de pruebas (n) por 100%:

$$\boldsymbol{Eficacia}(\%) = \left(\frac{\boldsymbol{p}}{\boldsymbol{n}}\right) * \mathbf{100}\%$$

6.4.2. Resultados obtenidos de cada voluntario

Tabla 4 - Resultado voluntario 1

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 1	✓	✓	✓	✓	✓	✓
Eficacia	100%			100%		
Eficacia Total	100%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 5 - Resultado voluntario 2

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 2	✓	✓	✓	✓	✓	X
Eficacia	100%			66.66%		
Eficacia Total	83.33%					

En las 6 subrutinas se obtuvo una incidencia en la posición del rostro hacia la derecha con accesorio teniendo un falso negativo ya que no se registró el evento solicitado, se consideran satisfactorias las otras 5 otras subrutinas.

$$\boldsymbol{Eficacia}(\%) = \left(\frac{\mathbf{5}}{\mathbf{6}}\right) * \mathbf{100}\% = \mathbf{83.88}\%$$

Tabla 6 - Resultado voluntario 3

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 3	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100%		
Eficacia Total	100%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 7 - Resultado voluntario 4

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 4	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100%		
Eficacia Total	100 %					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%

Tabla 8 - Resultado voluntario 5

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 5	✓	✓	✓	X	✓	X
Eficacia	100.00%			33.33%		
Eficacia Total	66.67%					

En las 6 subrutinas se tuvieron 2 lecturas con falsos positivos por lo que se consideran 4 subrutinas positivas teniendo una eficacia del 66.67%.

Tabla 9 - Resultado voluntario 6

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 6	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100%		
Eficacia Total	100%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 10 - Resultado voluntario 7

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 7	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%

Tabla 11 - Resultado voluntario 8

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 8	✓	✓	X	✓	✓	✓
Eficacia	66.66%			100%		
Eficacia Total	83.33%					

En las 6 subrutinas se obtuvo un falso negativo cuando el rostro del voluntario estaba posicionado hacia la derecha, con un total de eficacia de 83.33%.

Tabla 12 - Resultado voluntario 9

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 9	✓	X	✓	✓	X	✓
Eficacia	66.66%			66.66%		
Eficacia Total	66.66%					

En las 6 subrutinas se tuvieron 2 lecturas con falsos positivos, por lo que se consideran 4 subrutinas positivas.

Tabla 13 - Resultado voluntario 10

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 10	✓	X	✓	X	X	✓
Eficacia	66.66%			33.33%		
Eficacia Total	50%					

En las 6 subrutinas se tuvieron 3 incidencias entre falsos negativos y falsos positivos y 3 resultados correctos, cabe destacar que el ojo de este voluntario de 75 años era más pequeño que el promedio.

Tabla 14 - Resultado voluntario 11

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 11	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 15 - Resultado voluntario 12

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 12	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%

Tabla 16 - Resultado voluntario 13

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 13	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 17 - Resultado voluntario 14

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 14	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 18 - Resultado voluntario 15

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 15	✓	✓	✓	✓	✓	X
Eficacia	100.00%			66.66%		
Eficacia Total	83.33%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 19 - Resultado voluntario 16

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 16	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 20 - Resultado voluntario 17

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 17	✓	✓	X	✓	✓	X
Eficacia	66.66%			66.66%		
Eficacia Total	66.66%					

En las 6 subrutinas se obtuvieron 2 falsos positivos en la posición derecha con y sin accesorios.

Tabla 21 - Resultado voluntario 18

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 18	✓	X	✓	✓	X	✓
Eficacia	66.66%			66.66%		
Eficacia Total	66.66%					

En las 6 subrutinas se obtuvieron 2 falsos positivos en la posición izquierda con y sin accesorios.

Tabla 22 - Resultado voluntario 19

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 19	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

Tabla 23 - Resultado voluntario 20

	Rostro sin accesorio			Rostro con accesorio		
	Frente	Izquierda	Derecha	Frente	Izquierda	Derecha
Voluntario 20	✓	✓	✓	✓	✓	✓
Eficacia	100.00%			100.00%		
Eficacia Total	100.00%					

En las 6 subrutinas se tuvieron resultados satisfactorios por lo que se obtiene una eficacia total del 100%.

6.4.3. Cálculo de eficacia total

Tabla 24 - Resultados generales y obtención del cálculo de eficacia total del sistema

	Rostro sin accesorio			**Rostro con accesorio**			**Eficacia sin accesorio**	**Eficacia con accesorio**	**Eficacia total**
	Frente	Izquierda	Derecha	**Frente**	**Izquierda**	**Derecha**			
Voluntario 1	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 2	✓	✓	✓	✓	✓	X	100.00%	66.66%	83.33%
Voluntario 3	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 4	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 5	✓	✓	✓	X	✓	X	100.00%	33.33%	66.67%
Voluntario 6	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 7	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 8	✓	✓	X	✓	✓	✓	66.66%	100.00%	83.33%
Voluntario 9	✓	X	✓	✓	X	✓	66.66%	66.66%	66.66%
Voluntario 10	✓	X	✓	X	X	✓	66.66%	33.33%	50.00%
Voluntario 11	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 12	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 13	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 14	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 15	✓	✓	✓	✓	✓	X	100.00%	66.66%	83.33%
Voluntario 16	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 17	✓	✓	X	✓	✓	X	66.66%	66.66%	66.66%
Voluntario 18	✓	X	✓	✓	X	✓	66.66%	66.66%	66.66%
Voluntario 19	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
Voluntario 20	✓	✓	✓	✓	✓	✓	100.00%	100.00%	100.00%
	100%	**85%**	**90%**	**90%**	**85%**	**80%**	**91.67%**	**85.00%**	**88.33%**

VII. ANÁLISIS Y PRESENTACIÓN DE LOS RESULTADOS

Los resultados obtenidos de los individuos que están viendo de frente al sistema sin accesorios es del 100% en comparación con rostros con accesorios, el cual es del 90%, es decir, se considera que estas pruebas son exitosas.

De las pruebas realizadas con los individuos con el rostro posicionado hacia la izquierda con y sin accesorios es del 85% obteniendo el mismo resultado en ambas categorías, cabe destacar que los individuos que obtuvieron resultados negativos son de la tercera edad, esto se pude ver relacionado que la relación de aspectos de sus ojos era menor al promedio, lo que significa que sus parpados se ven decaídos por los cambios fisiológicos en su rostro asociado al envejecimiento en comparación con otros voluntarios que se consideran adultos jóvenes (18-40 años).

El resultado obtenido de los individuos con el rostro posicionado hacia la derecha sin accesorios fue del 90%, es decir, se considera esta prueba exitosa, mientras que con accesorios fue del 80%, cabe señalar que este resultado fue variable debido a que se registró falsos positivos y negativos en las lecturas realizadas.

En resumen, se tiene como resultado la eficacia sin accesorios del 91.67% y con accesorios del 85% para obtener como resultado la eficacia total del sistema como el promedio aritmético de las pruebas realizadas con rostro sin accesorio y rostro con accesorio. De los resultados obtenidos de los voluntarios se tiene:

$$\mathbf{Eficacia}(\%) = \left(\frac{\%\mathbf{Rostro\ sin\ accesorio} + \%\mathbf{Rostro\ con\ accesorio}}{\mathbf{2}}\right)$$

$$\mathbf{Eficacia}(\%) = \left(\frac{\mathbf{91.67\%} + \mathbf{85\%}}{\mathbf{2}}\right) = \mathbf{88.33\%}$$

VIII. CONCLUSIONES Y RECOMENDACIONES

8.1. Conclusiones

Se llevó a cabo el análisis de los fundamentos teóricos empleados en la somnolencia y visión por computador. Como resultado, se decidió abordar la somnolencia de tipo subjetiva a través del análisis de expresiones faciales, específicamente, el estudio del comportamiento de los parpadeos por medio de la relación de apertura de los ojos (EAR).

Se desarrolló un algoritmo que es capaz de detectar y rastrear el rostro del conductor, determinando su estado de vigilia (despierto, somnoliento o dormido). El sistema de detección de somnolencia diseñado incluye una interfaz gráfica de usuario (GUI) la cual presenta alarmas visuales y sonoras, estas se activan en caso en caso de detectar signos de somnolencia, promoviendo una conducción más segura y reduciendo el riesgo de accidentes.

Durante la primera versión del prototipo, se logró desarrollar un sistema no intrusivo, compacto, ligero y de bajo costo. Además, una de las características importantes a destacar fue la optimización del rendimiento del sistema donde se redimensionó el tamaño del vídeo, disminuyendo así el tiempo de procesamiento de los datos de entrada, por ende, el procesamiento en las imágenes consume menos recursos en comparación a otras técnicas de visión por computador, lo que significa una ventaja ya que, a menor potencia, menor consumo de recursos.

Finalmente, se realizaron pruebas a 20 voluntarios, en los cuales cada uno realizó 6 subrutinas con una duración de 1 minuto dónde se les solicitó generar un microsueño de forma aleatoria en ese tiempo, logrando una eficacia en la detección del 88.33%. Estos resultados confirman la efectividad del sistema, teniendo el potencial de reducir significativamente los accidentes causados por la somnolencia o los microsueños, el sistema puede ser utilizado en áreas más allá de la conducción particular, por ejemplo, en transporte comercial, control de seguridad (como aeropuertos) e incluso en el campo de la investigación médica para el estudio y análisis patrones de sueño y somnolencia en pacientes.

8.2. Recomendaciones

En base a las conclusiones que hemos obtenidos, se pueden sugerir las siguientes recomendaciones:

Evitar generalizar un umbral de cierre para EAR ya que este puede ser diferente según la forma del ojo del usuario, se recomienda investigar sobre otras técnicas como las redes neuronales convolucionales (CNN) para mejorar la detección de ojos abiertos o cerrados.

Se recomienda mejorar la detección que brinda el sistema en condiciones de poca iluminación o donde se reflejen sombras. Para lograrlo, se podrían utilizar técnicas de iluminación artificial o mejorar los algoritmos de detección de puntos de referencia.

El sistema puede ser escalable tanto en software como en hardware, lo que permite que pueda ser ampliado para incorporar otros elementos, por ejemplo:

- Las pruebas del prototipo se llevaron a cabo en un entorno controlado bajo condiciones específicas, se sugiere ampliar las pruebas a una implementación real en un vehículo, con el fin de evaluar como el sistema se comportaría respecto a parámetros como la vibración y los movimientos constantes del vehículo.
- Se podría considerar la implementación de una cámara infrarroja para que el sistema tenga la capacidad de detección de somnolencia nocturna, esta mejoraría la identificación del rostro en situaciones con poca iluminación. Además, elegir una cámara inalámbrica podría optimizar la eficiencia del sistema en ubicación e instalación en el vehículo.
- Se sugiere llevar a cabo un estudio acerca del ángulo de inclinación y distancia máxima a la cual pueda estar ubicada la cámara dentro del vehículo, sin que estos parámetros afecten la detección correcta de la somnolencia.

- Se recomienda que a partir del rango de valores obtenidos en la pendiente ocular o la detección del rostro se pueda desarrollar un sistema que alerte al usuario ante distracciones, por ejemplo, ante el uso del celular.
- Con base en los estudios mencionados en la etapa de discretización, se ha tomado la decisión de establecer la duración del tiempo de parpadeo en 1.5 segundos. No obstante, es importante destacar que este tiempo puede ser ajustado a un valor inferior considerando un escenario más realista teniendo en cuenta el tiempo que llevaría un objeto impactar a una velocidad determinada.
- Se recomienda la evaluación de otros patrones de comportamiento como los bostezos y cabeceos, en conjunto con la combinación de otras tecnologías para robustecer el sistema, como, sensores de movimiento, monitoreo del ritmo cardíaco, y patrones de conducción para detectar soñolencia con mayor precisión.
- Para mejora del rendimiento ante la incorporación de nuevos elementos, se podría considerar aumentar el hardware del sistema, utilizando una tarjeta de desarrollo como JETSON NANO NVIDIA y la GOOGLE CORAL, las cuales están diseñadas específicamente para aplicaciones de inteligencia artificial y aplicaciones de visión por computador.

Se invita a continuar investigando y desarrollando nuevas técnicas y algoritmos para la detección de somnolencia, con el objetivo de mejorar la precisión y la confiabilidad del sistema en el futuro, ya que existen diversos métodos y técnicas basados en inteligencia artificial para estudio de este tema, lo cual indica que no hay una sola forma de abordar esta aplicación.

IX. REFERENCIAS BIBLIOGRÁFICAS

[1] Arun Sahayadhas, Kenneth Sundaraj and Murugappan Murugappan, Detecting Driver Drowsiness Based on Sensors: A Review, Malaysia: IEEE, 2012.

[2] M. J. F. Calero, Sistema avanzado de asistencia a la conducción mediante visión por computador ára la detección de la somnolencia, Leganés: Universidad Carlos III de Madrid, 2009.

[3] M. P. F. Guillermo, «Prototipo de un sistema detector de somnolencia con alerta vía tuits para conductores vehiculares,» Universidad de Guayaquil, Guayaquil, Ecuador, 2017.

[4] Tereza Soukupova and Jan Cech, «Real-Time Eye Blink Detection using Facial Landmarks,» Czech Technical University in Prague, Prague, 2016.

[5] H. Díaz, «Informe sobre el Estado Mundial de la Seguridad Vial 2018,» 12 12 2018. [En línea]. Available: https://fundadeps.org/recursos/Informe-sobre-el-Estado-Mundial-de-la-Seguridad-Vial-2018/#:~:text=La%20Organizaci%C3%B3n%20Mundial%20de%20la%20Salud%20%28OMS%29%20ha,entre%20las%20personas%20de%205%20a%2029%20a%C3%B1os..

[6] F. Fajardo, «El 2021 cerró con 904 fallecidos en accidentes de tránsito,» Vos TV, 31 01 2022. [En línea]. Available: https://www.vostv.com.ni/nacionales/22076-el-2021-cerro-con-904-fallecidos-en-accidentes-de-/.

[7] Carlos Gómez Restrepo, Martín Rondon, Alvaro Ruiz, Juan Lozano, Juliana Guzman and Felipe Macías, «Blood Alcohol Concentration and Somnolence among Drivers Studied in Simulators: A Meta-Analysis,» Revista Colombiana de Psiquiatria, Colombia, 2011.

[8] María Agustina Garcés, José De Jesus Salgado, Jesus Andres Cruz and William Henry Cañón, «Sistemas de detección de somnolencia en conductores: inicio, desarrollo y futuro,» Revista Ingeniería y Región. 2015;13(1):159-168, 2015.

[9] RAE, «Somnolencia,» 2021. [En línea]. Available: https://dle.rae.es/somnolencia.

[10] Edmundo Rosales Mayor and Jorge Rey De Castro Mujica, «Somnolencia, Qué es, qué la causa y cómo se mide,» Acta Médica Peruana, 2010.

[11] L. Biolatto, «¿En qué consiste la somnolencia o fatiga diurna?,» 09 04 2022. [En línea]. Available: https://mejorconsalud.as.com/en-que-consiste-somnolencia-fatiga-diurna/.

[12] L. R. Mitjana, «Somnolencia: síntomas, causas y tratamientos,» 14 01 2021. [En línea]. Available: https://muysalud.com/enfermedades/somnolencia-sintomas-causas-tratamientos/.

[13] Juan Carlos Crespin Luis and Raúl Alexander Julián García, «Sistema detector de somnolencia en secuencias de vídeo de conductores manejando usando visión computacional,» Universidad Nacional de Trujillo, Trujillo, Perú, 2019.

[14] F. Checked, «¿Qué son los microsueños y cómo controlarlos?,» MejorconSalud, 20 Febrero 2023. [En línea]. Available: https://mejorconsalud.as.com/microsuenos-controlarlos/.

[15] Byung-Chan Chang, Jung-Eun Lim, Hae-Jin Kim and Bo-Hyeok Seo, A study of classification of the level of sleepiness for the drowsy driving prevention, Japan: Kagawa University, 2007.

[16] Carlos Guillermo Chaccere Rodríguez and Fidel Junior Sara García, «Diseño y simulación de un sistema de detección de somnolencia y alerta basado en el procesamiento digital de imágenes con algoritmos de correlación en tiempo real,» Universidad de San Martín de Porres, Lima, Perú, 2015.

[17] Jhon Iván Pilataxi and Willyam Marcelo Viñán Robalino, Diseño e implementación de un sistema de asistencia a la conducción de un robot tipo car-like ante la fatiga del usuario, Quito: Escuela Politécnica Nacional de Ecuador, 2014.

[18] T. Xiaomeng, «La solución de seguridad activa para la conducción inteligente de Quectel hace que la conducción irregular no tenga ningún lugar donde esconderse.,» 04 Abril 2019. [En línea]. Available: http://www.cww.net.cn/article?id=451123.

[19] Pia M. Forsmana, Bryan J. Vilaa,b, Robert A. Short c, Christopher G. Mott d, Hans P.A. Van Dongena, «Efficient driver drowsiness detection at moderate levels of drowsiness,» Research Center, Washington, Washington, 2012.

[20] H. A. C. Siguencia, «Sistema de detección de sueño en el vehículo,» Azogues, Ecuador, 2021.

[21] N. Wario, «Visión por Computadora o Visión Computación: Guía 2021,» BRITA, Inteligencia Artificial, 10 Enero 2021. [En línea]. Available: https://brita.mx/vision-por-computadora-o-vision-computacion-guia-2021.

[22] ICHI.PRO, «Introducción a CNN y clasificación de imágenes usando CNN en PyTorch,» 2020. [En línea]. Available: https://ichi.pro/es/introduccion-a-cnn-y-clasificacion-de-imagenes-usando-cnn-en-pytorch-19718109321276.

[23] V. Tabora, «Face Detection Using OpenCV With Haar Cascade Classifiers | by Vincent Tabora | Becoming Human: Artificial Intelligence Magazine,» 01 Febrero

2019. [En línea]. Available: https://becominghuman.ai/face-detection-using-opencv-with-haar-cascade-classifiers-941dbb25177.

[24] Pystyle, «OpenCV – Clasificador en cascada CascadeClassifier para detectar rostros y ojos a partir de imágenes,» 08 05 20202. [En línea]. Available: https://pystyle.info/opencv-cascade-classifier/.

[25] F. Ruh, «Aplicaciones de visión por computadora,» 20 Junio 2022. [En línea]. Available: https://www.linkedin.com/pulse/aplicaciones-de-visi%C3%B3n-por-computadora-frida-ru%C3%ADz/?originalSubdomain=es.

[26] G. bcnvision.blog, «Sistemas de iluminación para aplicaciones de visión artificial,» 11 Abril 2017. [En línea]. Available: https://www.bcnvision.es/blog-vision-artificial/iluminacion-vision-artificial2/.

[27] R. C. Catari, «Diseño de un algoritmo de procesamiento de imágenes del sistema de pesaje para el contro automático de una faja transportadora en la unidad minera Mallay,» Universidad Nacional del Altiplano, Puno, Perú, 2019.

[28] Miguel de la Cruz José Porras, «CLASIFICATION SYSTEM BASED ON COMPUTER VISION,» [En línea]. Available: https://www.urp.edu.pe/pdf/id/2881/n/clasification-system-based-on-computer-vision.

[29] W. Garage, «Github,» Intel Corporation, 23 December 2017. [En línea]. Available: https://github.com/topics/opencv?l=python.

[30] F. Cardellino, «La guía definitiva del paquete NumPy para computación científica en Python,» freeCodeCamp, 20 Marzo 2021. [En línea]. Available: https://www.freecodecamp.org/espanol/news/la-guia-definitiva-del-paquete-numpy-para-computacion-cientifica-en-python/.

[31] Dlib, «Dlib C++ Library,» 08 03 2022. [En línea]. Available: http://dlib.net/.

[32] F. Ucha, «Definición de redimensionar,» 05 2015. [En línea]. Available: https://www.definicionabc.com/general/redimensionar.php.

[33] Emerson Mayon Sanchez and Raul Limaquispe Miguel, «Sistema de detección de somnolencia mediante inteligencia artificial en conductores de vehículos para alertar la ocurrencia de accidentes de transito,» Universidad Nacional de Huancavelica, Huancavelica, 2018.

[34] D. E. King, «Dlib-ml: A machine learning toolkit,» *Journal of Machine Learning Research,* pp. 1755-1758, 10 julio 2009.

[35] N. &. T. B. Dalal, «Histograms of oriented gradients for human detection,» de *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, California, USA, 2005.

[36] V. K. a. J. Sullivan, «One Millisecond Face Alignment with an Ensemble of Regression Trees,» de *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, Jun. 2014.

[37] «Adjusting eye aspect ratio for strong eye blink detection based on facial landmarks,» Creative Commons , 2022.

[38] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou and M. Pantic, «300 Faces in-the-Wild Challenge: The First Facial Landmark Localization Challenge,» 2013 IEEE International Conference on Computer Vision Workshops: doi: 10.1109/ICCVW.2013.59., Sydney, NSW, Australia, 2013.

[39] Reza Ghoddoosian, Marnim Galib and Vassilis Athitsos, «A Realistic Dataset and Baseline Temporal Model for Early Drownsiness Detection,» Conference on Computer Vision and Pattern Recognition Workshops IEEE, Texas, 2019.

[40] J. MW, «The Amplitude-Velocity Ratio of Blinks: A New Method for Monitoring Drowsiness,» Melbourne, Australia., 2003.

[41] H. A. C. Siguencia, «Sistema de detección de sueño en el vehículo,» Azogues, 2021.

[42] S. T. Y. K. M. &. M. Y. Asano, «Influence of eye fatigue on eye blinking: Measurements using EOG and an infrared sensor,» *Journal of Eye Movement Research,* vol. 10, nº 2, p. 2, 2017.

[43] K. S. Y. N. M. T. H. &. O. T. Aoyama, «The influence of eye drop instillation on ocular surface and blinking behavior,» de *Investigative Ophthalmology & Visual Science*, 2019.

[44] E. C. Y. S. C. L. I. T. &. T. L. C. Chua, *Increased blink interval is associated with difficulty in visual attention tasks in patients with sleep apnea,* 1 ed., vol. 39, Sleep, 2016, pp. 175-182.

[45] Y. K. Y. W. T. M. Y. &. H. K. Hayashi, *Blink parameters predict sleepiness in obstructive sleep apnea syndrome patients,* 1 ed., vol. 41, Sleep, 2018.

[46] O. J. P.-G. V. Gómez-Jauregui V, *Microsleep detection through the joint analysis of EEG and EOG signals,* 236 ed., vol. 11, Sleep, 2020.

[47] D. D. Lim J, *A meta-analysis of the impact of short-term sleep derivation on cognitive variables,* 3 ed., vol. 136, Psychol Bull, 2010.

[48] B. D. Trotti LM, *No increased risk of next-day car crashes in patients with insomnia-disordered breathing comorbidities,* 3 ed., vol. 41, Sleep, 2018.

X. ANEXOS

1. Variables globales del código

```
#------------------------------VARIABLES--------------------------
# Parámetros de alineación de rostro
pos_ojoizq = (0.2, 0.2)
ancho_rostro = 120
alto_rostro = 120

# Valor limite de EAR
valorlim_EAR = 0.15

# Número de frames para calcular el promedio del EAR
num_frames = 15

# Inicializamos los parpadeos en 0
parpadeo = 0

# Inicializamos la duración del parpadeo en 0
parp_duracion = 0

# Variables para almacenar el tiempo en que se detecta un parpadeo
parp_inicio = None
parp_final = None

# Lista para almacenar los valores la pendiente ocular
valores_mo = []

# Lista para almacenar los valores de EAR
valores_EAR = []

# Lista para almacenar los tiempos de parpadeos
parp_tiempo = []


 # Variable de control para el bucle while
 running = True

 # Variable para el conteo de microsueños
 microsueno = 0
 ya_contado = False

 # Fuente de textos
 fuente = cv2.FONT_HERSHEY_SIMPLEX
```

2. Montaje de la carcasa del prototipo

Se trata de una carcasa para pantalla con soporte, puerta con ventilador, donde se ubica la Rapsberry Pi, cables de cinta para pantalla y cámara, panel frontal para poner cámara o no, etc.

El primer paso es la instalación de la cámara sobre el soporte de pantalla, para ellos se ubica en el compartimiento correspondiente a la cámara (cabe destacar que como una primera versión se hizo un sistema compacto, sin embargo, la cámara debería ir externa, con un soporte en el vidrio frontal del vehículo) esta se fija con los dos tornillos correspondientes:

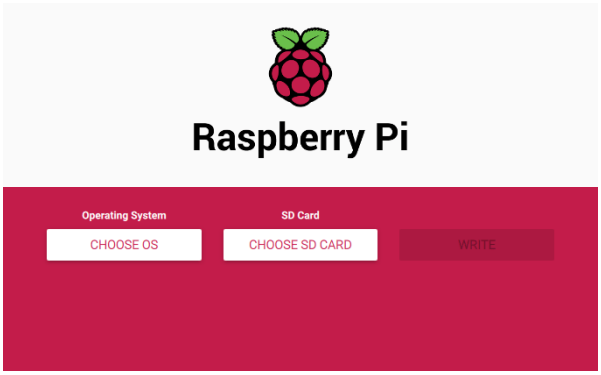
El segundo paso es montar la pantalla de 7 pulgadas sobre el soporte. Hay que destacar que tanto los cables planos flexibles de la pantalla y cámara tienen que ir como se muestran en las siguientes figuras. El tercer paso es insertar la Raspberry Pi en su lugar, lo que es intuitivo debido a que los puertos quedan expuestos. A continuación, se conectan los cables DSI para la pantalla y CSI para la cámara, y se conecta el ventilador de refrigeración a los GPIO de voltaje en "alta velocidad".

El ventilador también sirve como cubierta para proteger la Raspberry Pi dentro del estuche.

Como último paso se asegura el soporte de pantalla en su base. Lo ideal es que luego sea instalado en el tablero del vehículo ya que trae los orificios pertinentes para ser asegurado al mismo.

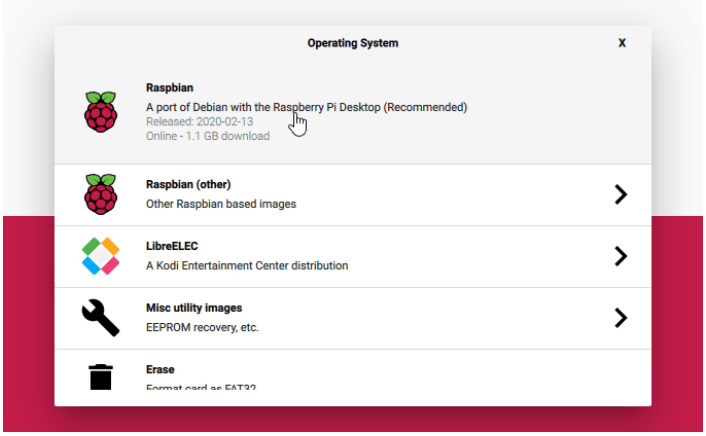
3. Instalación del sistema operativo

Se conecta la tarjeta Micro SD al ordenador y se abre el software Raspberry Pi Imager:

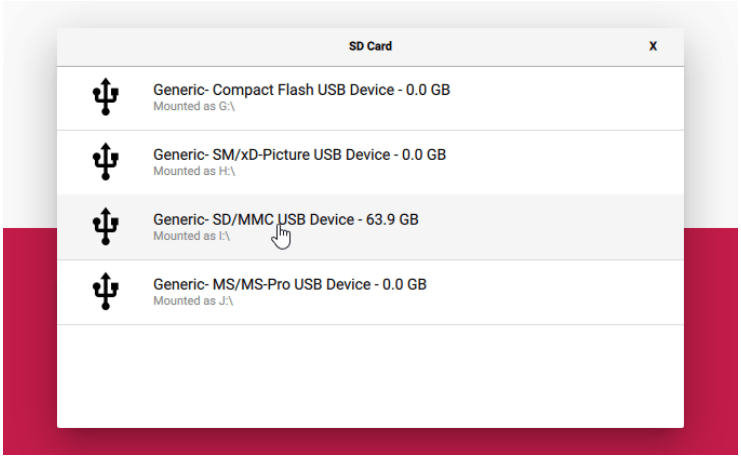


Se presiona sobre el botón CHOSSE OS y se selecciona el sistema operativo de preferencia, en este caso se seleccionó la opción recomendada clicando sobre la opción Raspbian.

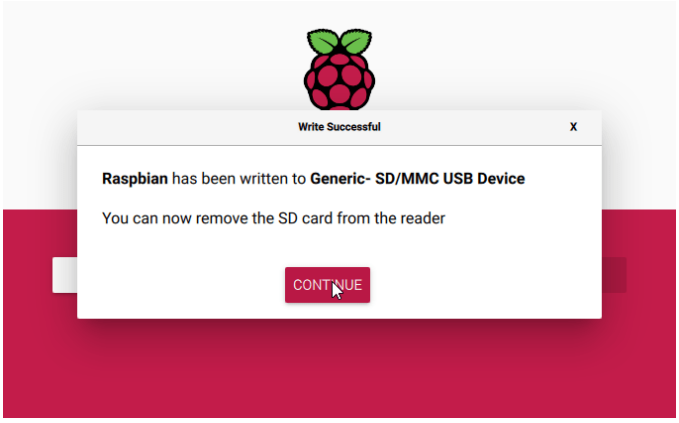
.



A continuación, se presionó el botón CHOSSE SD CARD y se selecciona la tarjeta Micro SD en la que se desea instalar Raspbian.



Finalmente, se tiene que clicar sobre el botón WRITE. Y de esta manera solo se procede a ingresar la SD a la Raspberry Pi.



Ya una vez que se ha configurado el sistema operativo, se procede a la actualización de todos los paquetes en la Raspberry Pi mediante “sudo apt-get update”, este comando actualiza el listado con todos los paquetes que se tiene disponibles para instalar, luego usando “sudo apt-get upgrade” se actualizarían los programas, pero no el sistema operativo.

4. Instalación de librerías

Cabe señalar que estas librerías son utilizadas para la creación del algoritmo, para la instalación del programa, solo se necesita copiar y pegar la carpeta del archivo ejecutable. Para la instalación inicial se verificó la versión de Python que este instalada y debido a que se trabajó con Python3, se escribe en la terminal:

```
pi@raspberrypi:~ $ python3
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Librería CV2

Lo primero que se realizó fue la instalación de algunos paquetes que serán los prerrequisitos. Esto se logra mediante la siguiente línea: “sudo apt-get install libhdf5-dev libhdf5-serial-dev libatlas-base-dev libjasper-dev libqtgui4 libqt4-test”.

```
pi@raspberrypi:~ $ sudo apt-get install libhdf5-dev libhdf5-serial-dev libatlas-
base-dev libjasper-dev libqtgui4 libqt4-test
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Nota, seleccionando «libhdf5-dev» en lugar de «libhdf5-serial-dev»
El paquete indicado a continuación se instaló de forma automática y ya no es nec
esario.
  rpi-eeprom-images
Utilice «sudo apt autoremove» para eliminarlo.
Se instalarán los siguientes paquetes adicionales:
  hdf5-helpers libaec-dev libaec0 libatlas3-base libaudio2 libhdf5-103
  libhdf5-cpp-103 libjasper1 libjpeg-dev libjpeg62-turbo-dev libjpeg8 libmng1
  libqt4-dbus libqt4-xml libqtcore4 libqtdbus4 libsz2 qdbus qt-at-spi
  qtchooser qtcore4-l10n
Paquetes sugeridos:
  libatlas-doc liblapack-doc nas libhdf5-doc libjasper-runtime libicu57
  qt4-qtconfig
```

Ahora bien, para la instalación de OpenCV se usó “pip3 install opencv-contrib-python” y la versión de OpenCV que deseamos en nuestro caso es la versión 4.5.5. Nota: Se usó pip3 ya que vamos a instalar Opencv en Python 3.

```
pi@raspberrypi:~ $ pip3 install opencv-contrib-python==4.1.0.25
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting opencv-contrib-python==4.1.0.25
  Downloading https://www.piwheels.org/simple/opencv-contrib-python/opencv_contr
ib_python-4.1.0.25-cp37-cp37m-linux_armv7l.whl (15.7MB)
    100% |████████████████████████████████| 15.7MB 17kB/s
Requirement already satisfied: numpy>=1.16.2 in /usr/lib/python3/dist-packages (
from opencv-contrib-python==4.1.0.25) (1.16.2)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.1.0.25
```

Por último, se comprobó que se ha instalado OpenCV, digitamos python3 en el terminal e importamos OpenCV para finalmente imprimir la versión instalada.

```
pi@raspberrypi:~ $ python3
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.5.5'
```

Librería DLIB

Lo primero fue la instalación de los prerrequisitos necesarios, igual que Opencv, Dlib necesita de unas dependencias, la cuales se instalaron mediante las siguientes líneas de comandos: “sudo apt-get update” “sudo apt-get install build-essential cmake” “sudo apt-get install libgtk-3-dev” “sudo apt-get install libboost-all-dev”.

```
pi@raspberrypi:~ $ pip3 install dlib
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: dlib in /usr/local/lib/python3.9/dist-packages/dlib
-19.24.0-py3.9-linux-armv7l.egg (19.24.0)
pi@raspberrypi:~ $
```

Ahora bien, para lo que fue la instalación usamos pip3 debido a que se trabajó en Python 3, se utilizaron las siguientes líneas de comando: “pip3 install numpy” “pip3 install scipy” “pip3 install scikit-image” “pip3 install dlib”. Por último, se comprobó que se ha instalado Dlib, de la siguiente manera:

```
pi@raspberrypi:~ $ python3
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import dlib
>>> dlib.__version__
'19.24.0'
```

Librería Imutils

Para esta librería fue necesario ingresar la siguiente línea de comando en el terminal: “pip3 install imutils”.

```
pi@raspberrypi:~ $ pip3 install imutils
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: imutils in ./.local/lib/python3.9/site-packages (0.
5.4)
pi@raspberrypi:~ $
```

Librería NumPy

Todos los requisitos para instalar NumPy ya están cumplidos, así que ahora finalmente podemos instalar NumPy. Para instalar NumPy, solo se auxilió del siguiente comando: “sudo pip3 install numpy”.

```
pi@raspberrypi:~ $ sudo pip3 install numpy
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: numpy in /usr/lib/python3/dist-packages (1.19.5)
pi@raspberrypi:~ $
```

Por último, se hace la comprobación que la librería se haya instalado correctamente.

```
pi@raspberrypi:~ $ pip show numpy
Name: numpy
Version: 1.24.1
Summary: Fundamental package for array computing in Python
Home-page: https://www.numpy.org
Author: Travis E. Oliphant et al.
Author-email:
License: BSD-3-Clause
Location: /home/pi/.local/lib/python3.9/site-packages
Requires:
Required-by: microdotphat, opencv-contrib-python, scrollphathd
pi@raspberrypi:~ $
```

5. Voluntarios haciendo pruebas con el dispositivo