Escuela Técnica Superior de Ingeniería de Telecomunicación

Universidad Politécnica de Cartagena

Desarrollo de un sistema inteligente de detección de fatiga en conductores

TRABAJO FIN DE ESTUDIOS

GRADO DE INGENIERÍA EN SISTEMAS DE TELECOMUNICACIÓN

Autor: Pedro Javier García Paterna
Directora: María Francisca Rosique

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi tutora del Trabajo Fin de Estudios, María Francisca Rosique, toda la ayuda y asesoramiento durante la elaboración de este proyecto, así como por los trámites realizados para que finalmente pueda presentar este trabajo.

Además, agradecer a todos los compañeros de grado que me he encontrado por el camino, y en especial a mi compañera María, por saber sacar siempre lo mejor de mi incluso en plena convocatoria de exámenes.

A mi familia y amigos, por siempre mostrarme el camino a seguir y en especial a mis padres, por permitirme estudiar lo que siempre ha sido mi vocación y por educarme siguiendo un camino cuyo final dirige a mis sueños.

Por último, quería agradecer a todo el personal docente y, en general, a la Universidad Politécnica de Cartagena por haberme formado de una manera tan ajena a la vez que eficiente, sin poner límites a mis oportunidades.

¡Gracias a todos!

Contenido

Tabla de figuras

1. Introducción

1.1. Motivación

En pleno siglo XXI, el automóvil es considerado una pieza clave en la historia de la evolución de la sociedad, ya que está involucrado en casi todos los aspectos de la vida moderna. La llegada del automóvil a la civilización ha permitido el desarrollo de grandes redes de distribución a nivel internacional, el despliegue de servicios públicos de emergencias, servicios de transporte e innumerables utilidades en todos los ámbitos. Pero también ha traído consigo una de las mayores causas de muerte en la sociedad, los accidentes de tráfico. Según un estudio realizado por la Dirección General de Tráfico, [1] alrededor de 1,25 millones de personas fallecen anualmente como consecuencia de accidentes en carretera, siendo los principales factores de riesgo:

- Velocidad
- Conducción bajo los efectos del alcohol y otras drogas
- No uso de sistemas de retención
- Distracción al volante
- Fatiga al volante
- Infraestructura vial insegura
- Vehículos inseguros
- Incumplimiento de las normas

Centrándonos en la fatiga como factor de riesgo a estudiar en este trabajo, se comienza con la definición en sí de este término. La definición pura de fatiga es “el cansancio experimentado después de un intenso y continuado esfuerzo físico o mental”[2]. La conducción se corresponde con un esfuerzo mental para mantener la atención, sobre todo en trayectos extensos y monótonos como autopistas. Este estado fisiológico está altamente relacionado con la somnolencia, cuya definición es “el estado entre sueño y vigilia donde todavía no se ha perdido la conciencia y se tiene la sensación de cansancio, pesadez, sueño y torpeza en los movimientos”. Aun así, no hay que confundir ambos conceptos, ya que la fatiga conduce al estado de somnolencia.

A veces, saber detectar la fatiga no es tan fácil como parece. Lo normal es que el sueño no se produzca de manera fulminante, pero sí es un mecanismo que, una vez lanzado, actúa rápidamente, por lo que es importante reconocer los primeros signos de la fatiga. [3]

- Picor en los ojos. Es uno de los primeros síntomas, junto con el aumento de la frecuencia de parpadeo.
- Sensación de inquietud. Antes de quedarse dormido, se cambiará de postura con frecuencia en el asiento.
- Aumenta el volumen de los sonidos. El instinto animal humano deja activo el sentido del oído cuando se duerme como mecanismo de alerta, antes de entrar en fase de sueño, se tendrá la sensación de oír el volumen de la radio y los sonidos en el coche más altos de lo normal.

- Bostezos. Es un síntoma claro y evidente de que se está fatigado y es necesario un descanso, sobre todo si se repiten o son más largos de lo normal.

La aparición de estos factores durante la conducción puede producir microsueños, lo que aumenta de manera desproporcionada la posibilidad de sufrir un accidente de tráfico. Un microsueño [4] es un episodio breve de sueño involuntario, que a menudo ocurre sin que la persona sea consciente de que está sucediendo. Los ojos se cierran al intentar permanecer despiertos durante una tarea monótona, como la conducción, resultando peligroso e incluso fatal.

Los datos recogidos en una campaña lanzada por Anfabra, DGT y el RACE [5], indican que el 70% de los conductores españoles asegura haber sufrido episodios de sueño, mientras que un 57% dice haber notado pérdidas de concentración y un 40% picor de ojos y visión borrosa.

Según un estudio realizado por el Real Automóvil Club de España [6], la fatiga y el cansancio se encuentran entre las primeras causas de los accidentes mortales en las vías españolas, causando hasta un 30% de los accidentes de tráfico en España. Esto nos lleva a afirmar que la fatiga es un factor de riesgo en la siniestralidad vial y por esto, es importante identificar sus síntomas y poner remedio en el caso de que aparezca.

Otro estudio realizado por la Fundación Línea Directa, [7] abordó 442.000 accidentes entre 2011 y 2015 con el fin de encontrar estadísticas fiables sobre la somnolencia como causa de un accidente, ya que normalmente estos datos se encuentran enmascarados en las distracciones al volante. Los resultados de este estudio concluyeron que los accidentes causados por la somnolencia o fatiga, han descendido paulatinamente sumando un total de 20.600 accidentes entre 2011 y 2015. Además, se consuma que el riesgo de morir en este tipo de incidentes es más del doble que si se sufre un accidente convencional, dejando un número de 800 fallecidos y 3.300 heridos graves entre los años en los que se realizó el estudio.

1.2. Objetivos

El objetivo general de este trabajo consiste en el desarrollo de un Sistema ADAS (Sistema Avanzado de Asistencia a la Conducción) orientado a la detección de fatiga en conductores, con el fin de mejorar la seguridad del automovilista minimizando el riesgo de ocasionar un accidente de tráfico.

En el estado del arte se estudian los distintos dispositivos utilizados hasta el momento para la detección de signos de fatiga y se puede distinguir entre dos técnicas muy desiguales. La primera consiste en la detección de fatiga a través de medidas fisiológicas del conductor, como la frecuencia cardiaca, lo que conlleva que el conductor esté monitorizado continuamente. Este tipo de técnicas son difíciles de desarrollar ya que es complicado encontrar un patrón que indique signos de fatiga, además de que suelen ser métodos invasivos e incomodos para el conductor. En cambio, la otra técnica utiliza métodos no invasivos extrayendo medidas del comportamiento del conductor, como la frecuencia de parpadeo o la apertura de los ojos, así como parámetros de la conducción

tales como los movimientos bruscos del volante o las correcciones de trayectoria que se ejecutan durante la conducción.

En este proyecto se utilizarán técnicas no invasivas con el fin de obtener distintos parámetros del conductor, a través del procesado de imágenes obtenidas en tiempo real desde una única cámara enfocada hacia el rostro del mismo. Estos parámetros pueden ser numerosos, pero con objeto de realizar un software ágil y robusto se han seleccionado los siguientes.

- Apertura de los ojos o PERCLOS.
- Detección de bostezos.

Una vez obtenidos los parámetros del automovilista, el siguiente paso consistirá en calcular una estimación del estado de fatiga del conductor en función de las medidas anteriores. Para el desarrollo del software de detección y procesado de imagen se utilizará la librería OpenCV 4.0.0 para Python 3.

2. Análisis de la fatiga

La fatiga es uno de los principales factores de riesgo junto con el consumo de bebidas alcohólicas, la distracción al volante y la velocidad. Se calcula que la fatiga ocasiona entre el 20 o 30% de los accidentes de manera directa o indirecta, según anuncia la Dirección General de Tráfico.

Centrándonos en la definición de fatiga, este concepto consiste en la disminución o pérdida de la actividad normal o habitual causada por un exceso de excitaciones y de trabajo, y que acaba en la aparición (manifiesta o no) de signos de deficiencia en la función muscular e intelectual [8]. Por lo que se pueden distinguir dos factores clave a la hora de detectarla, la fatiga puramente fisiológica, también llamada fatiga muscular, y la fatiga psíquica, causada fundamentalmente por el agotamiento intelectual.

La sensación de fatiga se divide en tres componentes clave: el cansancio o somnolencia, la disminución de la concentración y un conjunto de sensaciones como dolor de cabeza, malestar, mareos, etc. Estos se producen normalmente debido a un trabajo excesivo acompañado de un descanso inadecuado. Se puede afirmar que la fatiga crece de forma exponencial en relación al trabajo inadecuado que se realiza. Cabe destacar, que el trabajo inadecuado no se corresponde únicamente con el excesivo, sino también aquel que se repite de manera monótona y duradera o, mayormente, aquel que no se acompaña de un tiempo adecuado de sueño.

La conducción puede considerarse uno de estos trabajos inadecuados, sobre todo cuando se trata de largos trayectos donde se pueden producir algunos de estos factores: la repetición de estímulos, la contracción de determinados grupos musculares al mantener la posición, la necesidad de mantener la atención en la carretera o la estimulación simultanea del sujeto. Si a todo esto se le suma un escaso tiempo de descanso, se pronunciará la aparición de fatiga en el conductor, poniendo en peligro su seguridad y la de los demás conductores.

Para poder evitar la fatiga, el primer paso es saber reconocerla. Para ello se identifican los siguientes síntomas que aparecen con la fatiga en lo referente a la conducción:

- Disminución de la atención. La concentración al volante va disminuyendo conforme aumenta el trayecto, por lo que, para largos periodos de tiempo al volante, mantener la atención se convierte en una tarea difícil.
- Disminución de los reflejos. Ante una situación de peligro inminente es necesario que el conductor reaccione de manera ágil y acertada, lo que no es posible cuando se padece fatiga, ya que aumenta el tiempo de reacción ante un evento inesperado.
- Percepción alterada. La fatiga produce una alteración sensorial de lo que nos rodea, creando una falsa percepción de la información que llega a través de los sistemas sensoriales. Generalmente, las primeras manifestaciones son visuales, ya que, debido al cansancio, el movimiento ocular se reduce produciendo una menor percepción visual del entorno.
- Cambio en los movimientos. Durante la conducción nuestro cuerpo ejecuta varios movimientos con el fin de corregir la trayectoria, adelantar a otro automóvil o

evitar un obstáculo. En un estado de fatiga, la precisión, coordinación y velocidad de estos movimientos se reduce consideradamente.

- Malestar físico. Durante la fatiga aparecen sensaciones de malestar como dolor de cabeza, dolor muscular (en cuello, espalda y brazos), tensión muscular, mareos, rigidez y disminución de la flexibilidad.

Todos estos factores originan comportamientos inadecuados durante la conducción como la pérdida de la sensación de velocidad, el no respetar la distancia de seguridad con el vehículo contiguo, tomar las curvas de manera brusca o la modificación de la trayectoria. Esto conlleva a un aumento de las situaciones de riesgo durante la conducción en un intento de llegar cuanto antes al punto de destino.

En cuanto a las causas que llevan a padecer un estado de fatiga, se pueden distinguir tres componentes fundamentales. En primer lugar, el vehículo, que debido a las vibraciones constantes que produce, así como el ruido del motor o el mal estado de las luces, contribuye en el aumento de la fatiga. Por otro lado, el medio en el que se encuentra el conductor, puede también aumentar los síntomas del agotamiento cuando se transita por una carretera conocida o monótona, así como la nocturnidad o las condiciones climatológicas. Por último, se encuentra el estado del conductor. Una edad extrema, una mala colocación del asiento o un estado emocional negativo contribuyen de manera constructiva en el padecimiento de fatiga.

Como se ha comentado anteriormente, la fatiga afecta a los sentidos del conductor modificando la percepción del entorno que lo rodea. El sentido de mayor importancia y trascendencia durante la conducción es la visión, por lo que hay que tener muy en cuenta la fatiga visual, que disminuye la capacidad del ojo para mantener la imagen en la retina. Se produce sobre todo en entornos donde la estimulación es mínima, como paisajes monótonos y con pocas curvas, además de la nocturnidad, considerado un factor que repercute negativamente. La fatiga visual produce errores en la percepción que pueden ser identificados por el conductor, como son la confusión de luces, errores en la percepción de las distancias, ilusiones de falso movimiento, acomodación lenta del iris ante cambios de luminosidad o reducción de los contrastes.

La conducción nocturna ejerce un efecto sobre el conductor similar al de la miopía de una dioptría, si a esto se le suma la dificultad de visualizar objetos con poca iluminación, la adaptación del iris a cambios de luminosidad, la molestia de las luces de los vehículos que circulan por el carril contrario, la suciedad del parabrisas, etc., se llega a la conclusión de que la nocturnidad afecta a la fatiga visual consideradamente.

Todos estos síntomas mostrados son la manifestación del cuerpo ante la necesidad de descanso. Aunque parece lógico que la actitud normal del conductor ante la fatiga es la de concluir la conducción y parar a descansar durante un tiempo, desafortunadamente la mayor parte de la población no hace caso a su organismo y sigue conduciendo hasta llegar al destino. Otros intentan combatir el estado de fatiga a través de estimulantes como el café o la taurina, lo que no es una opción inteligente ya que estos productos no eliminan la fatiga, únicamente la enmascaran dando una falsa sensación de seguridad. Por esto, debe quedar claro que la fatiga, al igual que el sueño, únicamente desaparece de una manera, descansando.

2.1. Técnicas de análisis de la fatiga

Son muchas las técnicas desarrolladas para el análisis de la fatiga durante la conducción, pero todas ellas pueden dividirse en tres grandes grupos en función de los parámetros de entrada tomados para su posterior análisis.

2.1.1. Comportamiento del vehículo

Aunque no nos demos cuenta, cada individuo conduce de una manera única y distintiva su vehículo, lo que permite reconocer al conductor en función de sus métodos y patrones durante la conducción. Por ejemplo, a la hora de corregir la trayectoria durante travesías prolongadas, cada sujeto realiza una serie de movimientos escuetos del volante que, sometiéndolos a una monitorización, pueden llegar a identificar de manera inequívoca a la persona que se encuentra al volante.

Una vez establecido el estado de fatiga en el conductor, este comienza a variar sus hábitos durante el manejo de modo que, analizando constantemente ciertas métricas como pueden ser la posición del coche, los movimientos del volante, la presión ejercida en la frenada/aceleración o el cambio de marchas, es posible detectar y prevenir la llegada de la fatiga y sus probables consecuencias. El principal inconveniente de este método, generalmente, se encuentra en la dificultad a la hora de monitorizar el comportamiento del automóvil de manera eficaz, ya que depende de manera directa de las características del vehículo, conductor y carretera [9].

Son varios los sistemas implementados que hacen uso de estas características con el fin de predecir el estado de fatiga. En [10], se toma como parámetro de entrada la presión que se realiza sobre el asiento del conductor durante largas sesiones, relacionando esto con la posición del vehículo sobre el carril y el índice subjetivo de fatiga. Este algoritmo estime un estado de fatiga en intervalos de 10 minutos.

En [11], se analiza continuamente la distancia al vehículo precedente y el ángulo de giro del volante, con el objetivo de diferenciar cuando el conductor este concentrado en la carretera. Se desarrolla un modelo, que funciona en tiempo real, usando Modelos de Márkov con mezcla de Gaussianas para estimar la atención del conductor.

En [12], se utiliza la posición del pedal del acelerador, la posición del giro del volante, los extremos del carril y la curvatura de la carretera para evaluar el estado de somnolencia del conductor.

En [13], se utiliza un modelo de redes neuronales para caracterizar un patrón de conducción normal. Entrenando esta red neuronal en conjunto con un sistema SVM, es capaz de distinguir cuando el usuario del vehículo está ejecutando una conducción normal o una inatenta. El modelo de redes neuronales toma como parámetros de entrada la dinámica del vehículo y datos de la conducción como la posición del vehículo sobre el carril, la velocidad y la aceleración. Finalmente aproxima la conducción a un modelo estadístico donde la media y la desviación estándar son elegidas como entradas al clasificador SVM.

2.1.2. Variables fisiológicas del conductor

Este método de detección de fatiga consiste en la extracción de parámetros fisiológicos del conductor, como el ritmo cardiaco, con el fin de conocer el estado psicológico del sujeto. Las técnicas más efectivas se basan en la obtención del electroencefalograma (EEG), electromiograma (EMG), electrocardiograma (ECG) y electrooculograma (EOG), siendo el más utilizado el EEG [14].

Aunque esta técnica de detección de fatiga sea la más precisa, generando un mínimo de falsos positivos, es necesario el contacto directo con el conductor para su continua monitorización, lo que conlleva que su uso en entornos reales no sea nada práctica ni adecuada al tratarse de un sistema invasivo.

2.1.3. Comportamiento del conductor

Por último, se puede estimar el estado de fatiga en función de parámetros visuales que presenta el sujeto. Cuando el conductor se encuentra fatigado, son varios los signos faciales que lo demuestran, como la variación de la frecuencia de parpadeo, movimientos faciales, cabeceos, variación de la apertura media de los ojos o bostezos frecuentes. Estas características del individuo pueden obtenerse de manera sencilla y no invasiva, a través del procesamiento de imágenes en tiempo real de la cara del conductor, a partir de cámaras instaladas en el vehículo.

El inconveniente de este método consiste en que estos síntomas se producen en distintos estados de tiempo, más concretamente, los cabeceos aparecen en un estado avanzado de somnolencia, mientras que, los bostezos se producen en un estado previo de la misma. Es por esto que el estudio tanto de bostezos como de cabeceos, no producen un resultado concreto del estado de fatiga en el que se encuentra el sujeto. En cambio, los métodos basados en obtener información de los ojos pueden detectar con exactitud este punto, es decir, son los métodos más precisos para la evaluación del estado de somnolencia. En concreto, los patrones de parpadeo y el PERCLOS, definido como el porcentaje de tiempo en el que los ojos se encuentran cerrados un 80% por debajo del nivel medio de apertura, son los parámetros mas utilizados en los sistemas de prevención de fatiga o somnolencia en conductores [15].

El inconveniente de esta técnica se encuentra en que está limitada a unas condiciones ambientales controladas y requieren de un proceso complicado de calibración. Una solución a esto, puede ser la combinación de las distintas técnicas enumeradas (PERCLOS, bostezos y cabeceos) con el fin de proporcionar al sistema de detección de fatiga mayor robustez y precisión.

En Bergasa 2006, se toman como parámetros de entrada a un sistema de lógica difusa, el PERCLOS, la duración del cierre de los ojos, la frecuencia de parpadeo, y la variación de la dirección visual. Unifica todos estos parámetros con el fin de obtener de manera robusta y precisa una predicción del estado de fatiga del conducto

3. Visión artificial

Uno de los sentidos más importantes de los seres humanos es la visión. Ésta es empleada para obtener la información visual del entorno físico. Según Aristóteles, "Visión es saber que hay y donde mediante la vista". De hecho, se calcula que más de 70% de las tareas del cerebro son empleadas en el análisis de la información visual. El refrán popular de "Una imagen vale más que mil palabras" tiene mucho que ver con los aspectos cognitivos de la especie humana. Casi todas las disciplinas científicas emplean utillajes gráficos para transmitir conocimiento. Por ejemplo, en Ingeniería Electrónica se emplean esquemas de circuitos, a modo gráfico, para describirlos. Se podría hacerlo mediante texto, pero para la especie humana resulta mucho más eficiente procesar imágenes que procesar texto. La visión humana es el sentido más desarrollado y el que menos se conoce debido a su gran complejidad. Es una actividad inconsciente y difícil de saber cómo se produce. De hecho, hoy en día, se carece de una teoría que explique cómo los humanos perciben el exterior a través de la vista.

En el año 1826 el químico francés Nepe (1765-1833) llevó a cabo la primera fotografía, colocando una superficie fotosensible dentro de una cámara oscura para fijar la imagen. Posteriormente, en 1838 el químico francés Daguerre (1787-1851) hizo el primer proceso fotográfico práctico. Daguerre utilizó una placa fotográfica que era revelada con vapor de mercurio y fijada con trisulfuro de sodio. [16]

Desde que se inventó la fotografía se ha intentado extraer características físicas de las imágenes. La Fotogrametría dio sus primeros pasos desde imágenes capturadas en globos. La Astronomía avanzó enormemente con el análisis de imágenes recibidas por los telescopios. El análisis de radiografías transformó la Medicina. Se podrían citar muchos más ejemplos que durante décadas han transformado la percepción de la Ciencia con el procesamiento de las imágenes, algunas veces por separado y otras de forma multidisciplinar.

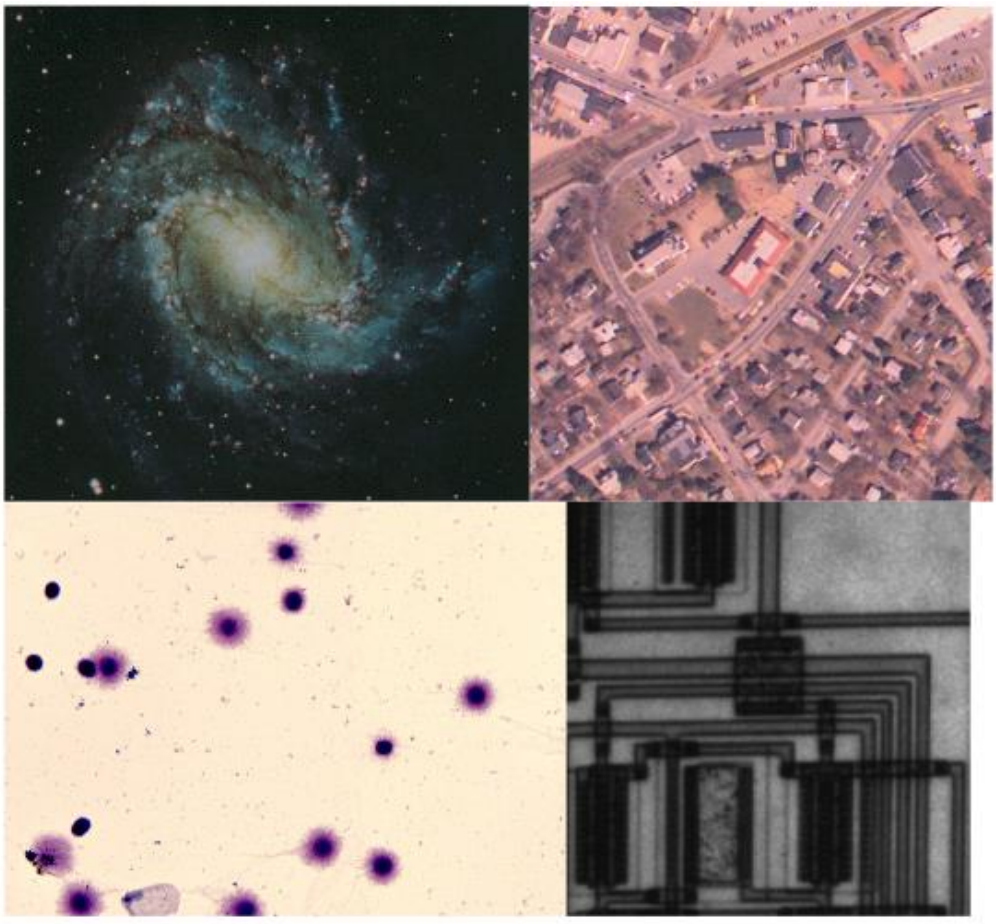


Ilustración 1. Aplicaciones del procesamiento de imágenes a) Astronomía, b) Fotogrametría, c) Medicina, d) Industria

Sin embargo, el momento histórico que hace que estas técnicas confluyan y den un cuerpo de conocimiento propio, surge en la década de los 80 del siglo XX. La revolución de la Electrónica, con las cámaras de vídeo CCD y los microprocesadores, junto con la evolución de las Ciencias de la Computación hace que sea factible la Visión Artificial.

Por tanto, la Visión Artificial o también llamada Visión por Computador, pretende capturar la información visual del entorno físico para extraer características relevantes visuales, utilizando procedimientos automáticos. Según Marr, "Visión es un proceso que produce a partir de imágenes del mundo exterior una descripción útil para el observador y no tiene información irrelevante".

3.1. Etapas de un sistema de visión artificial

Se ha visto que el ser humano captura la luz a través de los ojos, y que esta información circula a través del nervio óptico hasta el cerebro donde se procesa. Existen razones para creer que el primer paso de este procesado consiste en encontrar elementos más simples en los que descomponer la imagen (como segmentos y arcos). Después el cerebro interpreta la escena y por último actúa en consecuencia. La visión artificial, en

un intento de reproducir este comportamiento, define tradicionalmente cuatro fases principales:

- La primera fase, que es puramente sensorial, consiste en la captura o adquisición de las imágenes digitales mediante algún tipo de sensor.
- La segunda etapa consiste en el tratamiento digital de las imágenes, con objeto de facilitar las etapas posteriores. En esta etapa de procesamiento previo es donde, mediante filtros y transformaciones geométricas, se eliminan partes indeseables de la imagen o se realzan partes interesantes de la misma.
- La siguiente fase se conoce como segmentación, y consiste en aislar los elementos que interesan de una escena para comprenderla.
- Por último, se llega a la etapa de reconocimiento o clasificación. En ella se pretende distinguir los objetos segmentados, gracias al análisis de ciertas características que se establecen previamente para diferenciarlos.

Estas cuatro fases no se siguen siempre de manera secuencial, sino que en ocasiones deben realimentarse hacia atrás. Así, es normal volver a la etapa de segmentación si falla la etapa de reconocimiento, o a la de preproceso, o incluso a la de captura, cuando falla alguna de las siguientes.[17]

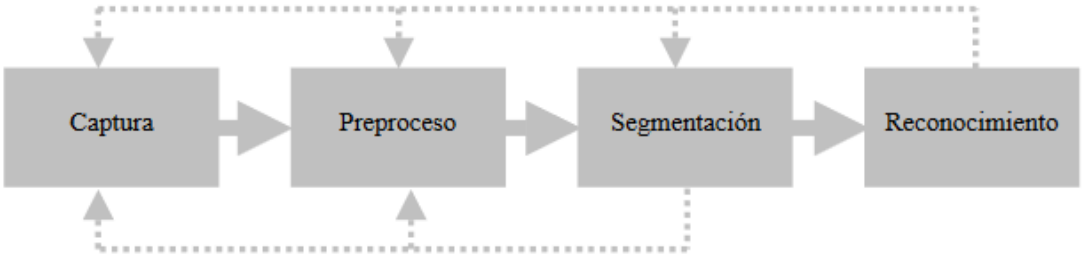


Ilustración 2. Diagrama de bloques de las etapas típicas en un sistema de visión artificial.

3.1.1. Captura y digitalización de imágenes

Las imágenes digitales son "señales" discretas, que suelen tener origen en una "señal" continua. Por ejemplo, una cámara digital toma imágenes del mundo real que es continuo (tanto el espacio, como el espectro emitido por los objetos se consideran continuos); otro ejemplo es el de un escáner, el cual digitaliza imágenes procedentes de documentos o fotografías que a efectos prácticos también se consideran continuos.

En el proceso de obtención de imágenes digitales se distinguen dos etapas. La primera, conocida como captura, utiliza un dispositivo, generalmente óptico, con el que obtiene información relativa a una escena. En la segunda etapa, que se conoce como digitalización, se transforma esta información, que es una señal con una o varias componentes continuas, en la imagen digital, que es una señal con todas sus componentes discretas. Los dispositivos de captura más comunes son:

Cámara fotográfica analógica

Está constituida por un recinto oscuro (la cámara), en la que se ha montado un objetivo. El objetivo forma la imagen luminosa en el interior de la cámara, en el plano de formación de la imagen, donde hay una superficie sensible a la luz. Entre el objetivo y la superficie sensible se encuentra el obturador, que sólo deja pasar la luz en el momento de captura de la imagen.

Cámara de video analógica

La cámara de vídeo es un aparato que transforma una secuencia de escenas ópticas en señales eléctricas. Está constituida por un objetivo, un tubo de cámara y diversos dispositivos electrónicos de control. La luz se enfoca dentro del tubo de cámara sobre una superficie fotosensible que convierte la señal lumínica en una señal eléctrica denominada señal de vídeo.

Cámara digital de fotografía y vídeo

El esquema de ambas cámaras es idéntico al de sus correspondientes analógicas, con la diferencia de que el dispositivo sensible es un CCD. Un CCD es un dispositivo constituido por una matriz de elementos fotosensibles, que se sitúa en el mismo lugar que el plano de formación de la imagen, de manera que se forma la imagen sobre él.

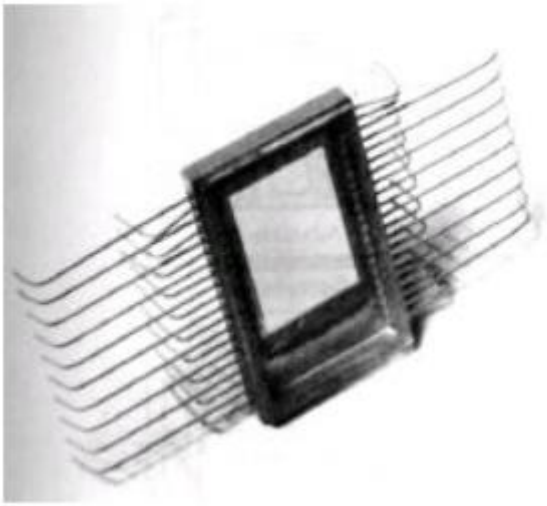


Ilustración 3. Primer CCD comercial. Constaba de 120.000 elementos y tenía un tamaño de 0.5x0.25 pulgadas.

El número de elementos fotosensibles, junto con el área que ocupan, definen la resolución espacial del dispositivo.

3.1.2. Preprocesamiento de la imagen

Una vez realizado el proceso de captura se aplica una serie de operaciones y transformaciones sobre las imágenes digitales en una etapa de procesamiento previa a las

de segmentación y reconocimiento. Su objetivo es mejorar o destacar algún elemento de las imágenes, de manera que las etapas posteriores sean posibles o se simplifiquen.

Todas las operaciones que se van a describir a continuación se pueden explicar desde la perspectiva ofrecida por la teoría de filtros. Un filtro puede verse como un mecanismo de cambio o transformación de una señal de entrada a la que se le aplica una función, conocida como función de transferencia, para obtener una señal de salida. En este contexto se entiende por señal una función de una o varias variables independientes. Los sonidos y las imágenes son ejemplos típicos de señales.

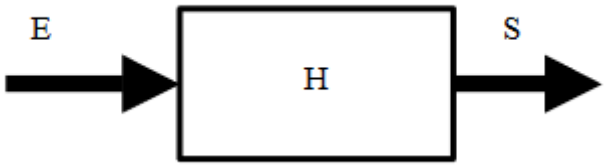


Ilustración 4. Esquema de funcionamiento de un filtro.

En el diagrama anterior se representa el esquema general de funcionamiento de un filtro, siendo E la función de entrada, S la de salida y H la función de transferencia del filtro. Todas estas señales y funciones pueden ser discretas o continuas, y aunque en el tratamiento de imágenes se procesan señales y funciones discretas, suele recurrirse al caso continuo para explicar sus comportamientos, ya que sobre las funciones continuas es posible emplear herramientas más potentes de cálculo matemático.

3.1.2.1. Conversión de una imagen RGB a escala de grises

La conversión de una imagen en color a escala de grises es el equivalente a la luminancia de la imagen. Como ya se sabe, el ojo percibe distintas intensidades de luz en función del color que se observe, esto es debido a la respuesta del ojo al espectro visible. Por esta razón el equivalente blanco y negro (escala de grises o luminancia) de la imagen debe realizarse como una media ponderada de las distintas componentes de color de cada pixel.

La ecuación de la luminancia es la expresión matemática de este fenómeno, y los factores de ponderación de cada componente de color nos indican la sensibilidad del ojo humano a las frecuencias del espectro cercanas a los tres colores básicos, rojo, verde y azul. [18]

$$E_y = 0.3 * R + 0.5 * G + 0.11 * B$$

Con esta simple ecuación se podrá transformar cualquier imagen a color (RGB) a una imagen en escala de grises con un solo canal de color, por lo que será más sencilla de procesar.

3.1.2.2. Operaciones sobre el histograma

El histograma de una imagen en niveles de gris proporciona información sobre el número de píxeles que hay para cada nivel de intensidad. En imágenes en color RGB se usan 3 histogramas, uno por cada componente de color. En el caso de imágenes de paleta, el histograma, si bien se puede calcular, tiene una utilidad menos evidente.

Se conoce como rango dinámico de una imagen al conjunto de todos los posibles valores que efectivamente se encuentran presentes en una imagen.

El análisis del histograma de una imagen permite conocer detalles sobre la calidad de la misma y del proceso de captura que se ha utilizado para obtenerla. Así, las imágenes de calidad suelen tener un rango dinámico amplio y no saturado. Un rango dinámico pobre implica que la imagen contiene poca información. Las imágenes muy saturadas o poco saturadas contienen menos información que las no saturadas, y por ello no son deseables para tareas de reconocimiento. También es importante para las tareas de reconocimiento que las imágenes tengan un alto nivel de contraste (sin llegar a estar saturadas), ya que esto implica que los detalles discriminantes se perciben con claridad. [17]

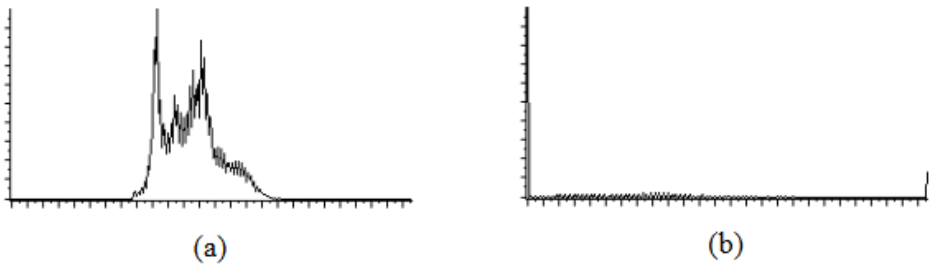


Ilustración 5. (a) histograma de una imagen con poco contraste. (b) histograma de una imagen saturada.

3.1.2.2.1. Aumento y reducción de contraste

El contraste se define como la diferencia de intensidad pronunciada en una imagen. Se puede hablar de alto contraste en una imagen digital en niveles de gris si sobre el histograma se aprecia masas separadas. En este contexto una buena medida podría ser la desviación típica del histograma.

Una función de transferencia que aclare los niveles claros y oscurezca los más oscuros, conseguirá sobre el conjunto de la imagen un efecto visual de aumento de contraste. Una función tal se puede obtener componiendo una función de transferencia del histograma que hasta el valor de 0’5 se comporte como la función cuadrado y que en adelante se comporte como la función raíz.

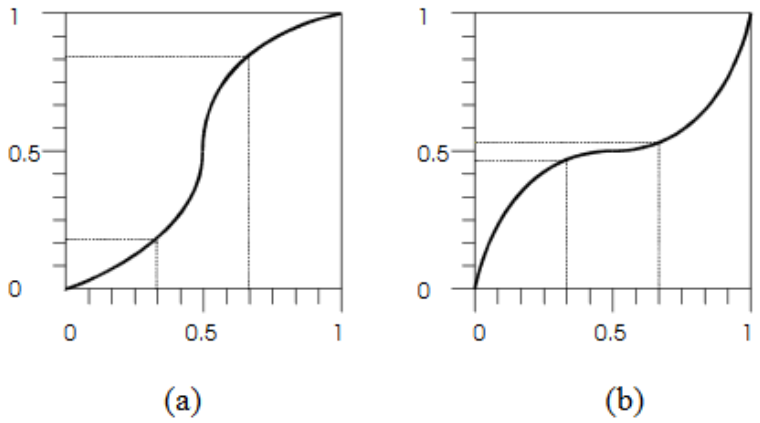


Ilustración 6. (a) Función de transferencia para aumento de contraste. (b) Función de transferencia para reducción de contraste.

En general, una función de transferencia con una pendiente inferior a la unidad produce un efecto de reducción de contraste. Esto se debe a que concentra los valores de las intensidades de un rango R en un rango más pequeño R'. Por otro lado, una función de transferencia con una pendiente superior a la unidad produce un efecto de aumento de contraste por razones análogas. [17]

3.1.2.2.1. Ecualización del histograma

El proceso de ecualizado tiene por objetivo obtener un nuevo histograma, a partir del histograma original, con una distribución uniforme de los diferentes niveles de intensidad.

Al transformar una distribución continua cualquiera en una distribución uniforme se está maximizando la cantidad de información que contiene. Y aunque en el caso discreto es imposible aumentar la cantidad de información, el ecualizado del histograma mejora la calidad visual de imágenes saturadas. Este efecto se debe a que se cambian los valores de intensidad de las zonas saturadas, en las que originalmente existen objetos que no se distinguen adecuadamente al inspeccionar visualmente la imagen.

Para exponer el ecualizado del histograma se modifica ligeramente la representación del histograma para asimilarla a la de una función de densidad de probabilidad. Así, los niveles de intensidad serán una variable aleatoria R que varía entre 0 y 1.

Esta transformación consiste en normalizar el número de intensidades a valores entre 0 y 1 (igual que previamente) y en dividir cada elemento del histograma por el número de píxeles de la imagen (para que su suma sea 1). Estas transformaciones consiguen que el área del histograma normalizado sea igual a la unidad.

Se ha dicho que el objetivo del ecualizado es transformar la distribución del histograma P_R en una distribución uniforme P_S. Como el área bajo P_R será igual al área bajo P_S, y el área bajo P_R es igual a 1, P_S debe ser una distribución uniforme con la forma $P_S=1$.

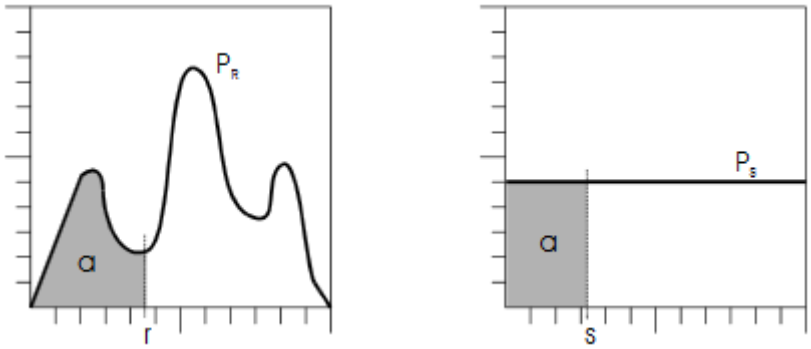


Ilustración 7. El proceso de ecualizado exige que el área a la izquierda de r debe ser la misma que la que hay a la izquierda de s.

Además, cualquier transformación del histograma cumple que el área a la izquierda de un punto *r* sobre la distribución original P_R es igual al área a la izquierda del correspondiente punto transformado s en la distribución P_S. Es decir, el área (a) bajo P_R de la ilustración 7 debe ser igual al área (a) bajo P_S si el nivel original *r* se corresponde con *s* en el histograma ecualizado.

Pasando al caso discreto se tiene que PR(r), la probabilidad de que un píxel tenga la intensidad r, se expresa como

$$P_R(r) = \frac{n_r}{n}$$

Donde n es el número total de píxeles en la imagen, y n_r el número de píxeles con nivel de intensidad r. La ecualización del histograma tiene como objetico obtener un histograma uniforme, es decir, que la probabilidad de cualquier nivel de gris en la imagen sea la misma. [17]

$$s(r) = \sum_{j=0}^{r} \frac{n_r}{n}$$

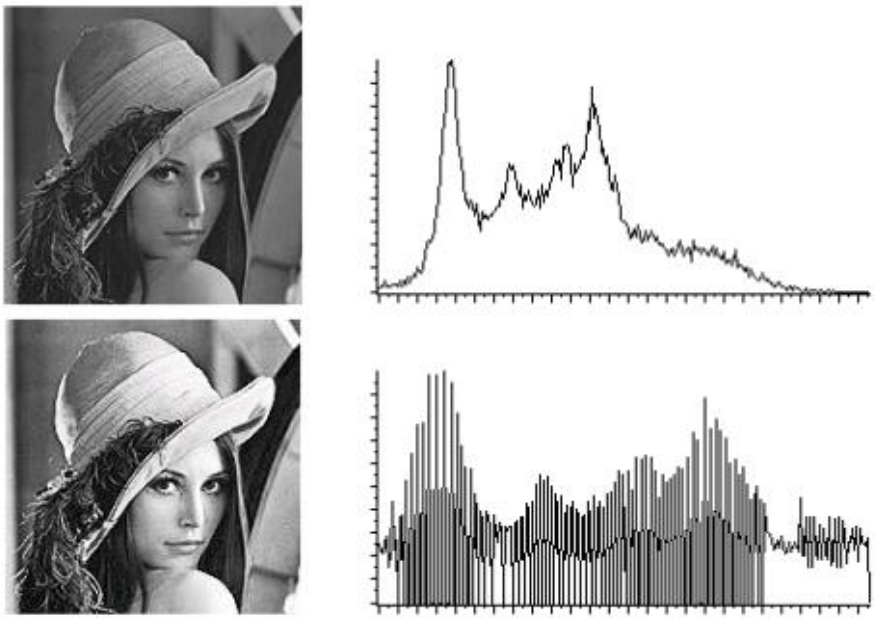


Ilustración 8. Ecualizado del histograma sobre la imagen: (a) imagen original con su correspondiente histograma; (b) ecualizado del histograma

3.1.2.3. Eliminación de ruido

El ruido es un factor a tener muy en cuenta a la hora de realizar un procesado a la imagen de interés, ya que muchas operaciones que se realizan sobre esta, producen resultados no deseados en presencia de ruido.

El proceso de filtrado consiste en la aplicación a cada uno de los pixeles de la imagen de una matriz de filtrado de tamaño NxN (generalmente de 3x3 aunque puede ser mayor) compuesta por números enteros y que genera un nuevo valor mediante una función del valor original y los de los pixeles circundantes. El resultado final se divide entre un escalar, generalmente la suma de los coeficientes de ponderación. Generalmente, el filtrado de ruido se realiza a través de filtros paso bajo con el objetivo de suavizar la imagen a través de operaciones geométricas como la media o mediana.

El filtrado paso bajo espacial se basa en el promediado de los píxeles adyacentes al píxel que se evalúa. Quizás el filtro paso bajo más simple que se puede diseñar se corresponde con una matriz de 3x3 con todos los elementos a 1. El resultado se deberá dividir por 9 para obtener valores dentro del rango de la paleta.

Otro filtro paso bajo es el filtro de la mediana. Éste se basa en sustituir el valor de un píxel por el de la mediana del conjunto formado por el mismo y sus ocho vecinos.

El filtro del bicho raro es otro ejemplo de filtro paso bajo. Consiste en comparar la intensidad de un píxel con la de sus 8 vecinos. Si la diferencia es superior a cierto umbral U (que debe elegirse previamente), se sustituye tal píxel por el valor promedio de los píxeles vecinos, en otro caso se mantiene su valor de intensidad.

Tanto el filtro de la mediana, como el filtro del "bicho raro" son filtros no lineales, es decir, no se pueden deducir de una convolución, y por tanto no tienen equivalente en el dominio de la frecuencia.[17]

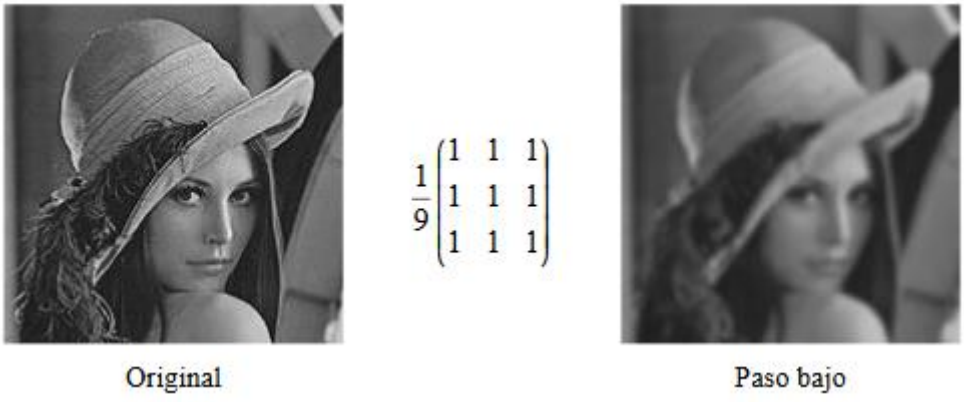


Ilustración 9. Aplicación de un filtrado espacial paso bajo.

3.1.2.4. Operaciones morfológicas

Clásicamente la morfología es una parte de la biología que estudia la forma de los animales y de las plantas. De la misma forma, la morfología matemática es una herramienta que ayuda a tratar problemas que involucran formas en una imagen. La morfología matemática tiene su origen en la teoría de conjuntos. Para ella las imágenes binarias son conjuntos de puntos 2D, que representan los puntos activos de una imagen, y las imágenes en niveles de gris son conjuntos de puntos 3D, donde la tercera componente corresponde al nivel de intensidad. En este apartado sólo se tratará detalladamente la morfología sobre imágenes bitonales, presentándose únicamente los operadores básicos para imágenes en niveles de gris. [17]

Dilatación

Siendo A y B dos conjuntos en Z2, la dilatación de A con B, denotada como A $\oplus$ B, se define:

$$A \oplus B = \{x/x = a + b \quad \forall a \in A \quad \forall b \in B\}$$

Es interesante notar que la dilatación cumple la propiedad conmutativa.

$$A \oplus B = B \oplus A$$

La implementación directa de la dilatación según la definición dada es demasiado costosa. La siguiente formulación, que puede demostrarse que es equivalente, da una idea de una implementación mucho más eficiente.

$$A \oplus B = \left\{ x/(\widehat{B})_x \bigcap A \neq \varphi \right\}$$

Escrito de otra forma:

$$A \oplus B = \left\{ x/\left[(\widehat{B})_x \bigcap A \right] \subseteq A \right\}$$

El elemento B es el elemento que dilata a A, y se conoce como elemento estructurante de la dilatación.

Intuitivamente esta operación produce el efecto de dilatar el aspecto del elemento A usando para ello a B. La posición del elemento B respecto del eje de ordenadas es importante, ya que influye en el proceso de dilatación. Por ello suele indicarse el centro de las figuras con un punto.

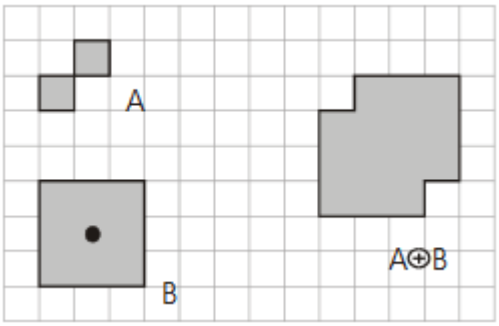


Ilustración 10. Ejemplo de dilatación.

Erosión

Siendo A y B dos conjuntos en Z^2, la dilatación de A con B, denotada como A $\oplus$ B, se define:

$$A \ominus B = \{x/x + b \in A \quad \forall b \in B\}$$

Nuevamente puede definirse con otra forma cuyo coste computacional es mucho más reducido.

$$A \ominus B = \left\{ x/(B)_x \subseteq A \right\}$$

La erosión adelgaza la imagen sobre la que se aplica siendo, en un sentido no estricto, opuesta a la dilatación. Si sobre la Ilustración 10 se erosiona A⊕B con B se obtiene de nuevo A, aunque esto no tiene por qué ocurrir en otro caso distinto.

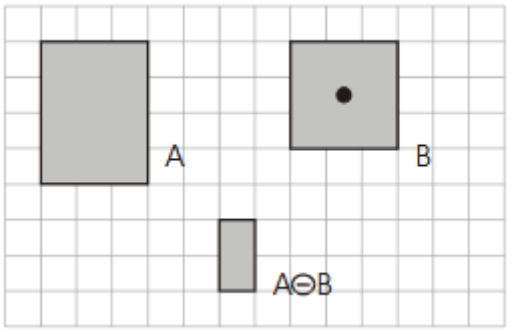


Ilustración 11. Ejemplo de erosión.

Apertura

La apertura de A con B se define como:

$$A \circ B = (A \ominus B) \oplus B$$

Sus propiedades son:

- $A \circ B$ es un subconjunto de A.
- $(A \circ B) \circ B = A \circ B$
- Si C es subconjunto de D → $C \circ B$ es un subconjunto de $D \circ B$

Intuitivamente la apertura de A con un elemento estructurante B equivale a determinar los puntos en los que puede situarse B cuando se desplaza por el interior de A.

Cierre

El cierre de A con B se define como:

$$A \odot B = (A \oplus B) \ominus B$$

Sus propiedades son:

- A es un subconjunto de $A \odot B$.
- $(A \odot B) \odot B = A \odot B$
- Si C es subconjunto de D → $C \odot B$ es un subconjunto de $D \odot B$

Intuitivamente el cierre de A con un elemento estructurante B equivale a los puntos en los que puede estar el origen de B cuando se desplaza tocando al menos un punto de A.

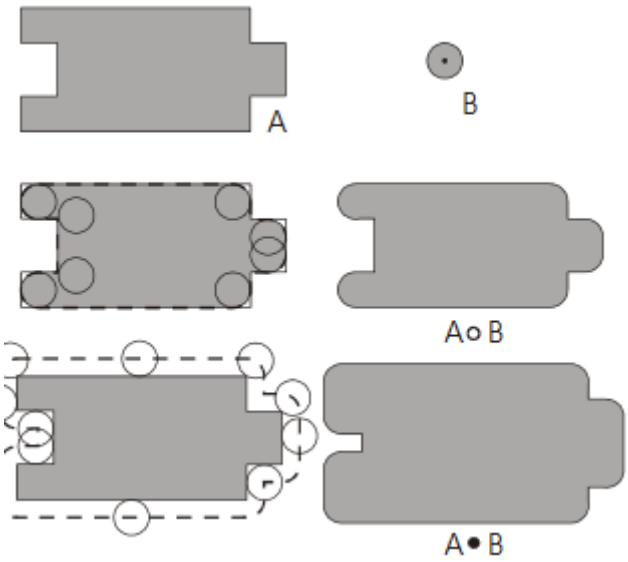


Ilustración 12. Arriba se presenta la figura A y el elemento estructurante B. En medio se presenta la ejecución de la operación de apertura y su resultado. Abajo se presenta la operación de cierre y su resultado.

3.1.3. Segmentación

La segmentación es un proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características (como por ejemplo el brillo o el color) con el fin de facilitar su posterior análisis y reconocimiento automático. Localizar la cara de una persona dentro de la imagen de una fotografía o encontrar los límites de una palabra dentro de una imagen de un texto, constituyen ejemplos de problemas de segmentación.

La segmentación termina cuando los objetos extraídos de la imagen se corresponden unívocamente con las distintas regiones disjuntas a localizar en la misma. En este caso se habla de segmentación completa de la escena o imagen y en el caso contrario, de segmentación parcial. En una escena compleja, el resultado de la segmentación podría ser un conjunto de regiones homogéneas superpuestas y en este caso, la imagen parcialmente segmentada deberá ser sometida después a un tratamiento posterior con el fin de conseguir una segmentación completa.

Los diferentes objetos que aparecen en una imagen pueden localizarse atendiendo a aspectos como: sus bordes o su textura. [17]

3.1.3.1. La textura

La textura de un objeto dentro de una imagen es el conjunto de formas que se aprecia sobre su superficie y que le dotan de cierto grado de regularidad. Una definición típica de textura es la siguiente: “uno o más patrones locales que se repiten de manera periódica”.

Para el estudio y comparación de algoritmos sobre imágenes que presentan texturas suelen utilizarse como referencia las imágenes de Brodatz, conocidas como álbum de Brodatz. [19]

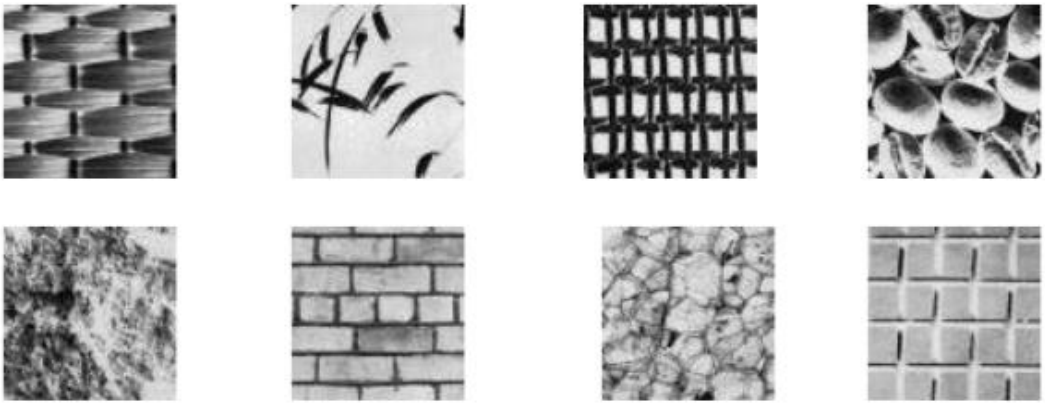


Ilustración 13. Imágenes del álbum de Brodatz.

3.1.3.2. El borde

Los bordes de un objeto en una imagen digital corresponden a la línea de píxeles que separa ese objeto del fondo de la imagen. Normalmente estos bordes se corresponden con los puntos donde se producen discontinuidades en los valores de los píxeles adyacentes (cambios en el matiz o el brillo) o en conjuntos de píxeles (cambios de textura)

3.1.3.3. Segmentación basada en umbralizado

La umbralización es un proceso que permite convertir una imagen de niveles de gris o en color en una imagen binaria, de tal forma que los objetos de interés se etiqueten con un valor distinto de los píxeles del fondo o background. Es una técnica de segmentación rápida, que tiene un coste computacional bajo y que incluso puede ser realizada en tiempo real durante la captura de la imagen usando un computador personal de propósito general.

Umbralización fija

El caso más sencillo, conocido como umbralización fija, se puede usar en aquellas imágenes en las que existe suficiente contraste entre los diferentes objetos que se desea separar. Así, se puede establecer un valor fijo que marque el umbral de separación sobre el histograma. Para obtener dicho umbral se debe disponer de información sobre los niveles de intensidad de los objetos a segmentar y del fondo de la imagen.

Como puede comprenderse, la elección de un valor de umbral correcto resulta decisiva para llevar a cabo la segmentación de una imagen de manera satisfactoria.[17]

Umbralización generalizada

En general, la obtención de un único valor de umbral fijo no es aplicable en imágenes más complejas y, además, debido a problemas de iluminación no uniforme de la escena, el estudio del histograma de la imagen no permite detectar un valor del umbral adecuado para separar los objetos del fondo o background. Esto lleva a considerar otros tipos de umbralización para segmentar una imagen, que resultan de generalizar la idea de umbral explicada.

La umbralización de banda permite segmentar una imagen en la que los objetos (regiones de píxeles) contienen niveles de gris dentro de un rango de valores y el fondo tiene píxeles con valores en otro rango disjunto.

La multiumbralización, como su nombre indica, consiste en la elección de múltiples valores de umbral dentro del proceso, permitiendo separar a diferentes objetos dentro de una escena cuyos niveles de gris difieran. El resultado no será ahora una imagen binaria, sino que los diferentes objetos (regiones) tendrán etiquetas diferentes

La semiumbralización persigue obtener una imagen resultado en niveles de gris, y para ello pone a cero el fondo de la imagen conservando los niveles de gris de los objetos a segmentar que aparecen en la imagen inicial.

En las técnicas anteriores, el o los rangos de umbralización se consideran fijos con independencia de las características locales de la imagen considerada. En muchas imágenes donde la iluminación no es uniforme puede ocurrir que píxeles del mismo objeto a segmentar tengan niveles de gris diferenciados. La umbralización adaptativa o variable permite resolver el problema de la segmentación haciendo que el valor del umbral varíe como una función de las características locales de la imagen. [17]

3.1.4. Reconocimiento por clasificadores

Los algoritmos de clasificación tienen la misión de distinguir entre objetos diferentes de un conjunto predefinido llamado universo de trabajo. Normalmente, el universo de trabajo se considera dividido en una colección K de clases (α_1, α_2... α_K), perteneciendo los diferentes tipos de objetos a algunas de estas clases.

Existen diferentes métodos que permiten determinar, de manera automática, en qué clase se encuentra un objeto de un universo de trabajo. Estos métodos se conocen como clasificadores. [17]

3.1.4.1. Características discriminantes

Para poder realizar el reconocimiento automático de los objetos se realiza una transformación que convierte un objeto del universo de trabajo en un vector X cuyas N componentes se llaman características discriminantes o rasgos.

Estas características deben permitir discriminar a qué clases puede pertenecer cualquier objeto del universo de trabajo.

La determinación de las N características discriminantes es un proceso difícil que puede no estar exento del uso de la imaginación. En general, suelen usarse características como los momentos de los objetos a reconocer, alguna transformación de los mismos (Fourier, cosenos...), las propias imágenes, o cualquier característica que se pueda obtener de los objetos mediante algún procedimiento algorítmico.

Una vez determinadas las características discriminantes para un problema concreto, la clasificación de un objeto comienza por la obtención de su patrón. El siguiente paso consiste en determinar la proximidad o grado de pertenencia de este patrón a cada una de las clases existentes, asignando el objeto a aquellas clases con las que el grado de semejanza sea mayor. A este efecto se definen las funciones discriminantes o funciones de decisión como aquellas funciones que asignan grados de semejanza de patrón a cada una de las diferentes clases. [17]

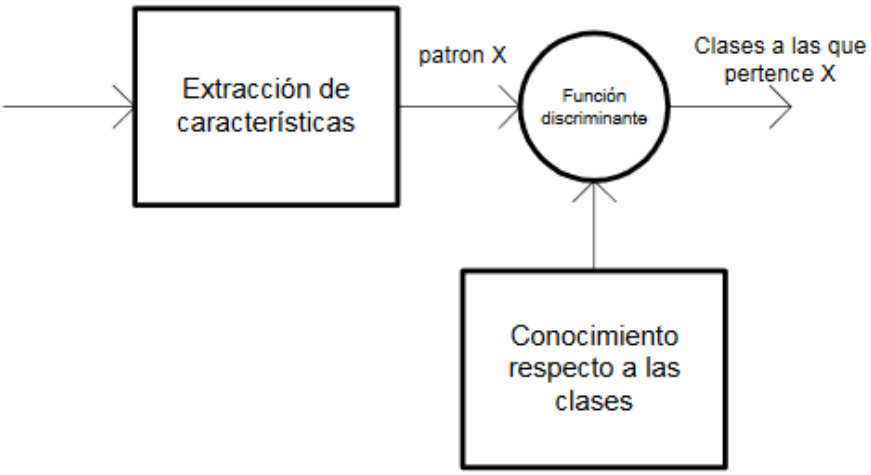


Ilustración 14. Esquema general de funcionamiento de un clasificador.

La muestra de aprendizaje

Para poder realizar el cálculo de las funciones discriminantes suele ser precisa la existencia de un conjunto de patrones similares a los que se desea reconocer, que se denomina conjunto de aprendizaje. Los patrones de este conjunto se utilizan a modo de modelo para encontrar la función discriminante que clasificará correctamente los patrones del universo de trabajo. El conjunto de aprendizaje debe ser por tanto un subconjunto representativo del universo de trabajo.

Cuando la muestra es abundante suele crearse otro conjunto con ella. Este segundo conjunto se utiliza para probar los resultados de las funciones discriminantes calculadas, y se conoce como conjunto de test. Es importante que estos dos conjuntos sean independientes, como norma general, en el caso de un universo de trabajo grande, es suficiente con asegurar que el conjunto de aprendizaje y el test no tengan elementos en común. De esta forma puede probarse que el clasificador desarrollado ha adquirido la propiedad de generalización. Esta propiedad garantiza que un sistema clasifica

correctamente patrones que no ha visto durante el proceso de cálculo de funciones discriminantes.

3.1.4.2. Clasificador Viola & Jones

Paúl Viola y Michael J. Jones desarrollaron un algoritmo cuyo objetivo es el reconocimiento de caras humanas y que tiene un coste computacional muy bajo. Consta de dos partes principales: clasificador en cascada, que garantiza una discriminación rápida y un entrenador de clasificadores.

Los primeros clasificadores son muy sencillos y permiten rechazar una gran cantidad de objetos que no se está buscando mientras que aceptan un porcentaje muy alto del objeto deseado. La cascada permite desechar gran parte de las regiones de la imagen y sólo concentrarse en las zonas en las que es más probable que haya un objeto que se desea encontrar por lo que la detección es bastante rápida. [20]

Haar-Like features

Este algoritmo toma características de tipo Haar (Haar-Like features) que se definen sobre regiones rectangulares de una imagen en escala de grises. Una característica está formada por un número finito de rectángulos y su valor escalar consistirá en la suma de los píxeles de cada rectángulo sumados aplicando un cierto factor de peso.

$$característica = \sum_{i=1}^{N} w_i * suma\_rectang(r_i)$$

Donde $\{r_1, \dots, r_N\}$ son los rectángulos que forman la característica y w_1 el peso de cada uno.

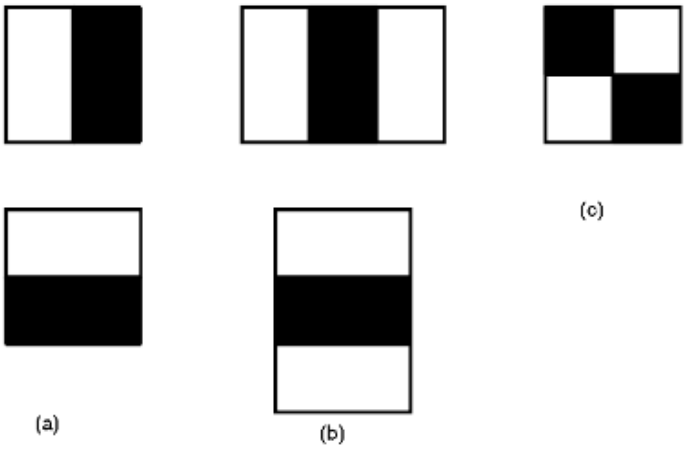


Ilustración 15. Características tipo Haar. (a) las de bordes, (b) las de líneas, (c) las de forma de X.

El valor de una característica se obtiene sumando todos los píxeles del rectángulo blanco y restándose todos los píxeles del rectángulo negro. Por ejemplo, la característica central de tres rectángulos trataría de representar que en general la región de los ojos es más oscura que las regiones de alrededor. [20]

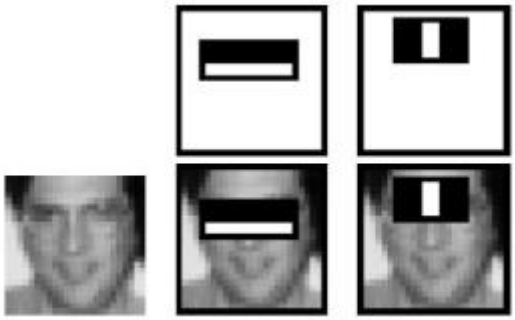


Ilustración 16. Características seleccionadas por la cascada de detectores Viola & Jones.

Integral de la imagen

Para computar rápidamente cada uno de los rectángulos se usa una representación de la imagen llamada "integral de la imagen". La integral de una imagen respecto a un punto (x,y) consiste en la suma de los píxeles por arriba y a la izquierda de dichos puntos, (x,y) incluidos.

$$ii(x,y) = \sum_{x' \leq x,\, y' \leq y} i(x',y')$$

La integral de la imagen representa en computación una manera elaborada de obtener valores de forma eficiente. Este método permite el uso de programación dinámica, que admite la obtención de valores dentro de la imagen a través de otros valores calculados previamente. [21]

Clasificador

Una vez extraídas las características de la imagen lo siguiente es clasificar. Para ello se emplea un entrenador para cada característica. Este entrenador es capaz de calcular el umbral óptimo para la clasificación, a través de las muestras positivas y negativas.

$$h_i(x) = \begin{cases} 1 \; si \; p_i f_i(x) < p_i \theta_i \\ 0 \; otro \; valor \end{cases}$$

Donde:

$h_i(x)$es el clasificador en función de x, que es la imagen.
p_i la paridad o la dirección de la inecuación
θ_i el umbral
$f_i(x)$ la característica que estamos evaluando.

Cascada de clasificadores

Los clasificadores se encadenan y en cada etapa se rechaza o se acepta una imagen para continuar en la siguiente etapa. Si atraviesa todas las etapas existe una alta probabilidad de que en esa zona se encuentre el objeto requerido. Cada etapa de la cascada de clasificadores se entrena haciendo que la tasa de aciertos aumenta lo suficiente y la de falsos positivos desciende drásticamente. [20]

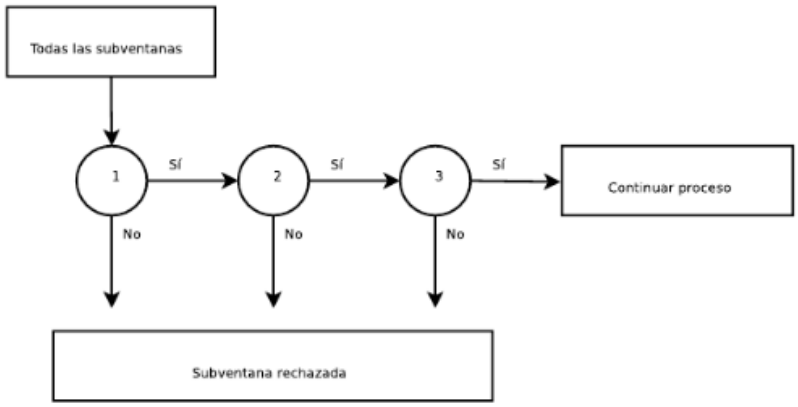


Ilustración 17. Diagrama de cascada de clasificadores.

3.1.4.3. OpenCV

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X, Windows y Android. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica. [22]

3.2. Dificultades de la visión artificial

Son varios los factores que hacen que la visión artificial sea difícil de manejar, conceptos tan básicos como la escala o la iluminación, que podemos controlar fácilmente con el sistema de visión humano, son factores a tener muy en cuenta a la hora de trabajar con visión artificial.

Ambigüedad en la definición de un concepto

Se tiene variabilidad dentro de una misma clase como se muestra en la siguiente figura que la clase es la silla, pero se tiene diferentes modelos y colores.

Ilustración 18. Ambigüedad en la definición de conceptos.[23]

Cambios de iluminación

Los cambios de iluminación pueden crear sombras o reflejos, y producir pérdidas importantes de información. Lo ideal para evitar estos problemas es trabajar con una iluminación uniforme.

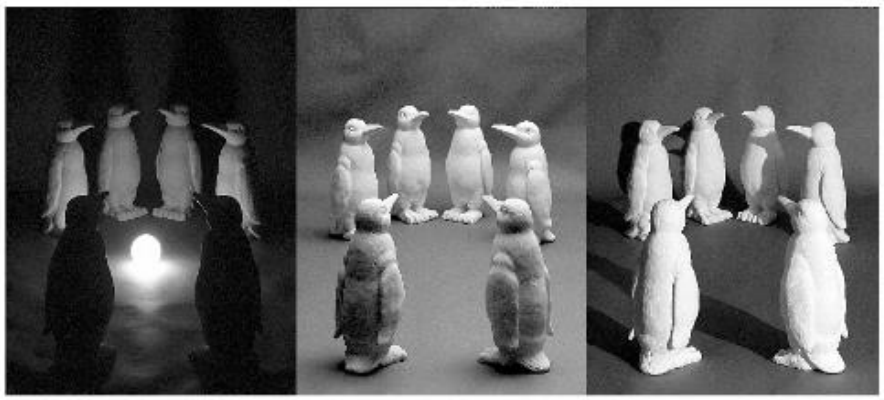


Ilustración 19. Cambios de iluminación.[23]

Cambios de escala

Los cambios de escala hacen que se pierda el seguimiento de objetos lo que hace que se inicie de nuevo la búsqueda del objeto para posteriormente su seguimiento provocando un mayor uso de recursos de hardware.

Ilustración 20. Cambio de escala.[23]

Deformación

La deformación de objetos da lugar a confusiones en el sistema, lo que conlleva que el seguimiento de objetos no funcione correctamente.

Ilustración 21. Deformación.[23]

Oclusión

Una oclusión es la percepción visual de un objeto que se encuentra detrás o en frente de otro objeto, otorgando información sobre el orden de las capas de la textura. Cuando un objeto es ocluido, el sistema visual solo tiene información sobre las partes del objeto que son visibles, por lo que el resto del procedimiento tiene que ser más profundo.

Ilustración 22. Oclusión.[23]

Movimiento

Con movimientos rápidos se pierde información del objeto, ya que la velocidad de refresco del sistema de captura debe ser muy potente para que ocurra lo contrario. Esto conlleva errores en el seguimiento del objeto de interés.

Ilustración 23. Movimiento.[23]

Pérdida de información

Las imágenes obtenidas por el sistema de captura suelen ser bidimensionales (2D) aunque las escenas reales capturadas sean tridimensionales, por lo que siempre habrá pérdida de información del entorno.

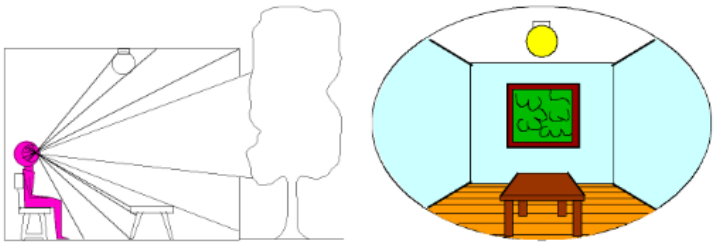


Ilustración 24. Pérdida de información del entorno

4. Metodología

El sistema a desarrollar en este trabajo consistirá en, a partir de una imagen tomada como entrada en el sistema, obtener un estado de fatiga a la salida. Para conseguir esto se pasará la imagen del conductor por distintos subsistemas, que se encargarán de detectar y reconocer el estado del sujeto en tiempo real para su posterior alerta, en caso de que sea necesario.

Para simplificar la realización del proyecto, se han desarrollado por separado cada uno de los subsistemas de detección, los cuales se enumeran a continuación:

- Detección de rostro
- Detección de ojos
- Seguimiento del ojo
- Estado del ojo
- Detección de boca
- Estado de la boca
- Estado de fatiga

En este capítulo, se muestra la funcionalidad de cada uno de los subsistemas nombrados, así como el método de desarrollo utilizado para el mismo.

4.1. Procesamiento previo

Antes de comenzar con los algoritmos de detección, se ha de procesar la imagen de entrada al sistema con el fin de adaptarla al medio. Para ello se realizan dos operaciones básicas en el procesamiento digital de la imagen: conversión a escala de grises y reescalado.

Conversión a escala de grises

Es muy importante convertir la imagen de entrada, que será RGB, a una imagen únicamente en escala de grises. Esto reduce en dos tercios la cantidad de píxeles a procesar ya que, con una imagen a color, se disponen de tres canales (uno para cada color) mientras que, una imagen en escala de grises, tiene un solo canal de color, tomando cada pixel un valor entre 0 y 255.

Además, al reducir el número de píxeles a procesar, también se reduce el tiempo de procesamiento, algo que es vital cuando se trabaja en un sistema que trabaja en tiempo real.

Para convertir la imagen original a una en escala de grises se utiliza el siguiente comando de OpenCV.

im=cv2.cvtColor(im_rgb,cv2.COLOR_RGB2GRAY)

Siendo:

- ***im_rgb***: imagen de entrada al sistema con tres canales de color
- ***im:*** imagen convertida a escala de grises

Ilustración 25. Imagen original RGB frente a imagen convertida a escala de grises.

Reescalado de la imagen

Una vez se tiene la imagen en escala de grises se pasa a realizar un reescalado de la imagen con el objetivo de disminuir considerablemente el número de píxeles a procesar. Para ello primero hay que comprobar la proporción de la imagen de entrada, si esta es 4/3 el tamaño a la salida será de 600x450 píxeles, en cambio, si la imagen es 16/9 su tamaño de reescalado será de 640x360 píxeles.

Este procedimiento es muy importante para reducir el tiempo de procesamiento del sistema de detección de fatiga, ya que reduciendo la resolución de la imagen a la mitad (pasar de 1200x900 a 600x450 como en el ejemplo), el número de píxeles de la misma se divide por cuatro, y por lo tanto el procesamiento será cuatro veces más rápido. El código utilizado para el reescalado de la imagen es el siguiente.

im_resize=cv2.UMat(cv2.resize(im,(600,450)))

Dónde:

- ***im_resize:*** imagen de salida ya reescalada.
- ***im:*** imagen en escala de grises de tamaño original.
- ***(600,450):*** tamaño de reescalado

Ilustración 26. Imagen de tamaño original frente a imagen reescalada.

Aceleración por hardware

En cuanto al comando cv2.UMat() que se ejecuta junto con el reescalado, se trata de un método de aceleración por hardware desarrollado por OpenCL (Open Computing Language).

OpenCL intenta aprovechar la potencia de los procesadores gráficos para realizar operaciones intensas repartidas entre el procesador del equipo (CPU) y la GPU de cualquier tarjeta gráfica compatible. Al contrario de CUDA o Stream, OpenCL fue creado originalmente por Apple quien luego la propuso al Grupo Khronos para convertirlo en un estándar abierto y libre que no dependa de un hardware de un determinado fabricante (CUDA sólo está disponible en gráficas NVidia y Stream en gráficas de ATI).

Que sea abierto y libre permite llevar OpenCL a un entorno multiplataforma, pudiendo ser aprovechado sobre cualquier plataforma y sistema operativo. Ya son muchos los dispositivos compatibles con OpenCL, indistintamente de la marca del fabricante, incluso es posible utilizarlo en las placas de bajo coste como Raspberry PI. [24]

En el caso de este proyecto, con el comando cv2.UMat() se carga la imagen, ya convertida a escala de grises y reescalada, en la memoria de la GPU para que las posteriores operaciones se realicen con apoyo del procesamiento gráfico.

Esto reducirá considerablemente el tiempo de operación siempre y cuando el tamaño de la imagen sea lo suficientemente grande, ya que, si la imagen es pequeña, el tiempo de carga de la imagen en la GPU será mayor que el tiempo de procesamiento de la misma, formando un cuello de botella innecesario.

4.2. Detección del rostro

Una vez se tiene la imagen adaptada al medio de procesamiento, se pasa a la detección del rostro del sujeto en cuestión dentro de la imagen de entrada del sistema. Para ello se ha optado por utilizar uno de los clasificadores en cascada que ofrece la

herramienta OpenCV, basados en el algoritmo de Viola & Jones del que se ha hablado en profundidad anteriormente.

Se trata de un detector basado en Características-Haar que ya ha sido entrenado para la localización de caras con un gran conjunto de imágenes, cuya información de entrenamiento se encuentra localizada en un documento XML. OpenCV ofrece algunos detectores Haar predeterminados para su uso libre, cada uno entrenado con una finalidad específica como detectar los ojos, la nariz o la boca. [20]

Para este caso se ha seleccionado un clasificador de caras frontales con el fin de localizar el rostro del sujeto al volante. El documento XML se ha importado al sistema a través de la siguiente línea.

face_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_frontalface_alt2.xml')

Se realizará una detección multi-escala del rostro con este clasificador, lo que permite detectar el objeto de interés independientemente de su tamaño dentro de la imagen. Esto es posible reescalando la imagen original según un factor de escala y buscando el objeto en cada una de las iteraciones. El comando utilizado para la detección multi-escala es el siguiente:

faces = face_classifier.detectMultiScale(img, scaleFactor=1.1, minNeighbors=3)

Dónde:

- ***faces***: vector donde se guardan las coordenadas, altura y anchura (x,y,w,h) del área donde se han detectado objetos de interés.
- ***img***: imagen a la que se va a realizar la detección en multi-escala.
- ***scaleFactor:*** indica cuánto se va a reescalar la imagen en cada iteración. Se ha introducido un valor del 10%, lo recomendado por la documentación de OpenCV.
- ***minNeighbors:*** este parámetro indica el número de veces que ha de detectarse la misma cara en escalas distintas, si se descubre el mismo rostro en 3 o más escalas de la imagen, se considera que es una detección válida.

Una vez obtenidos los rostros detectados se realiza una simple comprobación para verificar que únicamente se ha detectado una persona en la imagen, ya que este sistema de detección de fatiga está enfocado exclusivamente a conocer el estado del conductor del vehículo. Si únicamente se detecta una cara en la imagen, se dibuja un cuadrado delimitando el área donde esta se ha localizado.

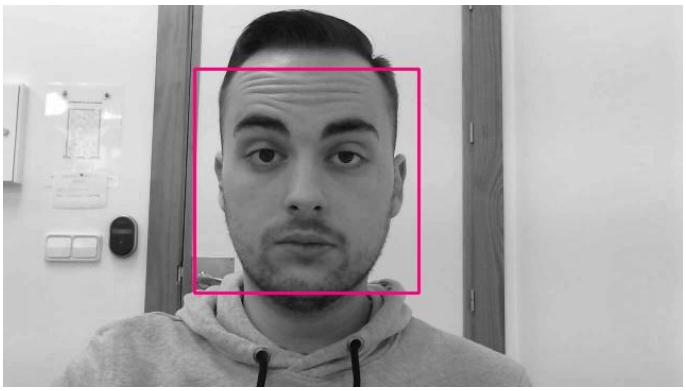


Ilustración 27. Imagen con rostro detectado por el clasificador multi-escala.

A continuación, se realiza un recorte del área donde se ha localizado la cara del sujeto, quedando únicamente la región de interés que será procesada más adelante.

Ilustración 28. Recorte del área localizada por el clasificador.

4.3. Segmentación de la cara

Una vez obtenido el rostro del sujeto, el siguiente paso sería localizar los ojos del individuo, así como su boca para poder realizar el seguimiento del estado de fatiga. Para simplificar este proceso, se realiza una segmentación de la imagen anterior en dos nuevas imágenes que delimiten el área donde se pueden localizar los ojos y la boca. Esto reducirá el tamaño de los archivos a procesar por los algoritmos de detección posteriores, lo que conlleva un menor tiempo de procesamiento del sistema.

La segmentación nombrada se realiza del siguiente modo:

- Para delimitar el área donde se puedan encontrar los ojos del sujeto, se hace un recorte de la imagen tomando el 80% de anchura respecto del centro de la cara y un 40% de altura, desde el 15% al 55% tomando como origen la esquina superior izquierda de la imagen.

- Para delimitar el área donde se pueda localizar la boca del individuo, se realiza un recorte de la imagen tomando un 60% de la anchura respecto al centro de la cara, así como el 40% de la parte inferior de la misma.

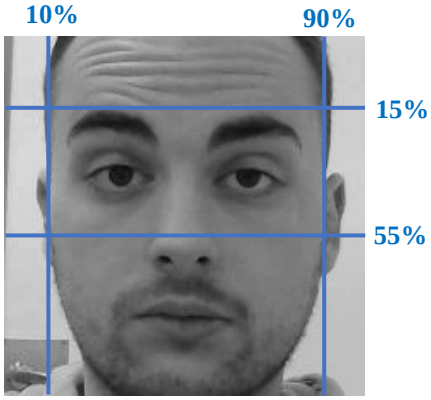


Ilustración 29. Segmentación de la cara del conductor para obtener área de los ojos.

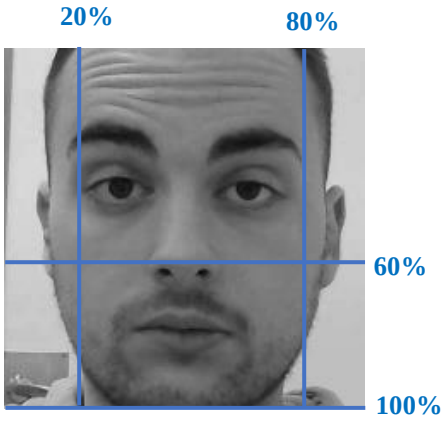


Ilustración 30. Segmentación de la cara del conductor para obtener área de la boca.

Para llevar a cabo este procedimiento, únicamente se ha hecho un recorte de la imagen que contiene la cara del sujeto a través de los siguientes comandos.

*ROI_eyes=ROI_face[int(dimX*0.15):int(dimX*0.55),int(dimY*0.1):int(dimY*0.9)]*

*ROI_mouth=ROI_face[int(dimX*0.6):int(dimX*1),int(dimY*0.2):int(dimY*0.8)]*

Con esto conseguimos delimitar las distintas zonas de interés correspondientes con el rostro del sujeto, atendiendo a la estructura facial de la que dispone cualquier individuo. Esto permitirá una mayor velocidad de procesamiento, ya que se pasarán estas nuevas imágenes a los futuros detectores y no la imagen del rostro completo.

Ilustración 31. Región de interés correspondiente a los ojos.

Ilustración 32. Región de interés correspondiente a la boca.

4.4. Detección de los ojos

Al igual que para la detección de la cara, se ha optado por utilizar un clasificador en cascada para la detección de los ojos del conductor. Para ello se ha seleccionado un clasificador de los que ofrece OpenCV, entrenado especialmente para la detección de ojos de personas. Se introduce en el proyecto a través del siguiente comando.

eye_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_eye.xml')

La imagen de entrada al clasificador en cascada, será la que se ha obtenido anteriormente en el proceso de segmentación que muestra únicamente los ojos del sujeto (Ilustración 31).

Antes de ejecutar el proceso de detección se comprueba que la iluminación de la imagen sea la correcta basándose en el valor medio de nivel de gris de la misma. Si está por encima de un umbral, se considerará una imagen poco iluminada y se realizará una ecualización del histograma con el fin de homogeneizar los distintos niveles de iluminación, quedando del siguiente modo.

imagen=cv2.equalizeHist(imagen)

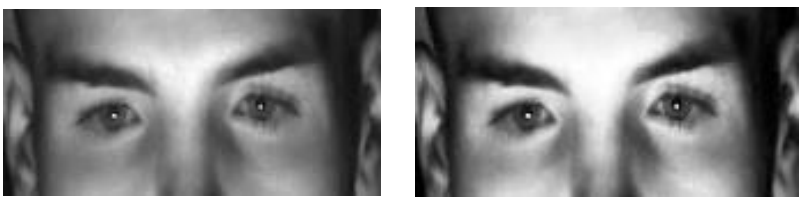


Ilustración 33. Imagen antes y después del proceso de ecualización de histograma en imagen con una iluminación no uniforme.

Se puede observar como la imagen ha mejorado una vez ejecutada la ecualización de histograma, esto reportará unos mejores resultados a la hora de la detección, proceso que es fundamental para el correcto funcionamiento del sistema.

Si la imagen ya está iluminada uniformemente, y su valor medio de nivel de gris está dentro de los valores aceptables, no es necesario hacer una ecualización de histograma, ya que el resultado sería peor que la imagen original como se puede apreciar en la siguiente ilustración.

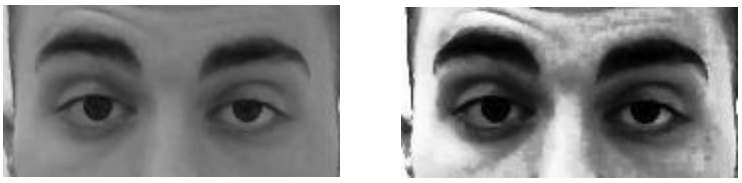


Ilustración 34. Ecualización de histograma sobre imagen uniformemente iluminada.

El objetivo del detector es localizar los ojos dentro de esta imagen para así conocer su ubicación, siempre y cuando estos se encuentren realmente en la imagen de entrada. La detección se realiza, al igual que se ha hecho anteriormente para la cara, a través del siguiente comando.

eyes = eye_classifier.detectMultiScale(imagen, scaleFactor=1.1, minNeighbors=3)

Donde:

- ***eyes***: vector donde se guardan las coordenadas, altura y anchura (x,y,w,h) del área donde se han detectado objetos de interés.
- ***imagen***: imagen a la que se va a realizar la detección en multi-escala.
- ***scaleFactor:*** indica cuánto se va a reescalar la imagen en cada iteración. Se ha introducido un valor del 10%, lo recomendado por la documentación de OpenCV.
- ***minNeighbors:*** este parámetro indica el número de veces que ha de detectarse la misma cara en escalas distintas, si se descubre el mismo rostro en 3 o más escalas de la imagen, se considera que es una detección válida.

Una vez obtenidas las coordenadas de localización de los ojos, se hace una simple comprobación para verificar que se han detectado al menos dos ojos, ya que, si únicamente se localiza uno o ninguno, no se considerará un resultado válido para el sistema que se está desarrollando y se pasará al procesamiento del siguiente frame. Si el resultado obtenido es admitido, es decir, se han detectado ambos ojos del sujeto, se dibuja un cuadrado que delimita el área de localización.

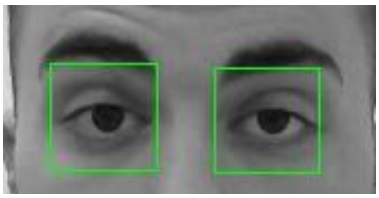


Ilustración 35. Detección de ojos en sujeto.

Conocida la localización de los ojos en la cara, se realiza un recorte de los mismos para obtener por separado ambos ojos y poder así, estudiar su estado de apertura en un proceso posterior. El recorte de las regiones de interés correspondientes a cada ojo se realiza del siguiente modo.

ROI_eye1=eyes[(ey1-expand_eyes):(ey1+eh1+expand_eyes) ,(ex1-expand_eyes):(ex1+ew1+expand_eyes)]

ROI_eye2=eyes[(ey2-expand_eyes):(ey2+eh2+expand_eyes) ,(ex2-expand_eyes):(ex2+ew2+expand_eyes)]

Siendo:

- ***ey:*** coordenada 'y' de la esquina superior izquierda del cuadrado que delimita el ojo.
- ***ex:*** coordenada 'x' de la esquina superior izquierda del cuadrado que delimita el ojo
- ***ew:*** anchura del cuadrado que delimita el ojo
- eh: altura del cuadrado que delimita el ojo
- ***expand_eyes:*** número de píxeles de margen tomado hacia todas las direcciones, con el fin de no perder información que pueda ser de interés.

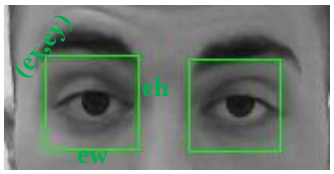


Ilustración 36. Explicación de variables para localización de los ojos.

Ilustración 37. Ojo izquierdo localizado por el detector.

Ilustración 38. Ojo derecho localizado por el detector.

4.5. Estado del ojo

El siguiente paso una vez obtenidas las imágenes correspondientes a los ojos del conductor que se quiere monitorizar, es el procesamiento de las mismas para conocer el estado del ojo. El objetivo de esto, es conocer en cada momento el estado de apertura del ojo del conductor, para poder estimar así, el estado de fatiga en el que se encuentra.

En este proyecto se han desarrollado distintos métodos que permiten evaluar la apertura del ojo en cualquier condición de iluminación y cualquier usuario, adaptando inicialmente la imagen de entrada con técnicas de preprocesamiento para posteriormente evaluar la apertura del ojo.

Por una parte, se ha apostado por el procesamiento morfológico de la imagen con el fin de localizar y estudiar el estado del iris, aprovechando la diferencia de luminosidad entre esta y el iris.

Por otro lado, se ha desarrollado un método basado en la proyección horizontal y vertical del ojo, con la finalidad también, de conocer el estado del iris y poder así estimar la apertura del ojo.

4.5.1. Procesado morfológico

El procesamiento morfológico, es una técnica de procesamiento no lineal de la señal, caracterizada en realzar la geometría y forma de los objetos. Su fundamento matemático se basa en la teoría de conjuntos. El objetivo de las transformaciones morfológicos es la extracción de estructuras geométricas en los conjuntos sobre los que se opera, mediante la utilización de otro conjunto de forma conocida, al que se le denomina elemento estructurante.[25]

Es por lo mencionado anteriormente, por lo que se ha decidido utilizar este tipo de procesamiento con el fin de localizar el iris del conductor, cuya forma es circular, y estimar así la apertura del ojo del sujeto. La imagen de entrada al subsistema de procesamiento morfológico será la del ojo del sujeto (*Ilustración 37*) y se obtendrá como salida un valor correspondiente a la apertura vertical del mismo.

En este tipo de operaciones morfológicas, el ruido de la imagen es un factor muy a tener en cuenta y es primordial que se encuentre bajo valores mínimos, por lo que el

primer paso es filtrar la imagen con un núcleo gaussiano. Al ser la imagen a procesar de un tamaño mínimo, se ha seleccionado un núcleo de convolución para el filtrado de tamaño 3x3, lo suficiente para eliminar cualquier muestra de ruido impulsivo existente. A cambio, la imagen de salida del filtro adquirirá un efecto de suavizado (*Ilustración 39*) que para el caso presente es irrelevante. El filtrado paso bajo mencionado se ha realizado a través de una herramienta que ofrece OpenCV del siguiente modo.

img_suav=cv2.GaussianBlur(imagen, (3,3),0)

Siendo:

- ***img_suav:*** imagen suavizada a la salida del filtro.
- ***imagen:*** imagen del ojo a filtrar.
- ***(3,3):*** dimensión del núcleo gaussiano.

Ilustración 39. Imagen original frente a imagen filtrada.

El siguiente paso consistirá en la localización del iris del sujeto a través de dos operaciones de cierre con un elemento estructural de tipo circular. Al aplicar un cierre a la imagen, lo que se hace es eliminar objetos oscuros más pequeños que el elemento estructurante, al ser este de tipo circular, se eliminarán todos los objetos oscuros con forma circular que sean mas pequeños que el elemento estructurante.

Para que el resultado de estas operaciones sea concluyente, se ha de conocer el tamaño en píxeles del iris de forma aproximada, de tal modo que dependiendo de la distancia a la que se encuentre el conductor de la cámara utilizada para la adquisición de imágenes, habrá que seleccionar un elemento estructurante adecuado. Además de esto, la resolución de las imágenes de entrada al sistema también tiene que tenerse en cuenta para ajustar el tamaño del elemento estructurante, por lo que cuando se proceda a la instalación del sistema de detección de fatiga, habrá que ajustar estos términos para su correcto funcionamiento.

En el caso de la imagen utilizada en este proyecto, el iris tiene un tamaño aproximado de 13 píxeles de diámetro, por lo que se han seleccionado unos elementos estructurantes de 9 y 15 píxeles, que se han declarado del siguiente modo.

kernel9=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9,9))

kernel15=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15,15))

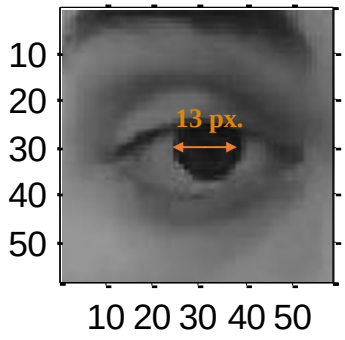


Ilustración 40. Tamaño del iris.

El proceso para la localización del iris va a consistir en dos cierres consecutivos con distintos elementos estructurantes. El primero se realizará con el de 9 píxeles de diámetro con el objetivo de eliminar todos los elementos oscuros de la imagen de menor tamaño que el iris. Esta operación se realiza sobre la imagen previamente suavizada del siguiente modo.

img1=cv2.morphologyEx(img_suav, cv2.MORPH_CLOSE, kernel9)

Siendo:

- ***img1:*** imagen se salida tras la operación de cierre morfológico.
- ***img_suav:*** imagen de entrada a la que se realiza la operación de cierre.
- ***cv2.MORPH_CLOSE:*** parámetro que indica el tipo de operación morfológica que se va a realizar.
- ***kernel9:*** elemento estructurante de forma circular y diámetro de 9 píxeles.

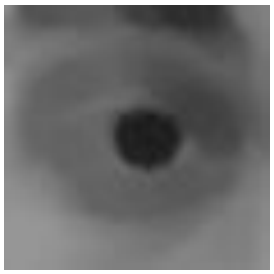


Ilustración 41. Imagen tras primer cierre.

Cómo se puede apreciar en la *Ilustración 41*, se han eliminado todos los detalles oscuros más pequeños que el elemento estructurante, es decir, todos los elementos oscuros menores de 9 píxeles de diámetro.

El siguiente paso es realizar otro cierre sobre esta imagen, pero ahora, con un elemento estructurante circular más grande que el iris del sujeto. Con esto se logrará

eliminar el iris de la imagen y poder aislarla de los demás elementos que no son de interés. El comando utilizado es el siguiente.

img2=cv2.morphologyEx(img1, cv2.MORPH_CLOSE, kernel15)

Siendo:

- ***img2:*** imagen de salida tras la operación del cierre morfológico.
- ***img1:*** imagen de entrada a la que se aplicará el cierre, en este caso correspondiente a la ilustración 41.
- ***cv2.MORPH_CLOSE:*** parámetro que indica el tipo de operación morfológica que se va a realizar.
- ***kernel15:*** elemento estructurante circular de 15 píxeles de diámetro.

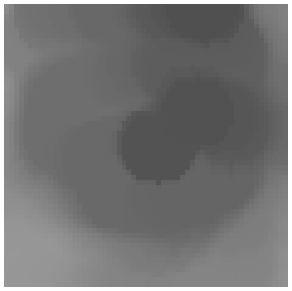


Ilustración 42. Imagen del ojo tras segundo cierre.

Una vez realizado el segundo cierre y eliminado el iris de la imagen, el siguiente paso será realizar una resta de la imagen de salida del segundo cierre (*Ilustración 42*) con la imagen de salida del primer cierre (*Ilustración 41*). Esto dará como resultado una imagen que contiene únicamente los elementos eliminados en el segundo cierre, es decir, el iris junto con algún otro objeto que se haya podido eliminar. Para realizar una resta de dos imágenes, se utilizará un comando ofrecido por OpenCV del siguiente modo.

img3=cv2.subtract(img2, img1)

Dónde:

- ***img3:*** resto o diferencia de la operación.
- ***img2:*** minuendo de la resta (*Ilustración 42*).
- ***img1:*** sustraendo de la resta (*Ilustración 41*).

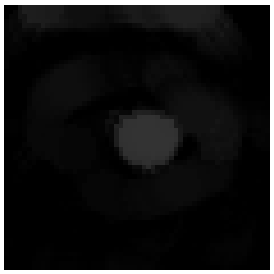


Ilustración 43. Resultado de la resta entre ambos cierres.

Tras realizar la resta entra ambas imágenes, ya se tiene aislado el iris del conductor como se puede apreciar en la *Ilustración 43*, el siguiente paso es realizar una umbralización de esta imagen con el fin de convertirla en una imagen binaria, donde el área coincidente con el iris se muestre en blanco y todo lo demás en negro.

El proceso de umbralización consiste en la transformación de una imagen en escala de grises con 256 niveles, a otra imagen binaria únicamente con 2 niveles, blanco y negro. Esta transformación se realiza a partir de un umbral especificado por el usuario, por lo que es muy importante que la elección del umbral sea la adecuada para el correcto funcionamiento del sistema.

En el caso que se presenta, se ha elegido el umbral en función del contenido de la imagen a umbralizar, tomando como referencia el valor de mayor luminosidad presente en la misma. En concreto, se ha establecido el umbral en el 75% del máximo valor de luminosidad, es decir, si un píxel presenta un nivel de gris mayor que el 75% del máximo, pasará a ser blanco, y sino se convertirá en negro. La umbralización se ha realizado de la siguiente manera.

*umbral1=np.amax(img3)*0.75*

ret,img_bw=cv2.threshold(img3, umbral1,255, cv2.THRESH_BINARY)

Dónde:

- ***ret:*** devuelve el umbral utilizado para la binarización.
- ***img_bw:*** imagen umbralizada.
- ***img3:*** imagen a umbralizar (*Ilustración 43*).
- ***umbral1:*** umbral utilizado, definido anteriormente.
- ***255:*** número máximo de niveles de gris en la imagen de entrada.
- ***cv2.TRESH_BINARY: tipo*** de umbralización a realizar, en este caso se indica que la imagen de salida va a ser de tipo binario (0 y 1).

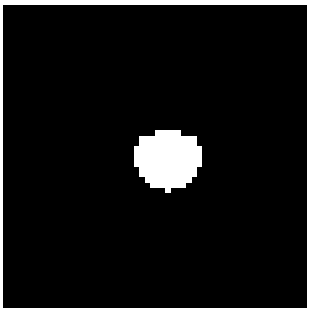


Ilustración 44. Imagen tras umbralización.

La imagen obtenida tras la umbralización (*Ilustración 44)* se corresponde con el área donde se localiza el iris del conductor del vehículo, por lo que a partir de esta se puede obtener el valor correspondiente con la apertura vertical del ojo. El siguiente paso es obtener el contorno del mismo para facilitar el proceso de extracción de valores, que se realizará también con ayuda de operaciones morfológicas.

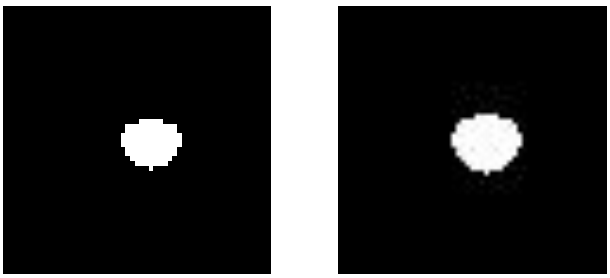
La manera mas simple de obtener el contorno de una figura es dilatándola con un elemento estructurante pequeño y realizando la resta con la imagen original. El proceso de dilatación es una operación morfológica que consiste en ensanchar las figuras existentes en una imagen en función del elemento estructurante escogido, para este caso se ha escogido uno circular con un tamaño de 3 píxeles de diámetro con el fin de obtener un contorno final de este grosor. Los comandos utilizados para esta operación son los mostrados a continuación.

kernel3=cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))

img4=cv2.dilate(img_bw, kernel3)

Dónde:

- ***img4:*** imagen dilatada.
- ***img_bw:*** imagen umbralizada que va a ser dilatada (*Ilustración 44*).
- ***kernel3:*** elemento estructurante utilizado para la dilatación.



(a) (b)

Ilustración 45. (a) Imagen antes de dilatación, (b) imagen tras dilatación.

Una vez realizada la dilatación se procede a restar ambas imágenes anteriormente mostradas (*Ilustración 45*), en concreto se realiza la diferencia de la imagen dilatada (b) menos la imagen sin dilatar (a) para obtener el contorno del iris. La resta, al igual que antes, se ejecuta con el siguiente comando.

img5=cv2.subtract(img4,img_bw)

Dónde:

- ***img5:*** resto o diferencia de la operación.
- ***Img4:*** minuendo de la resta (*Ilustración 45a*).
- ***Img_bw:*** sustraendo de la resta (*Ilustración 45b*).

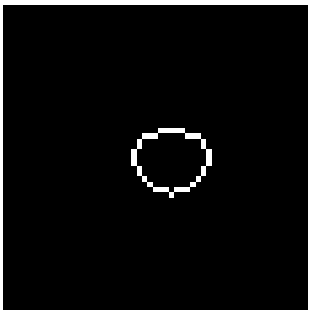


Ilustración 46. Contorno final del iris.

El contorno obtenido con esta operación, se corresponde con el contorno del iris del conductor del vehículo, lo que permitirá conocer de manera indirecta la apertura del ojo y poder intuir así el estado de fatiga en el que se encuentra el sujeto. Para comprobar que efectivamente el contorno obtenido coincide con el iris, se puede dibujar sobre la imagen del ojo en cuestión, tal y como se muestra en la siguiente figura.

Ilustración 47. Contorno obtenido sobre ojo procesado.

El siguiente objetivo es hallar la apertura del ojo, para lo que se han desarrollado dos técnicas distintas en este proyecto. Una de ellas se basa en la aproximación del contorno del iris a una elipse, de la que se pueden hallar sus parámetros básicos como son ambos ejes y la excentricidad de la misma. La otra técnica se basa en el cálculo del centro de masa del contorno obtenido para, a partir de él, poder trazar la línea vertical que se corresponderá con la apertura del ojo.

4.5.1.1. Obtención del centro de masa del iris

El objetivo de esta técnica es la obtención de la apertura del ojo del sujeto, a partir de una imagen de entrada que será la del contorno del iris hallado anteriormente (*Ilustración 46*). Para ello se hallará, en primer lugar, el centro de masa correspondiente al iris del conductor.

El centro de masa es una posición definida en relación a un objeto o a un sistema de objetos y se calcula promediando la posición de todas las partes del sistema. Al tener como imagen de entrada una imagen con dos niveles de gris, 0 y 255, se buscará la posición de todos los píxeles cuyo valor sea 255, que se corresponde con un píxel blanco.

```
im_contorno=np.where(img5==255)
```

En la variable *'im_contorno'* se almacenarán todas las posiciones, en formato (x,y), donde exista un píxel blanco dentro de la imagen del contorno del iris. El siguiente paso es promediar, por un lado, todos los valores correspondientes al "eje x" y, por otro lado, todas las posiciones correspondientes al "eje y".

```
coord_y=np.sum(im_contorno[0])/(im_contorno[0].shape[0])
coord_x=np.sum(im_contorno[1])/(im_contorno[1].shape[0])
```

De tal modo que se obtiene el centro de masa del iris en la posición *"(coord_x,coord._y)"*. Se puede dibujar sobre la imagen del contorno del iris para comprobar que el funcionamiento del código es correcto.

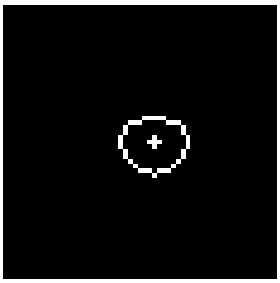


Ilustración 48. Centro de masa obtenido para el iris del conductor.

Una vez obtenido el centro de masa, es relativamente sencillo obtener la apertura del ojo, únicamente hay que buscar los píxeles blancos que tengan la misma posición en el *"eje x"* que el centro de masa, y entre esos píxeles, encontrar aquellos que tengan el valor máximo y mínimo en el *"eje y"*. La diferencia entre esas dos posiciones, dará como resultado el valor de apertura vertical del ojo. Este procedimiento es el equivalente a trazar la línea vertical que pase por el centro de masa y que corte con el contorno del iris, los píxeles frontera de esta línea serán los que aportarán el valor de apertura.

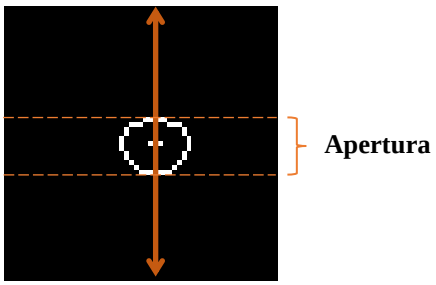


Ilustración 49. Método de obtención de la apertura por el método de centro de masa.

El problema de esta medida que se obtiene como resultado es que está expresada en píxeles y dependiendo de la distancia entre el conductor y la cámara, para un mismo valor de apertura del ojo, se pueden obtener distancias en píxeles distintas. Es por esto que el valor obtenido se dividirá entre el número de píxeles que contiene la imagen del ojo que toma como parámetro de entrada, obteniendo así un porcentaje de apertura.

$$apertura\ ojo\ (\%) = \frac{apertura\ vertical\ (píxeles)}{nº\ de\ píxeles\ de\ la\ imagen\ de\ entrada} * 100$$

4.5.1.2. Aproximación por elipse

Esta es otra de las técnicas desarrolladas en el proyecto para la obtención de la apertura del ojo del sujeto bajo monitorización. El funcionamiento de este subsistema consiste en la aproximación del iris, anteriormente detectado, a una elipse cuyos parámetros indicarán el valor de apertura vertical del ojo.

Para el desarrollo de esta técnica, se hará uso de una función ofrecida por la librería OpenCV, la cual permite aproximar cualquier contorno a una elipse de manera sencilla. Tras probar varios métodos, se llegó a la conclusión de que el contorno que necesita esta función como parámetro de entrada, no puede ser el obtenido anteriormente por procesos morfológicos, sino que debe de obtenerse a partir de otra función de OpenCV del siguiente modo.

contours, hierarchy = cv2.findContours(img_bw, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

Dónde:

- **contours:** variable de salida donde se acumulan los contornos encontrados. Cada contorno se guardará como un array con las coordenadas (x,y) de los puntos pertenecientes al mismo.
- **img_bw:** imagen donde se van a buscar los contornos, en este caso se correponde con la *Ilustración 44.*
- ***cv2.RETR_TREE:*** modo de recuperación de contornos.
- **cv2.*CHAIN_APPROX_SIMPLE:*** método de obtención de contornos. Este método únicamente guarda una serie de puntos relevantes sin necesidad de almacenar todos los puntos pertenecientes al contorno.

Se puede dibujar el contorno obtenido sobre la imagen del ojo de la siguiente manera.

cv2.drawContours(img_rgb, contours, -1, (0,255,0), 1)

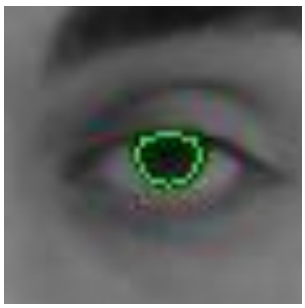


Ilustración 50. Contorno obtenido con la función ofrecida por OpenCV.

El mayor inconveniente de este método se encuentra en este punto, ya que si la imagen obtenida del iris contiene varios objetos no deseados, esta función dará como resultado varios contornos y consecuentemente, el sistema no funcionará de manera correcta. Si la extracción de contornos es satisfactoria, solo se encontrará el contorno correspondiente con el iris del conductor en cuestión, y el siguiente paso será aproximar este contorno a una elipse. La función que realiza este procedimiento tiene la siguiente estructura.

ellipse=cv2.fitEllipse(contours[0])

Dónde:

- ellipse: variable de salida de la función donde se almacenan los parámetros correspondientes a la elipse de aproximación.
- contours[0]: contorno correspondiente con el iris del sujeto.

Los valores devueltos por la función anterior serán: coordenadas donde se ubica el centro de la elipse, ángulo de orientación y las dimensiones de sus ejes. A partir de esto, se puede obtener la excentricidad de la elipse, valor que servirá de indicador para conocer la apertura del ojo. La excentricidad de una elipse es un valor que determina la forma de la misma, en el sentido de si es más redondeada o no, puede tomar valores entre 0 y 1, siendo 0 cuando la elipse es una circunferencia y distinta de 0 cuando su forma es achatada.

De este modo, cuando el conductor tenga los ojos completamiento abiertos, la excentricidad de la elipse que se aproxima a su iris tendrá un valor cercano a cero, mientras que, conforme vaya cerrando el ojo, el valor de excentricidad irá en aumento. La excentricidad se calcula del siguiente modo:

$$exc = \frac{\sqrt{a^2 - b^2}}{a^2}$$

Siendo ‘a’ el semieje menor de la elipse y ‘b’ el semieje mayor.

Se puede dibujar la elipse sobre la imagen del ojo del sujeto para comprobar que la aproximación es correcta. Para ello OpenCV ofrece una función específica que permite dibujar una elipse conociendo sus parámetros básicos. Con un ajuste de coordenadas, también es posible dibujar la figura sobre la imagen original de entrada al sistema de detección de fatiga, tal y como se muestra en la *Ilustración 52*.

cv2.ellipse(ROI_eye2,tuple(ellipse),(255,255,255),1)

Dónde:

- ***ROI_eye2:*** imagen sobre la que se va a dibujar la elipse.
- ***tuple(ellipse):*** parámetros de la elipse casteados a formato tuple.
- ***(255,255,255):*** color de la elipse, blanco en este caso.
- ***1:*** grosor de la línea.

Ilustración 51. Aproximación de elipse al iris del sujeto.

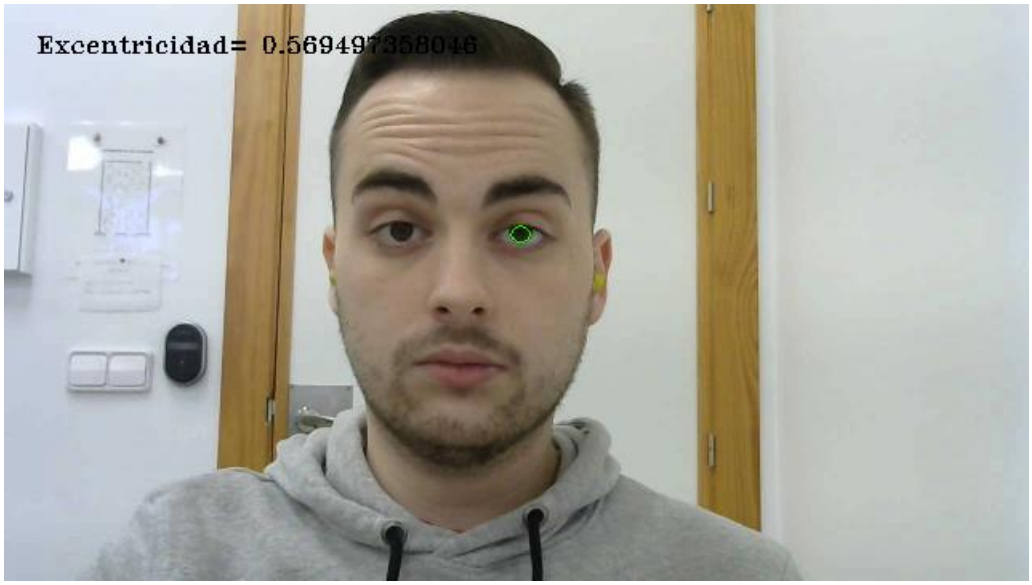


Ilustración 52. Imagen de entrada al sistema de detección de fatiga con elipse superpuesta.

4.5.2. Proyección vertical y horizontal

Además de la técnica anteriormente comentada basada en el procesamiento morfológico para la extracción del iris del conductor, se ha desarrollado un subsistema capaz de obtener la apertura del ojo a través de las proyecciones del mismo, tanto vertical como horizontal. Esta técnica se basa en la diferencia de niveles de gris que se observan entre el iris y la esclerótica del sujeto.

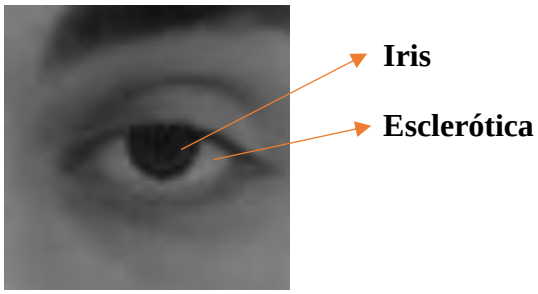


Ilustración 53. Partes del ojo.

La imagen de entrada a este subsistema será la del ojo del conductor (*Ilustración 53*) y se obtendrá a la salida el valor correspondiente a la apertura del ojo.

El primer paso de esta técnica consiste en, al igual que para el procesamiento morfológico, realizar una eliminación de ruido a través de un filtrado paso bajo con núcleo gaussiano. Se utilizará para ello el comando ofrecido por OpenCV para la aplicación de un filtro gaussiano.

img_suav=cv2.GaussianBlur(imagen,(3,3),0)

Siendo:

- ***img_suav:*** imagen suavizada a la salida del filtro.
- ***imagen:*** imagen del ojo a filtrar.
- ***(3,3):*** dimensión del núcleo gaussiano.

(a) (b)

Ilustración 54.(a) Imagen de entrada al sistema, (b) imagen filtrada con núcleo gaussiano 3x3

Una vez filtrada la imagen, se pasa a realizar una umbralización de la misma, con el objetivo de dejar únicamente en la imagen los objetos con un nivel de intensidad bajo, es decir, los oscuros. Es por esto que se impone un umbral de 50 unidades, para que únicamente permanezcan en la imagen aquellos píxeles cuyo nivel de gris sea inferior a este umbral, el resultado puede apreciarse en la *Ilustración 55*. El comando utilizado para esta operación de umbralización es el siguiente.

ret,img_umbr=cv2.threshold(img_suav, umbral2,255, cv2.THRESH_BINARY)

Dónde:

- ***ret:*** devuelve el umbral utilizado para la binarización.

- ***img_umbr:*** imagen umbralizada.
- ***img_suav:*** imagen a umbralizar (*Ilustración 54b*).
- ***umbral1:*** umbral utilizado, 50 en este caso.
- ***255:*** número máximo de niveles de gris en la imagen de entrada.
- ***cv2.TRESH_BINARY: tipo*** de umbralización a realizar, en este caso se indica que la imagen de salida va a ser de tipo binario (0 y 1).

Ilustración 55. Imagen del ojo umbralizada.

De la imagen resultado de la umbralización se obtienen únicamente los objetos que son de interés para obtener la apertura del ojo. Concretamente, será la proyección horizontal la que proporcione el valor de apertura. Esta proyección se obtiene realizando el sumatorio de cada una de las filas de la imagen, dando como resultado, el nivel de gris total de cada una de ellas.

proy_hor=np.sum(255-img_umbr,1)

Dónde:

- ***proy_hor:*** vector que almacena el valor total de cada fila.
- ***255-img_umbr:*** este parámetro de entrada corresponde con la imagen a la que se va a realizar el sumatorio, el objetivo es que ,a mayor cantidad de negros en la fila, mayor valor final, por lo que se realiza la inversa de la imagen restándola a 255.
- ***1:*** dimensión de la imagen a sumar, el valor 1 corresponde con la dimensión de las filas.

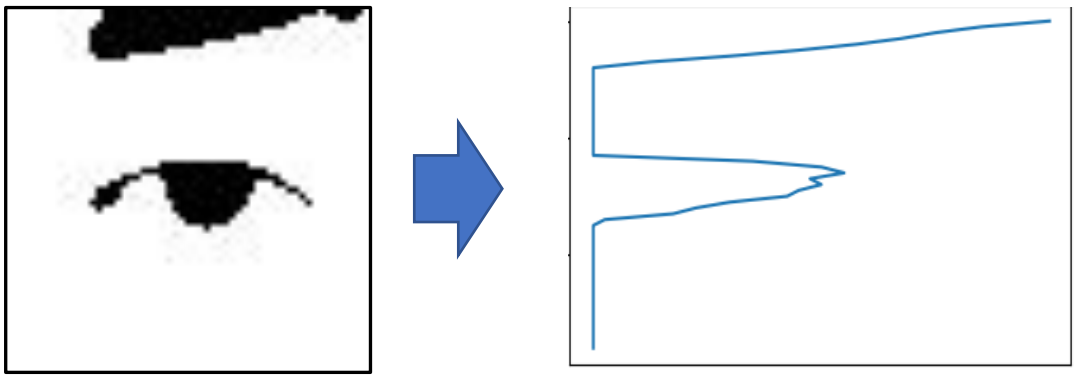


Ilustración 56. Proyección horizontal de la umbralización del ojo

Atendiendo a los resultados de la proyección horizontal mostrados en la *Ilustración 56*, se puede observar la presencia de dos picos, uno correspondiente a la ceja del usuario, y el otro al ojo del mismo, siendo este último el pico de interés para poder conocer la apertura del ojo.

Para la obtención de la apertura del ojo, únicamente se ha de establecer un umbral en la proyección horizontal de tal modo que, el número de píxeles del pico central correspondiente a la pupila del conductor que sobrepase este umbral, proporcionará la apertura del ojo en píxeles.

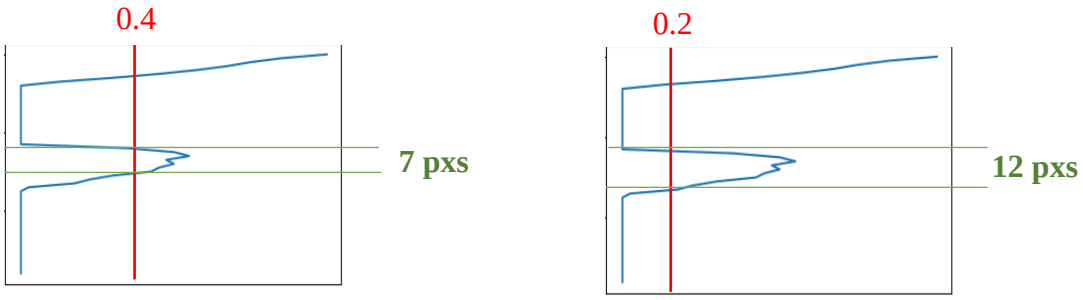


Ilustración 57. Obtención de la apertura con distintos umbrales.

El problema que presenta este método, es que la apertura del ojo se obtiene en píxeles, por lo que el tamaño real dependerá de la distancia que exista entre la cámara y el conductor del vehículo. De este modo, se pueden obtener distintos valores de píxeles para la misma apertura del ojo.

La solución propuesta para este problema consiste en transformar la medida en píxeles a un porcentaje de apertura en función de la anchura del ojo, que puede ser conocida con ayuda de la proyección vertical de la imagen del ojo umbralizada. Esta proyección se obtiene del mismo modo que la horizontal, pero sumando cada una de las

columnas de la imagen, por lo que el comando utilizado será el mismo pero modificando el segundo parámetro de entrada de la función (ahora la dimensión 0), del siguiente modo.

proy_ver=np.sum(255-img_umbr,0)

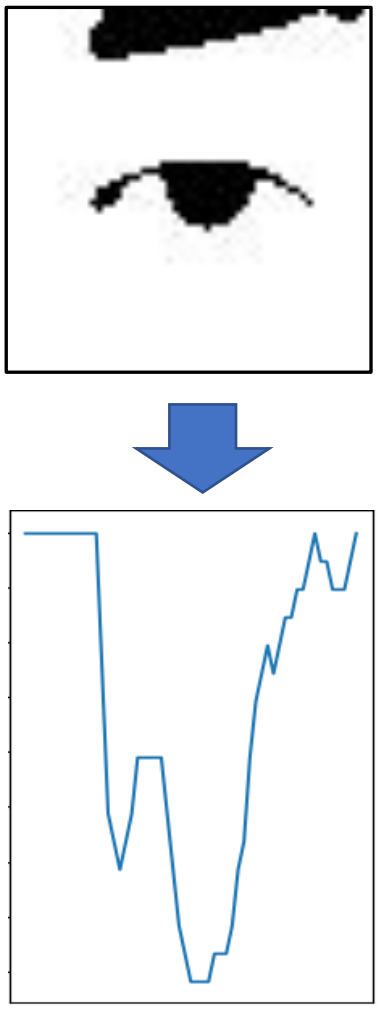


Ilustración 58. Proyección vertical de la umbralización del ojo.

Al igual que con la proyección horizontal, aplicando un umbral sobre la vertical se puede obtener fácilmente el valor en píxeles de la anchura del ojo. Conociendo la apertura y anchura del ojo del conductor en píxeles, se calculará el porcentaje de apertura del siguiente modo:

$$\%\ apertura = \frac{apertura\ en\ píxeles}{anchura\ en\ píxeles}$$

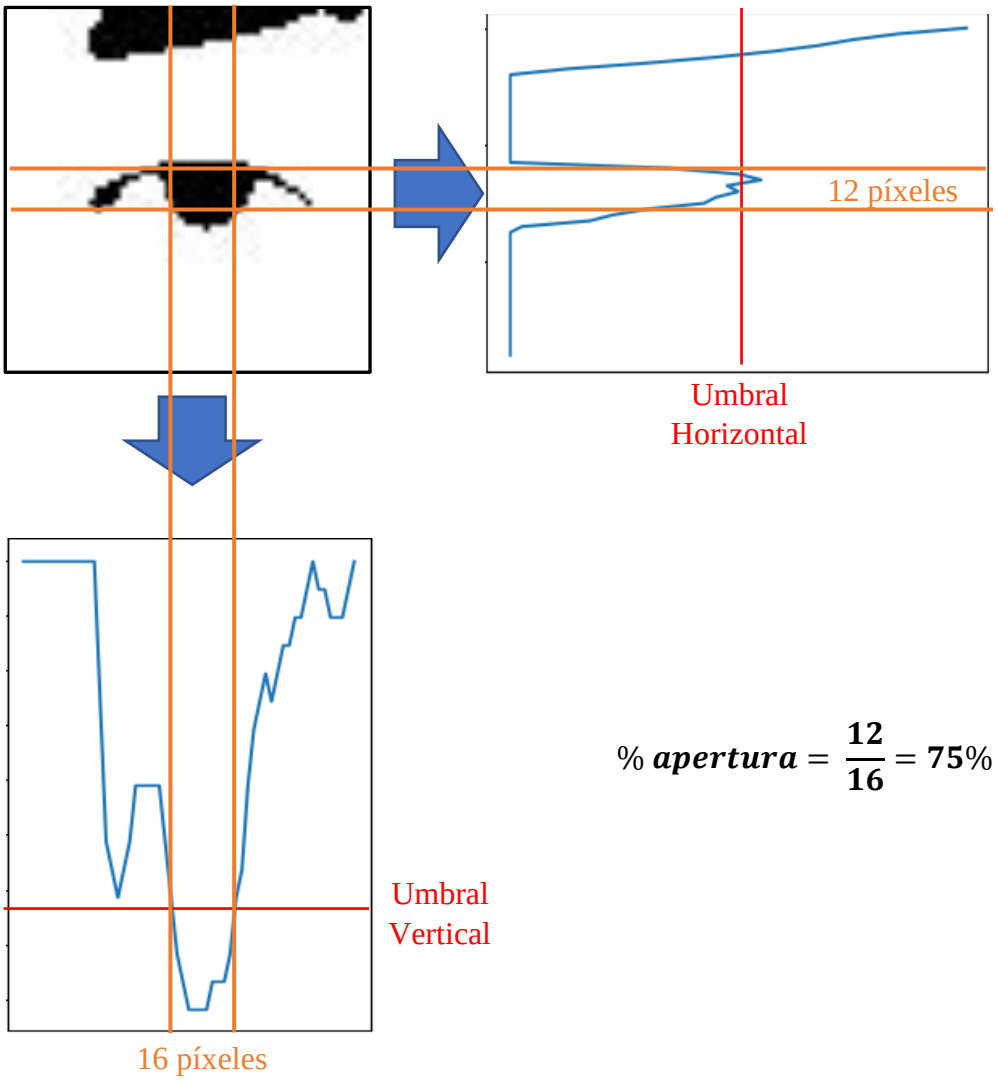


Ilustración 59. Método de obtención de porcentaje de apertura a través de ambas proyecciones.

4.6. Detección de la boca

El siguiente factor a tener en cuenta para la elaboración del sistema de detección de fatiga va a ser la apertura de la boca, mas concretamente, la detección de bostezos. Para el desarrollo de esta función, en primer lugar habrá que divisar la región donde se encuentra la boca del conductor, por lo que se usará de nuevo un detector en cascada igual que en apartados anteriores se usó para la localización de la cara y los ojos. Para ello se importa el documento XML con el detector previamente entrenado del siguiente modo.

mouth_classifier = cv2.CascadeClassifier('Haarcascades/mouth.xml')

La imagen sobre la que se realizará la detección será la anteriormente extraída, correspondiente a la región inferior de la cara (*Ilustración 32*). Para realizar la detección en multi-escala de la boca se procede con el siguiente comando.

mouth=mouth_classifier.detectMultiScale(imagen, scaleFactor=1.1,minNeighbors=3)

Donde:

- ***mouth***: vector donde se guardan las coordenadas, altura y anchura (x,y,w,h) del área donde se han detectado objetos de interés.
- ***img***: imagen a la que se va a realizar la detección en multi-escala.
- ***scaleFactor:*** indica cuánto se va a reescalar la imagen en cada iteración. Se ha introducido un valor del 10%, lo recomendado por la documentación de OpenCV.
- ***minNeighbors:*** este parámetro indica el número de veces que ha de detectarse la misma cara en escalas distintas, si se descubre el mismo rostro en 3 o más escalas de la imagen, se considera que es una detección válida.

Ilustración 60. Detección de boca.

El objetivo de este subsistema es la detección de bostezos, por lo que se han realizado pruebas del detector con imágenes en las que aparecen personas bostezando. La detección en estas imágenes ha sido nula, ya que el detector está entrenado únicamente con imágenes de bocas cerradas.

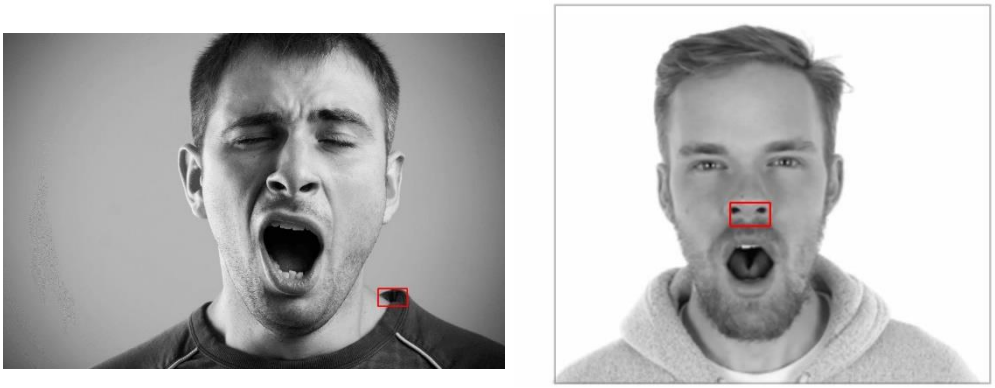


Ilustración 61. Detección errónea de la boca cuando el usuario bosteza.

4.7. Apertura de la boca

Tras comprobar que la detección de la boca del usuario no funciona cuando este se encuentra bostezando, se decide usar la imagen anterior a la detección (*Ilustración 32*), en la que se localiza la boca recortando la parte inferior de la cara detectada, para hallar su apertura y averiguar cuando el conductor está bostezando.

El método para la obtención de la apertura de la boca va a ser el de procesamiento morfológico utilizado anteriormente en la apertura del ojo. Para la explicación de su funcionamiento se utilizará la imagen mostrada a continuación, la cual se ha pasado a escala de grises y reescalado.

Ilustración 62. Imagen usada para la detección de bostezos.

Tras pasar la imagen por el detector de rostro y segmentar el resultado, se obtiene el área correspondiente a la boca del sujeto, que se muestra en la siguiente figura.

Ilustración 63. Boca del sujeto.

4.7.1. Procesamiento morfológico

La función de procesamiento morfológico utilizada para obtener la apertura de la boca va a ser la misma que la que se ha explicado anteriormente para conocer la apertura

de los ojos. La única modificación va a ser en las dimensiones de los elementos estructurantes, ya que, el tamaño de la boca va a ser mayor que el de los ojos.

Para poder elegir los elementos estructurantes adecuadamente, se debe conocer aproximadamente el tamaño de la boca en píxeles.

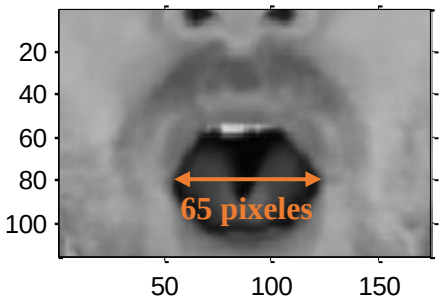


Ilustración 64. Imagen de la boca con escala en píxeles.

Para el caso de este usuario, se utilizarán unos elementos estructurantes de forma circular, con un tamaño de 40 píxeles para el primer cierre y 70 píxeles para el segundo cierre, quedando del siguiente modo.

img1=cv2.morphologyEx(img_suav, cv2.MORPH_CLOSE, kernel40)

img2=cv2.morphologyEx(img1, cv2.MORPH_CLOSE,kernel70)

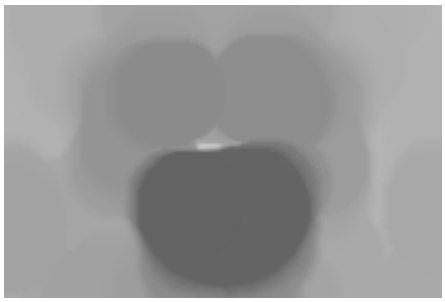


Ilustración 65. Imagen de la boca tras primer cierre (ee 40).

Ilustración 66. Imagen de la boca tras segundo cierre (ee70).

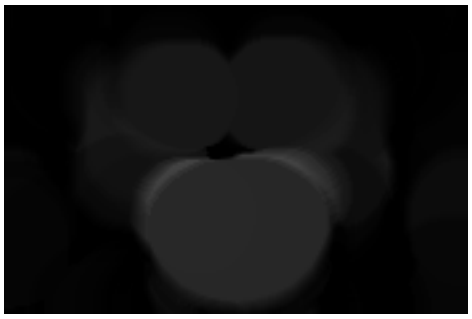


Ilustración 67. Resta del segundo cierre menos el primero.

Tras aplicar los dos cierres y realizar la diferencia entre ellos, se obtiene una imagen donde se encuentran los objetos eliminados en la operación del segundo cierre, donde el área de la boca abierta, al ser más oscura que el resto de la imagen, tiene un valor de luminosidad más alto.

Es el momento de realizar una umbralización, seleccionando un umbral en función del valor más alto de nivel de gris que se encuentra en la imagen (objeto más claro). En concreto, el umbral seleccionado va a ser el 50% del valor de luminosidad más alto encontrado.

```
umbral1=np.amax(img3)*0.5
ret,img_bw=cv2.threshold(img3, umbral1,255, cv2.THRESH_BINARY)
```

Ilustración 68. Imagen umbralizada.

Al igual que anteriormente se hizo en la obtención de la apertura del ojo, se va a realizar una dilatación de la imagen umbralizada con el objetivo de obtener el contorno de la boca.

```
img4=cv2.dilate(img_bw,np.ones(5))
img5=cv2.subtract(img4,img_bw)
```

Ilustración 69. Contorno de la boca abierta.

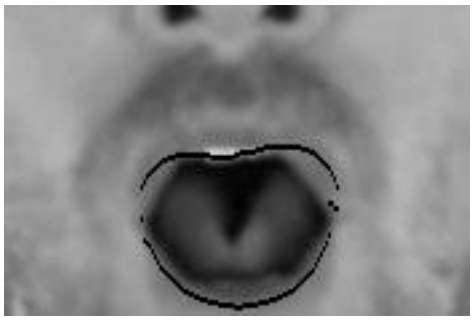


Ilustración 70. Contorno detectado sobre boca del sujeto.

Para calcular el valor de apertura de la boca, se va a utilizar el subsistema mostrado anteriormente basado en la extracción del centro de masa del contorno de la boca. De este modo se podrá trazar la línea vertical que pase por el centro y conocer así la apertura de la boca. El centro de masas de un objeto se calculaba como el promedio de cada uno de los puntos pertenecientes al mismo, obteniendo la coordenada (x,y) donde se encuentra el centro.

coord_y=np.sum(im_contorno[0])/(im_contorno[0].shape[0])

coord_x=np.sum(im_contorno[1])/(im_contorno[1].shape[0])

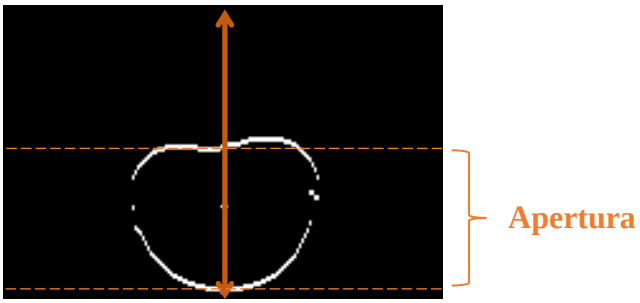


Ilustración 71. Obtención de la apertura de la boca.

El problema de esta medida que se obtiene como resultado es que está expresada en píxeles y dependiendo de la distancia entre el conductor y la cámara, para un mismo valor de apertura de la boca, se pueden obtener distancias en píxeles distintas. Es por esto que el valor obtenido se dividirá entre el número de píxeles que contiene la imagen de la boca que toma como parámetro de entrada, obteniendo así un porcentaje de apertura.

$$apertura\ boca\ (\%) = \frac{apertura\ vertical\ (píxeles)}{nº\ de\ píxeles\ de\ la\ imagen\ de\ entrada} * 100$$

Para la detección de bostezos se seleccionará un umbral de apertura de la boca de tal modo que, cuando la apertura sobrepase el valor umbral durante un cierto tiempo, se indicará la detección de un bostezo durante la conducción.

El motivo de seleccionar el método de procesamiento morfológico para la detección de bostezos es debido a que la apertura de la boca sigue una forma circular, por lo que utilizando elementos estructurantes circulares se puede obtener relativamente fácil el contorno de la boca abierta.

El otro método presentado anteriormente, basado en las proyecciones de las imágenes, se ayuda de la diferencia de niveles entre el objeto de interés y su entorno con el fin de detectar la cantidad de píxeles oscuros que hay en la imagen. Para el caso del ojo es muy efectivo, al tener un contraste muy grande entre el iris y la esclerótica, pero en el caso de los bostezos, no es tan abultada la diferencia de niveles entre el interior de la boca y la cara del sujeto, por lo que se ha descartado utilizar este procedimiento.

4.8. Sistema de detección de fatiga

Una vez extraídos del conductor los valores de apertura tanto de los ojos como de la boca, se establece el método para la detección del estado de fatiga. El conductor ya se encuentra monitorizado a tiempo real, por lo que ahora queda estimar un estado de fatiga a partir de los valores anteriormente calculados.

En primer lugar, se debe diferenciar cuando el usuario ha ejecutado un parpadeo, por lo que se ha de evaluar constantemente el valor de la apertura del ojo y atender a los mínimos encontrados. Para ello, se establecerán unos umbrales que decidirán si un valor pertenece a un parpadeo o no.

Los distintos valores de apertura del ojo que se obtendrán dependerán de cada persona, es por esto por lo que los umbrales establecidos no pueden ser estáticos, sino que se estimarán de forma dinámica adaptándose de acuerdo al comportamiento observado del usuario en concreto. Teniendo en cuenta esto, se va a calcular el valor medio de apertura del ojo constantemente para tomarlo como referencia, el cual se puede observar en la línea roja de la siguiente figura.

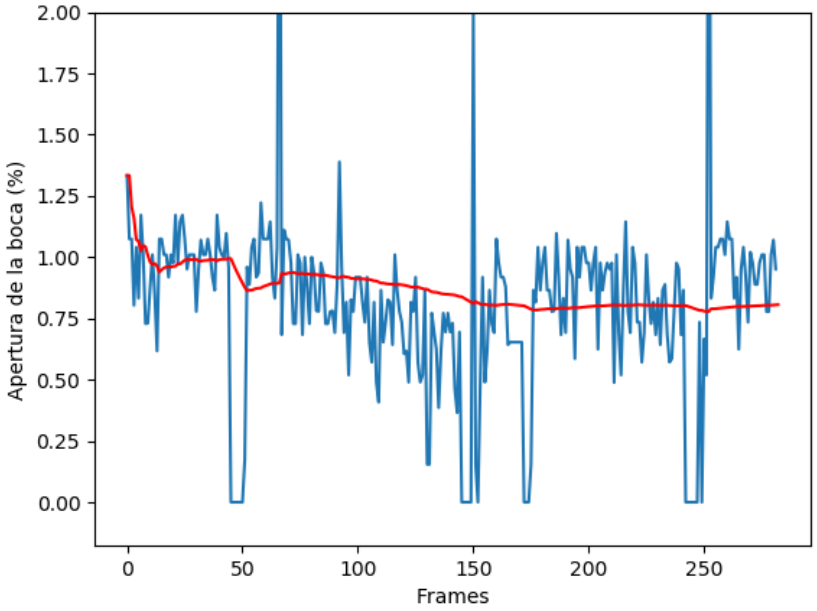


Ilustración 72. Evolución de la apertura del ojo junto con valor medio expresado en rojo.

Para evitar una sobrecarga de memoria se ha limitado el promedio del valor de apertura a una ventana de 1000 frames, de tal modo que solo se realizará la media de los mil últimos valores de apertura. Para efectuar esta operación, los valores de apertura se almacenan en una lista de Python modelada como una cola FIFO, que cuando llega a una dimensión de mil posiciones, comienza a descartar los datos más antiguos.

En la *Ilustración 72* se puede observar la existencia de ruido, esto es debido a que la extracción del valor de apertura del ojo no es perfecta, aun así, el valor medio de ruido es mucho menor que el rango total del ojo, por lo que los valores mínimos correspondientes a los parpadeos se pueden distinguir fácilmente.

Volviendo a la selección del umbral para la detección de parpadeos, se seguirá lo establecido por el procedimiento PERCLOS (PERcentage of eyelid CLOSure) , el cual estima un estado de fatiga en función del tiempo que el ojo pasa cerrado más de un 80%. En el caso que se presenta, al no conocer exactamente el valor máximo de apertura, se impondrá este umbral del modo en que, cuando el valor de apertura este un 80% por debajo del valor medio, se supondrá que el ojo está cerrado y existe un parpadeo.

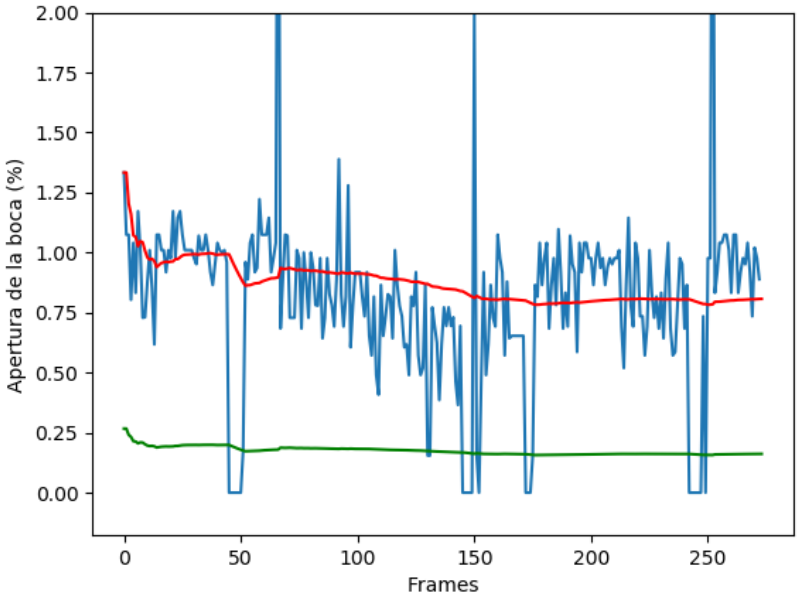


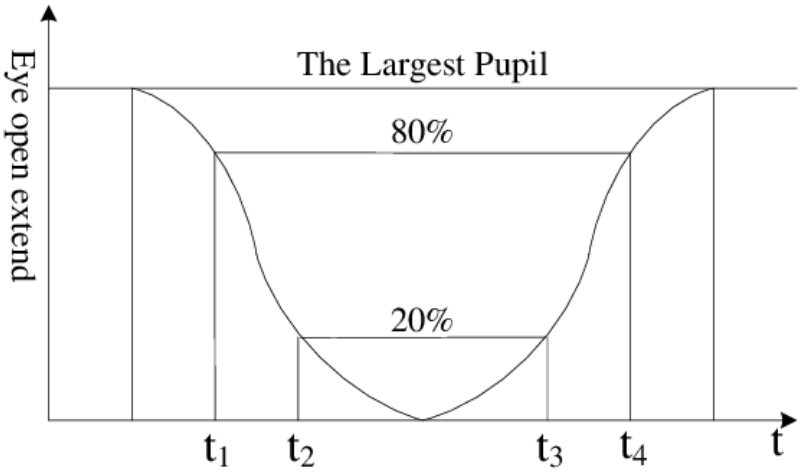
Ilustración 73. Evolución de la apertura del ojo con umbral de ojo cerrado en verde.

Estableciendo el umbral anterior se pueden identificar sin problema los mínimos de apertura evitando los falsos positivos que puedan aparecer. Una vez establecido el umbral de cierre, se pasa a obtener un porcentaje de PERCLOS. El PERCLOS, como su nombre indica, expresa el porcentaje de tiempo que el ojo pasa cerrado por debajo de un 80% respecto al periodo de parpadeo, y se calcula del siguiente modo:

$$PERCLOS\ (\%) = \frac{t_3 - t_2}{t_4 - t_1} * 100$$

Siendo:

- t1: instante de tiempo en el que la apertura del ojo es inferior al 80%
- t2: instante de tiempo en el que la apertura del ojo es inferior al 20%
- t3: instante de tiempo en el que la apertura del ojo es superior al 20%
- t4: instante de tiempo en el que la apertura del ojo es superior al 80%



Ya establecido el porcentaje de PERCLOS en cada instante, es el momento de introducir los bostezos en la ecuación. Estos influirán en el estado de fatiga cuando se detecten un elevado número de ellos en un breve periodo de tiempo. Para la detección de bostezos habrá que definir una norma observando las muestras obtenidas a continuación.

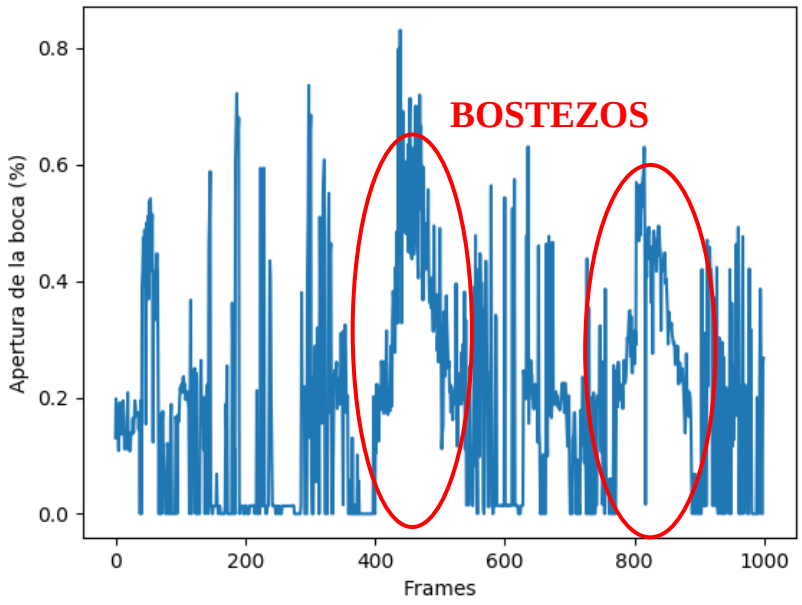


Ilustración 74. Evolución de la apertura de la boca.

La señal obtenida correspondiente a la apertura de la boca es muy ruidosa, tal y como se aprecia en la figura anterior, pero aun así es posible localizar los periodos donde se han producido bostezos atendiendo a que, durante un número consecutivo de frames, el valor de la apertura sobrepasa cierto umbral. Por ello, la condición impuesta para la detección de bostezos es que durante 60 frames consecutivos (2 segundos), la apertura obtenida en el 80% de los frames esté correlada un 90% y, además, sobrepasando un cierto umbral.

Para la elección de este umbral, no se puede atender al valor medio de la apertura de la boca, ya que, el mayor porcentaje de tiempo debería estar cerrada. Por ello, se decide tomar como valor de referencia las dimensiones en píxeles de la imagen que delimita la región de la boca (*Ilustración 63*), promediando ambas dimensiones y estableciendo el umbral en la mitad del valor obtenido.

Ya se tiene un porcentaje de PERCLOS y detección de bostezos a la salida del sistema. El último paso a realizar es estimar un nivel de fatiga del conductor en función de ambos parámetros, para ello se establecen 3 estados distintos: fatiga nula, indicios de fatiga, fatiga extrema.

- **Fatiga nula:** El porcentaje de PERCLOS se mantiene por debajo del 30%. No se da ninguna señal al conductor.
- **Indicios de fatiga:** El porcentaje de PERCLOS está entre el 30% y 60%. Aviso leve al conductor del estado en el que se presenta.
- **Fatiga extrema:** El porcentaje de PERCLOS se encuentra por encima del 60%. Aviso urgente y llamativo por pantalla para que el conductor despierte.

Ilustración 75. Distintos estados de fatiga.

Por último, referente a la detección de bostezos, se ha incluido que cuando el conductor del vehículo bostece 3 veces o más en un tramo temporal de 2 minutos, se aumente el estado de fatiga en un nivel, independientemente del porcentaje de PERCLOS en ese instante.

5. Resultados

En esta sección se presentan los resultados obtenidos con distintas imágenes de prueba en las que varía la apertura de los ojos y la boca, con la finalidad de comprobar el funcionamiento de los distintos métodos expuestos anteriormente.

5.1. Apertura de los ojos

Para probar el funcionamiento de los distintos subsistemas encargados de obtener el valor de apertura de los ojos, se usará el siguiente conjunto de imágenes.

Ilustración 76. Conjunto de imágenes de test.

5.1.1. Procesamiento morfológico

Centro de masa del contorno

Los resultados obtenidos con el método mostrado en la sección 4.5.1.1, basado en la obtención del centro de masa para el contorno del iris del conductor, y a partir de esto, poder calcular la apertura del ojo se muestran a continuación.

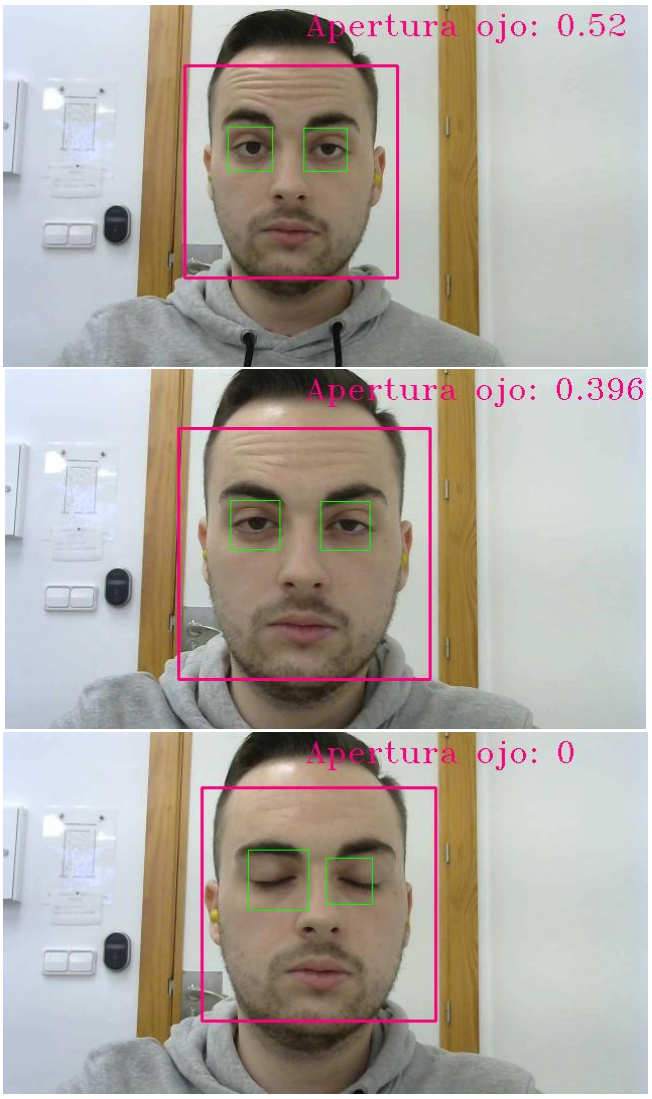


Ilustración 77. Testeo del método basado en el centro de masa del contorno de la pupila.

Los resultados obtenidos se consideran como válidos ya que el valor de apertura en la segunda imagen es menor que el de la primera, y en el caso de los ojos cerrados se detecta a la perfección. Recordar que el valor mostrado en las imágenes se expresa en porcentaje debido a la siguiente ecuación.

$$apertura\ ojo\ (\%) = \frac{apertura\ vertical\ (píxeles)}{nº\ de\ píxeles\ de\ la\ imagen\ de\ entrada} * 100$$

Aproximación por elipses

Los resultados obtenidos con el método de aproximación del iris a una elipse, expuesto en la sección 4.5.1.2, son los mostrados a continuación:

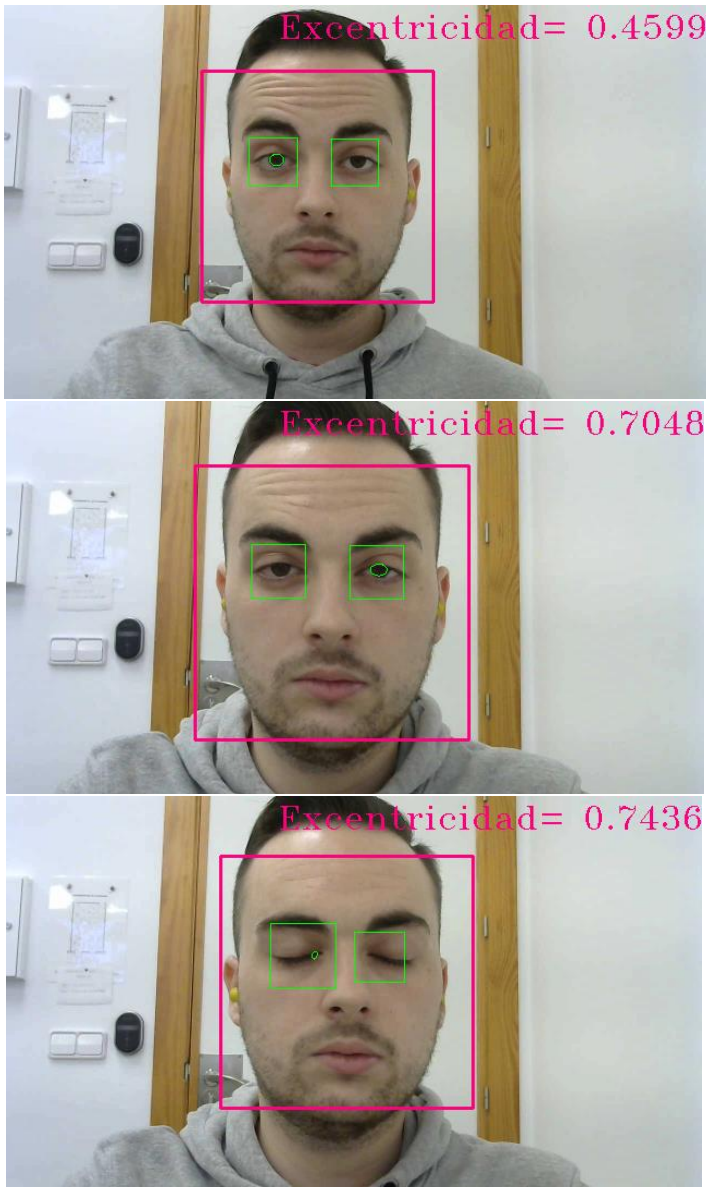


Ilustración 78. Testeo del método basado en la aproximación de elipses al iris.

Observando los resultados se llega a la conclusión de que el método que se presenta es muy débil frente a irregularidades. En el caso de la imagen con los ojos cerrados, ha detectado el lagrimal como un contorno y lo a aproximado a una elipse, algo que no pasa con el método anterior.

En cuanto a las imágenes con los ojos abiertos las ha clasificado de una manera correcta, devolviendo una excentricidad menor para aquella en la que los ojos están completamente abiertos. Aun así, para que este método funcione correctamente necesita de un descarte de contornos en función de las coordenadas, de tal modo que, cuando un contorno esté alejado del centro del ojo, no lo aproxime a una elipse. Se deja el desarrollo para líneas futuras.

5.1.2. Proyección binaria

Los resultados obtenidos con el método mostrado en la sección 4.5.2, basado en la proyección de la imagen binaria del ojo, se muestran a continuación.

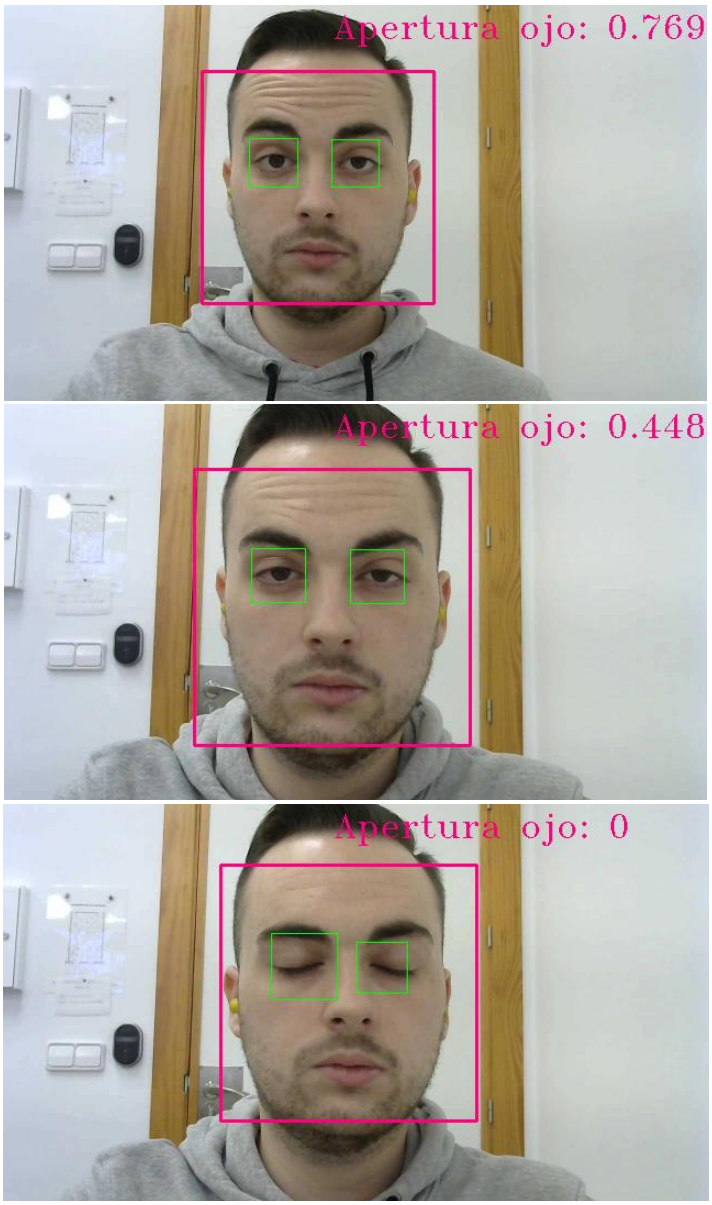


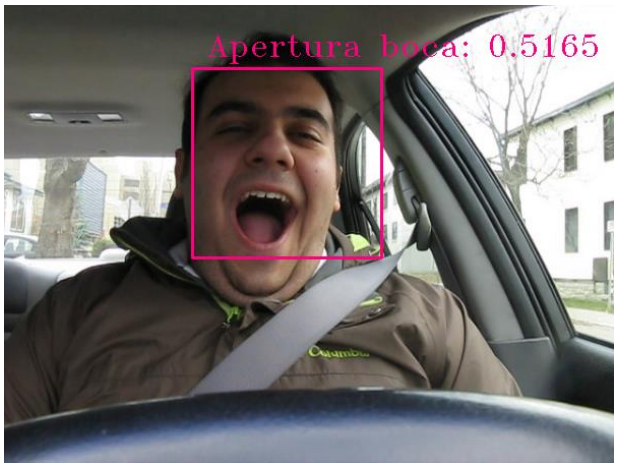
Ilustración 79. Testeo del método basado en las proyecciones de la imagen binaria del ojo.

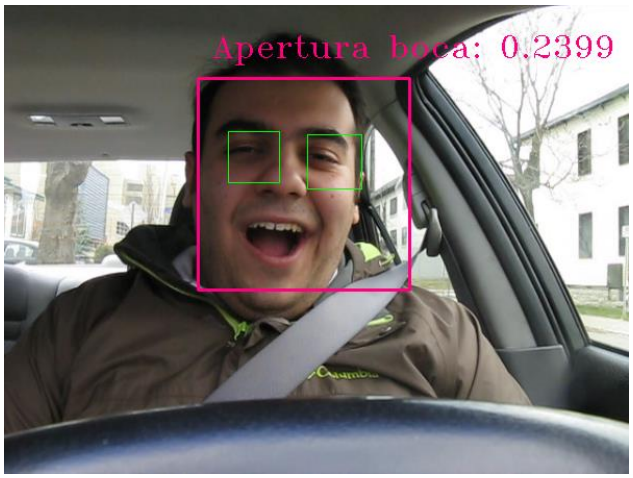
Los resultados obtenidos se consideran como válidos ya que el valor de apertura en la segunda imagen es menor que el de la primera, y en el caso de los ojos cerrados se detecta de manera correcta. Recordar que el valor mostrado en las imágenes se expresa en porcentaje de apertura respecto a la anchura del ojo debido a la siguiente ecuación.

$$\%\ apertura = \frac{apertura\ en\ píxeles}{anchura\ en\ píxeles}$$

5.2. Apertura de la boca

Para la detección de bostezos se ha escogido el método de procesamiento morfológico basado en la extracción del centro de masa del contorno de la boca con el fin de obtener la apertura de la misma. El conjunto de imágenes seleccionado para su testeo es el siguiente:





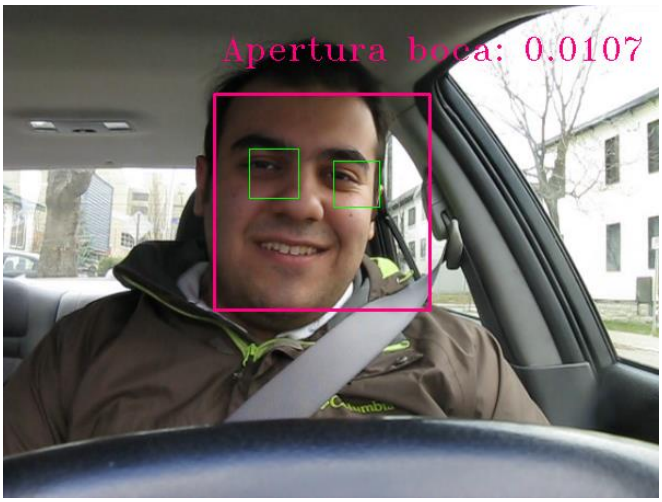


Ilustración 80. Conjunto de imágenes de testeo para la apertura de la boca.

Observando los resultados se aprecia que el método desarrollado funciona como era de esperar, ofreciendo un valor cada vez menor para una apertura de boca que decrece en cada imagen. Recordar, al igual que en apartados anteriores, que las unidades que se muestran en las imágenes anteriores son porcentajes de apertura, según la siguiente ecuación.

$$apertura\ boca\ (\%) = \frac{apertura\ vertical\ (píxeles)}{nº\ de\ píxeles\ de\ la\ imagen\ de\ entrada} * 100$$

5.3. Evaluación de los resultados

Atendiendo a los resultados anteriores, en los que se ha comprobado el funcionamiento de todos los sistemas desarrollados para obtener el estado en el que se encuentra el ojo y la boca, hay que elegir el método que se utilizará en el sistema final de detección de fatiga.

Para el subsistema del estado del ojo, tanto la proyección binaria como el método de procesamiento morfológico basado en la extracción del centro de masa han dado unos resultados concluyentes.

Si se atiende al tiempo de computación de cada uno, el que mejor resultados da es el de proyección binaria, aun así, se ha optado por escoger el segundo método de procesamiento debido a que es más robusto frente a irregularidades. Además, es una técnica que aún no se ha usado en ningún sistema de detección de fatiga anterior y que ha sido desarrollado exclusivamente por el autor de este proyecto, por lo que se desea comprobar el funcionamiento del sistema y poder mejorarlo en líneas futuras.

En cuanto a la detección del estado de la boca, también se utilizará el método de procesamiento morfológico basado en la extracción del centro de masa del contorno del iris, ya que ha sido el único desarrollado para esta función.

6. Conclusiones

En el presente proyecto se ha implementado un Sistema Avanzado de Asistencia a la Conducción (ADAS) basado en la detección de fatiga en conductores. Se ha dividido en dos etapas principales, por un lado, la detección de cara y ojos, y por otro, la extracción de los valores correspondientes a la apertura de los ojos y la boca.

Respecto a la primera etapa de detección, los resultados han sido muy favorables gracias al uso de librerías previamente entrenadas con gran cantidad de imágenes, tanto positivas como negativas. No se ha realizado un testeo de los métodos utilizados para la detección, pero viendo el funcionamiento del sistema final y otros proyectos en los que se han manejado estos detectores, se puede asegurar que su funcionamiento es muy robusto. Esto ha constituido la base del sistema, ya que para obtener una buena precisión del estado de fatiga es primordial adquirir una localización de la cara y ojos con un mínimo de fiabilidad.

La segunda parte del sistema, correspondiente al estado de los ojos y la boca, requiere un sistema de extracción del contorno del ojo lo más robusto posible, por lo que se ha desarrollado un buen preprocesado de la imagen para evitar las iluminaciones no uniformes y los factores de ruido no deseados. Además, se ha implementado un algoritmo completamente nuevo basado en el procesamiento morfológico de la imagen para la extracción de contornos, así como para la obtención de la apertura del ojo y la boca. En un sistema de detección en tiempo real es vital el periodo de procesamiento total desde que entra una imagen hasta que se devuelve el estado de fatiga, por esto se ha apostado por técnicas de procesamiento morfológico cuyo tiempo de procesamiento es mínimo.

En cuanto al sistema final de detección de fatiga, se puede decir que se han cumplido los objetivos marcados en este proyecto, ofreciendo a la salida el estado de fatiga en el que se encuentra el conductor. Aun así, queda como tarea pendiente entrenar con situaciones reales este detector y poder así ajustar los umbrales que marcan el funcionamiento del sistema.

Durante el desarrollo y evaluación del sistema implementado se han detectado algunas limitaciones, el trabajo futuro estará dirigido a solucionar estas limitaciones de modo que el detector tenga un funcionamiento más robusto.

Se plantea para el curso que viene, durante el desarrollo de mi Trabajo Fin de Máster, una mejora del sistema actual con la fusión de sensores para medidas fisiológicas, como el pulso cardiaco, teniendo así una monitorización completa del conductor. Esto dará al sistema actual una gran mejora en la estimación del estado de fatiga, ya que basará su resultado en función de distintos sensores y parámetros extraídos por visión artificial.

6.1. Valoración personal

La valoración que hago de la realización de este trabajo es muy positiva, ya que me ha permitido adquirir experiencia en el desarrollo de un proyecto en solitario de unas

dimensiones mucho mayores que todo lo desarrollado anteriormente durante el grado. Además, al tratarse de un proyecto cuyo objetivo es la reducción de los accidentes de tráfico e, indirectamente, la disminución de muertes al volante, me ha resultado una gran experiencia poder desarrollar algo que puede ayudar a la población.

También, he podido aprender Python de una manera divertida y dinámica con el procesamiento de imágenes, algo a lo que le doy mucha importancia debido a la gran acogida que está teniendo este lenguaje de programación en el ámbito laboral, y que no se ejerce en el grado.

Por último, me ha encantado introducirme en el ámbito de la Visión Artificial y el Machine Learning, algo que no se ejerce en el grado y que quería tener como competencia personal. Son dos aspectos que están al orden del día y que su evolución en los años prósperos va a producir, según mi opinión, una revolución en el mundo de las TIC.

Bibliografía

[1] «Los accidentes de tráfico, principal causa de muerte en jóvenes». [En línea]. Disponible en: http://revista.dgt.es/es/noticias/internacional/2018/1218oms-informe-mundial-accidentes-trafico.shtml#.XGwbWbiCHIU. [Accedido: 19-feb-2019].

[2] autosportmoraleja, «Fatiga, somnolencia y sueño: cómo afectan a la conducción.», *Auto Sport Moraleja*, 25-oct-2017. .

[3] «Cómo evitar el sueño al conducir de noche», *Autocasión*, 23-may-2017. .

[4] «Salud IDEAL - ¿Qué son los microsueños y cómo prevenirlos?» [En línea]. Disponible en: https://salud.ideal.es/lomonaco/2304-que-son-los-micro-suenos.html. [Accedido: 19-feb-2019].

[5] «¿Te has quedado dormido alguna vez al volante? Te contamos qué hacer para evitarlo | www.eurotaller.com». [En línea]. Disponible en: https://www.eurotaller.com/noticia/te-has-quedado-dormido-alguna-vez-al-volante-te-contamos-que-hacer-para-evitarlo#. [Accedido: 19-feb-2019].

[6] RACE, «La Fatiga en la Conducción: Consejos y Recomendaciones», *RACE*, 26-dic-2017. .

[7] Fundación Linea Directa, «Influencia de la somnolencia en los accidentes de tráfico en España (2011-2015)».

[8] «Cuestiones de seguridad vial, conducción eficiente, medio ambiente y contaminación (Edición 2015)», p. 440.

[9] R. U. F.´andez A. F.´andezVill´an y Ruben Casado Tejedor, «Sistema Automático Para la Detección de Distracción y Somnolencia en Conductores por Medio de Características Visuales Robustas».

[10] Y. Furugori, S N. y M. Iname, C Y., «Estimation of driver fatigue by pressure distribution on seat in long term driving. 26. 053-058.», 2005.

[11] M. K. Farid MN y A. E. E. Heiner Bubb, «Methods to develop a driver observation system used in an active safety system», 2006.

[12] K. Torkkola, N. Massey, y C. Wood, «Driver inattention detection through intelligent analysis of readily available sensors. 326 - 331. 10.1109/ITSC.2004.1398919.», 2004.

[13] T. Ersal, H. J.A.Fuller, y O. Tsimhoni, «Model-basedanalysis and classification of driver distraction under secondary tasks.», 2010.

[14] F. J, «Detection of different levels of vigilance by eeg pseudo spectra.Neural Network World», 2004.

[15] «PERCLOS: A VALID PSYCHOPHYSIOLOGICAL MEASURE OF ALERTNESS AS ASSESSED BY PSYCHOMOTOR VIGILANCE», *Tech Brief*, oct. 1998.

[16] M. Cazorla, «Robótica y visión artificial». Grupo de Visión Robótica/Universidad Politécnica de Alicante.

[17] J. F. Velez Serrano, A. B. Moreno iDiaz, y A. Sanchez Calle, *Visión por computador*. .

[18] «Curso de Procesado de Imagen (c)GPI». [En línea]. Disponible en: https://www.uv.es/gpoei/eng/Pfc_web/generalidades/grises/grey.htm. [Accedido: 18-abr-2019].

[19] P. Brodartz, *Textures: A Photographic Album for Artists and Designers*. New York: Dover Publications, 1966.

[20] W. L. López Romero, «Sistema de control del estado de somnolencia en conductores de vehículos», *Univ. Téc. Ambato*, 2016.
[21] E. Mayon y M. Raúl, «SISTEMA DE DETECCIÓN DE SOMNOLENCIA MEDIANTE INTELIGENCIA ARTIFICIAL EN CONDUCTORES DE VEHÍCULOS PARA ALERTAR LA OCURRENCIA DE ACCIDENTES DE TRANSITO», 2018.
[22] «OpenCV», *Wikipedia, la enciclopedia libre*. 25-abr-2019.
[23] N. L. Fernández García, «Introducción a la Visión Artificial», *Dep. Indormática Análisis Numér. Univ. Córdoba*.
[24] P. Santamaría, «OpenCL, la alternativa libre a CUDA y el futuro de la computación GPGPU», *Applesfera*, 24-nov-2010. [En línea]. Disponible en: https://www.applesfera.com/aplicaciones-os-x-1/opencl-la-alternativa-libre-a-cuda-y-el-futuro-de-la-computacion-gpgpu. [Accedido: 20-abr-2019].
[25] C. Platero, «Procesamiento Morfológico», *Apunt. Visión Artif. UPM*.

ANEXO I: Instalación

En esta sección se indican los pasos a seguir para la instalación del software necesario para la elaboración de este proyecto. Como ya se ha indicado anteriormente, el presente trabajo se ha realizado bajo el entorno de Python y OpenCV.

Ilustración 81. Software utilizado.

En primer lugar, se procede con la instalación de Python, en su página web oficial se puede encontrar el instalador tanto de la última versión como de todas las releases hasta la fecha (https://www.python.org/downloads/). Para la ejecución de este proyecto se recomienda la instalación de Python3, pero también es compatible con versiones anteriores como Python 2.7. Se recomienda que durante la instalación se marque la casilla "Add Python X.XX to Path" para incluirlo en las variables del entorno y poder ejecutarlo desde la consola de Windows.

Una vez tenga Python en su sistema, se procede a la instalación de la librería de libre distribución OpenCV, concretamente la versión 4.0.0. Esta se instala a través de la consola de Windows, siempre y cuando haya añadido Python a las variables del entorno, a través del siguiente comando: `pip install opencv-python`

Si la versión de Python que hay en su sistema es inferior a la 3.0, deberá instalar OpenCV de manera manual, ya que la herramienta 'pip' para la instalación de paquetes se incuye a partir de Python3. Para la instalación manual debe descargar OpenCV de aquí y copiar el archivo **'cv2.pyd'** en la ruta de instalación de Python: '*C/PythonX/libs/site-packages'*.

En cuanto al software desarrollado en este proyecto, puede encontrarlo en el siguiente repositorio de github: https://github.com/pedrolo22/DetectorFatiga

Las librerías necesarias para su ejecución son:

- Numpy
- Time
- Math
- Matplotlib

ANEXO II: Código fuente

I. Documento con todas las funciones

```
import cv2
import numpy as np
import time as t
import math
from matplotlib import pyplot as plt

kernel1=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(1,1))
kernel2=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(2,2))
kernel3=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
kernel4=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(4,4))
kernel7=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7))
kernel8=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,8))
kernel9=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(9,9))
kernel10=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10,10))
kernel11=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))
kernel12=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(12,12))
kernel13=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(13,13))
kernel14=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(14,14))
kernel15=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(15,15))
kernel16=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(16,16))
kernel17=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(17,17))
kernel31=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(31,31))
kernel32=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(32,32))
kernel33=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(33,33))
kernel34=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(34,34))
kernel35=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(35,35))
kernel36=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(36,36))
kernel37=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(37,37))
kernel38=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(38,38))
kernel39=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(39,39))
kernel40=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(40,40))
kernel50=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(50,50))
kernel61=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(61,61))
kernel62=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(62,62))
kernel63=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(63,63))
kernel64=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(64,64))
kernel65=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(65,65))
kernel66=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(66,66))
kernel67=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(67,67))
kernel68=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(68,68))
kernel69=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(69,69))
kernel70=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(70,60))
```

```
PREDICTOR_PATH="shape_predictor_68_face_landmarks.dat"
face_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_frontalface_alt2.xml')
face_classifier_LBP = cv2.CascadeClassifier('Haarcascades/lbpcascade_frontalface.xml')
eye_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade_eye.xml')
mouth_classifier = cv2.CascadeClassifier('Haarcascades/mouth.xml')
cv2.ocl.setUseOpenCL(True)

'''Funcion que localiza la cara del ususario con OpenCV (haarcascade_frontalface_default,
ademas dibuja un rectangulo en el area de la cara, si hay mas de una persona lo indica'''
def detect_careto_OpenCV(imagen):
    x,y,w,h = [0,0,0,0]
    detect=False;
    texto="Mas de una cara detectada";
    #img=cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY);
    img=imagen
    faces = face_classifier.detectMultiScale(img, scaleFactor=1.1, minNeighbors=3)
    if len(faces) > 1:
        cv2.putText(imagen, texto, (100,100), cv2.FONT_HERSHEY_TRIPLEX, 1, (127,0,255), 2)
        detect=False
        return imagen,x,y,w,h,detect

    if faces is ():
        return imagen,x,y,w,h,detect

    detect=True
    x,y,w,h=faces[0,:]
    #cv2.rectangle(imagen,(x,y) , (x+w,y+h), (127,0,255), 3)
    return imagen,x,y,w,h,detect

def detect_eyes_OpenCV(imagen):

    graylevel=sum(map(sum, imagen))/(imagen.shape[0]*imagen.shape[1])
    if(graylevel<60):
        imagen=cv2.equalizeHist(imagen)

    eyes = eye_classifier.detectMultiScale(imagen, scaleFactor=1.1, minNeighbors=3)
    detect_eyes=True
    eye1=0
    eye2=0
    if len(eyes) < 2:
        detect_eyes=False
        return imagen,eye1,eye2,detect_eyes
    eye1=eyes[0,:]
    eye2=eyes[1,:]
    ex1,ey1,ew1,eh1=eye1
    ex2,ey2,ew2,eh2=eye2
    #cv2.rectangle(imagen,(ex1,ey1),(ex1+ew1,ey1+eh1),(0,255,0),1)
    #cv2.rectangle(imagen,(ex2,ey2),(ex2+ew2,ey2+eh2),(0,255,0),1)
```

```
    return imagen,eye1,eye2,detect_eyes

def detect_mouth_OpenCV(imagen):

    mouth=mouth_classifier.detectMultiScale(imagen, scaleFactor=1.1,minNeighbors=3)
    imagen=cv2.cvtColor(imagen,cv2.COLOR_GRAY2BGR)
    print('BOCA:', mouth)
    detect_mouth=True
    if(len(mouth) == 0):
        detect_mouth=False
    else:
        mx,my,mw,mh=mouth[0]
        cv2.rectangle(imagen,(mx,my),(mx+mh,my+mw),(0,0,255),2)
        cv2.imwrite('capturas/mouth_rectangle.jpg',imagen)
    return imagen,mouth,detect_mouth


def proy_bin(imagen):
    img_suav=cv2.GaussianBlur(imagen,(3,3),0)
    umbral2=50
    ret,img_umbr=cv2.threshold(img_suav, umbral2,255, cv2.THRESH_BINARY)
    proy_ver=np.sum(255-img_umbr,0)
    proy_ver_norm=proy_ver.astype(float)/float(np.max(proy_ver))
    proy_hor=np.sum(255-img_umbr,1)
    proy_hor_norm=proy_hor.astype(float)/float(np.max(proy_hor))
    umbral_ver=0.4
    umbral_hor=0.4
    index_ver=sum(proy_ver_norm>umbral_ver)
    index_hor=sum(proy_hor_norm>umbral_hor)
    if(index_hor!=0 and index_ver!=0):
        apertura=float(index_hor)/float(index_ver)
    else:
        apertura=0

    #print(index_ver)
    #print(index_hor)
    #print(apertura)

    #cv2.imwrite('./capturas/umbr_proy.jpg',img_umbr)
    #plt.subplot(121)
    #plt.imshow(img_umbr,cmap='gray')
    #plt.title('Ojo')
    #plt.subplot(122)
    #plt.plot(proy_ver_norm)
    #plt.title('')
    #plt.subplot(133)
    #plt.imshow(img_suav,cmap='gray')
    #plt.title('ojo')
```

```
    #plt.show()
    return apertura

def morf_proc(imagen,type_aprox):

    if(imagen.shape[0]==0 or imagen.shape[1]==0):
        return 0
    else:

        img_rgb=cv2.cvtColor(imagen, cv2.COLOR_GRAY2BGR);
        img_suav=cv2.GaussianBlur(imagen,(3,3),0)
        img1=cv2.morphologyEx(img_suav, cv2.MORPH_CLOSE, kernel9)
        img2=cv2.morphologyEx(img1, cv2.MORPH_CLOSE,kernel15)
        img3=cv2.subtract(img2,img1)
        umbral1=np.amax(img3)*0.8
        ret,img_bw=cv2.threshold(img3, umbral1,255, cv2.THRESH_BINARY)
        img4=cv2.dilate(img_bw,kernel3)
        img5=cv2.subtract(img4,img_bw) #imagen con el contorno umbralizado

        if(type_aprox==0):
            #Proceso para aproximar el iris a una elipse
            contours, hierarchy = cv2.findContours(img_bw, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
            cv2.drawContours(img_rgb, contours, -1, (0,255,0), 1)
            ellipse=cv2.fitEllipse(contours[0])
            return ellipse

        else:

            #Proceso para encontrar el centro de masa del contorno de la pupila
            im_contorno=np.where(img5==255)
            coord_y=int(np.sum(im_contorno[0])/(im_contorno[0].shape[0]))
            coord_x=int(np.sum(im_contorno[1])/(im_contorno[1].shape[0]))
            distancia=np.where(im_contorno[1]==coord_x) #Devuelve indices donde se
cumple la condicion
            xcol=im_contorno[0]
            aux=[]
            for i in distancia[0]:
              aux.append(xcol[i])
            if(len(aux) != 0):
                apertura=(max(aux)-min(aux))*100/(imagen.shape[0]*imagen.shape[1])
#Valor de apertura vertical del ojo en funcion de la pupila
            else:
                apertura=0


            # print('Apertura Ojo',apertura)
            cv2.drawMarker(img5, (int(coord_x), int(coord_y)), (255,255,255),
```

```
cv2.MARKER_CROSS,  markerSize = 2)

            # cv2.imwrite('./capturas/contorno_pupila_1.png',img5)
            # cv2.imwrite('./capturas/morf_proc/elipse.png',img_rgb)
            # cv2.imwrite('./capturas/morf_proc/close1.png',img1)
            # cv2.imwrite('./capturas/morf_proc/close2.png',img2)
            # cv2.imwrite('./capturas/morf_proc/closeresta.png',img3)
            # cv2.imwrite('./capturas/morf_proc/umbralizada.png',img_bw)
            # cv2.imwrite('./capturas/morf_proc/contorno_pupila.png',img5)
            # cv2.imwrite('./capturas/morf_proc/ojo_suav.jpg',img_suav)
            # cv2.imwrite('./capturas/morf_proc/dilatacion.jpg',img4)
            # cv2.imshow('Apertura1',img1)
            # cv2.imshow('Apertura2',img2)
            # cv2.imshow('1-2',img3)
            # cv2.imshow('Umbralizacion',img_bw)

            # cv2.imshow('Dilatacion',img4)
            # cv2.imshow('Contorno Pupila', img5)

            return apertura

def morf_proc_mouth(imagen):

    #imagen=cv2.equalizeHist(imagen)
    img_suav=cv2.GaussianBlur(imagen,(3,3),0)
    img1=cv2.morphologyEx(img_suav, cv2.MORPH_CLOSE, kernel11)
    img2=cv2.morphologyEx(img1, cv2.MORPH_CLOSE,kernel31)
    img3=cv2.subtract(img2,img1)
    umbral1=np.amax(img3)*0.5
    ret,img_bw=cv2.threshold(img3, umbral1,255, cv2.THRESH_BINARY)
    img4=cv2.dilate(img_bw,np.ones(5))
    img5=cv2.subtract(img4,img_bw) #imagen con el contorno umbralizado
    im_contorno=np.where(img5==255)
    coord_y=int(np.sum(im_contorno[0])/(im_contorno[0].shape[0]))
    coord_x=int(np.sum(im_contorno[1])/(im_contorno[1].shape[0]))
    distancia=np.where(im_contorno[1]==coord_x)
    xcol=im_contorno[0]
    aux=[]
    for i in distancia[0]:
       aux.append(xcol[i])
    if(len(aux) != 0):
        apertura=(max(aux)-min(aux))*100/(imagen.shape[0]*imagen.shape[1]) #Valor de
apertura vertical de la boca
    else:
        apertura=0
```

```
        # print('Apertura Boca',apertura)
        # cv2.drawMarker(img5, (int(coord_x), int(coord_y)), (255,255,255),
cv2.MARKER_CROSS,  markerSize = 2)

        # cv2.imshow('Apertura1',img1)
        # cv2.imshow('Apertura2',img2)
        # cv2.imshow('1-2',img3)
        # cv2.imshow('Umbralizacion',img_bw)
        # cv2.imwrite('./capturas/close1.png',img1)
        # cv2.imwrite('./capturas/close2.png',img2)
        # cv2.imwrite('./capturas/closeresta.png',img3)
        # cv2.imwrite('./capturas/umbralizada.png',img_bw)
        # cv2.imwrite('./capturas/contorno_pupila.png',img5)
        # cv2.imwrite('./capturas/ojo_suav.jpg',img_suav)
        # cv2.imwrite('./capturas/dilatacion.jpg',img4)
        # cv2.imshow('Dilatacion',img4)
        # cv2.imshow('Contorno Pupila', img5)

        return apertura
```

II. Código para procesamiento de imágenes

```
import cv2
import numpy as np
import time as t
import math
from matplotlib import pyplot as plt
import functions as fun
#import dlib

start_time=t.time()
im_rgb=cv2.imread('images/bostezo_medio.png')
im=cv2.cvtColor(im_rgb,cv2.COLOR_RGB2GRAY)

y,x=im.shape
if ((float(x)/float(y))==(float(4)/float(3))):
    im_rgb_resize=cv2.resize(im_rgb,(600,450))
    im_resize=cv2.UMat(cv2.resize(im,(600,450)))
if((float(x)/float(y))==(float(16)/float(9))):
    im_rgb_resize=cv2.resize(im_rgb,(640,360))
    im_resize=cv2.UMat(cv2.resize(im,(640,360)))
else:
    im_rgb_resize=im_rgb_resize
    im_resize=cv2.UMat(im)


face_UMat,fx,fy,fw,fh,fdetect=fun.detect_careto_OpenCV(im_resize)
face=cv2.UMat.get(face_UMat)

im_resize_gray_rgb=cv2.cvtColor(im_resize,cv2.COLOR_GRAY2RGB)
cv2.rectangle(im_rgb_resize,(fx,fy) , (fx+fw,fy+fh), (127,0,255), 2)
#cv2.imshow('pedrolo',im_resize)
face_time=t.time()-start_time
print('Tiempo ejecucion cara',face_time)

ROI_face=face[fy:fy+fh , fx:fx+fw]
#cv2.imshow('face',ROI_face)
dimY,dimX=ROI_face.shape
print('Tamano cara',dimY,dimX)

ROI_eyes=ROI_face[int(dimX*0.25):int(dimX*0.55),int(dimY*0.1):int(dimY*0.9)]
ROI_mouth=ROI_face[int(dimX*0.6):int(dimX*1),int(dimY*0.2):int(dimY*0.8)]

# ROI_eyes=ROI_face[int(dimX*0.25):int(dimX*0.55),int(dimY*0.1):int(dimY*0.9)]
# ROI_mouth=ROI_face[int(dimX*0.7):int(dimX*1),int(dimY*0.2):int(dimY*0.8)]

cv2.imshow('Cara',ROI_face)
```

```
cv2.imshow('Ojos',ROI_eyes)
cv2.imshow('Boca',ROI_mouth)


print('Tamano boca',ROI_mouth.shape)
print('Tiempo ejecucion crops',t.time()-start_time)

#Procesamiento para conocer el estado de los ojos

eyes,eye1,eye2,detect_eyes=fun.detect_eyes_OpenCV(ROI_eyes)
if detect_eyes==1:

    ex1,ey1,ew1,eh1=eye1
    ex2,ey2,ew2,eh2=eye2
    expand_eyes=3
    ROI_eye1=eyes[(ey1-expand_eyes):(ey1+eh1+expand_eyes) ,(ex1-
expand_eyes):(ex1+ew1+expand_eyes)]
    ROI_eye2=eyes[(ey2-expand_eyes):(ey2+eh2+expand_eyes) ,(ex2-
expand_eyes):(ex2+ew2+expand_eyes)]

    img_rgb_eyes=cv2.cvtColor(eyes, cv2.COLOR_GRAY2RGB)
    ex1=ex1+fx+int(dimY*0.1)
    ey1=ey1+fy+int(dimX*0.25)
    ex2=ex2+fx+int(dimY*0.1)
    ey2=ey2+fy+int(dimX*0.25)
    cv2.rectangle(im_rgb_resize,(ex1,ey1),(ex1+ew1,ey1+eh1),(0,255,0),1)
    cv2.rectangle(im_rgb_resize,(ex2,ey2),(ex2+ew2,ey2+eh2),(0,255,0),1)



    type_aprox=1 # 0 para aproximacion a elipses 1 para centro de masa


    if(type_aprox==1):

        #Calculo de apertura por centro de masa del contorno
        #apertura=fun.morf_proc(ROI_eye2,type_aprox)
        apertura=fun.proy_bin(ROI_eye2)
        texto=('Apertura ojo: ' + str(apertura))
        #cv2.putText(im_rgb_resize, texto, (300,30), cv2.FONT_HERSHEY_TRIPLEX, 1,
(127,0,255), 1)
    else:
        # Procesamiento morfologico y aproximacion de elipse a pupila

        ellipse=fun.morf_proc(ROI_eye1,type_aprox)
        cv2.ellipse(ROI_eye2,tuple(ellipse),(255,255,255),1)
        center,axis,angle=ellipse
        a=float(axis[1])
```

```
            b=float(axis[0])
            excentricidad=math.sqrt(a**2-b**2)/a
            center=(center[0]+ex2-expand_eyes,center[1]+ey2-expand_eyes)
            ellipse=[center,axis,angle]
            cv2.ellipse(im_rgb_resize,tuple(ellipse),(0,255,0),1)
            cv2.putText(im_rgb_resize, 'Excentricidad= '+ str(excentricidad), (250,30),
cv2.FONT_HERSHEY_TRIPLEX, 1, (127,0,255), 1)

        # print('Semieje menor',axis[0])
        # print('Semieje mayor',axis[1])
        # print('Excentricidad', excentricidad)
        #
        cv2.imwrite('./capturas/eyes_eqal.jpg',img_rgb_eyes)
        cv2.imwrite('./capturas/eye1.jpg',ROI_eye1)
        cv2.imwrite('./capturas/eye2.jpg',ROI_eye2)



        print('Tamano ojo',ROI_eye1.shape)
        print('Tiempo ejecucion ojos',t.time()-start_time)

else:
        print('No se ha detectado ningun ojo')


#Procesamiento para conocer el estado de la boca
cv2.imshow('mouth', ROI_mouth)
print(ROI_mouth.shape)
im_mouth,mouth,detect_mouth=fun.detect_mouth_OpenCV(im)
cv2.imshow('mouth rectangle', im_mouth)
apertura_boca=round(fun.morf_proc_mouth(ROI_mouth),4)
texto2=('Apertura boca: '+str(apertura_boca))
cv2.putText(im_rgb_resize, texto2, (200,50), cv2.FONT_HERSHEY_TRIPLEX, 1, (127,0,255), 1)

cv2.imwrite('./capturas/im_orig.jpg', im_rgb)
cv2.imwrite('./capturas/im_byw.jpg', im)
cv2.imwrite('./capturas/im_byw_face.jpg', im_resize_gray_rgb)
cv2.imwrite('./capturas/im_resize.jpg',face)
cv2.imwrite('./capturas/face.jpg',ROI_face)
cv2.imwrite('./capturas/mouth.jpg',ROI_mouth)
cv2.imwrite('./capturas/eyes.jpg',ROI_eyes)
cv2.imwrite('./capturas/elipse_rgb.png',im_rgb_resize)



cv2.waitKey(0)
```

III. Código para procesamiento de videos

```
import cv2
import numpy as np
import time as t
import math
from matplotlib import pyplot as plt
from collections import deque
import functions as fun


start_time=t.time()
cap = cv2.VideoCapture('dataset/9-MaleNoGlasses.avi')
glob_aper=deque([])
glob_aper_med=deque([])
glob_aper_umb=[]
frame_num=-1

while(cap.isOpened()):
    ret, frame = cap.read()
    frame_num=frame_num+1
    cv2.imwrite('dataset/T001/captura.png',frame)
    im_rgb=frame
    im=cv2.cvtColor(frame,cv2.COLOR_RGB2GRAY)
    y,x=im.shape
    if ((float(x)/float(y))==(float(4)/float(3))):
        im_rgb_resize=cv2.resize(im_rgb,(640,480))
        im_resize=cv2.UMat(cv2.resize(im,(640,480)))
    if((float(x)/float(y))==(float(16)/float(9))):
        im_rgb_resize=cv2.resize(im_rgb,(640,360))
        im_resize=cv2.UMat(cv2.resize(im,(640,360)))
    else:
        im_rgb_resize=im_rgb
        im_resize=cv2.UMat(im)

    face_UMat,fx,fy,fw,fh,fdetect=fun.detect_careto_OpenCV(im_resize)
    if (fdetect==1):
        cv2.rectangle(im_rgb_resize,(fx,fy) , (fx+fw,fy+fh), (127,0,255), 1)
        face=cv2.UMat.get(face_UMat)
        ROI_face=face[fy:fy+fh , fx:fx+fw]
        dimY,dimX=ROI_face.shape
        ROI_eyes=ROI_face[int(dimX*0.25):int(dimX*0.55),int(dimY*0.1):int(dimY*0.9)]
        ROI_mouth=ROI_face[int(dimX*0.7):int(dimX*1),int(dimY*0.2):int(dimY*0.8)]
        eyes,eye1,eye2,detect_eyes=fun.detect_eyes_OpenCV(ROI_eyes)

        if detect_eyes==1:
            ex1,ey1,ew1,eh1=eye1
```

```
            ex2,ey2,ew2,eh2=eye2
            expand_eyes=-5
            ROI_eye1=eyes[(ey1-expand_eyes):(ey1+eh1+expand_eyes) ,(ex1-
expand_eyes):(ex1+ew1+expand_eyes)]
            ROI_eye2=eyes[(ey2-expand_eyes):(ey2+eh2+expand_eyes) ,(ex2-
expand_eyes):(ex2+ew2+expand_eyes)]
            ex1=ex1+fx+int(dimY*0.1)-expand_eyes
            ey1=ey1+fy+int(dimX*0.25)-expand_eyes
            ex2=ex2+fx+int(dimY*0.1)-expand_eyes
            ey2=ey2+fy+int(dimX*0.25)-expand_eyes
            cv2.rectangle(im_rgb_resize,(ex1,ey1),(ex1+ew1,ey1+eh1),(0,255,0),1)
            cv2.rectangle(im_rgb_resize,(ex2,ey2),(ex2+ew2,ey2+eh2),(0,255,0),1)

               # Procesamiento morfologico y aproximacion de elipse a pupila
           apertura_ojo=fun.morf_proc(ROI_eye2,1)


        else:
            cv2.putText(frame, 'No se detecta ningun ojo', (100,100),
cv2.FONT_HERSHEY_TRIPLEX, 1, (127,0,255), 2)

        #Procesamiento para conocer el estado de la boca
        apertura_boca=fun.morf_proc_mouth(ROI_mouth)
    else:
        cv2.putText(frame, 'No se detecta ninguna cara', (100,100), cv2.FONT_HERSHEY_TRIPLEX,
1, (127,0,255), 2)
        apertura_boca=0
        apertura_ojo=0


    time_now=t.time()
    end=apertura_ojo
    if(len(glob_aper)<1000):
        glob_aper.append(end)
    else:
        glob_aper.popleft()
        glob_aper.append(end)

    if (len(glob_aper_med)==0):
        glob_aper_med.append(apertura_ojo)
        glob_aper_umb.append(apertura_ojo*0.2)
        med=0
    if(len(glob_aper_med)>=1000):
        med=float(sum(glob_aper))/float(len(glob_aper))
        glob_aper_med.popleft()
        glob_aper_med.append(med)
    else:
        med=float(sum(glob_aper))/float(len(glob_aper))
```

```
        glob_aper_med.append(med)
        umb=float(med)*float(0.2)
        glob_aper_umb.append(umb)

    print(len(glob_aper),apertura_ojo)
    cv2.imshow('Video',im_rgb_resize)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        plt.plot(glob_aper)
        plt.plot(glob_aper_med,'r')
        plt.plot(glob_aper_umb, 'g')
        plt.ylabel('Apertura de la boca (%)')
        plt.xlabel('Frames')
        plt.ylim(top=2)
        plt.show()
        break

cap.release()
cv2.destroyAllWindows()
```

IIII. Código para procesamiento desde webcam

```
import cv2
import numpy as np
import time as t
import math
from matplotlib import pyplot as plt
from collections import deque
import functions as fun



start_time=t.time()
cap = cv2.VideoCapture(0)
glob_aper=deque([])
glob_aper_med=deque([])
```

```
glob_aper_umb=[]
frame_num=-1

while(True):
   ret, frame = cap.read()
   frame_num=frame_num+1
   cv2.imwrite('dataset/T001/captura.png',frame)
   im_rgb=frame
   im=cv2.cvtColor(frame,cv2.COLOR_RGB2GRAY)
   y,x=im.shape
   if ((float(x)/float(y))==(float(4)/float(3))):
      im_rgb_resize=cv2.resize(im_rgb,(640,480))
      im_resize=cv2.UMat(cv2.resize(im,(640,480)))
   if((float(x)/float(y))==(float(16)/float(9))):
      im_rgb_resize=cv2.resize(im_rgb,(640,360))
      im_resize=cv2.UMat(cv2.resize(im,(640,360)))
   else:
      im_rgb_resize=im_rgb
      im_resize=cv2.UMat(im)

   face_UMat,fx,fy,fw,fh,fdetect=fun.detect_careto_OpenCV(im_resize)
   if (fdetect==1):
      cv2.rectangle(im_rgb_resize,(fx,fy) , (fx+fw,fy+fh), (127,0,255), 1)
      face=cv2.UMat.get(face_UMat)
      ROI_face=face[fy:fy+fh , fx:fx+fw]
      dimY,dimX=ROI_face.shape
      ROI_eyes=ROI_face[int(dimX*0.25):int(dimX*0.55),int(dimY*0.1):int(dimY*0.9)]
      ROI_mouth=ROI_face[int(dimX*0.7):int(dimX*1),int(dimY*0.2):int(dimY*0.8)]
      eyes,eye1,eye2,detect_eyes=fun.detect_eyes_OpenCV(ROI_eyes)

      if detect_eyes==1:
         ex1,ey1,ew1,eh1=eye1
         ex2,ey2,ew2,eh2=eye2
         expand_eyes=-5
         ROI_eye1=eyes[(ey1-expand_eyes):(ey1+eh1+expand_eyes) ,(ex1-
expand_eyes):(ex1+ew1+expand_eyes)]
         ROI_eye2=eyes[(ey2-expand_eyes):(ey2+eh2+expand_eyes) ,(ex2-
expand_eyes):(ex2+ew2+expand_eyes)]
         ex1=ex1+fx+int(dimY*0.1)-expand_eyes
         ey1=ey1+fy+int(dimX*0.25)-expand_eyes
         ex2=ex2+fx+int(dimY*0.1)-expand_eyes
         ey2=ey2+fy+int(dimX*0.25)-expand_eyes
         cv2.rectangle(im_rgb_resize,(ex1,ey1),(ex1+ew1,ey1+eh1),(0,255,0),1)
         cv2.rectangle(im_rgb_resize,(ex2,ey2),(ex2+ew2,ey2+eh2),(0,255,0),1)

            # Procesamiento morfologico y aproximacion de elipse a pupila
         apertura_ojo=fun.morf_proc(ROI_eye2,1)
```

```
        else:
            cv2.putText(frame, 'No se detecta ningun ojo', (100,100),
cv2.FONT_HERSHEY_TRIPLEX, 1, (127,0,255), 2)

        #Procesamiento para conocer el estado de la boca
        apertura_boca=fun.morf_proc_mouth(ROI_mouth)
    else:
        cv2.putText(frame, 'No se detecta ninguna cara', (100,100), cv2.FONT_HERSHEY_TRIPLEX,
1, (127,0,255), 2)
        apertura_boca=0
        apertura_ojo=0


    time_now=t.time()
    end=(apertura_ojo,apertura_boca,time_now)
    if(len(glob_aper)<1000):
        glob_aper.append(end)
    else:
        glob_aper.popleft()
        glob_aper.append(end)

    if (len(glob_aper_med)==0):
        glob_aper_med.append(apertura_ojo)
        glob_aper_umb.append(apertura_ojo*0.2)
        med=0
    if(len(glob_aper_med)>=1000):
        med=float(sum(glob_aper))/float(len(glob_aper))
        glob_aper_med.popleft()
        glob_aper_med.append(med)
    else:
        med=float(sum(glob_aper))/float(len(glob_aper))
        glob_aper_med.append(med)
        umb=float(med)*float(0.2)
        glob_aper_umb.append(umb)

    print(len(glob_aper),apertura_ojo)
    cv2.imshow('Video',im_rgb_resize)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        #plt.plot(glob_aper)
        #plt.plot(glob_aper_med,'r')
        #plt.plot(glob_aper_umb, 'g')
        #plt.ylabel('Apertura de la boca (%)')
        #plt.xlabel('Frames')
        #plt.ylim(top=2)
        #plt.show()
        break
```

```
cap.release()
cv2.destroyAllWindows()
```