Universitat Autònoma de Barcelona

escola d'enginyeria

5751-1: DETECCIÓN DE FATIGA AL VOLANTE

Memoria del Trabajo Final de Carrera
Ingeniería Informática
Realizado por
Abdelilah Choukri
y dirigido por
Katerine Diaz Chito
Bellaterra, a 15 de Septiembre de 2014

Universitat Autònoma de Barcelona

escola d'enginyeria

El Sotasignat, Katerine Diaz Chito

Profesor/a de l'Escola d'Enginyeria de la UAB,

CERTFICA:

Que el treball a que correspon aquest memòria ha estat realitzada sota la seva direcció per en/na Abdelilah Choukri.

I per a que consti firma la present.

Signat: Katerine Diaz Chito

Bellaterra, 15 de setembre de 2014

Índice:

Índice de Figuras:

Capítulo 1

Introducción y Objetivos

1.1 Introducción

¿Qué es la fatiga?, Según *Medline Plus*[3] "Es una sensación de falta de energía, de agotamiento o de cansancio", y se puede determinar con síntomas[9] como: la temperatura corporal, el movimiento de los ojos, el ritmo de respiración y las actividades del cerebro.

Según el *Wrold Health Organization* [2], La fatiga es una de las principales causas de los accidentes de tráfico. En las navidades de 2013, el 40% de los accidentes fueron causados por la fatiga [1], en el mismo año, la fatiga provoco entre 20%~30% de accidentes[15], especialmente cuando se combina con el alcohol. Desde aquí surge la necesidad de sistemas de asistencia al conductor que detecten la fatiga al volante, y den una señal de alerta al conductor.

Actualmente, los sistemas de asistencia al conductor que detentan el grado de fatiga los encontramos en coches de gama alta [4], como son el *Driver Alert Control* de *Volvo, Attention Assist* de *Mercedes-Benz* o *Driver Alert* de *Ford*. Estos sistemas monitorean al conductor a partir de sensores que observan: el giro del volante, la desviación de los carriles, Etc….

También existen sistemas independientes del coche que monitorean la fatiga, como por ejemplo *Vigo*[19] . Vigo es un auricular Blutooth que mesura el parpadeo de los ojos y los movimientos del cuerpo, usando sensores de infrarrojo y acelerómetro, con un algoritmo que sigue los patrones de los ojos y el cuerpo.

Muchos artículos académicos y revistas de la ciencia de la computación [9-13], proponen sistemas de detección de fatiga basado en características faciales, especialmente características del ojo que permiten determinar su apertura para monitorear el parpadeo de los ojos, que según estos artículos es la síntoma más relevante de la fatiga durante la conducción.

El objetivo de este proyecto es diseñar e implementar un sistema de bajo coste de asistencia al conductor, que permita cuantificar el nivel de fatiga del

conductor. Basamos la detección de fatiga visualmente a partir de la caracterización de los ojos, usando tácticas de procesamiento de imágenes, inteligencia artificial y visión por computador. Durante el desarrollo del proyecto se ha experimentado con varias técnicas, pero no se han incorporado todas en la implementación final. Las técnicas analizadas y testeadas, que se han descartado están mencionadas en el anexo 4: métodos descartados.

1.2 Objetivos

Para llegar a cuantificar y detectar la fatiga a tiempo real, debemos pasar por algunos objetivos esenciales:

- **El estado del arte actual**: Buscar y estudiar los métodos que se usan actualmente en visión por computador en el reconocimiento de ojos, y la caracterización de la fatiga.
- **Determinar las características relevantes**: Analizar y determinar, las características más relevantes y discriminatorias, para definir el estado del conductor.
- **Detección y seguimiento de características faciales:** Estudiar los mejores métodos que nos permitan detectar y seguir las características faciales, que permitan caracterizar al conductor.
- **Cuantificación de fatiga**: A partir de la caracterización facial, determinar el grado de fatiga del conductor.
- **Diseño e implementación del sistema de asistencia**: Diseñar e implementar un sistema de detección de fatiga.

1.3 Viabilidad del Proyecto

1.3.1 Viabilidad Económica

El grupo de ADAS de Centre de Visio per Computador nos facilitó unas capturas de conductores al volante para desarrollar la detección y seguimiento del ojo. Luego hemos usado capturas de caras frontales con ojos abiertos y cerrados para llevar acabo la determinación del estado del ojo. El software y librerías usados son de carácter Libre u Open-Source como:

- QtCreator 3.0.1 basado en Qt 5.2.1. y MinGW (versión no comercial).
- OpenCV 2.48.
- CMake 2.8.12.2.

Al nivel de HardWare, el proyecto fue desarrollado y testeado sobre unLaptop personal del alumno, que lleva un procesador Intel Core i5-3230M Dual Core, de 2.6 GHz y con 6GB de RAM. La cámara usada es la cámara del laptop con una resolución que llega hasta 1280 x 720.

1.3.2 Viabilidad Técnica

El sistema fue desarrollado con usando QtCreator que facilita la programación de GUIs con C++, y contiene la librería de Qt 5 que ofrece estructuras y funcionamiento como: el acceso a directorios del sistema operativo, manejo de Strings, estructuras de datos (listas, pilas, diccionarios…), el patrón Observer para crear un Timer o detectar interrupciones de eventos externos (clics, movimiento ratón…). Un gran parte del programa fue desarrollado con OpenCV, que facilita el uso de las técnicas y los algoritmos del procesamiento de imágenes y visión por computador.

Estos dos componentes se usan mucho en el ámbito de crear software de visión por computador, además los desarrolladores ofrecen una documentación amplia online y un mantenimiento continuo que garantiza la reusabilidad de las aplicaciones desarrolladas con QtuOpenCV.

Capítulo 2

Conceptos Fundamentales

Como hemos mencionado antes en los objetivos, basamos la detección de fatiga sobre las características faciales, para conseguir esto, hemos estudiado varios artículos [9-13] para ver cómo se consigue la detección de fatiga desde una captura de cámara hasta la cuantificación de la fatiga.

En esta sección explicaremos 3 conceptos: primero como es el flujo principal de un sistema que detecta la fatiga y algunos métodos que se usan en cada componente del flujo, luego pasamos a explicar los métodos que hemos usado en el sistema implementado y finalmente los métodos que nos han servido para reducir el ruido.

2.1 El Flujo Principal

Los sistemas propuestos por los artículos mencionados arriba, comparten el mismo flujo principal, solo varían en algunos flujos adicionales o en los métodos usados en cada fase del flujo. El siguiente diagrama representa el flujo principal:

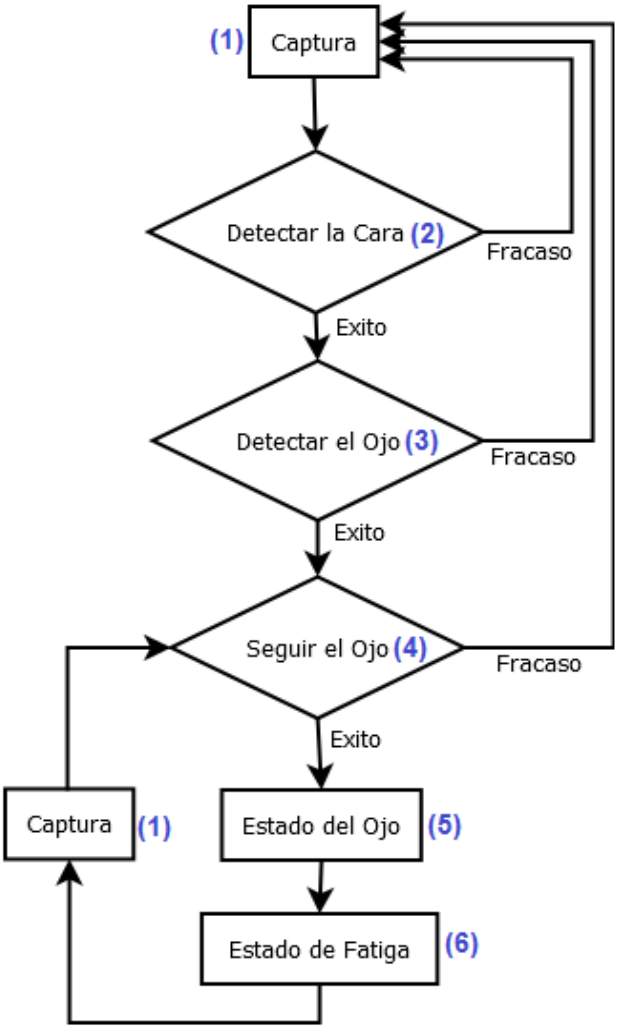


Figura 1: Flujo Principal de un sistema de sección de fatiga

2.1.1 Captura

La entrada principal de todo el sistema, actúa como el proveedor de las imágenes (por ejemplo capturas de video o cámara). Los sistemas propuestos en los artículos, usan cámaras digitales (CCD).

2.1.2 Detección de la Cara

Recibe una captura de imagen, y su rol es analizar y determinar dónde está el rostro, en caso de detectar la cara, se recorta el área donde se detectó para proceder a detectar el ojo, sino se procede a detectar la cara en el siguiente frame.

De los métodos que hemos encontrado en la literatura, podemos mencionar los siguientes:

- Detección basada en el color de piel[10-12]: En esta técnica, se determina el área de una cara obteniendo un bounding-box sobre los pixeles que se han clasificado como piel, luego se pasa a la detección de otros componentes, como los ojos (Figura 2).

Figura 2: Detección del rostro basada en el color de piel.

- Clasificador en Cascada[13]: Se usa para detectar objetos basándose en sus características, en este caso el objeto es el rostro. Como es el método que hemos usado en el proyecto, esta explicado con más detalle abajo (sección 2.2.1).

2.1.3 Detección del Ojo

Aquí, se procede a detectar el área del ojo dentro del área de la cara detectada previamente, en caso que se ha fracasado la detección del ojo, el sistema vuelve a la fase anterior para detectar la cara.

Algunos métodos para conseguir el área de ojo pueden ser:

- Template Matching [13]: Este método requiere una plantilla del ojo, que a partir de la cual se calcula la posición donde hay más similitud entre la plantilla y el área de la cara. Este método esta explicado en la sección 2.2.2

- Proyección Horizontal [11-12]: Primero se coge la mitad superior del área de la cara para detectar los contornos, así se obtiene una imagen binaria, que luego se proyecta horizontalmente para conseguir la posición vertical del ojo (Figura 3).

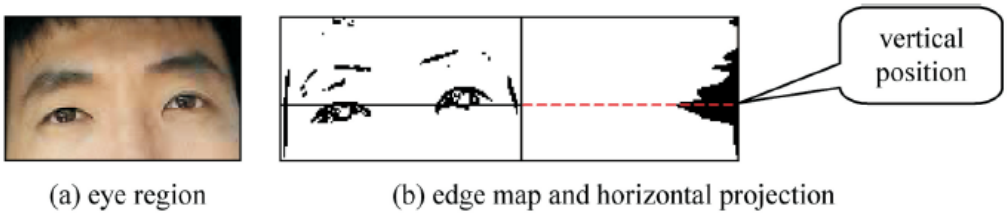


Figura 3: Detectar la posición de los ojos usando proyección horizontal.

2.1.4 Seguir el Ojo

Una vez que se conoce el área del ojo, se pasa a seguir lo en un área reducido en vez de toda la captura, considerando que el conductor no se mueve mucho, esto es ideal para un sistema a tiempo real para conseguir una eficiencia al nivel de cómputo. Mientras que el seguimiento no fracase, se procede a determinar el estado del ojo sino, consideramos que el ojo ya no existe en el área actual, entonces volvemos otra vez a la fase de detección de la cara. En la mayoría de los casos se usa el Template Matching [12], esto requiere guardar una plantilla del ojo en la fase anterior para proceder a detectar la durante el seguimiento, cuando no se cumple un criterio de similitud entre la plantilla y la área actual se considera un fracaso. Esto es metodología

2.1.5 Estado del Ojo

El objetivo de esta fase es determinar el estado del ojo, si está cerrado o abierto, o el porcentaje de apertura, con este dato se procede a actualizar el estado de fatiga.

Aquí hemos notado una variedad de técnicas como por ejemplo:

- Comparar el área del ojo con una plantilla de ojo abierto [10][13], el porcentaje de similitud será el grado de apertura del ojo.

- Usar un detector de contornos (como el Canny por ejemplo), paraestimar del grado de la apertura del ojo[11], a partir de medir la distancia entre los parpados (Figura 4).

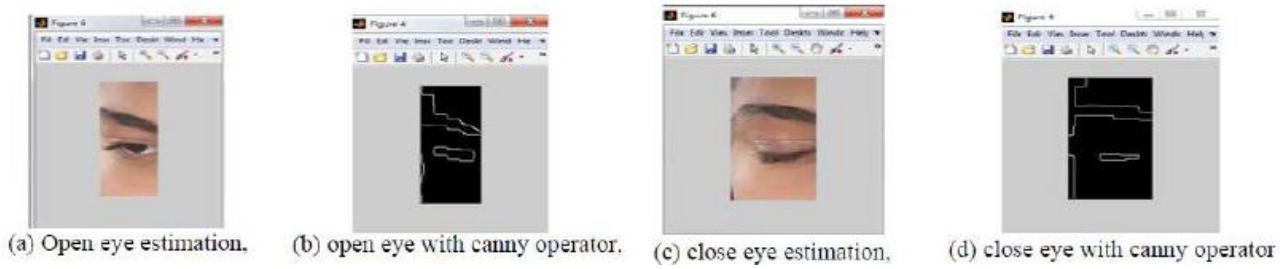


Figura 4: Estimación de la distancia entre los parpados

- Convertir el color desde el espacio RGB a HSI, y determinar si hay pixeles que tienen una saturación menor que un valor determinado[12] (Figura 5), sino de determina que el ojo está cerrado.

(a) open eye (b) eyeball pixels (c) closed eye (d) no eyeball pixels

Figura 5: Detectar la apertura a partir de la saturación HSI del ojo.

2.1.6 Estado de la Fatiga

La salida principal del sistema, aquí se calcula el estado de fatiga basándose en la variación del estado del ojo durante un periodo de tiempo. Una vez calculado el estado actual de fatiga, el sistema regresa al estado del seguimiento del ojo.

El método más usado para cuantificar la fática desde la variación de la apertura del ojo es el PERCLOS [9], y es el método que hemos usado en nuestro proyecto, y por lo tanto esta explicado en la siguiente sección con más detalle.

2.2 Técnicas usadas en el Proyecto

Después de determinar cómo es el flujo principal y las fases de un sistema de detección de fatiga, falta escoger que método vamos a usar en dichas fases, a continuación explicamos las técnicas que hemos incorporado en la implementación final:

2.2.1 Cascade Classifier

El algoritmo más usado para detectar objetos, fue propuesto por Paul Viola y Michael Jones con su algoritmo Viola-Jones [6] para detectar caras basándose en AdaBoost [7] (un conjunto de clasificadores débiles que se agrupan para crear un clasificador fuerte).

Cada objeto está formado por características Haar (Figura 6), además cada característica está presente con frecuencia deferente que la otra según el objeto en cuestión, la idea aquí es ordenar estas características en conjuntos y cada conjunto forma una cascada (Figura 7). En la fase del entrenamiento se determina la ordenación de estas características en cada cascada.

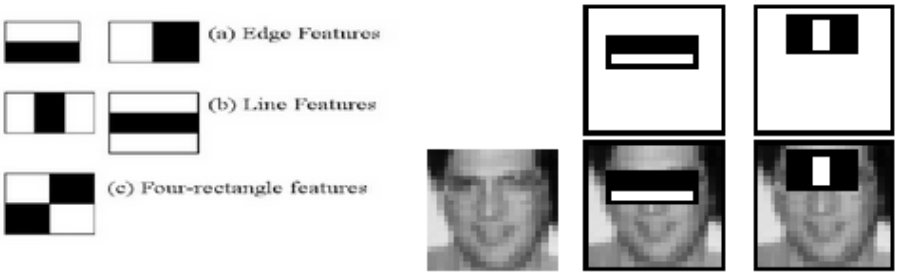


Figura 6: Características Haar, ejemplo Características del rostro

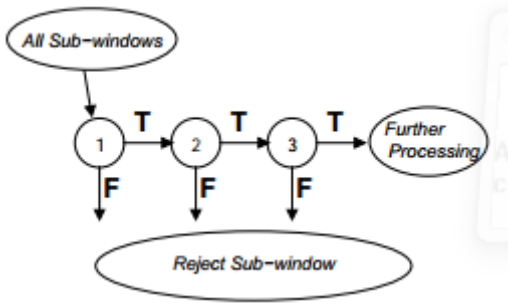


Figura 7: Clasificador en Cascada de Viola-Jones

Para detectar un objeto en una imagen, se desliza una sub-ventana sobre esta imagen, después cada cascada determina si la sub-ventana no contiene el objeto (en este caso se rechaza la sub-ventana), o si hay una posibilidad que el objeto se detectara con la siguiente cascada, si la sub-ventana pasa por todas las cascadas, entonces se determina que objeto en cuestión está en esta sub-ventana.

2.2.2 Template Matching

Otro método que detecta objetos en una imagen, pero al contrario que el Clasificador en Cascada, esta técnica no requiere entrenamiento, el objeto aquí se representa con una imagen como plantilla. Para detectar el objeto en cuestión en una imagen, se desliza una sub-ventana del tamaño de la plantilla y se calcula la suma de la deferencia entre los pixeles de la plantilla con los pixeles de la sub-ventana de la imagen, la posición del pixel donde hay el mínimo valor calculado (mínimo absoluto) será la posición donde se ha detectado el objeto. No siempre se calcula la diferencia, por ejemplo se puede usar la correlación entre los pixeles de la sub-ventana y la plantilla, pero en estos casos se coge la posición del pixel con el valor máximo calculado (pico absoluto).

2.2.3 SVM (Support Vector Machine)

Es un clasificador binario, que detecta patrones entre dos clases, calculando un hiperplano durante la fase del entrenamiento [8](Figura 8), el hiperplano separa las dos clases durante la fase del entrenamiento.

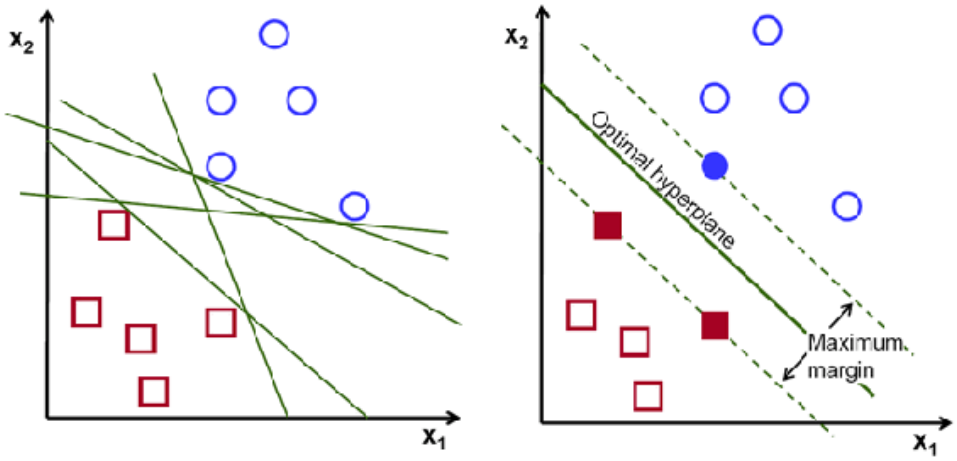


Figura 8: Ejemplo de Entrenar un SVM en 2D

Para clasificar las nuevas muestras, se determina a qué lado del hiperplano pertenecen.

2.2.4 PCA (Principal Component Analisis)

Es una técnica que nos permite proyectar datos originales a un espacio donde las imágenes de estos datos cogen una distribución más discriminante [20](Figura 9), así se puede detectar los patrones que distinguen entre clases, y como consecuencia nos permite reducir la dimensión de los datos originales.

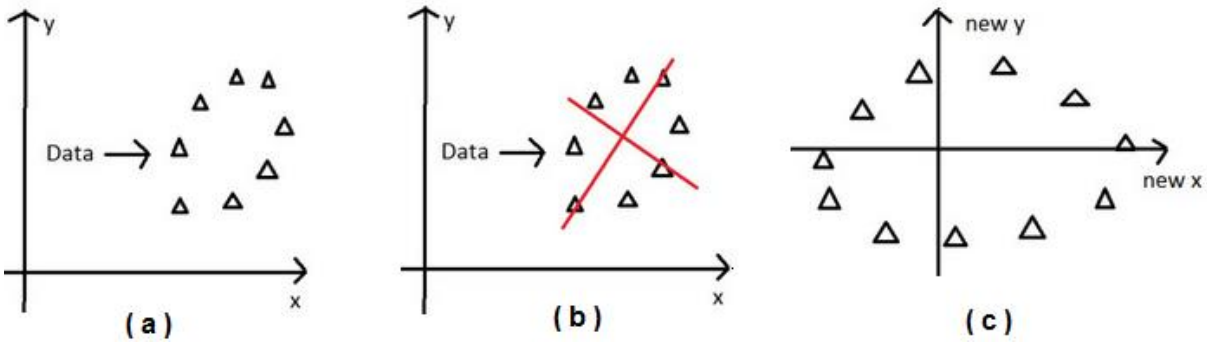


Figura 9: a-Datos Originales, b-Calcular el nuevo espacio, c-Proyectar los Datos

2.2.5 Perclos

Es un algoritmo que determina el porcentaje del cierre de los ojos durante un intervalo de tiempo, primero se determina un threshold de cierre de los ojos (Figura10 - Rojo), se calcula la suma de las duraciones de tiempo en las que el porcentaje del cierre de los ojos ha sido menor que el threshold (Figura 10- Verde), luego se divide esta suma sobre un periodo de tiempo (Figura 10 - Lila) para calcular el Perclos.

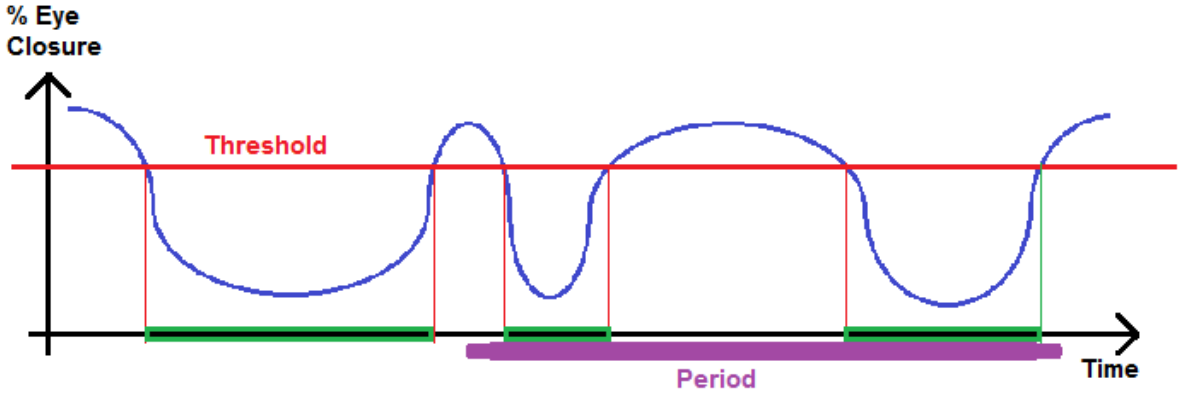


Figura 10: Componentes del algoritmo del Perclos.

2.3 Métodos para reducir el Ruido

Durante el desarrollo del proyecto hemos usado algunas técnicas para quitar efectos del ruido, como: sobras o cambios de iluminación, pixeles deformados…,

2.3.1 Histogram Equalization

Este método se usa para mejorar el contraste de las imágenes, estrechando el rango de las intensidades [14] (Figura 11). Así conseguimos que las intensidades con frecuencias muy altas no se queden centradas en un rango pequeño.

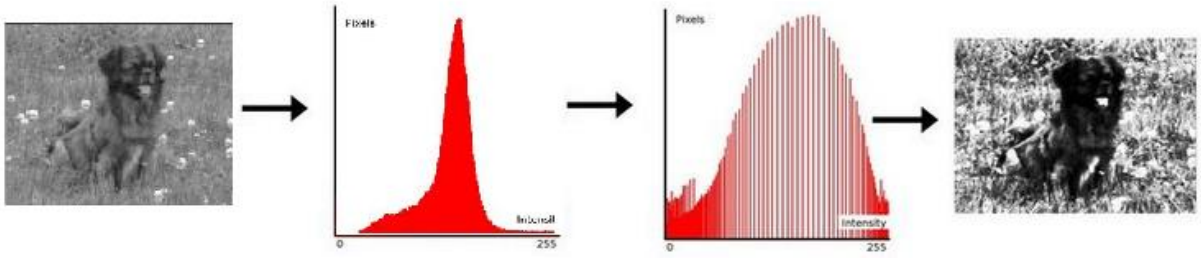


Figura 11: Ejemplo de Ecualización del Histograma

2.3.2 Morphology (Dilatation)

Uno de los usos de la dilatación[16] es reducir el ruido en imágenes binarias, quitando pixeles en puntos separados u reducir el ancho de los objetos en pixeles continuos (Figura 12).

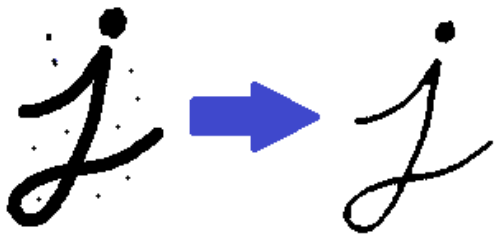


Figura 12: Ejemplo de dilatación.

Capítulo 3

Metodología y Resultados

En el flujo principal mencionado en la sección 4.1 (Figura 1), el sistema tiene como entrada una captura y la salida es el estado de fatiga, para conseguir esto pasamos por varios sub-sistemas que a su vez receben entradas y envían salidas uno al otro. Entonces para simplificar el desarrollo de este proyecto hemos desarrollado cada sub-sistema aparte y luego hemos juntado todas las partes en el sistema final, tomando en cuenta los métodos usados en cada sub-sistema:

- Detección de la Cara: CascadeClassifier.
- Detección del Ojo; CacadeClassifier.
- Seguimiento del Ojo: TemplateMatching
- Estado del Ojo: Proyección Binaria y Clasificador SVM (con PCA)
- Estado de Fatiga: PerClos

En este capítulo, empezamos con la preparacion previa que consiste en crear un ground-truth, luego, en las siguientes secciones explicamos la metodología usada en cada fase del flujo principal, acompañada con su implementación y resultados.

3.1 Preparación Previa

Antes de implementar la detección de la cara y el ojo, necesitamos un GroundTruth para medir el porcentaje de acierto de la detección.

A partir de los métodos disponibles en la librería OpenCV, en el ámbito de detección de caras, ojos y boca, se ha implementado una GUI (con Qt) para etiquetar las muestras del dataset proporcionado por el grupo de ADAS. En cada captura se usa el clasificador en cascada que ofrece OpenCV para detectar la cara, ojos y boca. Cuando la detección de algún componente no es correcta, se edita su posición a través de la GUI (Figura 13). Además, el estado de cada componente se etiquita

como cerrado o abierto, para su posterior uso en el desarrollo del proyecto.

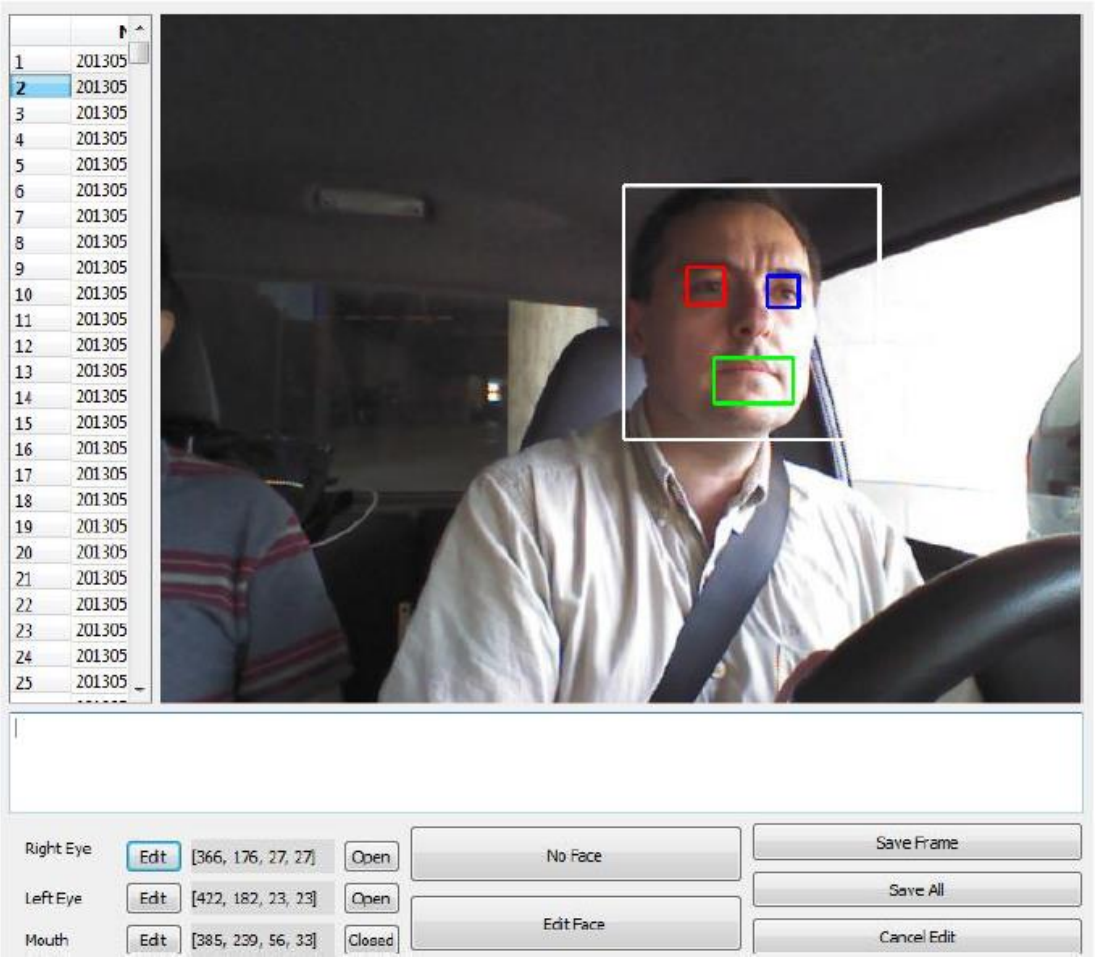


Figura 13: GUI para etiquetar el dataset.

Al final hemos descartado el estado de la boca en la implementación final, debido a que los artículos mencionados en el estado del arte y otros artículos que hemos buscado durante el desarrollo, ignoran totalmente el estado de la boca para la detección de fatiga, y se centran más en el estado del ojo.

3.2 Detección de la Cara

Después de recibir la primera captura, debemos determinar el área de la cara, hemos optado con el clasificador en cascada de OpenCV que nos permite clasificar en multi-escala [17], es decir, que nos permite detectar un objeto como la cara de cualquier tamaño dentro de una imagen, esto se consigue re-escalando la imagen original según un factor

de escala (Figura 14), luego en cada se procede a detectar el objeto usando el clasificador en cascada simple.

Otro parámetro que se usa en la detección multi-escala es MinNeighbors (El mínimo de los vecinos), cuando se ha detectado la cara en la misma área en diferentes escalas consecutivas (Figura 15) y el número de las veces que se ha detectado es mayor o igual que el MinNeighbors, consideramos que esta área contiene el objeto en cuestión sino, la rechazamos.

Para los valores de los parámetros ScaleFactor y MinNeighbors del clasificador multi-esclar, hemos usado la recomendación en los documentos de OpenCV, re-escalar la imagen original un 10% en cada iteración, y considerar 3 vecinos mínimos.

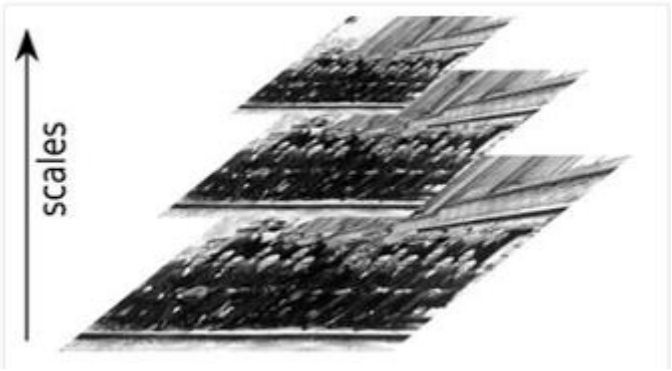


Figura 14: Re-Escalar una imagen

Figura 15: Detección de una cara en escalas diferentes

3.2.1 Implementación

OpenCV ofrece un clasificador en cascada de caras entrenado con una gran cantidad de muestras, hemos incorporado este clasificador en el deseño de la fase de detección de cara según el siguiente flujo (Figura 16):

(1) Primero debemos convertir la imagen en color a la escala de gris, que facilita el procesamiento de la imagen y nos permite usar el clasificador en cascada.

(2) Después de recibir la imagen en gris, pasamos a ecualizar su histograma para quitar efectos de la sombra y variación de contraste

(3) Pasamos la imagen ecualizada al clasificador para detectar el área de la cara.

(4) Si el ancho y el alto del área es cero, entonces no se detectó la cara, en este caso volvemos a coger el siguiente frame, sino seguimos a la siguiente fase.

(5) Una vez detectado el área, recortamos esta área para obtener una imagen de la cara en escala de gris.

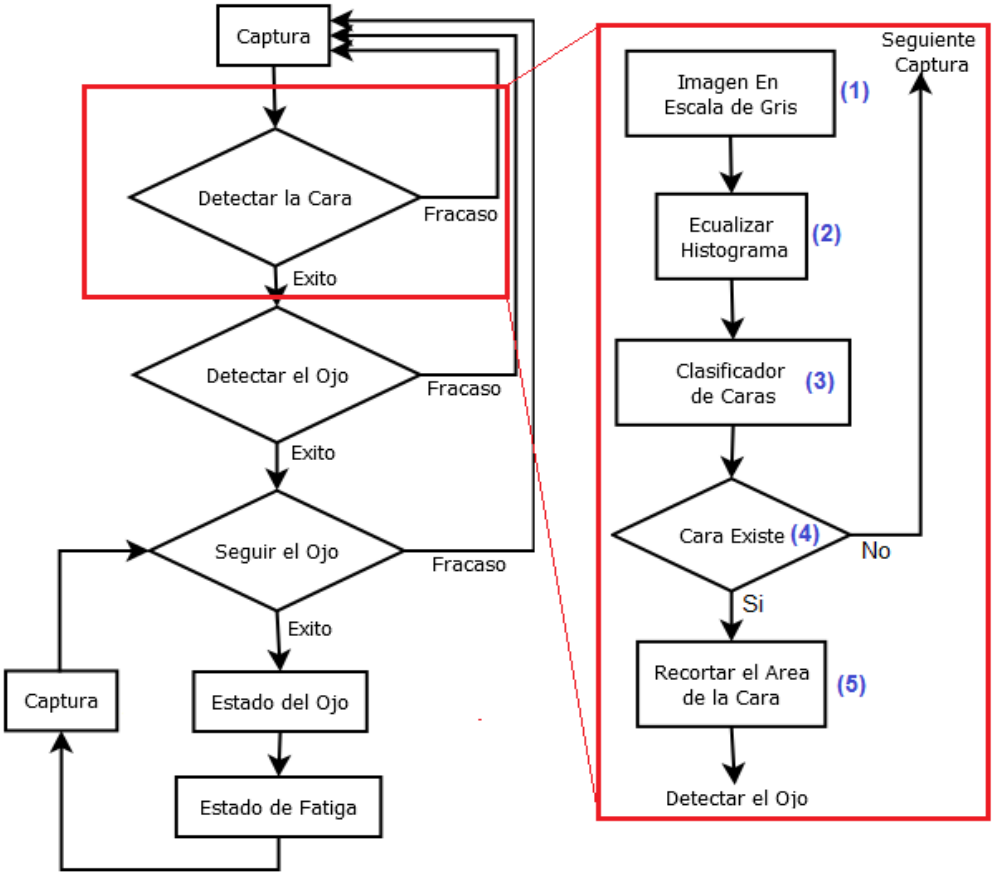


Figura 16: El flujo de la detección de la cara

3.2.2 Resultados

Hemos probado la tasa de acierto del clasificador implementado sobre el groudntruth creado con el data set de ADAS, y nos dio un 70% de acierto, pero más adelante hemos vuelto a testear el detector con otro datatsetde caras frontales, y la tasa de acierto llego a un 96%. Entonces para detectar características faciales, es mejor tener capturas con una orientación frontal lo más posible.

3.3 Detección del Ojo

Con la fase anterior, obtenemos una imagen de la cara en escala de gris, nuestro objetivo ahora es determinar el área del ojo dentro de esta imagen, además debemos guardar el área donde se detectó en la captura para la fase del seguimiento.

3.3.1 Implementación

Como en la fase anterior, hemos usado un clasificador en cascada de ojos que proporciona OpenCV, que hemos usado siguiente el siguiente flujo (Figura 17):

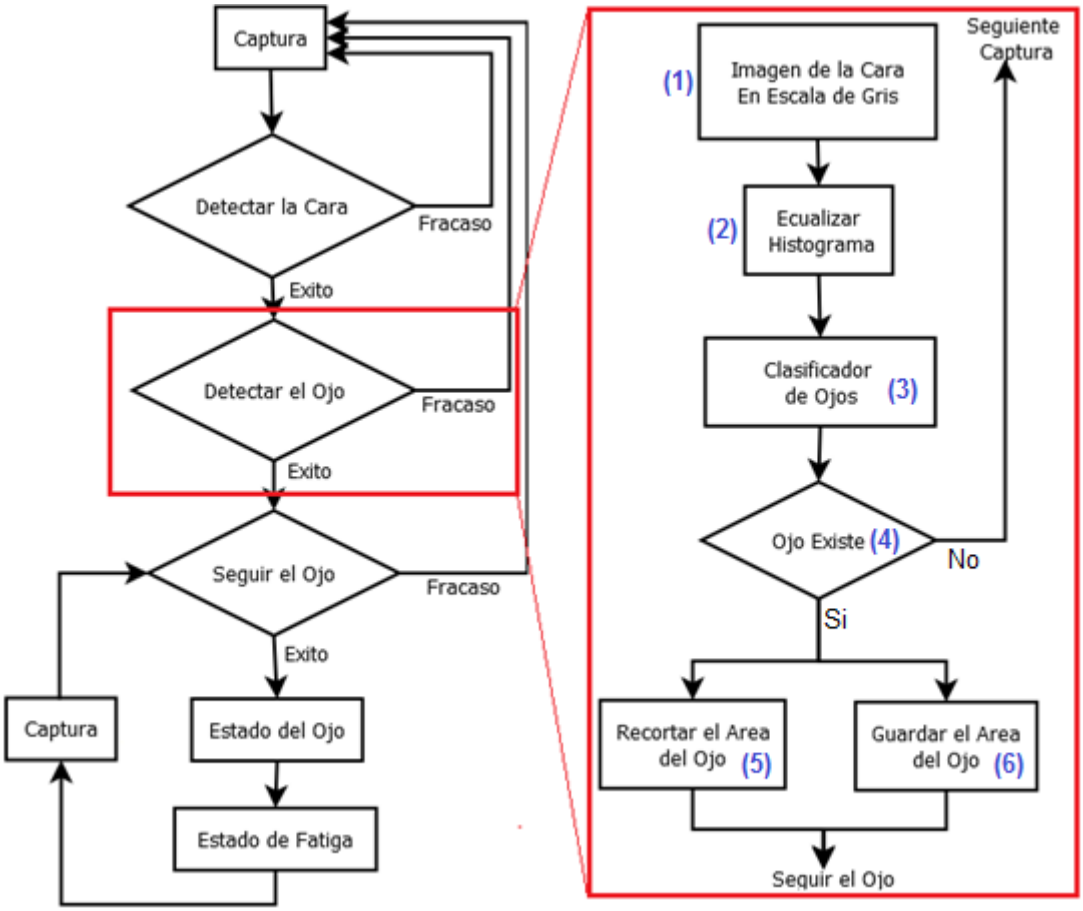


Figura 17: El flujo de la detección de los ojos.

(1) Recibimos la imagen de la cara que hemos detectado en la fase anterior.

(2) Ecualizamos el histograma de la imagen.

(3) Pasamos la imagen ecualizada al clasificador para detectar el área del ojo.

(4) Si el ancho y el alto del área es cero, entonces no se detectó el ojo, en este caso volvemos a coger el siguiente frame, sino seguimos a la siguiente fase.

(5) Una vez detectado el área, recortamos esta área de la imagen ecualizada para guardar un plantilla del ojo que nos servirá en la fase del seguimiento.

(6) Ademas de la plantilla del ojo, debemos guardar el área donde se detectó el ojo para la fase del seguimiento.

3.3.2 Resultados

Al testear la tasa de acierto del clasificador sobre las muestras del groundtruth, hemos notado que solo se consiguió clasificar el 20% de las muestras de forma correcta. Aparte de que el groundtruth contiene caras no frontales, el problema reside en que la mayoría de las personas llevan gafas, lo que provoca una alteración de las características faciales alrededor de los ojos, como por ejemplo: el reflejo de las gafas (Figura 18 - a) o el marco de las gafas (Figura 18 - b).

Figura 18: Alteraciones que provocan las gafas.

Más adelante en el proyecto, hemos usado otro dataset de caras frontales, la tasa de acierto con este dataset aumento hasta un 64%, la mayoría de los casos de fallo han sido con las muestras de caras con gafas.Despues de excluir estas muestras del dataset nuevo, la detección mejoró hasta un 78%.

3.4 Seguimiento del Ojo

Hasta ahora, hemos conseguido una plantilla del ojo y el área donde se detectó. Consideramos que el conductor no se mueve mucho, entonces no hace falta detectar el área en cada frame, solo marcando un área alrededor del área donde hemos confirmado la existencia del ojo previamente, que simplemente será una oscilación del área original según un factor de escala (Figura 19). Nuestro objetivo aquí es a partir de las siguientes capturas, conseguir donde está el ojo en un área limitada.

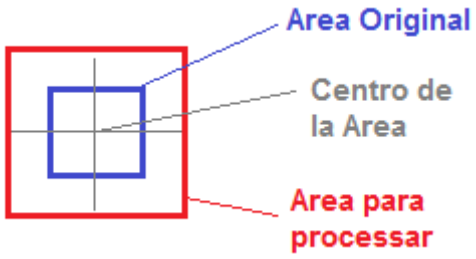


Figura 19: Escalar un área respecto al centro

3.4.1 Implementación

Como hemos mencionado en el estado del arte, el TmeplateMatching se usa para calcular la similitud entre una plantilla y una imagen, obteniendo la posición con más similitud. OpenCV nos ofrece una implementación del algoritmo con dos opciones como criterio de similitud: calcular la diferencia o calcular la correlación. Hemos optado con el cálculo de la diferencia, en este caso la implantación de OpenCV usa la fórmula de la diferencia normalizada (Figura 20). En seguida explicamos los pasos del flujo para el seguimiento del ojo en una captura (Figura 21).

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Figura 20: Fórmula para calcular la diferencia entre la plantilla y la imagen.

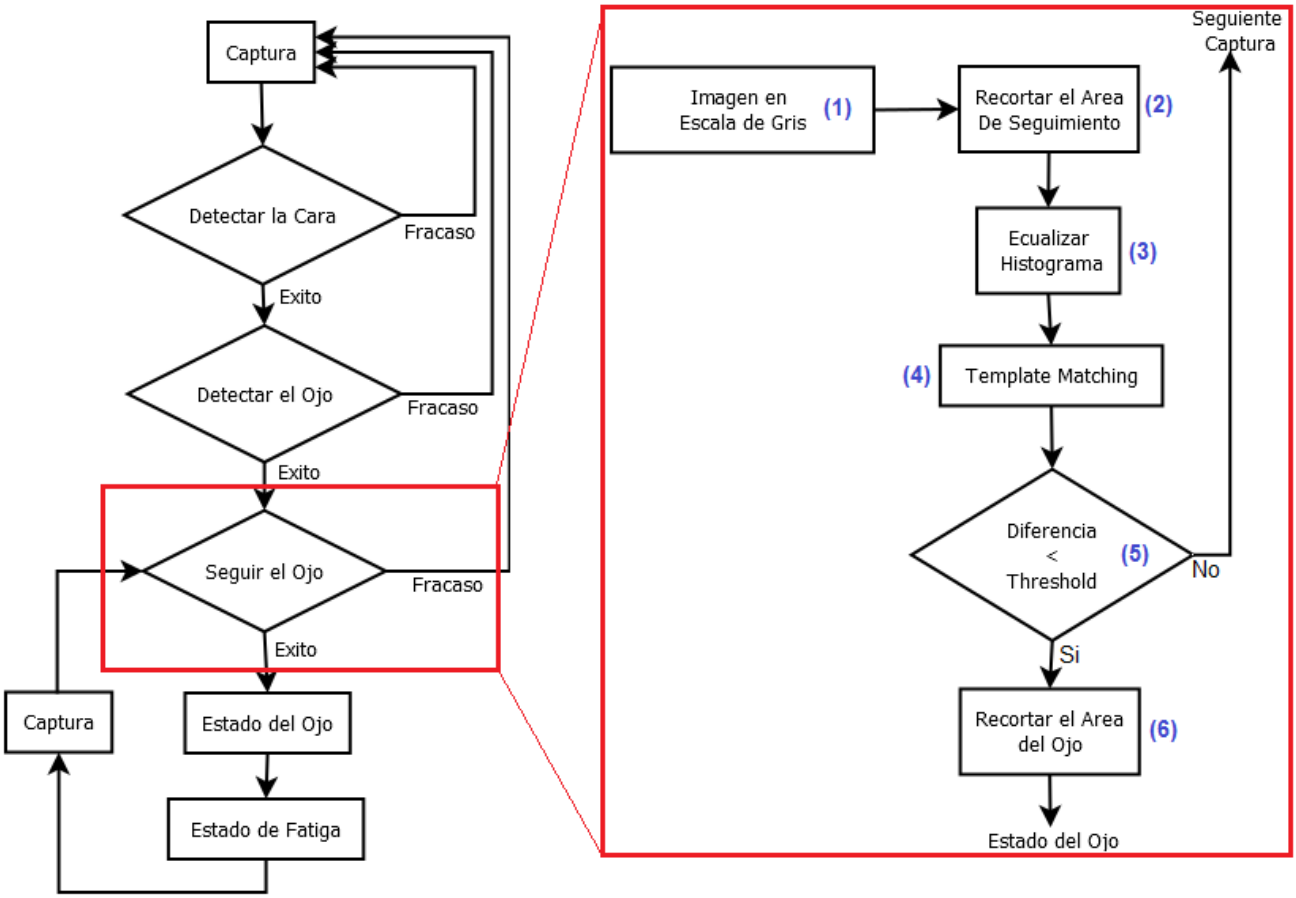


Figura 21: El Flujo del Seguimiento del Ojo

(1) Convertimos la captura en color a escala de gris.

(2) Recortamos el área a procesar.

(3) Ecualizamos el histograma de la imagen recortada.

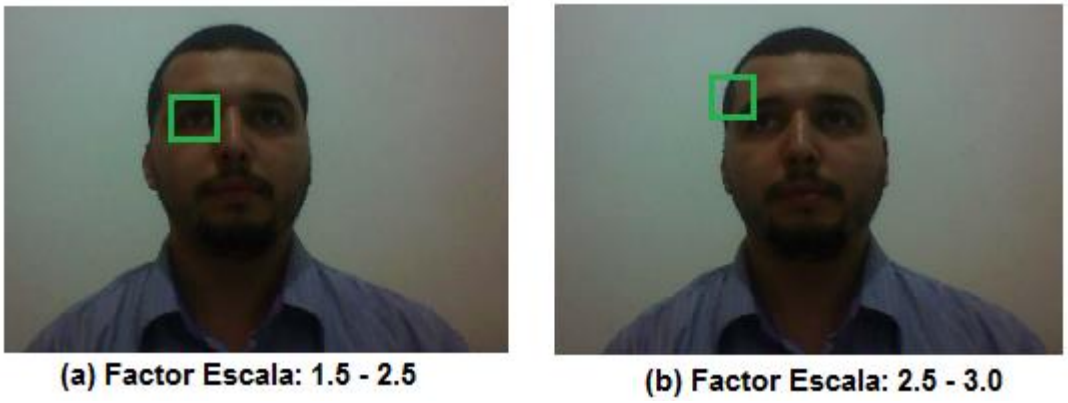
(4) Calculamos la diferencia entre la plantilla del ojo y la imagen usando el algoritmo de TemplateMatching.

(5) Desde el algoritmo, obtenemos dos datos, el valor mínimo de la diferencia calculado y su posición dentro de la imagen. Si el valor es mayor que un threshold determinado, consideramos que el ojo ya no existe en el área de seguimiento, con lo cual debemos regresar a la fase de detección.

(6) Recortamos el área del ojo, desde la posición del valor mínimo de la diferencia, así obtenemos una imagen del ojo para la siguiente fase.

3.4.2 Resultados

Para determinar el factor de escalar del área original del ojo, hemos experimentado con capturas de cámara a tiempo real (30 FPS), variando el factor de escala entre 1.5 y 3.0, el seguimiento era correcto entre 1.5 y 2.5 (Figura 22 - a), mientras que valores superiores, el seguimiento era erróneo (Figura 22 - b).



(a) Factor Escala: 1.5 - 2.5

(b) Factor Escala: 2.5 - 3.0

Figura 22: Efecto del Factor de Escala sobre el seguimiento.

Falta determinar un thresholdmínimo de la diferencia entre la plantilla y el área del seguimiento, para esto, hemos guardado los valores que retorna el retorna el TemplateMatching durante una sesión de seguimiento correcto a tiempo real (con cámara a velocidad de 30 FPS). Hemos observado que los valores guardados varían entre 0.08 y 0.13, entonces hemos establecido que el threshold deberá tener un valor entre 0.15 y 0.2 para asegurar que el ojo este presente dentro del área del seguimiento.

3.5 Estado del Ojo

En esta fase, nuestro objetivo es conseguir el estado del ojo a partir de una imagen del ojo que hemos obtenido en la fase anterior. Hemos experimentado con varias técnicas inspiradas de los artículos mencionados en el estado del arte, pero como al final no los hemos incorporado en la implementación final, están explicados en el anexo 4.

En esta fase hemos implementado dos métodos (Figura 23). El primero es nuestra propuesta que está basada en proyecciones para obtener el porcentaje de la apertura del ojo. El segundo es un clasificador SVM con dos clases: ojo abierto y ojo cerrado.

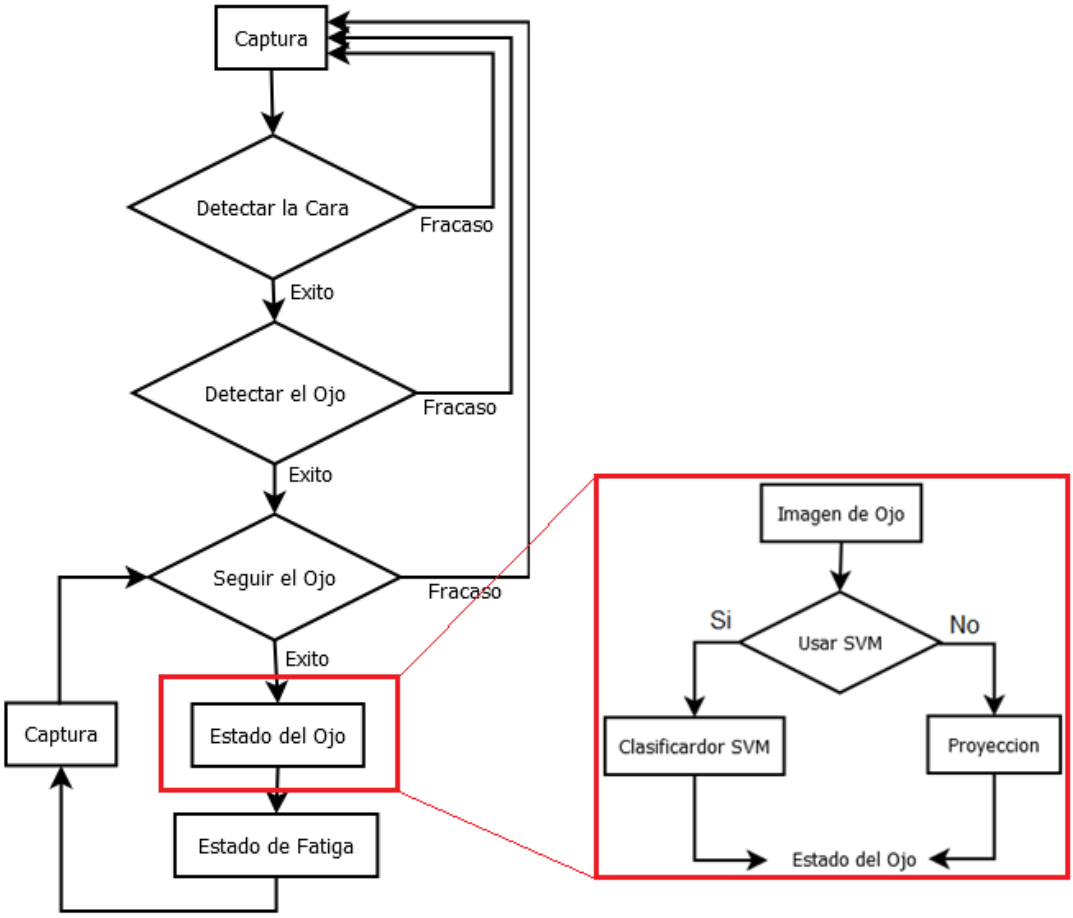


Figura 23: El Flujo de detectar el estado del ojo

3.5.1 Proyección Horizontal y Vertical

Al ecualizar el histograma de las imágenes de los ojos, hemos observado que los pixeles más centrales del ojo (iris, parpados cerrados) tienen una intensidad muy baja (color negro), si aplicamos un threshold obtendremos una imagen binaria que nos simplificara el proceso de detectar el estado del ojo (Figura 24).

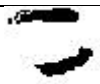
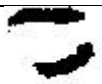
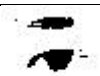
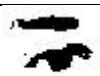
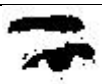
Threshold / Imagen Ecualizada	0.10	0.15	0.20
			
			

Figura 24: Aplicar un threshold sobre imágenes ecualizadas del ojo

3.5.1.a Análisis

Para obtener la apertura del ojo, usamos una proyección horizontal de las imágenes binarias, como en la siguiente figura:

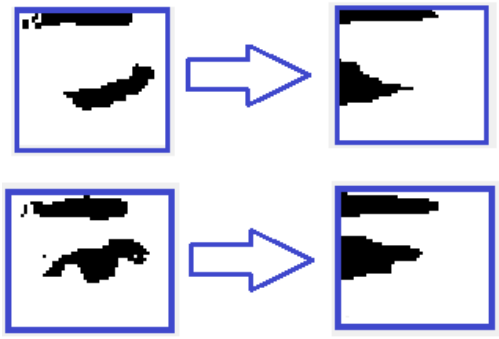


Figura 25: Proyección horizontal.

Tenemos dos picos, uno en la parte superior de la ceja y el otro en la parte central, entonces para evitar la confusión durante el cálculo de la apertura del ojo, siempre cogeremos la medida del mico más central. La siguiente figura muestra ejemplos de coger medidas con thresholds horizontales diferentes:

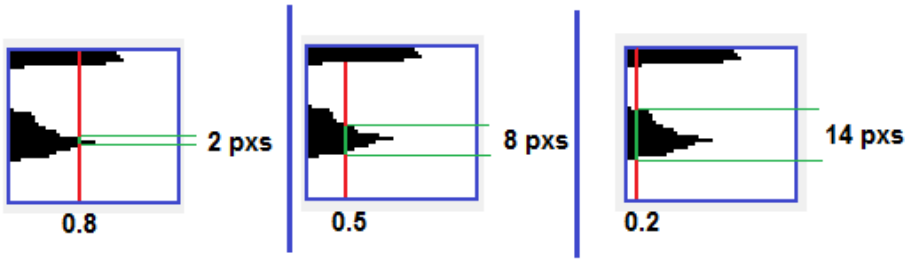


Figura 26: Coger medidas de la apertura con diferentes thresholds.

El threshold horizontal es un porcentaje del pico más central, es decir, si cogemos 0.5 estamos cogiendo la medida del alto en la mitad del pico. Como vemos con la figura anterior, cogiendo thresholds diferentes resulta en medidas diferentes. El problema aquí es que estamos midiendo pixeles, y sabes que depende de la distancia entre el ojo y la cámara, podemos tener varias medidas en pixeles para la misma apertura del ojo. Entonces, habrá que transformar esta medida a un porcentaje independiente del tamaño del ojo y la distancia entre el conductor y la cámara.

Para transformar los pixeles a un porcentaje, podemos proyectar la imagen binaria verticalmente (Figura 27), y como hemos hecho antes, usar un threshold vertical para coger la medida del ancho del área del ojo, luego dividimos la medida de los pixeles verticales sobre la medida de los pixeles horizontales del ojo, que nos dará como resultado el porcentaje de la apertura del ojo (Figura 28).

Figura 27: Proyección vertical.

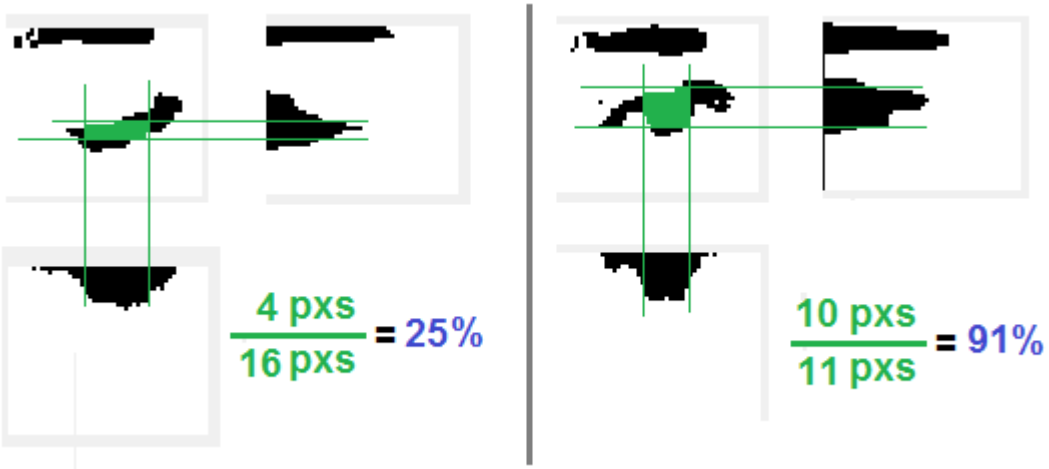


Figura 28: Calcular el porcentaje de apertura.

Ahora necesitamos determinar los valores de: el threshold de ecualización para obtener la imagen binaria, el treshold horizontal y el threshold vertical. Para esto, hemos recortado imágenes de ojos desde el dataset con caras frontales, para crear un nuevo data set de ojos que están etiquetados si están abiertos o cerrados. Luego hemos calculado la mediana el porcentaje de apertura de las imágenes abiertas y cerradas, variando los 3 thresholds en cuestión. La siguiente figura de muestra algunos resultados:

Threshold	Thr. Horizontal	Thr. Vertical	% Ojo Cerrado	% Ojo Abierto
0.15	0.5	0.3	27.27	32.14
0.15	0.5	0.5	40.90	52.94
0.15	0.5	0.8	64.28	90.00
0.15	0.3	0.8	57.14	95.00
0.15	0.3	0.5	36.36	70.58
0.15	0.3	0.3	24.24	42.85
0.10	0.5	0.3	47.62	38.09
0.10	0.5	0.5	62.50	61.53
0.10	0.5	0.8	96.03	95.20
0.10	0.3	0.8	70.00	94.06
0.10	0.3	0.5	43.75	69.23
0.10	0.3	0.3	33.33	42.85

Figura 29: Algunos resultados de variar los 3 Thresholds.

Como vemos la mejor combinación de valores que nos permite destinguir entre ojo abierto y ojo cerrado es: Threshold = 0.15, Threshold Horizontal = 0.3 y Threshold Vertical = 0.8.

3.5.1.b Implementación

Después de implementar un flujo directo, hemos añadido una fase opcional de dilatación sobre la imagen binaria, que esta explicado más abajo en resultados. Con esto la implementación resultante es (Figura 30):

(1) Recibimos como entrada la imagen del ojo en escala de gris.

(2) Ecualizamos el histograma de la imagen.

(3) Seleccionamos los pixeles que son menores que el threshold.

(4) Es una fase que hemos añadido más adelante para disminuir el ruido en las imágenes binarias. Es opcional porque en algunos casos no hace falta aplicarla.

(5) Proyectamos la imagen binaria horizontalmente, y como hemos explicado antes, obtenemos una medida de la altura del ojo en pixeles.

(6) Proyectamos la imagen binaria horizontalmente, y como hemos explicado antes, obtenemos una medida de la anchura del ojo en pixeles.

(7) Aquí, dividimos las medidas obtenidas con las proyecciones anteriores para obtener el estado del ojo.

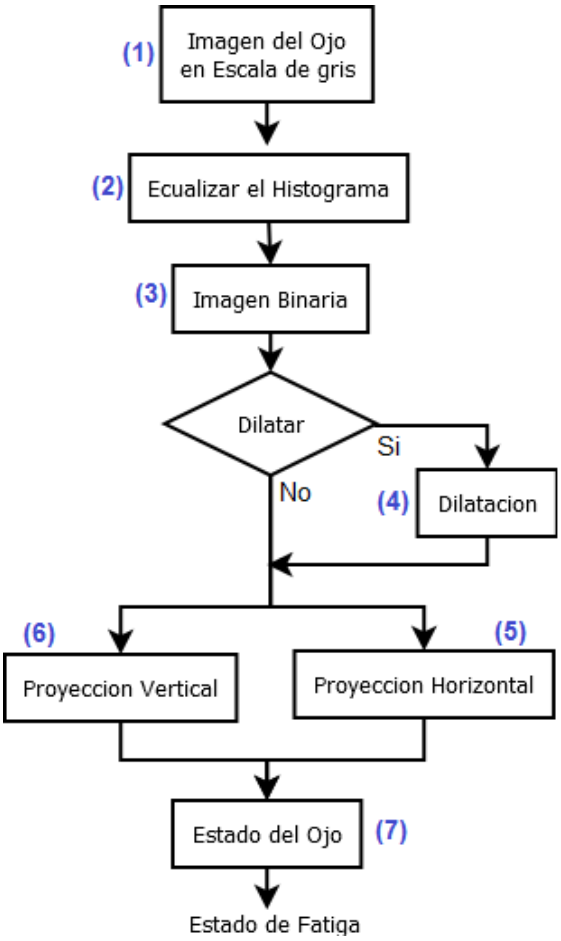


Figura 30: El flujo de calcular el estado del ojo usando proyecciones.

3.5.1.c Resultados

Al principio hemos implementado el flujo en la figura anterior sin la dilatación, al testear esta implementación a tiempo real con capturas de cámara (a 30 FPS), hemos notado que la estimación de la apertura del ojo era errónea. Esto sucede a veces cuando tenemos pixeles ruidosos después de obtener la imagen binaria (Figura 31 - a), por eso, hemos decidido de probar la dilatación (Figura 31 - b).

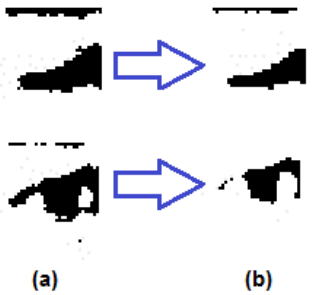


Figura 31: Efecto de la dilatación sobre la imagen binaria.

Después de incorporar la dilatación de la imagen binaria en el flujo, hemos notado una mejora en la estimación de la apertura del ojo a tiempo real.

3.5.2 SVM (con PCA)

Uno de los métodos que hemos probado es la clasificación con SVM, con una imagen del ojo en escala de gris queremos obtener el estado del ojo, abierto o cerrado. Primero entrenamos un SVM con dos clases: ojo abierto e ojo cerrado.

3.5.2.a Entrenamiento

Para entrenar el SVM, hemos usado el dataset de los ojos que hemos mencionado en la sección anterior (6.3.1), primero debemos transformar estas imágenes a vectores siguiendo el siguiente procedimiento para cada imagen:

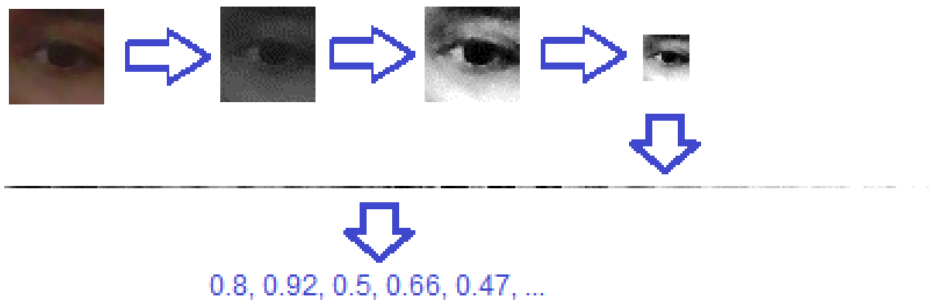


Figura 32: Convertir una imagen a un vector de números.

1) Convertir laimagen a escala de gris.
2) Ecualizar el histograma de la imagen en gris.
3) Escalara la imagen ecualizada a un tamaño fijo: 20x20.
4) Concatenar las filas de la imagen anterior en una fila: 1x400.
5) Normalizar los valores de la fila entre 0 e 1.

Al final obtenemos un data set numérico de todas las imágenes (Figura 33), donde cada muestra tiene 400 características.

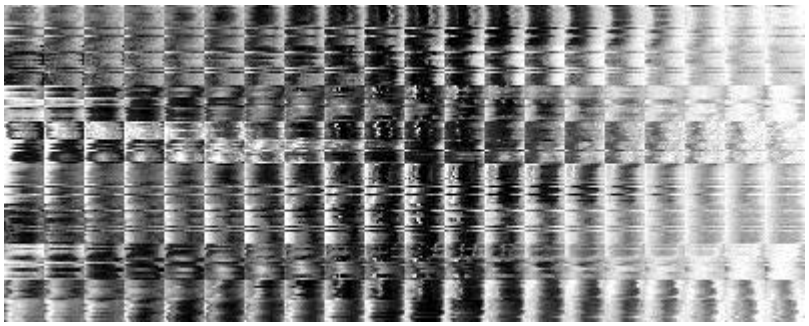


Figura 33: Data set de imágenes en filas.

3.5.2.b PCA

Con el PCA, obtenemos características más discriminantes del dataset anterior, y con una dimensión reducida. Para usar el PCA en OpenCV, primero se crea un objeto PCA con el dataset original indicando cuantos componentes principales queremos. Con este objeto podemos transformar una muestra en la dimensión del dataset original (400) a la nueva dimensión.

3.5.2.c Implementación

Aquí veremos el flujo (Figura 34) que hemos implementado en sistema final para clasificar el estado del ojo, el flujo tienen los pasos necesarios para transformar la imagen del ojo a un vector de números para clasificar el estado del ojo:

(1) Como entrada tenemos la imagen del ojo en gris, que hemos obtenido en el seguimiento.

(2) Ecualizamos el histograma de la imagen en gris.

(3) Re-escalar la imagen a un tamaño fijo: 20x20.

(4) Concatenamos la imagen escalada en filas: 1x400.

(5) Normalizamos la fila anterior entre 0 e 1.

(6) Transformamos la muestra con 400 características, a la dimensión que hemos obtenido con el PCA. En la sección de resultados, hemos mencionado algunos resultados de variar el número de los componentes principales.

(7) Finalmente, usamos el clasificador entrenado para predecir a que clase pertenece la imagen del ojo (cerrado o abierto).

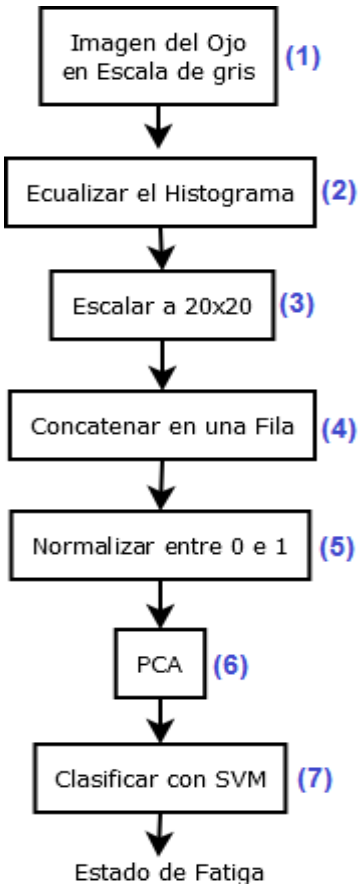


Figura 34: Flujo de determinar el estado de ojo con SVM y PCA.

3.5.2.d Resultados

Hemos divido el dataset de los ojos entre dos conjuntos, un conjunto de entrenamiento y otro conjunto de validación, luego hemos obtenido el PCA del primero conjunto para entrenar un SVM, finalmente testeado el SVM con el conjunto de validación. La siguiente figura muestra algunos resultados que hemos obtenido variando el número de los componentes principales:

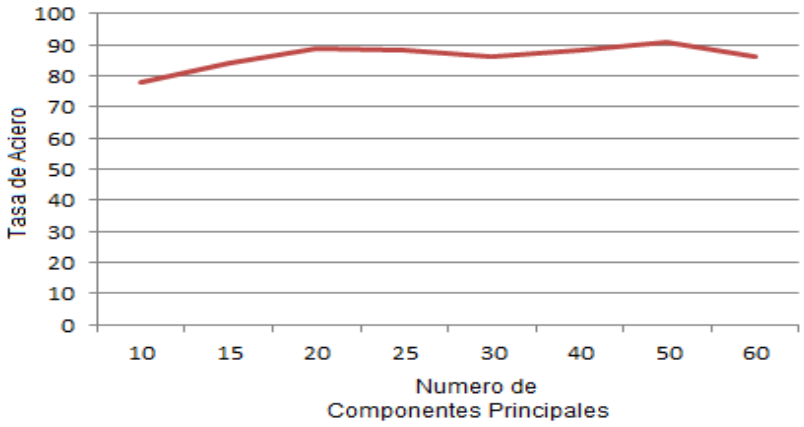


Figura 35: Tasas de acierto de SVM.

Como vemos la tasa de acierto es bastante alta, especialmente cuando escogemos 20 componentes principales (88%) o 50 (91%).

3.6 Cuantificar la Fatiga

Una vez que llegamos a esta fase, procedemos a actualizamos el estado de la fatiga basándonos en el algoritmo de Perclos, como hemos explicado en la sección 2.2.5, este algoritmo determina el estado de fatiga basándose en la variación de la apertura del ojo durante un periodo de tiempo, entonces necesitamos guardar los estados del ojo durante una duración determinada.

La figura 10 muestra un ejemplo del Perclos en tiempo continuo, mientras que con un sistema a tiempo real, tendremos valores del estado del ojo en tiempo discreto. Entonces necesitamos guardara dos datos cada vez que entramos en esta fase, el estado del ojo, y la instancia del tiempo del estado en milisegundos.

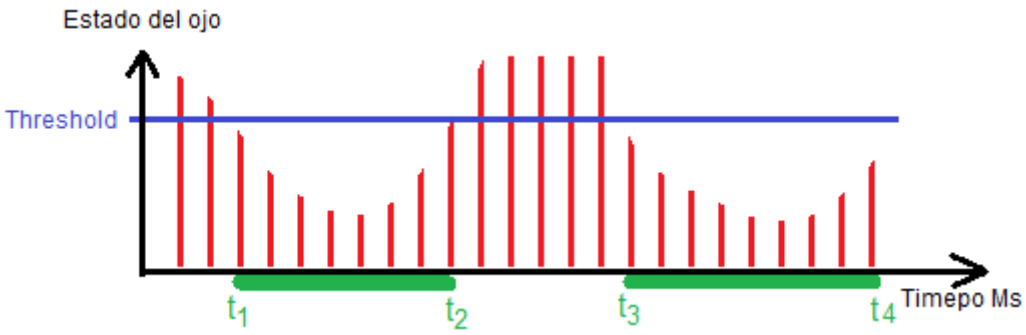


Figura 36: Ejemplo de calcular Perclos en tiempo discreto.

3.6.1 Implementación

En la implementación final, tenemos un vector donde gradamos cronológicamente: el estado del ojo, tiempo del estado y un valor booleano que nos indica si es menor que el threshold del Perclos. Como resultado, la detección de fatiga sigue el siguiente flujo:

(1) Depende de que método hemos usado en la fase anterior, el estado del ojo puede ser, el porcentaje de apertura del ojo, o un valor binario 0 u 1.

(2) Como hemos mencionado antes, necesitamos guardar el estado del ojo, el tiempo actual y si es el estado es menor que el threshold determinado, aquí usamos un threshold de 0.8.

(3) Borramos las instancias que ya no están dentro del periodo que hemos establecido.

(4) Calculamos el valor del Perclos como hemos explicado en la Figura 36.

(5) Finalmente, el estado de fatiga se determina a partir del Perclos, así cuantificamos la fatiga entre 0 y 1 (0% y 100%).

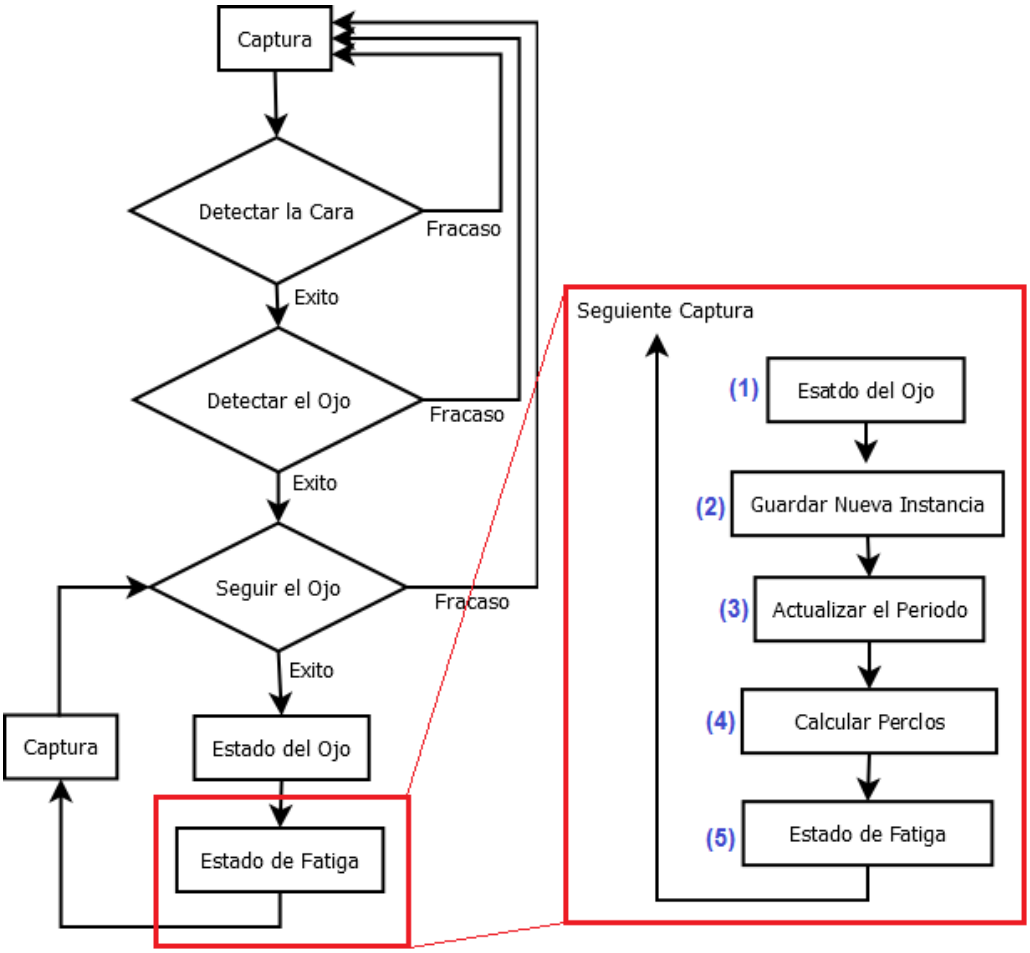


Figura 37: El flujo de cuantificar la fatiga.

3.6.2 Resultados

Para testear el algoritmo hemos cogido 3 conjuntos del estado del ojo (0s e 1s), el primero conjunto simula un ojo abierto (1s), el segundo simula un el parpadeo del ojo (alterar entre 0s e 1s), e el último simula un ojo cerrado (0s). Fijando un threshold a 0.8, y variando el periodo de Perclos, en la siguiente figura mostramos los resultados:

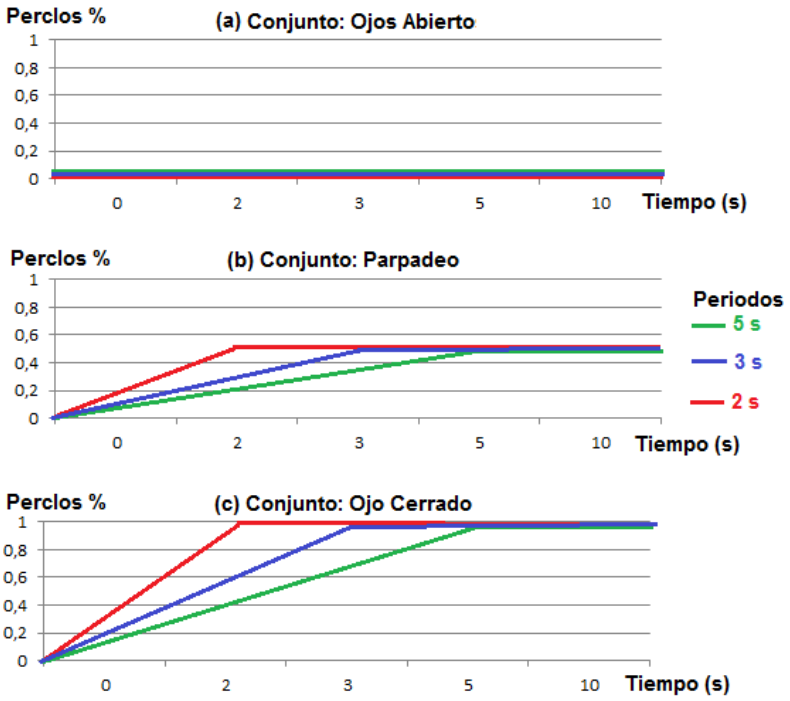


Figura 38: Probar el efecto del periodo del Perclos.

Como vemos en la gráfica del conjunto de ojos abiertos (Figura 38 - a), el perclos no varía del 0%, mientras que con el resto de los conjuntos el Perclos empezara a subir hasta un valor máximo (50% en caso de parpadeo, y 100% en caso de ojo cerrado).

En las dos últimas gráficas, notamos que la variación del periodo afecta a la velocidad del aumento del perclos, entonces efecta el tiempo de respuesta del sistema en tiempo real.

3.7 Sistema Final

El resultado final es un software que lleva dentro un sistema a tiempo real, este sistema está basado en el flujo principal que hemos mencionado en el mostrado en la Figura 1. En el anexo 2 esta explicado el diseño del sistema. La GUI del software permite una interacción ágil con el sistema, que nos permite cambiar algunos parámetros de las fases implementadas, más detalles sobre la GUI están en el anexo 3.

Al lanzar el software, el sistema estará en modo parado, cuando se aprieta el botón Arrancar, el sistema empieza a coger las capturas a una frecuencia de 30 FPS, al principio se notara unos saltos de frames, esto

pasa en la detección de la cara y el ojo, debido a que el clasificador en cascada necesita mucho computo. Una vez que se encuentre el ojo, el sistema entra en estado de seguimiento, mientras que el seguimiento no falle, se procesa el estado del ojo, con locual se actualiza el estado de fatiga calculando el preclos.

El sistema dispone de un logger (anexo 4), que registra los pasos intermedios de cada flujo, así podemos analizar los errores que suceden a tiempo real como el fallo de detección o seguimiento, o valores erróneos de la apertura del ojo.

Un sistema de detección de fatiga debe tener un método de alerta (luz, sonido,…). Hemos simulado la alerta cambiando el color de un rectángulo que se sitúa la parte inferior de la GUI. Nuestro sistema tiene 3 niveles de alerta según el valor del Perclos, y cada nivel se indica con un color diferente, como se muestra en las siguientes figuras (39-41):

- Verde: Perclos< 50%.
- Naranja: Perclos entre 50% y 80%.
- Rojo: Perclos> 80%.

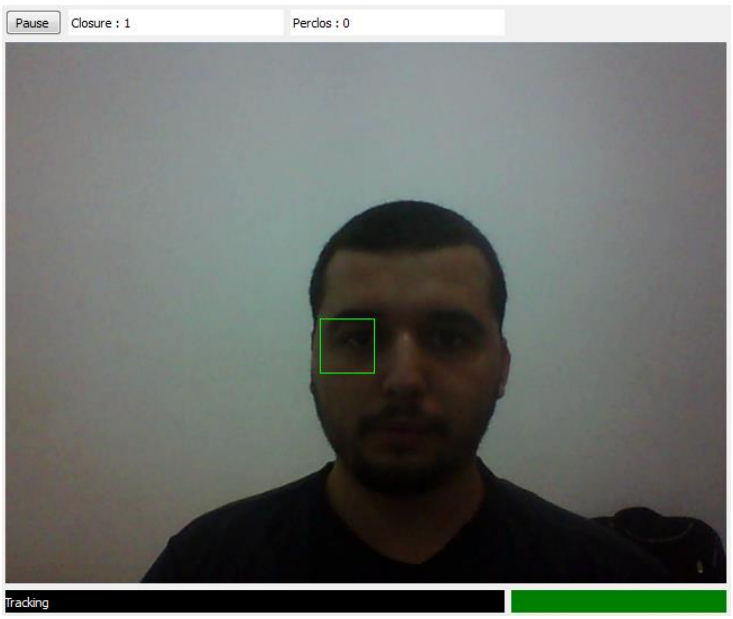


Figura 39: No hay Alerta

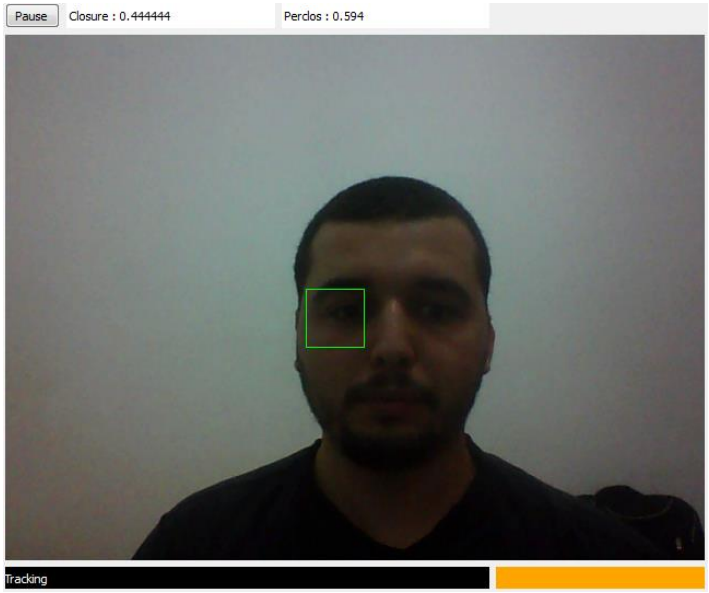


Figura 40: La fatiga es superior al 50%

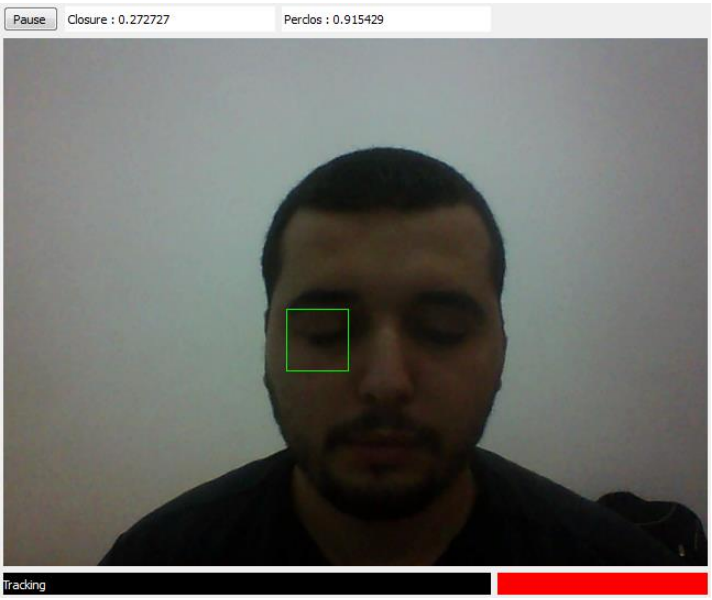


Figura 41: La fatiga alcanza el 90%

Capítulo 4

Conclusiones

4.1 Conclusión

En primer lugar hemos estudiado técnicas que se usan actualmente en visión por computador, luego hemos creado un ground truth para testear y evaluar estas técnicas. Después de estudiar los artículos mencionados al principio [9-13] hemos decidido basar la fatiga sobre el parpadeo de los ojos, como consecuencia, teníamos que analizar las características faciales que nos permite medir el parpadeo.

Seguidamente hemos pasado a diseñar el flujo principal del sistema, que nos permite obtener el estado de fatiga desde unas capturas de imágenes, pasando por unas fases: La primera era la detección de la cara dentro de la imagen, luego detectar el ojo dentro del área de la cara, una vez obtenida la imagen del ojo, procedemos a seguir el ojo en un área reducida, después obtenemos el estado del ojo, finalmente, podemos calcular el estado de fatiga.

En cada fase del flujo principal, se ha implementado la metodología más adecuada. Se ha usado el clasificador en cascadas de OpnCV para detectar la cara y luego el ojo, y para ahorrar tiempo de computo usamos el Template Matching para seguir una plantilla del ojo detectado, para seguir el ojo en las siguientes capturas.

Para determinar el estado del ojo hemos experimentado con varias técnicas, al final hemos optado con dos métodos: Proyección Horizontal y Vertical que nos devuelve el porcentaje de apertura, y Clasificación con SVM con PCA que nos determina si el ojo está cerrado o abierto. En cuanto a la cuantificación de la fatiga, se ha usado el Perclos que nos devuelve el porcentaje de la fatiga.

Una vez que se ha implementado una fase hemos psasdo a probar su tasa de acierto, hemos tenido tasas muy bajas con el ground-trith original, pero se ha conseguido una mejora con otro conjunto de imágenes con caras frontales.

Una vez implementado y testado cada componente del flujo principal, se ha creado un software a tiempo real, que lleva dentro la implementación del flujo principal. La una GUI que nos permite ajustar los parámetros de cada fase. Además el programa puede depurara los procesos de cada flujo para analizar casos de errores o fallos.

Se puede usar el flujo principal del sistema como una plantilla de un sistema de detección de fatiga, cambiando la metodología usada en cada fase, o ampliar la con otro método.

4.2 Mejoras

Se pude mejorar la detección de las características del ojo, entrenado el clasificador a tiempo real, usado muestras de la cara y los ojos del conductor, en este caso el conductor se sitúa frente de la cámara con sus ojos abiertos durante un tiempo determinado, para conseguir sus características faciales.

Como hemos mencionado, algunos coches usan sistemas de detección de fatiga basándose en algunos sensores que detectan: los giros del volante, la desviación en de las carreteras. Podemos incorporar estos señales el sistema para mejorar la calidad de la detección de fatiga.

Capítulo 5

Anexos

Anexo 1. Métodos descartados para determinar el estado del ojo

Anexo 1.1 Template Matching

Queríamos aprovechar el valor de similitud que nos devuelve el TemplateMatching en determinar el estado del ojo. Para probar esta idea, hemos cogido cuatro plantillas de ojos abiertos, que pertenecen a cuatro personas diferentes en el ground-truth original, y luego para cada persona hemos cogido 2 muestras de ojos cerrados y 2 muestras de ojos abiertos.

Las siguientes figuras muestran los resultados usando la difrenecia entre la plantilla y los ojos como criterio de similitud (Figura 42) y luego usando la correlación (Figura 43):

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Sample(D)	Eye	Close	Open
01	Left	0.0372044	0.0344777
	Right	0.0587593	0.022411
02	Left	0.19739	0.0816539
	Right	0.229846	0.291502
03	Left	0.0349566	0.179983
	Right	0.0225668	0.160493
04	Left	0.0495955	0.0516645
	Right	0.0219282	0.0628581

(a) Usar Escala de Gris

Sample(H)	Eye	Close	Open
01	Left	0.0896902	0.120733
	Right	0.124596	0.0775563
02	Left	0.217308	0.208612
	Right	0.324592	0.337023
03	Left	0.204801	0.223405
	Right	0.253793	0.197148
04	Left	0.13066	0.180336
	Right	0.409942	0.426784

(b) Ecualizar el Histograma

Figura 42: Resultados del Template Matching con la diferencia

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Sample(D)	Eye	Close	Open
01	Left	0.991487	0.990348
	Right	0.989049	0.992179
02	Left	0.971529	0.963616
	Right	0.98227	0.98014
03	Left	0.989406	0.971434
	Right	0.989911	0.99451
04	Left	0.981422	0.974976
	Right	0.989063	0.968647

(a) Usar Escala de Gris

Sample(H)	Eye	Close	Open
01	Left	0.955703	0.958981
	Right	0.9382	0.962284
02	Left	0.893903	0.908778
	Right	0.841537	0.845598
03	Left	0.900646	0.888461
	Right	0.873124	0.905241
04	Left	0.937431	0.91912
	Right	0.871403	0.800681

(b) Ecualizar el Histograma

Figura 43: Resultados del Template Matching con la correlación

Como vemos en los resultados, el Templatematching no discrimina mucho entre un ojo cerrado y un ojo abierto, aunque ecualicemos el histograma, con lo cual no podemos usar este metodopara obtener el estado del ojo. Pero esto nos sirve para seguir el ojo en tiempo real, aun que cambie de estado.

Anexo 1.2 Canny

Unos de los enfoques que hemos probado era detectar los contornos del ojo para transformar una imagen del ojo a una imagen binaria, luego proyectar esta última horizontalmente para medir la distancia entre los parpados. Para detectar los contornos, hemos usado el algoritmo del Canny, que esta basad calcular el gradiente de la imagen, y luego usando dos thresholds, se detectan los contornos. La siguiente figura muestra los resultados de las pruebas que hemos hecho, variando los valores de los dos thresholds:

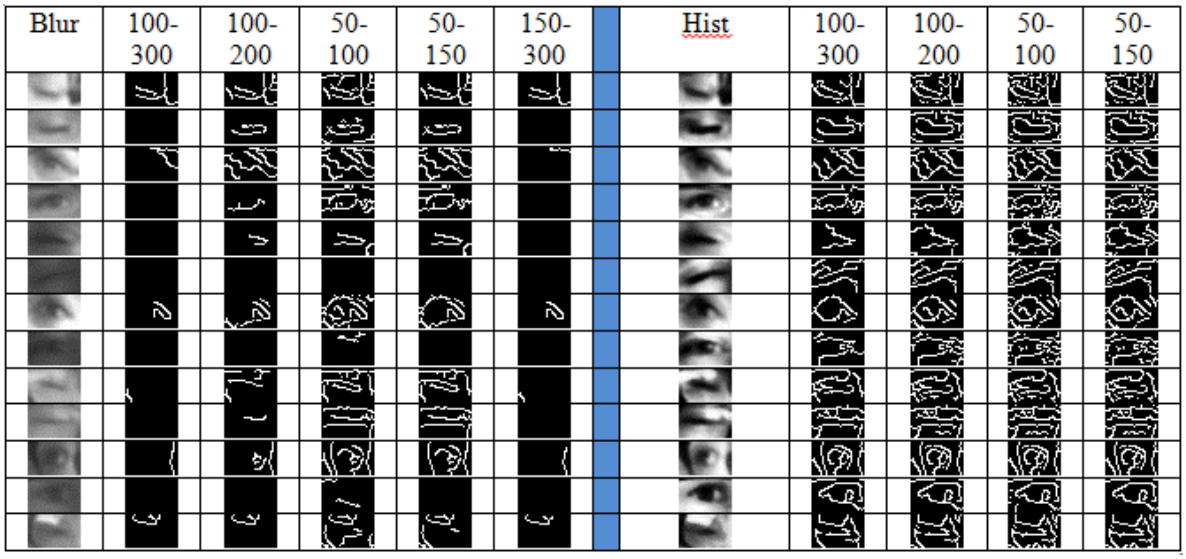


Figura 44: Resultados de aplicar el Canny
(Derecha sin ecualizar, izquierda con la ecualización)

Como vemos, en las los contornos detectados, no tiene una forma coherente que nos permite medir la apertura del ojo, por esto, hemos descartado este enfoque.

Anexo 1.3 Saturación HSI

Otro enfoque que teníamos en cuenta era usar un método propuesto en [12], primero se convierte la imagen en RGB a modelo de colores HSI, luego se seleccionan algunos pixeles que tienen una saturación menor que un threshold.

La siguiente figura muestra algunas pruebas que hemos hecho con la saturación, variando el threshold. A la derecha son resultados después de ecualizar el histograma de la saturación, y a la izquierda son resultados de normalizar la saturación.

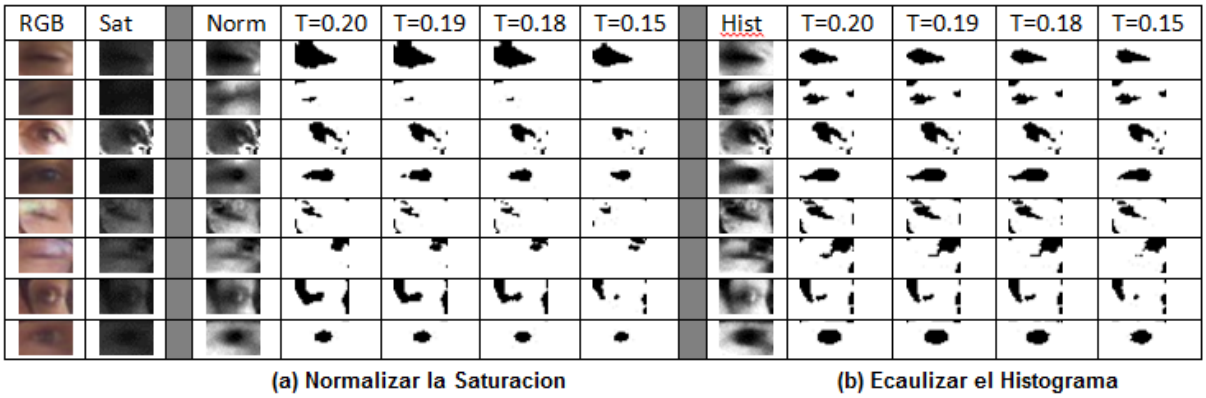


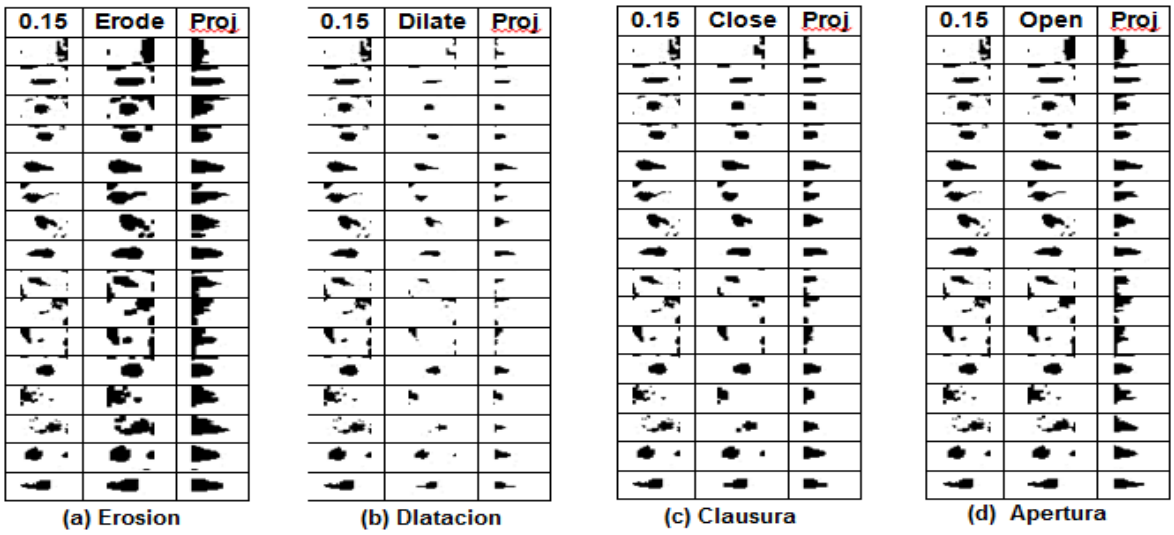
Figura 45: Determinar el área del ojo con la saturación HSI

Como vemos, la mayoría de los resultados no son válidos para determinar el estado del ojo, esto es debido a que el ruido desde las imágenes en RGB causa errores en calcular la saturación en el modelo HSI. Entonces hemos descartado este método para detectar el estado del ojo.

Anexo 1.4 Morfología

En la fase de detección de la apertura del ojo con la proyección horizontal y vertical, hemos notado que a veces obtenemos imágenes binarias con ruido, como pixeles adicionales. Para reducir este ruido, hemos optado con las operaciones morfológicas: Erosión, Dilatación, Clausura y Apertura.

La siguiente figura muestra los resultados de aplicar las cuatro operaciones sobre una muestra de imágenes binarias, y luego proyectar el resultado horizontalmente.



46: Resultados de las operaciones morfológicas, con su proyección.

Como vemos las operaciones no mejora mucho los resultados, pero hemos observado un efecto de la dilatación que nos ha servido más adelante en el proyecto. Este efecto es la reducción de la altura de las imágenes binarias de los ojos cerrados.

Anexo 2. Diseño del Sistema (Diagrama de Clases)

El sistema fue diseñado como una estructura jerárquica, donde cara clase representa un componente del flujo principal del sistema, así, cualquier cambio en el diseño se puede aplicar con facilidad.

La siguiente figura muestra el diagrama de clases

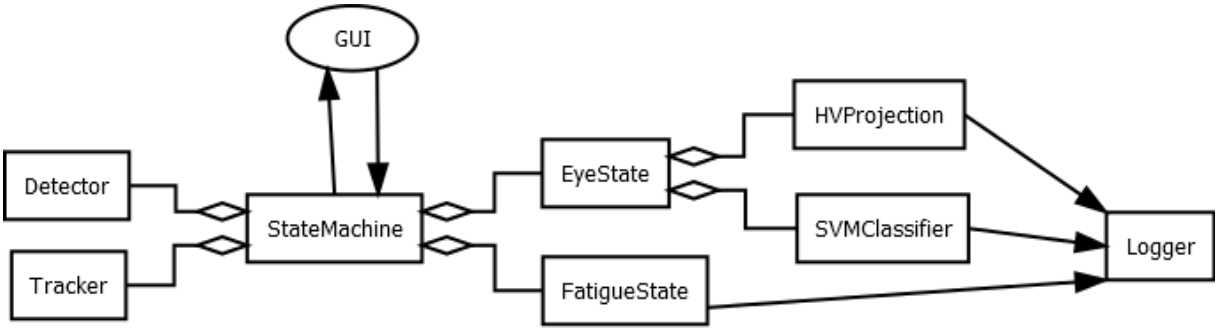


Figura 47: Diagram de Clases del Sistema

- StateMachine: esta clase representa el flujo principal del sistema, desde la GUI recibe mensajes como: la captura para determinar la fatiga, cambio de configuraciones o parámetros y mensaje de inicio o pausa. Como salida, envía a la GUI: el estado del ojo, el estado de la fatiga y si está en la fase de detección o seguimiento.

- Detector: tenemos dos instancias de esta clase en el StateMachine, una para detectar la cara y la otra para detectar el ojo. Pero en general se encarga de detectar un objeto (cara u ojo) dentro de una imagen.

- Tracker: la clase responsable del seguimiento del ojo.

- EyeState: esta clase contiene dos clases, uno para obtener el estado del ojo con el clasificador SVM y otro para obtener el porcentaje de apertura aplicando la proyección.

- HVProjection: la clase responsable de calcular la apertura del ojo, con proyecciones.

- SVMClassifier: esta clase facilita la clasificación con SVM con PCA. Además se entrena el clasificador al iniciar el programa

- FatigueState: en esta clase se guarda el estado del ojo durante un periodo para calcular el Perclos.

- Logger: un singleton, para guardar datos durante la ejecución del programa, recibe estos datos desde el HVProjeccion, SVMClassifier y FatigueState.

Anexo 3. GUI

La GUI nos permite arrancar el sistema, y configurar algunos parámetros. Con las siguientes figuras indicamos el uso de la GUI:

1) GUI Principal:

Figura 48: GUI Principal

(1) El botón "Run" nos permite iniciar/pausar el sistema.
(2) Indica la apertura del ojo.
(3) Indica el valor del Perclos.
(4) Muestra la captura actual en color.
(5) Indica el estado actual del sistema.
(6) Es la alerta visual (verde, rojo y naranja).
(7) File: Nos permite elegir la fuente de la captura.

2) Menu File:

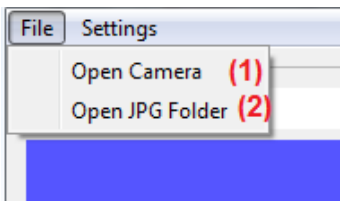


Figura 49: GUI - Opciones de Captura

(1) Indica que las capturas serán desde la cámara.
(2) Indica que las capturas estarán en una carpeta de imágenes en formato JPG. Luego sale un dialogo para navegar a la ubicación de la carpeta.

3) Menu de Configuraciones:

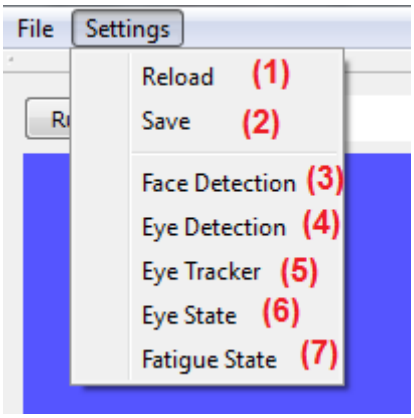


Figura 50: GUI - Opciones de configuración.

(1) Recargamos el fichero de configuración.
(2) Guardamos las configuraciones actuales en el fichero de configuración.
(3) Abre un dialogo para configurar la detección de la cara.
(4) Abre un dialogo para configurar la detección del ojo.
(5) Abre un dialogo para configurar el seguimiento del ojo.
(6) Abre un dialogo para configurar el estado del ojo.
(7) Abre un dialogo para configurar la cuantificación del ojo.

4) Configurar la detección de la cara:

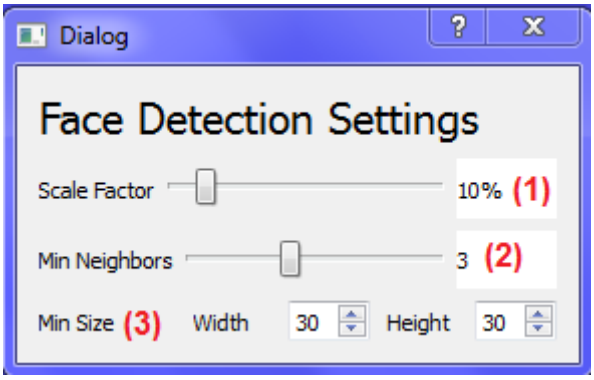


Figura 51: GUI - Configuración de la detección de la cara

(1) Nos Permite cambiar el factor de escala del clasificador en cascada.

(2) Nos Permite cambiar el valor de los vecinos mínimos del clasificador en cascada.
(3) Para Cambiar el tamaño mínimo del área de la cara en pixeles.

5) Configurar la detección del ojo:

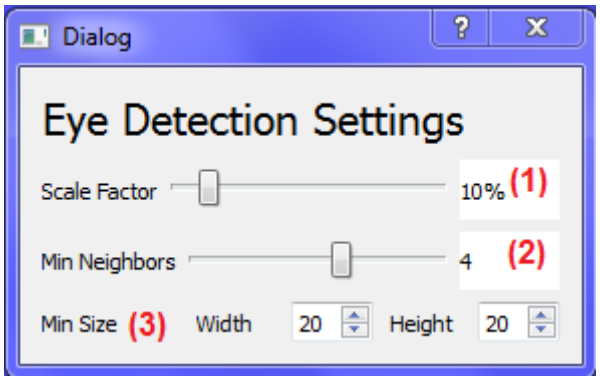


Figura 52: GUI - Configuración de la detección del ojo

(1) Nos Permite cambiar el factor de escala del clasificador en cascada.
(2) Nos Permite cambiar el valor de los vecinos mínimos del clasificador en cascada.
(3) Para Cambiar el tamaño mínimo del área del ojo en pixeles.

6) Configuración del estado del ojo:

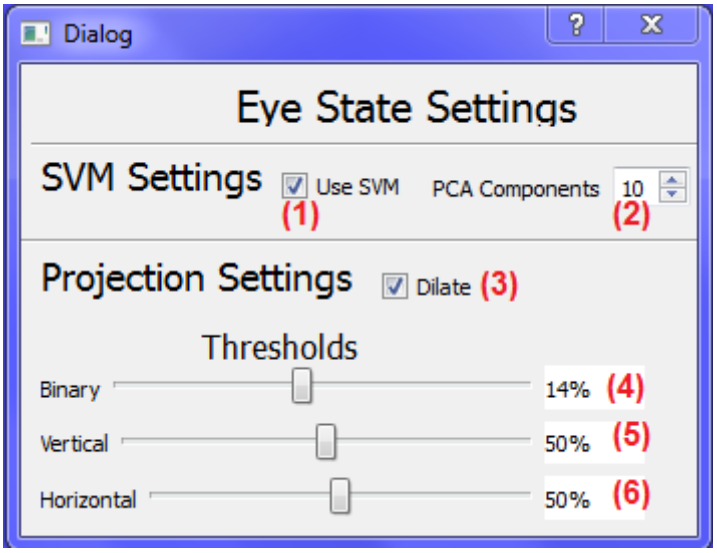
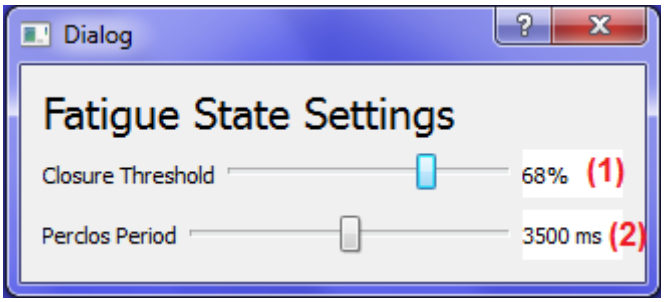


Figura 53: GUI - Configuración del estado del ojo

(1) Indica si queremos usar el calificador SVM o la proyección.
(2) Para Indicar cuantos componentes principales escogemos para clasificar con SVM.

(3) Indica si queremos usar la dilatación después de obtener la imagen binaria.
(4) No permite indicar el valor del treshold para obtener la imagen binaria.
(5) Indica el valor del threshold de la proyección vertical.
(6) Indica el valor del threshold de la proyección horizontal.

7) Configurar los parámetros del Perclos:



Fatiga 54: GUI - Configuración del Perclos

(1) Indica el threshold del Perclos.
(2) Indica el periodo del Perclos.

8) Fichero de configuración:

La siguiente figura muestra el fichero de configuración en formato JSON.


```
"EyeDetection": {
    "MinHeight": 20,
    "MinNeighbors": 4,
    "MinWidth": 20,
    "ScaleFactor": 1.1000000000000001
},
"EyeState": {
    "SVM": true,
    "SVMClassifier": {
        "Log": true,
        "MaxComponents": 10
    },
    "VHProjector": {
        "Dilate": true,
        "HorThreshold": 0.5,
        "Log": false,
        "Threshold": 0.14000000000000001,
        "VerThreshold": 0.5
    }
},
"FaceDetection": {
    "MinHeight": 30,
    "MinNeighbors": 3,
    "MinWidth": 30,
    "ScaleFactor": 1.1000000000000001
},
"FatigueState": {
    "Log": false,
    "Period": 3500,
    "Threshold": 0.68000000000000005
},
"Tracking": {
    "Scale": 1.5,
    "Threshold": 0.20000000000000001
}
```


Figura 55: GUI - Fichero de Configuración.

Anexo 4. Logger

El sistema lleva un singleton, que gurda resultados intermedios de cada fase, para activar el logger. El fichero de configuración nos permite indicar si queremos guardar los datos del Perclos, clasificación SVM o la Proyección. El logger crea un fichero con los resultados numéricos, y crea una carpeta con imágenes resultantes de procesar el ojo en la fase de detección del estado del ojo.

La siguiente figura, muestra las imagen creadas por el logger, todas la imagen empiezan con números que indican el tiempo en que se empezó a procesar la imagen del ojo, seguido del tipo de la imagen (color, gris, ecualizada…), las siguientes figuras muestran dos ejemplos que ha guardado el logger:

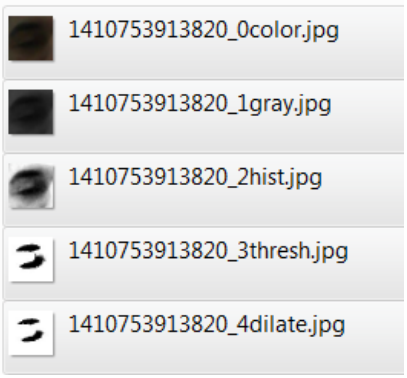


Figura 56: Resultados intermedios el ojo antes de aplicar la proyección.

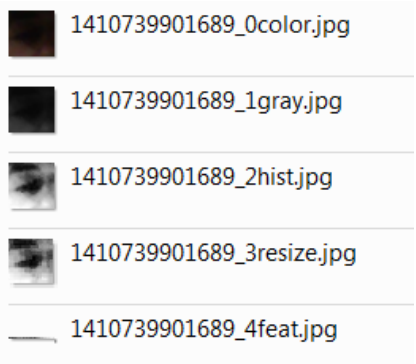


Figura 57: Resultados intermedios del ojo antes de aplicar el PCA.

Referencias

[1] La fatiga y el alcohol, principales causas de accidente, Publicado por Irene Mendoza, 02 Ene 2013.

[2] World Health Organization, Global Status Report On Road Safty 2013.

[3] MedlinePlus, Institutos Nacionales de la Salud, Articulo sobre la Fatiga, 21 Abr 2013.

[4] Julie Halpert, Drowsiness Detection Systems, Jean Knows Cars, Cool Tech, Tech News, 8 Oct 2012.

[5] Wing Teng, Robust Real-Time Face Detection by Paul Viola & Michael Jones, 27 Sep 2007.

[6] Paul Viola, Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001.

[7] Robert E. Schapire, Princeton University, Dept. of Computer Science, Explaining AdaBoost.

[8] OpenCV Documentaion, Machine Learning Module, Introduction to SVM.

[9] Mohamad-Hoseyn S., Mahmood F., Mohsen S., A Driver Face Monitoring System for Fatigue and Distraction Detection, International Journal of Vehicular Technology, Volume 2013, Article ID 263983, November 2012.

[10] Yong D., Peijun M., Xiahong S., Yingjun Z., Driver Fatigue Detection based on Eye State Analysis, Department of Technology, Harbin Institute of Technology.

[11] Saeid F., ParisaEsfehani, Tracking Eye State for Fatigue Detection, ICACEE'2012, 17-18, November 2012

[12] Wen-Bing Horng, Chih-Yuan Chen, A Real-Time Driver Fatigue Detection System Based on Eye Tracking and Dynamic Template Matching, Tamkang Journal of Science and Engineering, Vol. 11, No. 1, pp. 65-72 (2008).

[13] Wu Qing, Sun BingXi, Xie Bin, Zhao Junjie, A PERCLOS-Based Driver Fatigue Recognition Application for Smart

Vehicle Space, Information Processing (ISIP), Third International Symposium on 2010, 437-441, 15-17 Oct 2010.

[14] OpenCVDocumentaion, Image Processing Module, Histogram Equalization.

[15] RACC, Cifras, La fatiga: presente en un 20% a 30% de accidentes, Junio 2013.

[16] OpenCVDocumentaion, Image Processing Module, Erosion and Dilatation.

[17] Mostafiz Hossain, OpenCv face detection procedure and Explanation, 12 Aug 2013.

[18] OpenCV Documentation, Object Detection Module, Cascade Classifiaction, Detect Multi Scale.

[19] Catherine Shu,Bluetooth Headset Vigo Knows When You Are Tired Before You Do, TechCrunch, 17 Ene 2014.

[20]George Dallas, Principal Component Analysis 4 Dummies, UK based Information Engineer/Internet Social Scientist, WordPress, 30 Oct 2013.