ESCUELA TÉCNICA SUPERIOR DE INGENIEROS

INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN,
ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

DESARROLLO DE APLICACIONES DE DETECCIÓN DE OJOS BASADAS EN ALGORITMOS DE DETECCIÓN FACIAL Y TRACKING

José Javier Bengoechea Irañeta

Arantzazu Villanueva Larre

Pamplona, 26 de junio de 2012

AGRADECIMIENTOS

Gracias especialmente a Arantxa, mi tutora en este proyecto, por todo lo que me ha ayudado desde el primer día, por el tiempo dedicado a las grabaciones y por abrirme las puertas a un campo que desconocía completamente.

Gracias a Victoria, Mikel y Rafael, por el tiempo que pasamos grabando los vídeos que tantas veces he visto y por sus ideas en las reuniones.

Gracias a mis compañeros de carrera, entre todos habéis hecho las cosas más fáciles, especialmente Diego, nuestro Lenny, capaz de conseguir exámenes que en teoría no existían.

Gracias a mis compañeros de teleco superior, que aunque no estamos juntos en clase, seguimos pasando buenos ratos en la biblioteca.

Gracias a mi familia, por no desesperar tras tantos años de altibajos que parecían no tener fin, y que ahora no quiero que terminen.

ÍNDICE

1 – INTRODUCCIÓN

El concepto de eye-tracking [1][2] hace referencia a un conjunto de tecnologías que permiten monitorizar y registrar la forma en la que una persona mira una determinada escena o imagen, en concreto en qué áreas fija su atención, durante cuánto tiempo y qué orden sigue en su exploración visual.

Las técnicas de eye-tracking tienen un gran potencial de aplicación en una amplia variedad de disciplinas y áreas de estudio, desde el marketing y la publicidad hasta la investigación médica o la psicolingüística, pasando por los estudios de usabilidad.

Existe una gran variedad tecnológica de sistemas de eye-tracking, cada uno con sus propias ventajas e inconvenientes. Una de las técnicas más precisas implica el contacto físico con el ojo a través de un mecanismo basado en lentes de contacto, pero inevitablemente estos sistemas resultan muy incómodos para los participantes de la prueba. La mayoría de sistemas actuales son mucho menos molestos, ya que se basan en el uso de cámaras (eye-trackers) que proyectan rayos infrarrojos hacia los ojos del participante, sin necesidad de contacto físico.

Los sistemas basados en rayos infrarrojos se limitan a entornos de laboratorio, donde la iluminación artificial es controlada para obtener una iluminación óptima y donde la fuente de rayos infrarrojos es la única fuente de luz infrarroja. Además para esta tecnología se utilizan unas cámaras y unos ordenadores con capacidad de procesado superiores a la media.

La intención de los investigadores es crear un sistema de eye-tracking para imágenes tomadas con una webcam en un entorno de iluminación natural variable con la capacidad computacional de un ordenador medio. Para ello se hace necesario trabajar en la detección y seguimiento de nuevas características como la cara previamente al análisis de la zona del ojo.

El seguimiento de caras consta de dos partes: la primera consiste en la localización facial, mediante un sistema de detección de caras y de un sistema de reconocimiento facial, mientras que la segunda consiste en el seguimiento de estas a lo largo de una secuencia de imágenes, es decir, un vídeo.

La detección de caras puede ser considerada como un caso específico de la detección de objetos. Lo que se pretende con la detección de caras es localizar la cara o las caras, si existen, y devolver su ubicación a la imagen y el tamaño de estas. La localización facial se puede realizar mediante la detección de características faciales, como los ojos o la nariz.

La detección de caras [3] lleva desarrollándose desde hace varias décadas. Fruto de estos trabajos han surgido multitud de algoritmos y métodos para detectar caras en imágenes, entre los que a día de hoy, destaca el desarrollado por Paul Viola y Michael Jones [4]. Los extraordinarios resultados que han obtenido, unidos a la eficiencia y versatilidad de su esquema, hacen del detector de Viola-Jones un punto de partida muy interesante para cualquier aplicación donde se necesite detectar objetos con variabilidad estructural en tiempo real.

El seguimiento de objetos [5] es el proceso de estimar en el tiempo la ubicación de uno o más objetos móviles mediante el uso de una cámara. La rápida mejora en cuanto a calidad y resolución de los sensores de imagen, juntamente con el dramático incremento en cuanto a la potencia de cálculo en la última década, ha favorecido la creación de nuevos algoritmos y aplicaciones mediante el seguimiento de objetos.

2 – OBJETIVO

El objetivo de este proyecto es la evaluación y optimización de un programa de detección facial y un programa de tracking genérico aplicado al entorno de la conducción, con los requerimientos que ello implica en cuanto a variaciones de iluminación y movimientos del sujeto y la implementación de dos nuevos programas de eye-tracking que aúnen los puntos fuertes de los anteriores. Dichos programas tienen como objetivo principal la detección de las zonas de los ojos.

Figura 2.1 – Ejemplo de imagen a procesar (izquierda) y objetivo de la detección (derecha)

Como punto de partida se evaluará el programa de tracking *Predator*/*OpenTLD* [6] y se modificará para aplicarlo al seguimiento de los ojos.

Se evaluará y optimizará el programa de detección de ojos *facedetector*, un programa de detección facial previamente implementado por los desarrolladores del proyecto, basado en el detector de patrones de Viola-Jones.

A partir de los dos programas anteriores se implementarán dos nuevos programas que combinen los mejores aspectos de los dos primeros para obtener unas funciones de detección y seguimiento de las zonas de los ojos más robustas.

Una vez analizados los resultados de los cuatro programas y observados los errores principales de cada uno se implementará un sistema de filtrado para reducir esos errores y aumentar la fiabilidad de los detectores.

Finalizados y optimizados los cuatro programas, se añadirá un detector de iris, previamente implementado, para completar la detección.

Como aplicación secundaria orientada a la seguridad en la conducción se implementará un programa que alerte al usuario en caso de que no mire a la carretera durante un periodo determinado de tiempo.

Para analizar los programas se creará una base de datos de imágenes tomadas a cuatro personas conduciendo en tres momentos del día diferentes. Esas imágenes serán de hombres y mujeres de diferentes características faciales, con y sin gafas, a diferentes alturas, con diferentes condiciones de iluminación y a diferentes resoluciones. También se dispondrá de imágenes tomadas en interiores para la prueba inicial de los programas en condiciones de iluminación constantes.

3 – BASE DE DATOS DE IMÁGENES

Para la base de datos de imágenes se ha grabado a cuatro personas conduciendo en tres días diferentes.

Las imágenes son de hombres y mujeres de diferentes características faciales, con y sin gafas.

Las grabaciones se han realizado en diferentes vehículos para disponer de imágenes de caras a diferentes alturas y diferentes perspectivas de cámara.

Se han tomado las imágenes en diferentes momentos del día para tener muestras con diferente iluminación.

En la tabla 3.1 se muestra el día de grabación de los diferentes vídeos con cada persona. Se han nombrado los vídeos en función de la persona grabada y el número de vídeo de esa persona. Por ejemplo, el tercer vídeo de la persona número dos tiene de nombre P2-V3.

Persona-Día	D1	D2	D3
P1	V1, V2	V3	V4
P2	V1	V2	V3
P3	-	V1, V2	V3
P4	-	V1	V2

Tabla 3.1 – Relación Persona-Día de grabación para los vídeos

También se dispone de imágenes tomadas en interiores para la prueba inicial de los programas en condiciones de iluminación constante.

4 – PROCEDIMIENTO DE ANÁLISIS DE RESULTADOS

El análisis de los resultados se ha realizado de forma visual para todas las imágenes.

El procesado de una imagen es correcto si el programa detecta los dos ojos y éstos se encuentran completamente dentro de los bounding box. Se considera que un ojo está dentro de un bounding box si lo están el iris y la esclera.

Dado que *facedetector* detecta la cara y a partir de ésta estima la posición de los ojos, puede darse el caso de detectarse los dos ojos o de no detectarse ninguno, pero no sólo uno. Con *Predator*, por el contrario, sí puede darse el caso de detectarse un único ojo. Para que el criterio de clasificación entre procesado correcto e incorrecto sea el mismo para todos los programas se establece que en los casos en los que sólo se detecta un ojo la detección es incorrecta.

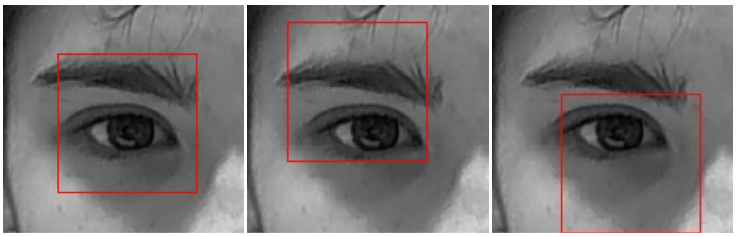


Figura 4.1. Ejemplos de detecciones correctas

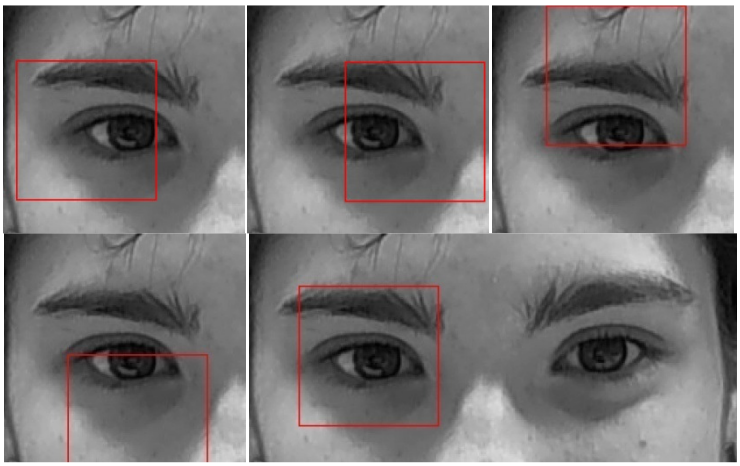


Figura 4.2. Ejemplos de detecciones incorrectas

5 – PREDATOR

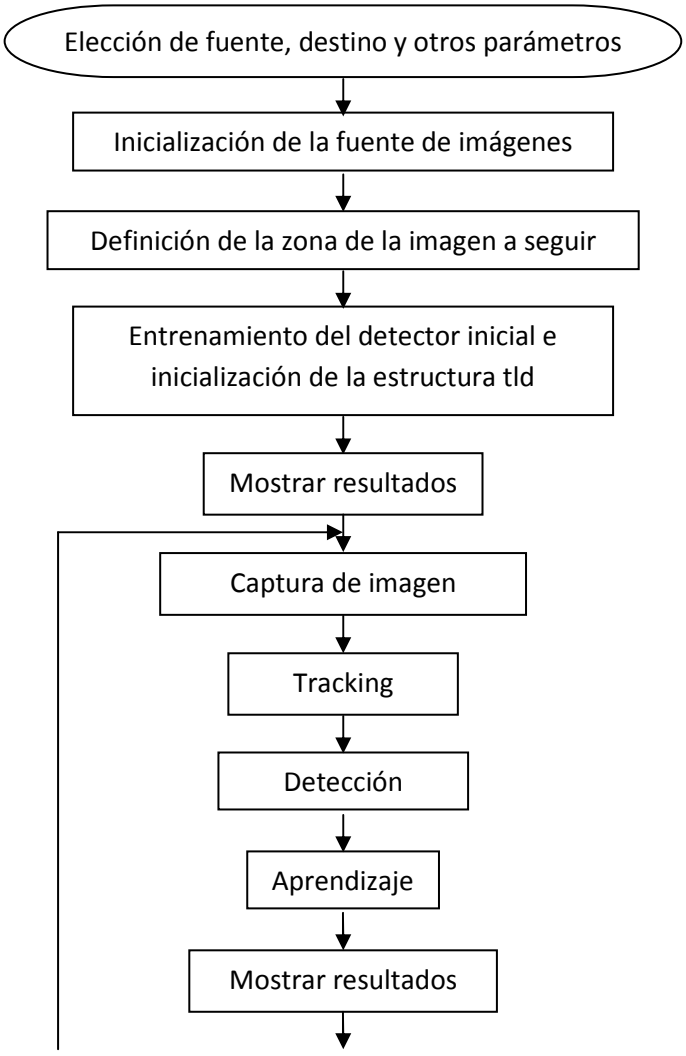
5.1 – Introducción

El algoritmo *Predator* (también conocido como *TLD Tracking-Learning-Detection*) [6] es un algoritmo de tracking (rastreo o seguimiento). Combina las funciones de tracking, detección y aprendizaje, es decir, además de detectar y seguir un objeto en una serie de imágenes es capaz de aprender online.

5.2 – Funcionamiento general

Al inicio del programa se elige la región de la imagen a seguir y el programa hace el seguimiento de esa región en las sucesivas imágenes. Mediante el proceso de aprendizaje amplía su base de datos de detecciones válidas o no válidas para mejorar el detector y no dar como válidas en el futuro aquellas detecciones que hasta ese momento consideraba no válidas. De esta manera el programa sabe qué características debe buscar y qué características debe evitar. Tras procesar las imágenes devuelve las coordenadas de las regiones donde ha encontrado el objeto a buscar y un índice de confianza que indica cómo de fiable ha sido la detección.

5.3 – Diagrama de flujo



5.4 – Funcionamiento paso a paso

Elección de fuente, destino y parámetros

Como inicialización del programa se debe definir una serie de parámetros que determinarán el comportamiento final del programa. Se explican a continuación los parámetros más importantes y los que se modifican más adelante:

- Fuente de imágenes: puede ser una carpeta con imágenes o una cámara de vídeo. En el campo *camera* en la estructura *opt.source* se define la fuente como directorio de imágenes o cámara de vídeo mediante el valor 0 ó 1 respectivamente. En el campo *input* se indica la ruta del directorio de imágenes, y en el campo *bb0* se indica el bounding box (área) inicial. Este último parámetro no se ha usado en este proyecto, ya que se calcula más adelante.
- Directorio de salida: en la estructura *opt.output* se indica la carpeta en la que se almacenan los datos resultantes del proceso.
- Min_win: tamaño mínimo válido del bounding box del objeto a buscar.
- Patchsize: tamaño de la zona de búsqueda normalizada en el detector.
- Fliplr: ofrece la posibilidad de aprender automáticamente el reflejo de los objetos detectados.
- Maxbbox: fracción de los bounding boxes evaluados en cada frame. Si se le asigna el valor 0 se desactiva la detección.
- Update_detector: aprendizaje. Si se le asigna el valor 0 el programa no aprende sobre la marcha.

Inicialización de la fuente de imágenes

La función *tldInitSource* inicializa la fuente de imágenes, o bien la cámara o bien la carpeta con imágenes. Si la fuente es una cámara, el programa enciende la cámara y comienza con la toma de imágenes en modo preview. Si la fuente es una carpeta, la función *img_dir* busca todas las imágenes existentes en esa carpeta (en formato jpeg, jpg, png y pgm) y almacena sus nombres y rutas completas.

Definición de la zona de la imagen a seguir

Tras las inicializaciones anteriores llega el turno del usuario de indicarle al programa qué región de la imagen desea seguir, para ello usa la función *tldInitFirstFrame*. Si el usuario sabe de antemano las coordenadas de la región a buscar puede indicarlo en un archivo *init.txt*, en caso contrario se abre una interfaz en la que el usuario indica con el ratón la región a seguir. Si la región es incorrecta (o bien no existe bounding box o bien su tamaño es menor que *min*_win) se repite la lectura o petición de datos hasta que sea correcta.

Figura 5.1 – Ejemplo de definición de la zona de la imagen a seguir

Entrenamiento del detector inicial e inicialización de la estructura tld

La función *tldInit* entrena el detector a partir de la zona definida en la función anterior para las siguientes imágenes. Para ello inicializa por un lado la estructura *tld*, que almacena todos los datos generados por *Predator* e inicializa por otro las funciones lk.cpp, fern.cpp y bb_overlap.cpp compiladas en formato mex. Una vez inicializado el detector y generados los parámetros de funcionamiento se entrena a partir de la imagen y bounding box anteriores, detectando las zonas de la imagen válidas y las zonas no válidas para el aprendizaje.

Mostrar resultados

La función *tldDisplay* muestra en pantalla la primera imagen junto con su bounding box y todos los puntos relevantes para el detector. Esta función muestra más o menos datos dependiendo de si la imagen a mostrar es la primera (que aún no ha sido procesada por *Predator*) o si es una de las siguientes imágenes (que ya han sido procesadas por *Predator*). En este momento se trata del primer caso, y se muestra la imagen, el bounding box del objeto, los ejemplos que hasta el momento considera válidos (en el lado derecho de la imagen), los que considera no válidos y la región de interés (en el lado izquierdo de la imagen).

Figura 5.2 – Ejemplo de primer tracking

Procesado de las siguientes imágenes

La función *tldProcessFrame* es la función principal de *Predator*, en la que se hace el tracking, la detección y el aprendizaje. Con esta función se procesan todas las imágenes salvo la primera, en la que el bounding box no se calcula sino que es definido por el usuario. Consta de las siguientes partes principales:

- Captura de la imagen: toma la i-ésima imagen de la fuente (cámara/carpeta) y la convierte en imagen en escala de grises. Admite la posibilidad de reescalar la imagen.
- Tracking: la función *tldTracking* realiza el seguimiento del objeto en la nueva imagen. El método que usa para ello es el algoritmo de seguimiento de Lucas-Kanade [7].
- Detección: la función *tldDetection* realiza la detección del objeto en la nueva imagen. Escanea la imagen con una ventana deslizante y devuelve un conjunto de bounding boxes y la confianza con la que se asemejan al objeto. Para ello utiliza principalmente la función *fern* [8].
- Análisis de tracking y detección: A partir del bounding box resultante del tracking y los candidatos de la detección, calcula la región que más se asemeja al objeto a seguir. Inicialmente considera correcto el resultado del tracking. En caso de haber una única detección con un índice de confianza superior al del tracking desecha el tracking y considera correcta esa detección. Si hay más de una detección con índice de confianza superior al del tracking, ajusta el resultado con el tracking y las detecciones más cercanas al tracking.
- Aprendizaje: a partir del resultado anterior, mediante la función *tldLearning* se actualiza la base de datos con la región válida y las regiones no válidas.

Mostrar resultados

Nuevamente se utiliza la función *tldDisplay* para mostrar los resultados del procesado. Tras *tldProcessFrame* muestra algunos datos más que la primera vez, al ser el procesado más complejo: imagen, bounding box del objeto (si lo encuentra), los ejemplos que hasta el momento considera válidos (en el lado derecho de la imagen), los que considera no válidos y la región de interés (en el lado izquierdo de la imagen), un punto rojo en el centro del bounding box detectado, puntos azules utilizados como referencia para el procesado, el índice de confianza y un mensaje de texto que indica el número de imagen y la velocidad de procesado.

Figura 5.3 – Ejemplo de detección en imagen cualquiera

Guardar resultados

Al acabar el procesado de las imágenes se guarda las coordenadas de los bounding box y el intervalo de confianza de cada imagen en el archivo de texto *tld.txt* en la carpeta de salida definida al comienzo del programa.

6 – FACEDETECTOR

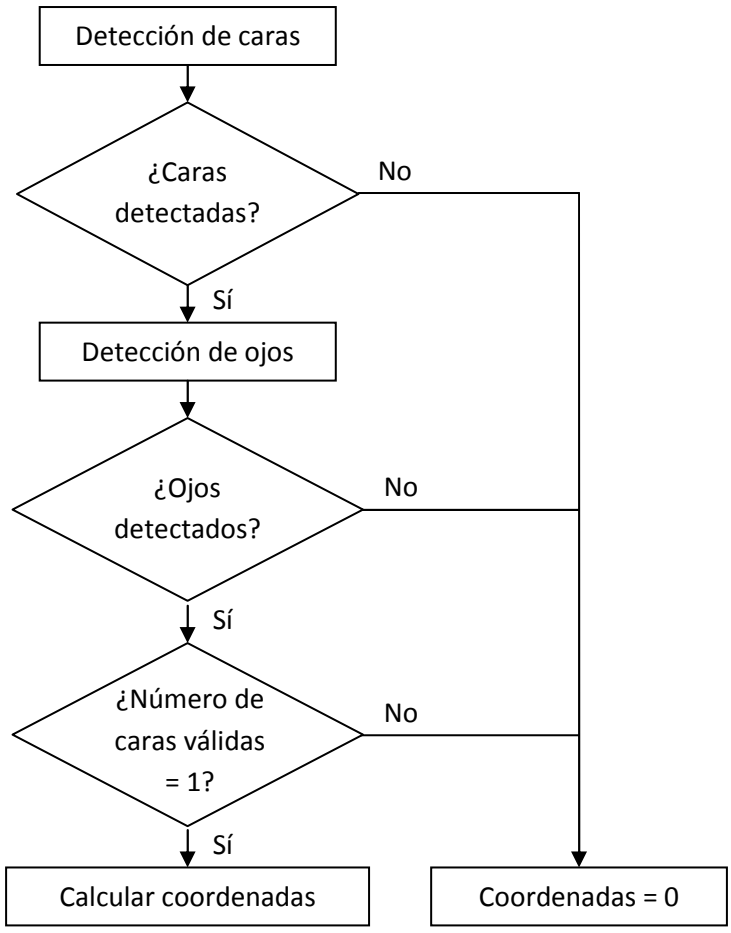
6.1 – Introducción

Facedetector es un programa de detección facial basado en el algoritmo de detección de patrones de Viola-Jones [4]. Utilizando ese detector localiza la cara en una imagen y recorta la zona de los ojos según conocimientos estructurales de la cara para devolver la zona de interés.

6.2 – Funcionamiento general

A partir de una imagen, la función de detección de patrones *cvod200uint_v03* y el archivo de entrenamiento de caras *haarcascade_frontalface_alt.xml*, encuentra las posibles caras en la imagen. Una vez localizada la región o regiones candidatas a ser caras afina la búsqueda detectando un ojo, usando la misma función *cvod200uint_v03* con el archivo de entrenamiento *haarcascade_eye_tree_eyeglasses.xml*. Tras la segunda detección acepta las caras que contienen ojos y rechaza las que no los contienen. En caso de que el número de caras sea diferente de uno considera que la detección no es válida. Si la detección es válida aísla los ojos en base a las características faciales medias. La función de detección está implementada en la librería *OpenCV* [9].

6.3 – Diagrama de flujo



6.4 – Funcionamiento paso a paso

Detección de caras

Se detectan las posibles caras presentes en la imagen con la función de detección de patrones *cvod200uint_v03.* Para la detección facial utiliza el archivo*haarcascade_frontalface_alt.xml*. Esta detección se limita a caras frontales, funcionando también para caras con una inclinación baja pero no para caras de perfil.

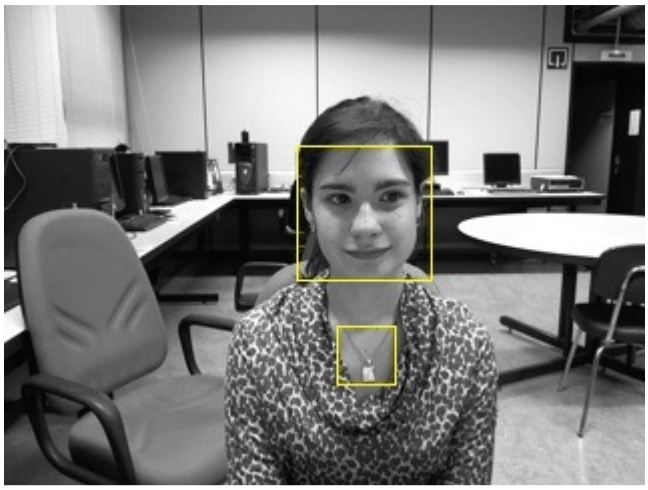


Figura 6.1 – Detección de dos posibles caras en la imagen

Detección de ojos

A partir de las regiones consideradas como posibles caras se buscan los ojos presentes. Se utiliza nuevamente la función de detección *cvod200uint_v03*, con el archivo de características de ojos *haarcascade_eye_tree_eyeglasses.xml*.

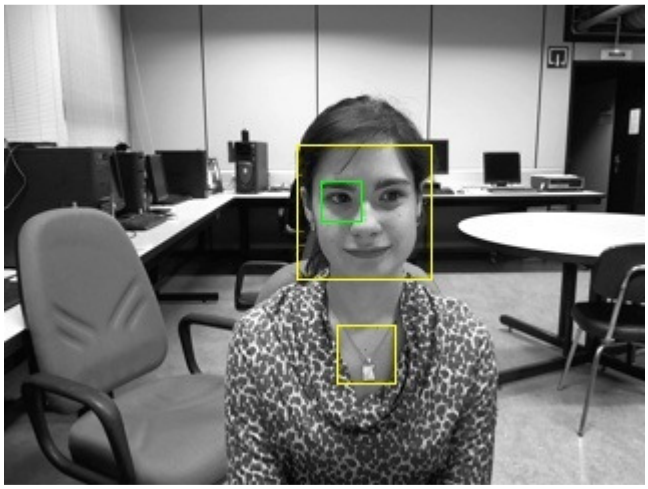


Figura 6.2 – Detección de ojos en las posibles caras en la imagen

Cálculo de coordenadas

A partir de las detecciones de caras y ojos se desechan las regiones candidatas a caras que no contienen ojos. El resto de regiones que sí contienen ojos se consideran caras válidas. La detección es válida si tras el filtrado queda una única cara. Finalmente se recorta la región de interés según conocimientos estructurales de la cara. La región de interés puede ser la cara completa, la mitad superior de la cara, los ojos izquierdo o derecho o la zona de los dos ojos. En este proyecto la región de interés se corresponde a los dos ojos.

Figura 6.3 – Recorte de los ojos en la cara de la imagen

7 – EYE TRACKER PREDATOR

7.1 – Introducción

La primera función de tracking es *Eye_Tracker_Predator*. Está basada principalmente en el programa *Predator*, pero con una serie de modificaciones para detectar y seguir las zonas de los ojos.

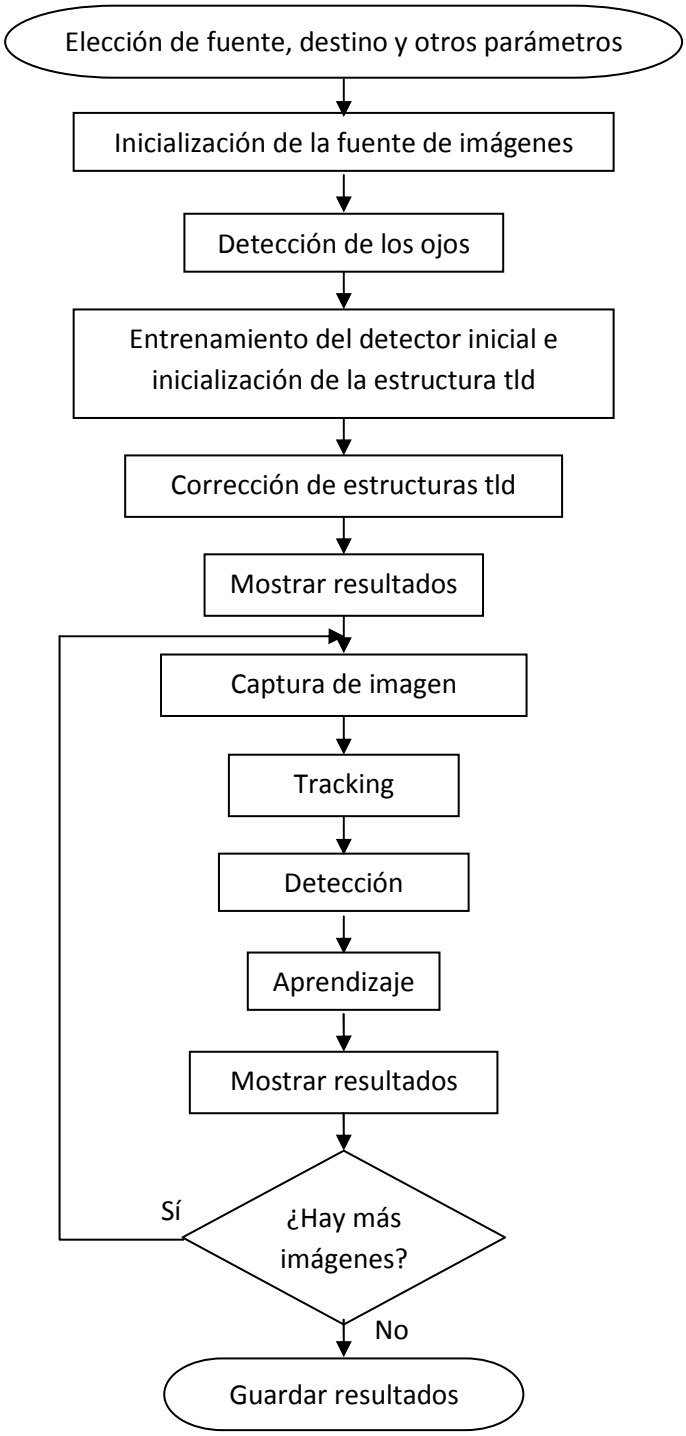
7.2 – Modificaciones

Predator está diseñado para seguir un único objeto en la imagen, por lo que no puede procesar los dos ojos independientemente y de manera simultánea. Dado que se necesita seguir dos ojos es necesario procesar el doble de información. Se plantean dos opciones para llevarlo a cabo. La primera opción es duplicar los campos de las estructuras *opt* y *tld*. Esa opción supone modificar todas las funciones que procesan esos campos, prácticamente todas. La segunda opción es tener dos estructuras *opt* y dos estructuras *tld*. De esta manera se llama dos veces a las funciones que utilicen esas estructuras pero no se modifica su código. Por simplicidad y para evitar errores se ha optado por esta segunda opción.

El problema a la hora de duplicar las estructuras se halla en la función *fern*, que realiza una reserva de memoria dinámica al ejecutarse por primera vez. Cuando se ejecuta por primera vez en la inicialización de *tld1*, en *tldInit,* reserva memoria a la que accede mediante un puntero, y lo hace correctamente. A la hora de inicializar *tld2* vuelve a reservar memoria, y es en ese momento cuando el puntero cambia de dirección, perdiéndose la primera dirección. La siguiente vez que se utiliza la función *fern* intenta acceder a la primera zona de memoria, pero al haberla perdido se produce un error. Para evitar ese problema se ha optado por duplicar la función *fern*, usando en este nuevo programa las funciones *fern1* y *fern2*. De esta manera cada función reserva su zona de memoria independiente. Dado que en este programa existen dos funciones *fern*, es necesario duplicar todas las funciones que directa o indirectamente utilizan esa función, y asociar a cada una la función *fern1* o *fern2*. Esas funciones son *tldInit*, *tldProcessFrame*, *tldTracking*, *tldDetection*, *tldLearning*, *tldGeneratePositiveData* y *tldGenerateNegativeData*.

El siguiente cambio principal ha sido sustituir la elección del objeto a seguir por una función de detección facial, de manera que el nuevo programa no depende del usuario para localizar los ojos. El algoritmo original ofrece la opción de elegir el objeto al usuario sobre la primera imagen, y en caso de que el área definida sea incorrecta vuelve a pedirle el área sobre esa primera imagen. El problema de *Predator* con la función de detección facial es que en caso de no detectar la cara vuelve a buscarla sobre la misma imagen, derivando en un bucle infinito del que no sale. Se ha solucionado modificando la función para que en caso de no encontrar la cara cargue la segunda imagen, y si es necesario la tercera, y así sucesivamente hasta encontrar la cara en una de ellas. Las sucesivas imágenes se procesan con *Predator*.

7.3 – Diagrama de flujo



7.4 – Funcionamiento paso a paso

Elección de fuente, destino y parámetros

Se ha agrupado en una función, *opt_init*, para dar al programa mayor modularidad, pero mantiene las mismas instrucciones que el original. Para aumentar la compatibilidad con diferentes versiones de MatLab se utilizan diferentes funciones mex según indicaciones del desarrollador de *Predator* [6]. La elección de las funciones correspondientes a cada versión de MatLab se realiza en la función *init_mex*.

Inicialización de la fuente de imágenes

Se ha modificado la captura de imágenes para hacerla algo más completa. La función *tld_init_source* es exactamente igual a *tldInitSource*, pero la nueva función *img_dir*, que recoge los nombres de todas las imágenes existentes en un directorio, funciona además de con los 4 formatos de imagen originales (jpeg, jpg, png y pgm) con gif y bmp.

Detección de los ojos

Para la detección de los ojos en la primera imagen se ha utilizado la función *facedetector*, basada en el detector de objetos de Viola-Jones, entrenado para buscar caras. En capítulos posteriores se describe detalladamente este detector. Como se ha introducido anteriormente, la función *tld_init_first_frame* (equivalente a *tldInitFirstFrame*), encargada de capturar la imagen y buscar los ojos, se ha modificado para cargar el número de imágenes necesario hasta detectar la cara en lugar de centrarse únicamente en la primera imagen.

Entrenamiento inicial de los detectores e inicialización de las estructuras tld

El entrenamiento inicial de los detectores y la inicialización de las estructuras tld1 y tld2 sigue la misma línea que el *Predator* original con la función *tldInit*. La diferencia en este caso es que para evitar los problemas de la función *fern* con la reserva de memoria se utilizan dos funciones de inicialización, *tld_init_1* y *tld_init_2*, asociadas a *fern1* y *fern2* respectivamente. También relacionada con la función *fern* está la función *tldGenerateFeatures*, que genera una serie de datos que posteriormente utiliza *fern*. Parte de la generación de esos datos se hace de manera aleatoria (*randperm*), y es ahí donde se encuentra el problema. A la hora de usar *randperm* en la inicialización de *tld1* el generador de números aleatorio de MatLab no ha sido usado y genera cierta serie de números, pero la segunda vez que se ejecuta para inicializar *tld2* ya ha sido usado y genera una serie de números diferente a la anterior, así que la inicialización es diferente en ese aspecto. La diferencia no es significativa, ya que el procesado de los dos ojos es correcto, pero hace que los resultados no sean exactamente los mismos si invertimos el orden de procesado de los ojos. Para eliminar esa diferencia (aunque el resultado no es incorrecto) se ha modificado el estado del generador de números aleatorios en la segunda inicialización para dejarlo igual que en la primera inicialización.

Corrección de estructuras tld

Las funciones de inicialización *tld_init_1* y *tld_init_2* tienen el problema de generar los datos suponiendo que la imagen tratada es la número 1, guardándolos en la posición 1 de los vectores de datos. Debido al uso del *facedetector* es posible que la primera imagen válida no sea la número 1, así que almacenar los datos en esa posición no siempre es correcto. En lugar de modificar las funciones *tld_init* debido al número de funciones de las que depende, se ha optado por hacer la corrección al finalizar la inicialización con la función *tld_corregir*. Esa corrección sitúa en la posición correcta los campos *bb* (bounding box), *conf* (intervalo de confianza) e *img* (imagen).

Mostrar resultados

La función *tldDisplay* es muy compleja y muestra mucha información en pantalla, lo que por un lado es bueno de cara a analizar el funcionamiento del programa. Sin embargo, a la hora de analizar los datos en la aplicación de eye tracking resulta muy incómodo ver esa serie de puntos y números en la pantalla, que imposibilitan determinar a simple vista si la detección ha sido buena o no, si los ojos están completamente dentro de los bounding box o no. Para ello se ha hecho una nueva función, *display_colores*, que muestra únicamente la imagen, los bounding box si los hay, y el texto informando del número de imagen y la velocidad de procesado. Además permite cambiar el color del bounding box, utilizando en este proyecto el color verde para mostrar los resultados obtenidos con *facedetector* y rojo los de *Predator.* Esta diferenciación resulta interesante a la hora de analizar los datos para comparar ambos detectores.

Figura 7.1 – Primera detección de los ojos

Captura de imágenes

La función de captura de imágenes *img_alloc*, dependiente de *img_get*, ha sido modificada para reescalar la imagen de entrada a un tamaño de 240 pixeles de altura y la anchura correspondiente a su relación de aspecto. De esta manera se evitan los problemas de falta de memoria con imágenes muy grandes, se aumenta la velocidad de ejecución y no es necesario reajustar los parámetros *min_win* o *patchsize* para cada resolución.

Procesado de las siguientes imágenes

La función *tldProcessFrame* ha derivado en las funciones *tld_process_frame_1* y *tld_process_frame_2*, muy parecidas a la anterior pero salvando el problema con *fern*. Estas dos funciones han sido modificadas para no realizar la lectura de la imagen sino para tomarla como variable de entrada, así no se produce la lectura de la misma imagen dos veces, evitando ese retraso innecesario. El tracking, la detección y el aprendizaje se realiza exactamente igual que en el *Predator* original con la salvedad del cambio de nombre por la duplicidad de las funciones.

Mostrar resultados

Para mostrar los resultados de los *tld_process_frame* se utiliza nuevamente la función *display_colores*, mostrando los bounding box de color rojo.

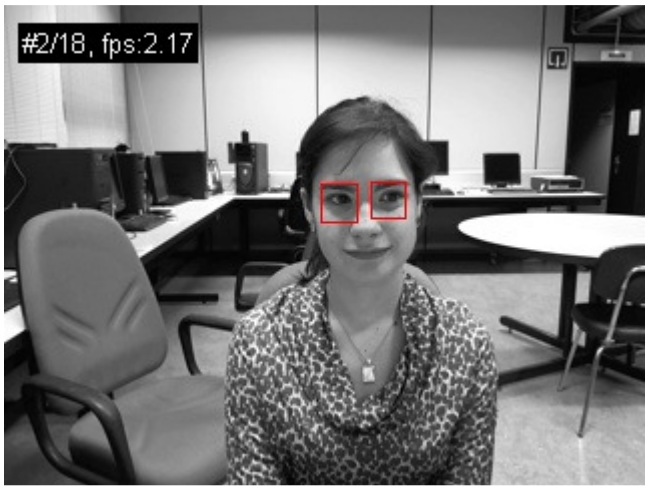


Figura 7.2 – Ejemplo de tracking

Guardar resultados

Dado que el método de guardar los resultados de *Predator* no es exactamente lo que necesita este proyecto se ha hecho una nueva función de guardado de datos, *guardar_resultados*. Genera un archivo de texto en el que almacena los datos de bounding box e intervalos de confianza, pero además identifica la imagen a la que corresponden esos datos y a qué ojo se refieren. Además, en lugar de guardar las coordenadas de las esquinas opuestas de los bounding box guarda las coordenadas de la esquina superior izquierda y las dimensiones, resultando un archivo más cómodo de analizar.

Guardar resultados en vídeo

Además de guardar los datos en el archivo *tld.txt* se ofrece la posibilidad de capturar las imágenes mostradas durante el procesado y crear con ellas un vídeo para facilitar el posterior análisis de los datos. La función creada para ello es *img_a_video*. El vídeo final está en formato *avi* con compresión *cinepak*.

Ajuste de parámetros del detector

Para optimizar el funcionamiento del detector se han hecho una serie de pruebas modificando en *opt_init* los parámetros de inicialización *min_win, patchsize, fliplr y maxbbox*. De esas pruebas se concluye que los valores óptimos para la aplicación que aquí se trata son:

- Min_win: Se reduce a 12. Al trabajar con imágenes pequeñas en ocasiones el tamaño mínimo de 24 píxeles resulta demasiado grande.
- Patchsize: Se mantiene en 15. Valores mayores o menores dan como resultado un menor número de detecciones válidas.
- Fliplr: Se mantiene en 0. No interesa que el detector aprenda el reflejo de la imagen válida porque puede centrarse en el ojo que no le corresponde.
- Maxbbox: Se mantiene en 1. Si se le asigna el valor 0 realiza el tracking pero no la detección, aumentando la velocidad de procesado pero disminuyendo el número de detecciones válidas.

8 – EYE_TRACKER_VJ

8.1 – Introducción

La función *Eye_Tracker_VJ* se presenta como alternativa a la función *Eye_Tracker_Predator*. En este caso la detección de la zona de los ojos se realiza para cada imagen en lugar de únicamente en la imagen inicial y se prescinde del tracking.

8.2 – Modificaciones

Se ha creado la función *bb_get*, que a partir de una imagen obtiene las posiciones de los dos ojos. Su funcionamiento se reduce a la función *facedetector*, más una corrección de coordenadas para centrar los ojos en los bounding box. En caso de no detectar ninguna cara las coordenadas de los bounding box son dos vectores de NaNs en lugar del cero original para evitar errores de dimensiones con *Predator*. Para mantener la uniformidad en el archivo de resultados en las cuatro funciones principales se asigna a los resultados válidos de *bb_get* un índice de confianza de 1, y a los resultados inválidos un índice de confianza de NaN.

A partir del programa *Eye_Tracker_Predator* se ha sustituido la parte de procesado de imágenes (*tld_process_frame*) para tener los diferentes programas idénticos en todos los aspectos salvo la función de detección o seguimiento de los ojos.

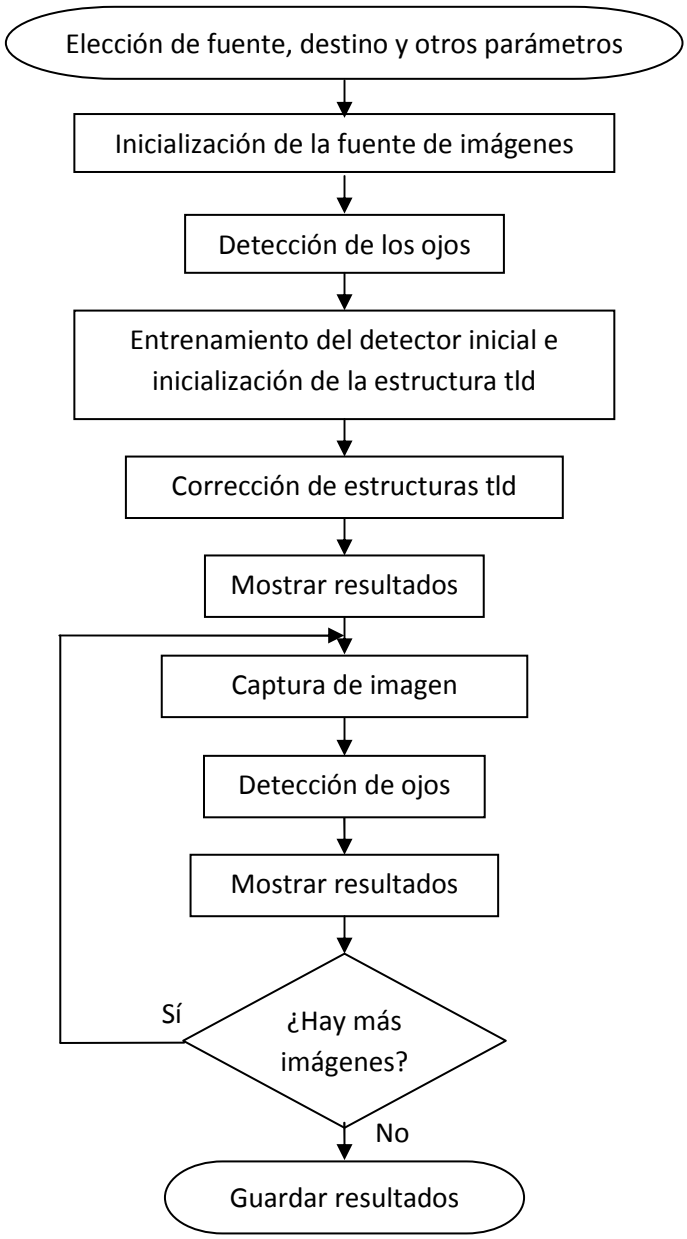
A partir de las coordenadas de los ojos generadas por *facedetector* se han separado los dos ojos y se han centrado los bounding box en los ojos para evitar cortes.

Figura 8.1 – Zona de los ojos según *facedetector* (izquierda) y zonas corregidas (derecha)

La función *cvod200uint_v03* consta de los siguientes parámetros de entrada: *archivo xml, imagen, escala, minsize* y *debug*. Tras realizar una serie de pruebas con diferentes valores para los parámetros *escala* y *minsize* se concluye que para imágenes de 240 pixeles de altura los valores óptimos en cuanto a rapidez y calidad de la detección son 1 para *escala* y 2 para *minsize*. Cuanto mayor es *minsize* menor es el número de detecciones válidas, y cuanto menor es *escala* mayor es el tiempo de ejecución.

A pesar de que las funciones *tld_init_1* y *tld_init_2* no son estrictamente necesarias al no procesar las imágenes con *Predator*, se han mantenido para inicializar las variables que almacenan los resultados. Estas funciones inicializan más campos de los necesarios en las estructuras *tld1*y *tld2,* pero de esta manera hay una única función de inicialización para las cuatro funciones de detección principales, manteniendo las mismas estructuras y variables en todas ellas.

8.3 – Diagrama de flujo



9 – EYE_TRACKER_VJ_LEARNING_PREDATOR

9.1 – Introducción

La función *Eye_Tracker_VJ_Learning_Predator* combina la capacidad de detección de *facedetector* y la capacidad de tracking de *Predator*. La diferencia con la función *Eye_Tracker_Predator* es que *Eye_Tracker_VJ_Learning_Predator* utiliza *facedetector* para detectar los ojos en todas las imágenes en lugar de sólo en la primera. En aquellos casos en los que la detección no es correcta, se realiza el tracking con *Predator*. De este modo el número de detecciones es mayor que con cualquiera de los dos métodos anteriores. Para que el tracking sea lo más correcto posible se provoca en *Predator* el aprendizaje (*Learning*) con las detecciones de *facedetector*.

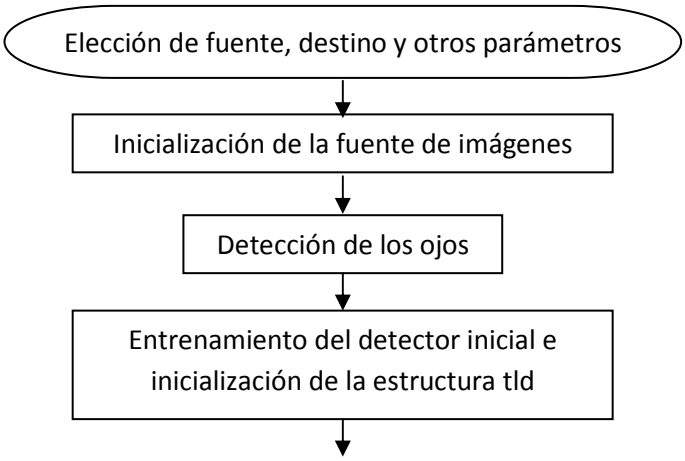
9.2 – Modificaciones

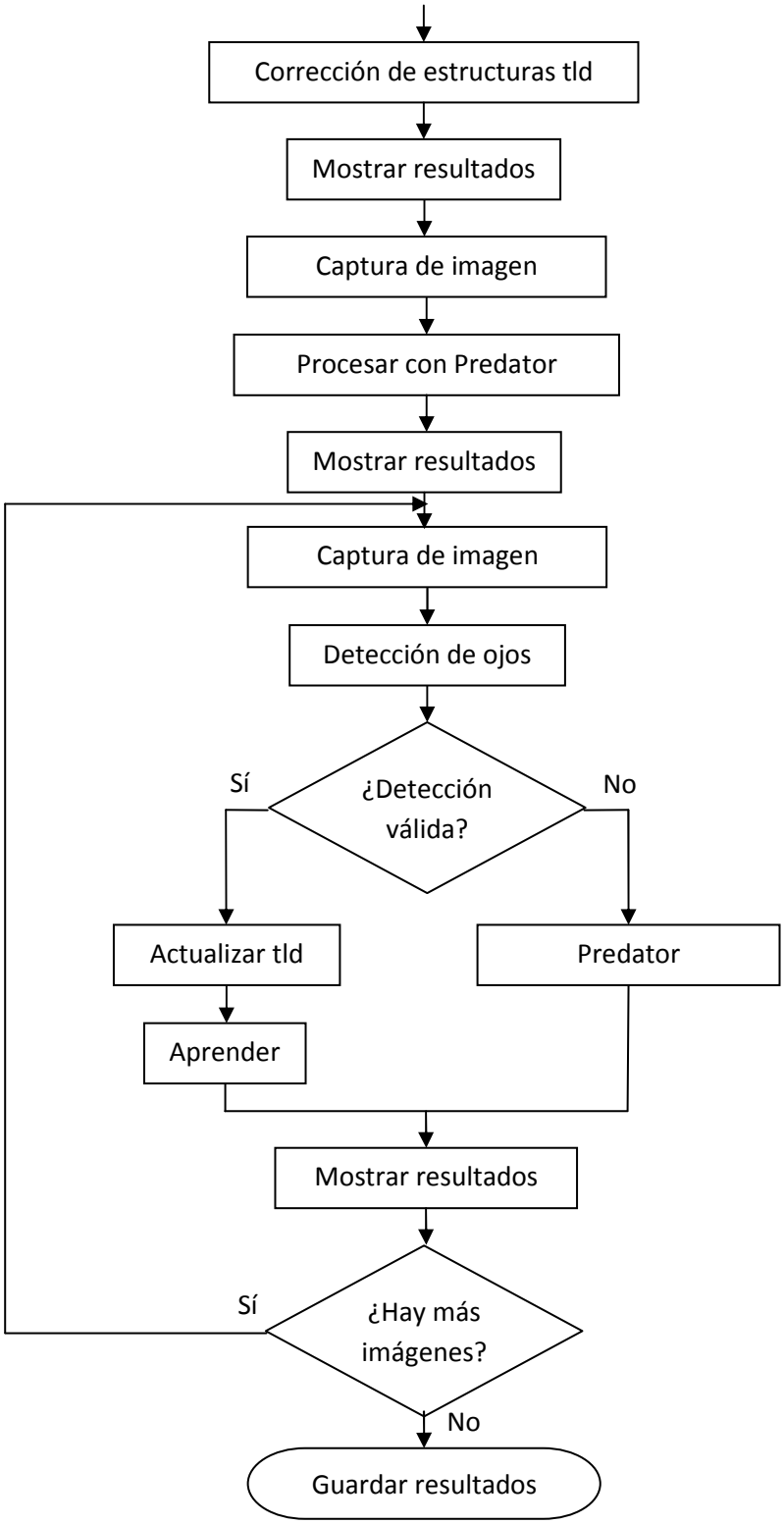
En líneas generales se parte de la función *Eye_Tracker_VJ*, se añade el *Learning* en las detecciones válidas y *Predator* en las detecciones no válidas.

El *learning* requiere una serie de campos en las estructuras *tld* que sólo se generan en el *tracking* y la *detección* en *tld_process_frame*. Dado que es posible que *tld_learning* intervenga por primera vez antes que *tld_process_frame*, es necesario inicializar esos campos para que *tld_learning* pueda hacer uso de ellos independientemente de si ya se han inicializado por *tld_process_frame* o no. El método utilizado para la inicialización consiste en aplicar la función *tld_process_frame* en la segunda imagen procesada. De esta manera, todas las llamadas a las funciones *tld_learning* a partir de la tercera imagen disponen de todas las variables necesarias.

Del mismo modo que *tld_process_frame* inicializa ciertas variables en la primera ejecución, las actualiza en cada nueva llamada. En las imágenes en las que no interviene *tld_process_frame* no se produce esa actualización, por lo que es necesario corregirlo para que en futuros usos de *tld_process_frame* se disponga de datos no nulos en esos campos. Para ello se rellenan los campos *tld.img* y *tld.bb*, relativos a la imagen a procesar y al bounding box detectado respectivamente, para cada imagen procesada por *get_bb* exclusivamente. El resto de campos se duplican con la información de la imagen anterior. Esta actualización se lleva a cabo en la función *tld_actualizar*.

9.3 – Diagrama de flujo





10 – EYE_TRACKER_VJ_PREDATOR

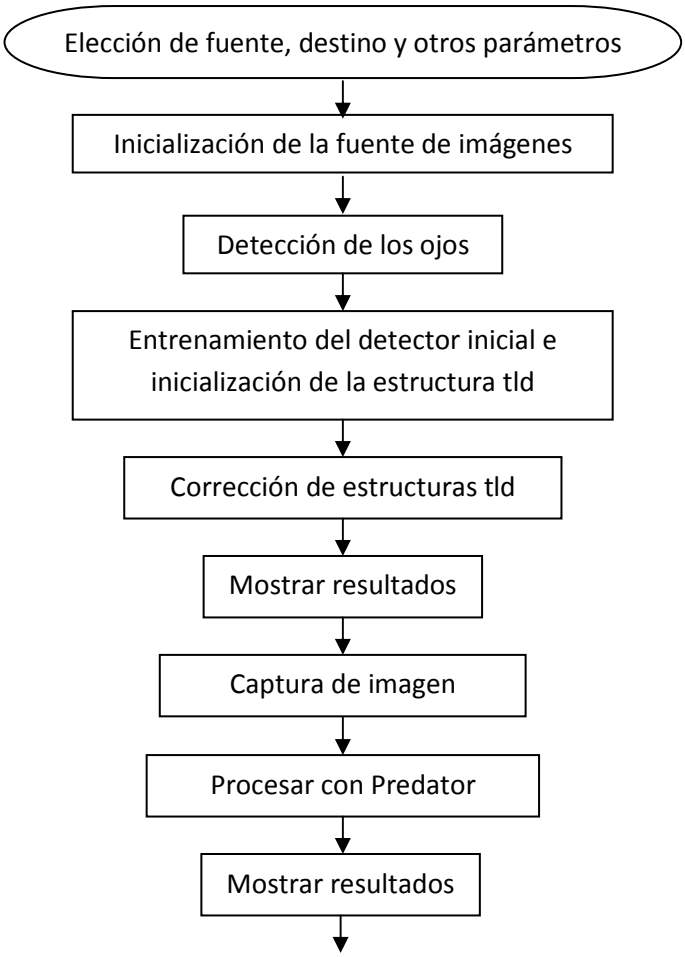
10.1 – Introducción

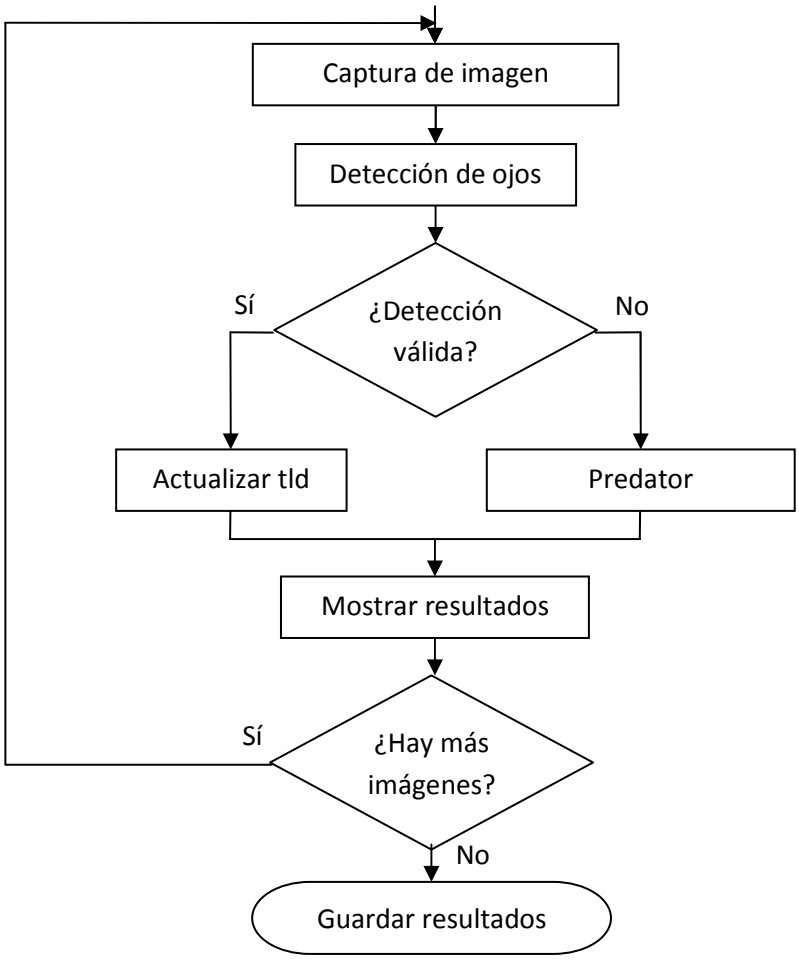
La función *Eye_Tracker_VJ_Predator* es similar la función *Eye_Tracker_VJ_Learning_Predator*, combina *Eye_Tracker_VJ* con *Eye_Tracker_Predator* para aumentar el número de detecciones, pero se prescinde del aprendizaje para reducir el tiempo de procesado.

10.2 – Modificaciones

Se mantiene exactamente la misma estructura que en *Eye_Tracker_VJ_Learning_Predator* con la salvedad de la eliminación del aprendizaje. Se ha optado por mantener el procesado con *predator* en la segunda imagen, a pesar de no ser necesario por no haber aprendizaje aislado, para que *predator* procese exactamente las mismas imágenes que en *Eye_Tracker_VJ_Learning_Predator* y poder compararlos posteriormente imagen a imagen.

10.3 – Diagrama de flujo





11 – REDUCCIÓN DE FALSOS POSITIVOS

Las funciones *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* aumentan el número de detecciones correctas con respecto a *Eye_Tracker_Predator* y *Eye_Tracker_VJ*. Sin embargo, el número de falsos positivos se considera elevado para poder afirmar que los detectores son fiables. Tras analizar las imágenes en las que la detección es incorrecta se ha obtenido una serie de criterios con los que se puede afirmar con una seguridad muy alta que la detección es errónea.

Diferencia de tamaños de los bounding box

Una gran diferencia de tamaños de los bounding box indica que o bien un bounding box es demasiado pequeño o bien es demasiado grande. Se ha considerado que una detección es correcta si el tamaño del menor bounding box es superior al 60% del tamaño del mayor bounding box.

Figura 11.1 – Ejemplo de diferencia significativa de tamaño de bounding box

Ojos muy separados horizontalmente

Una gran separación horizontal de los bounding box indica que al menos uno de los dos ojos no ha sido detectado correctamente. Se ha considerado que una distancia superior al doble del ancho del bounding box promedio supone una detección incorrecta.

Figura 11.2 – Ejemplo de distancia horizontal excesiva de los bounding box

Ojos muy separados verticalmente

Una gran separación vertical de los bounding box indica que al menos uno de los dos ojos no ha sido detectado correctamente. Se ha considerado que una distancia superior al 50% de la altura del bounding box promedio supone una detección incorrecta.

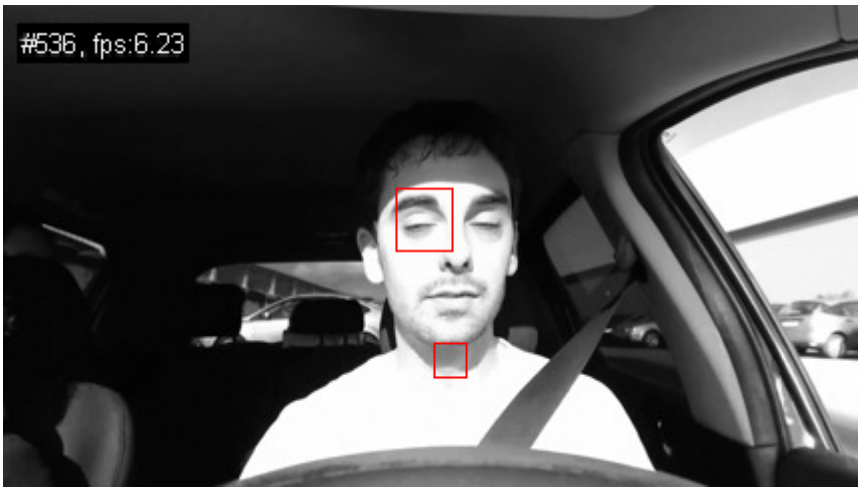


Figura 11.3 – Ejemplo de distancia vertical excesiva de los bounding box

Ojos cruzados o mismo ojo

En el caso de que el bounding box del ojo izquierdo se encuentre a la derecha del bounding box del ojo derecho o los dos bounding box estén centrados en el mismo ojo la detección es incorrecta. Se ha considerado que un cruce de los bounding box o un solapamiento superior al 33% supone una detección incorrecta.

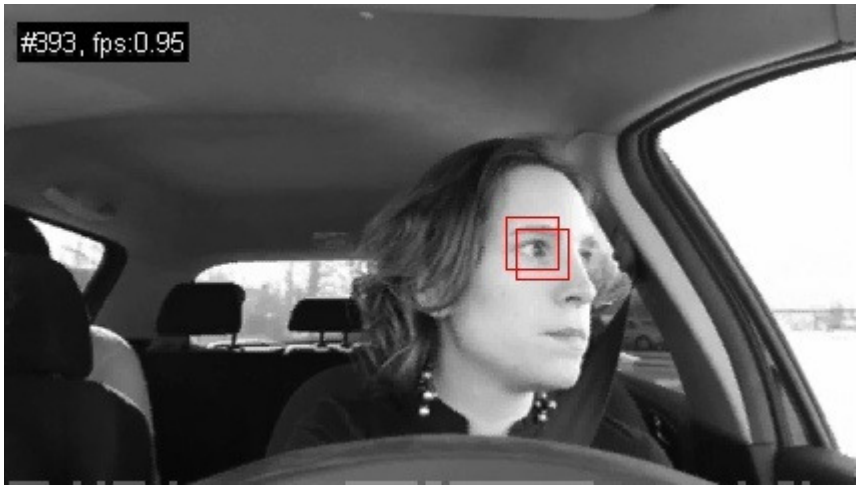


Figura 11.4 – Ejemplo de los bounding box centrados en el mismo ojo

Sólo un ojo detectado

En el caso de que no se detecten los dos ojos la detección es incorrecta.

Figura 11.5 – Ejemplo de detección de un único ojo

Resultado muy diferente al primero

Se ha asumido que la detección de los ojos en la primera imagen es correcta. Usando esa detección como referencia se ha considerado erróneas aquellas detecciones que difieren significativamente de ésta. Se ha considerado que una diferencia de tamaño o distancia superior al 300% o inferior al 33% supone una detección incorrecta.

Figura 11.6 – Ejemplo de diferencia significativa de resultados con respecto al original.

Se ha implementado la función *filtrado_bbox* para analizar los resultados de cada imagen y considerar nulos aquellos que cumplan alguna de las condiciones anteriores.

12 – RESULTADOS

A continuación se muestra los resultados obtenidos por los diferentes algoritmos para los videos contenidos en la base de datos. Los resultados se dividen en dos partes. En una primera se muestran los resultados obtenidos sin el filtrado de falsos positivos y en la segunda los obtenidos añadiendo el filtrado de falsos positivos.

Resultados de Eye_Tracker_Predator sin el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	887	708	1863	2919	416	227	1612
Detecciones incorrectas	554	1216	937	2334	361	1467	940
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	399	213	574	2302	87	538	857
Verdaderos positivos (%)	62%	37%	67%	56%	54%	13%	63%
Falsos positivos (%)	28%	11%	21%	44%	11%	32%	34%
Verdaderos negativos (%)	11%	52%	13%	1%	35%	55%	3%
Tiempo por imagen (s)	0,67	0,39	0,65	0,72	0,51	0,52	0,67

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	2332	1271	2580	763	0		**15578**
Detecciones incorrectas	1201	699	350	686	1410		**12155**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	1086	693	166	461	755		**8131**
Verdaderos positivos (%)	66%	65%	88%	53%	0%		**56%**
Falsos positivos (%)	31%	35%	6%	32%	54%		**29%**
Verdaderos negativos (%)	3%	0%	6%	16%	46%		**15%**
Tiempo por imagen (s)	0,80	0,73	0,56	0,42	0,77		**0,64**

Tabla 12.1 – Resultados de *Eye_Tracker_Predator* sin filtrado de falsos positivos.

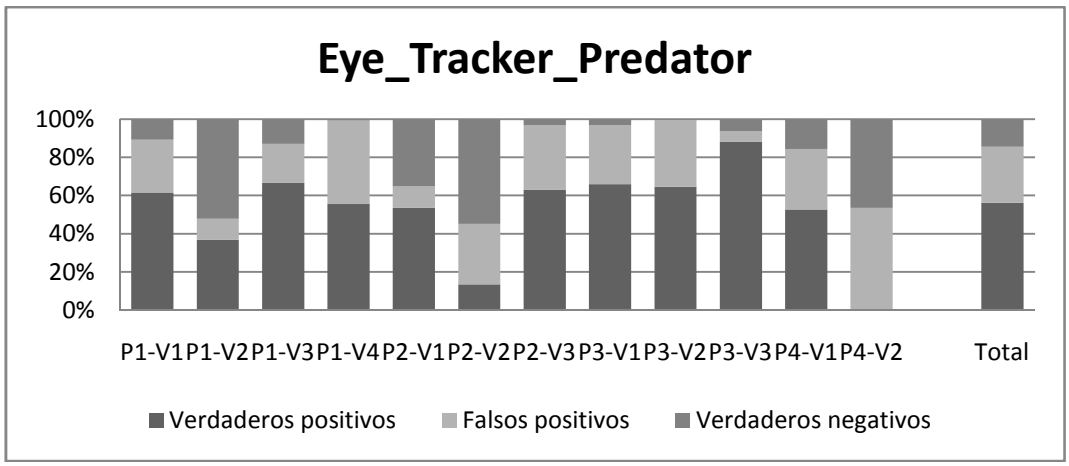


Figura 12.1 – Resultados de *Eye_Tracker_Predator* sin filtrado de falsos positivos.

Resultados de Eye_Tracker_VJ sin el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	1083	1268	1800	4248	376	903	2191
Detecciones incorrectas	358	656	1000	1005	401	791	361
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	3	37	34	59	4	25	44
Verdaderos positivos (%)	75%	66%	64%	81%	48%	53%	86%
Falsos positivos (%)	0%	2%	1%	1%	1%	1%	2%
Verdaderos negativos (%)	25%	32%	35%	18%	51%	45%	12%
Tiempo por imagen (s)	0,52	0,50	0,61	0,62	0,47	0,61	0,61

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	1422	1377	1808	518	683		**17677**
Detecciones incorrectas	2111	593	1122	931	727		**10056**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	44	12	81	81	104		**528**
Verdaderos positivos (%)	40%	70%	62%	36%	48%		**64%**
Falsos positivos (%)	1%	1%	3%	6%	7%		**2%**
Verdaderos negativos (%)	59%	29%	36%	59%	44%		**34%**
Tiempo por imagen (s)	0,66	0,65	0,60	0,61	0,67		**0,61**

Tabla 12.2 – Resultados de *Eye_Tracker_VJ* sin filtrado de falsos positivos.

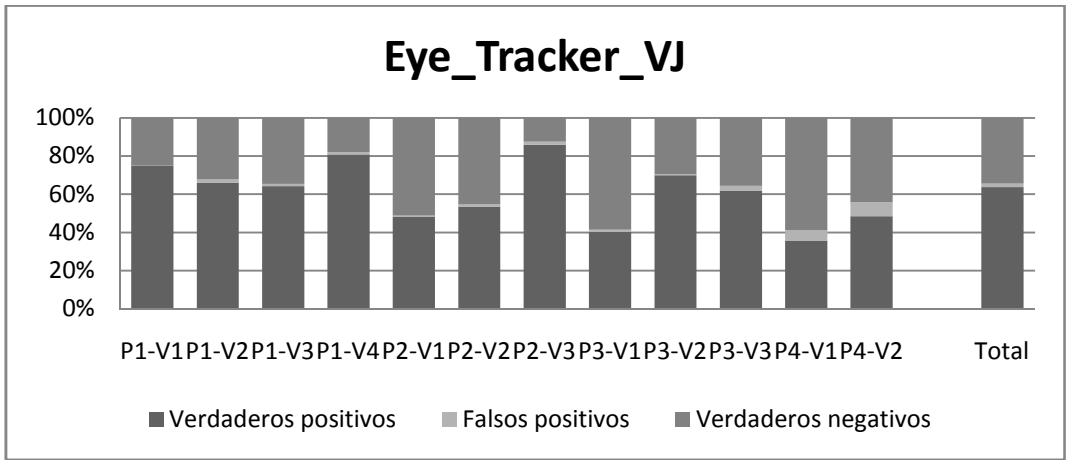


Figura 12.2 – Resultados de *Eye_Tracker_VJ* sin filtrado de falsos positivos.

Resultados de Eye_Tracker_VJ_Learning_Predator sin el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	1356	1392	2430	4885	430	1109	2365
Detecciones incorrectas	85	532	370	368	347	585	187
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	34	167	215	326	35	195	157
Verdaderos positivos (%)	94%	72%	87%	93%	55%	65%	93%
Falsos positivos (%)	2%	9%	8%	6%	5%	12%	6%
Verdaderos negativos (%)	4%	19%	6%	1%	40%	23%	1%
Tiempo por imagen (s)	0,66	0,67	0,80	0,73	0,70	0,86	0,70

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	2942	1885	2320	985	1059		**23158**
Detecciones incorrectas	591	85	610	464	351		**4575**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	506	77	361	460	345		**2878**
Verdaderos positivos (%)	83%	96%	79%	68%	75%		**84%**
Falsos positivos (%)	14%	4%	12%	32%	24%		**10%**
Verdaderos negativos (%)	2%	0%	8%	0%	0%		**6%**
Tiempo por imagen (s)	1,11	0,85	0,80	1,05	1,05		**0,83**

Tabla 12.3 – Resultados de *Eye_Tracker_VJ_Learning_Predator* sin filtrado de falsos positivos.

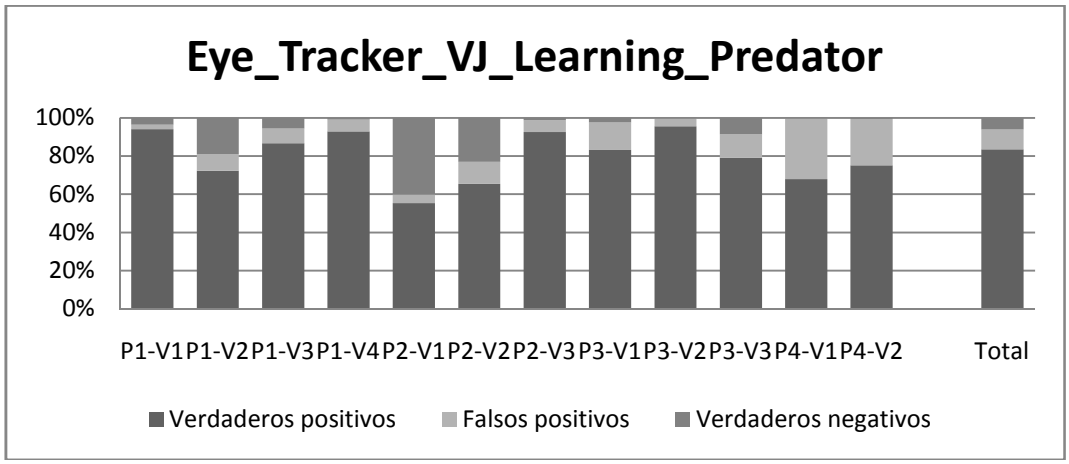


Figura 12.3 – Resultados de *Eye_Tracker_VJ_Learning_Predator* sin filtrado de falsos positivos.

Resultados de Eye_Tracker_VJ_Predator sin el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	1250	1377	2292	4748	421	1126	2452
Detecciones incorrectas	191	547	508	505	356	568	100
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	136	154	363	416	88	49	66
Verdaderos positivos (%)	87%	72%	82%	90%	54%	66%	96%
Falsos positivos (%)	9%	8%	13%	8%	11%	3%	3%
Verdaderos negativos (%)	4%	20%	5%	2%	34%	31%	1%
Tiempo por imagen (s)	0,61	0,56	0,78	0,69	0,63	0,82	0,67

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	2943	1858	2306	1138	1109		**23020**
Detecciones incorrectas	590	112	624	311	301		**4713**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	386	104	345	305	292		**2704**
Verdaderos positivos (%)	83%	94%	79%	79%	79%		**83%**
Falsos positivos (%)	11%	5%	12%	21%	21%		**10%**
Verdaderos negativos (%)	6%	0%	10%	0%	1%		**7%**
Tiempo por imagen (s)	1,00	0,85	0,74	0,86	0,86		**0,76**

Tabla 12.4 – Resultados de *Eye_Tracker_VJ_Predator* sin filtrado de falsos positivos.

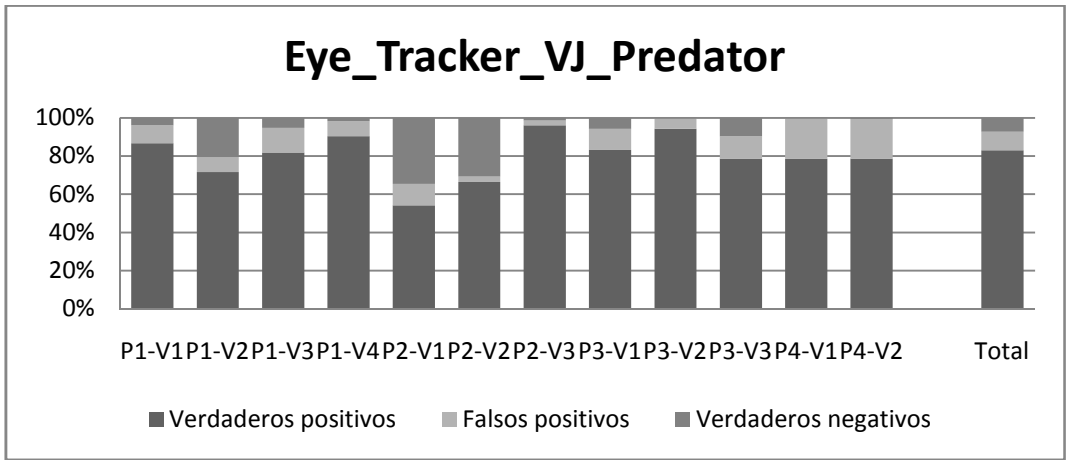


Figura 12.4 – Resultados de *Eye_Tracker_VJ_Predator* sin filtrado de falsos positivos.

Resultados de Eye_Tracker_Predator con el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	992	802	1780	4281	422	343	2088
Detecciones incorrectas	449	1122	1020	972	355	1351	464
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	91	80	298	353	7	495	135
Verdaderos positivos (%)	69%	42%	64%	81%	54%	20%	82%
Falsos positivos (%)	6%	4%	11%	7%	1%	29%	5%
Verdaderos negativos (%)	25%	54%	26%	12%	45%	51%	13%
Tiempo por imagen (s)	0,51	0,34	0,58	0,58	0,43	0,51	0,58

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	2380	1682	1674	70	0		**16514**
Detecciones incorrectas	1153	288	1256	1379	1410		**11219**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	155	105	87	16	0		**1822**
Verdaderos positivos (%)	67%	85%	57%	5%	0%		**60%**
Falsos positivos (%)	4%	5%	3%	1%	0%		**7%**
Verdaderos negativos (%)	28%	9%	40%	94%	100%		**34%**
Tiempo por imagen (s)	0,68	0,68	0,53	0,35	0,50		**0,55**

Tabla 12.5 – Resultados de *Eye_Tracker_Predator* con filtrado de falsos positivos.

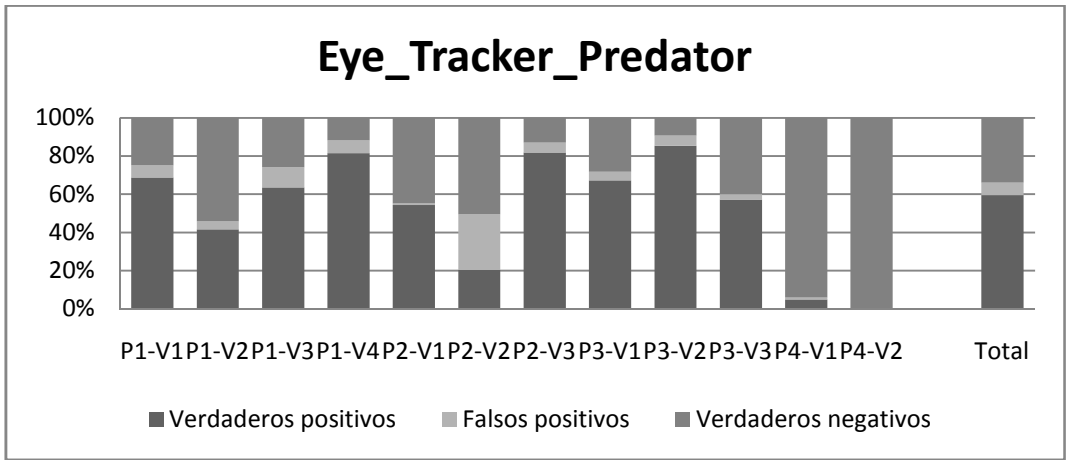


Figura 12.5 – Resultados de *Eye_Tracker_Predator* con filtrado de falsos positivos.

Resultados de Eye_Tracker_VJ con el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	1080	1268	1797	4208	372	899	2185
Detecciones incorrectas	361	656	1003	1045	405	795	367
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	5	28	36	70	8	17	40
Verdaderos positivos (%)	75%	66%	64%	80%	48%	53%	86%
Falsos positivos (%)	0%	1%	1%	1%	1%	1%	2%
Verdaderos negativos (%)	25%	33%	35%	19%	51%	46%	13%
Tiempo por imagen (s)	0,54	0,52	0,65	0,63	0,51	0,62	0,62

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	1443	1375	1815	522	687		**17651**
Detecciones incorrectas	2090	595	1115	927	723		**10082**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	22	13	56	76	62		**433**
Verdaderos positivos (%)	41%	70%	62%	36%	49%		**64%**
Falsos positivos (%)	1%	1%	2%	5%	4%		**2%**
Verdaderos negativos (%)	59%	30%	36%	59%	47%		**35%**
Tiempo por imagen (s)	0,71	0,68	0,85	0,64	0,69		**0,63**

Tabla 12.6 – Resultados de *Eye_Tracker_VJ* con filtrado de falsos positivos.

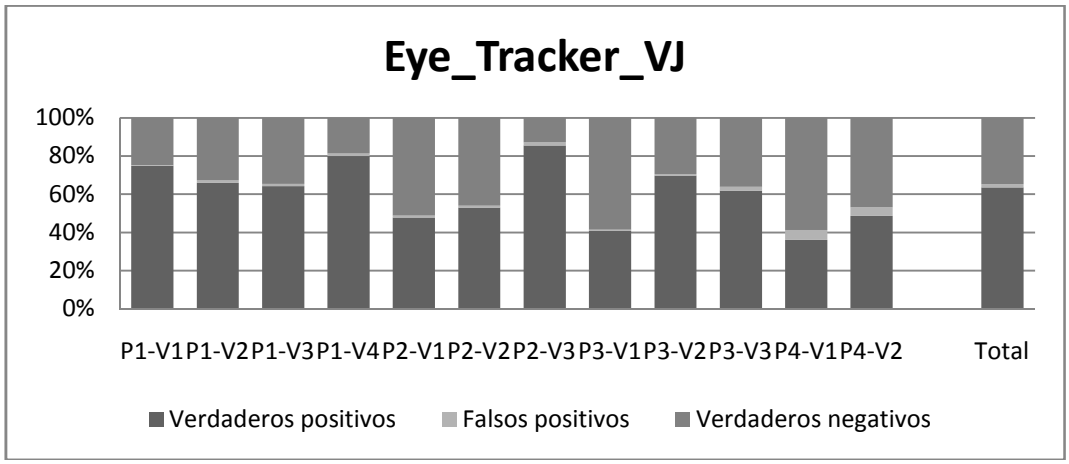


Figura 12.6 – Resultados de *Eye_Tracker_VJ* con filtrado de falsos positivos.

Resultados de Eye_Tracker_VJ_Learning_Predator con el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	1308	1378	2338	4688	420	1105	2369
Detecciones incorrectas	133	546	462	565	357	589	183
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	52	101	105	252	27	20	127
Verdaderos positivos (%)	91%	72%	84%	89%	54%	65%	93%
Falsos positivos (%)	4%	5%	4%	5%	3%	1%	5%
Verdaderos negativos (%)	6%	23%	13%	6%	42%	34%	2%
Tiempo por imagen (s)	0,66	0,67	0,80	0,74	0,71	0,83	0,67

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	2836	1833	2221	1141	1083		**22720**
Detecciones incorrectas	697	137	709	308	327		**5013**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	84	39	185	206	246		**1444**
Verdaderos positivos (%)	80%	93%	76%	79%	77%		**82%**
Falsos positivos (%)	2%	2%	6%	14%	17%		**5%**
Verdaderos negativos (%)	17%	5%	18%	7%	6%		**13%**
Tiempo por imagen (s)	1,08	0,85	0,80	1,05	1,03		**0,82**

Tabla 12.7 – Resultados de *Eye_Tracker_VJ_Learning_Predator* con filtrado de falsos positivos.

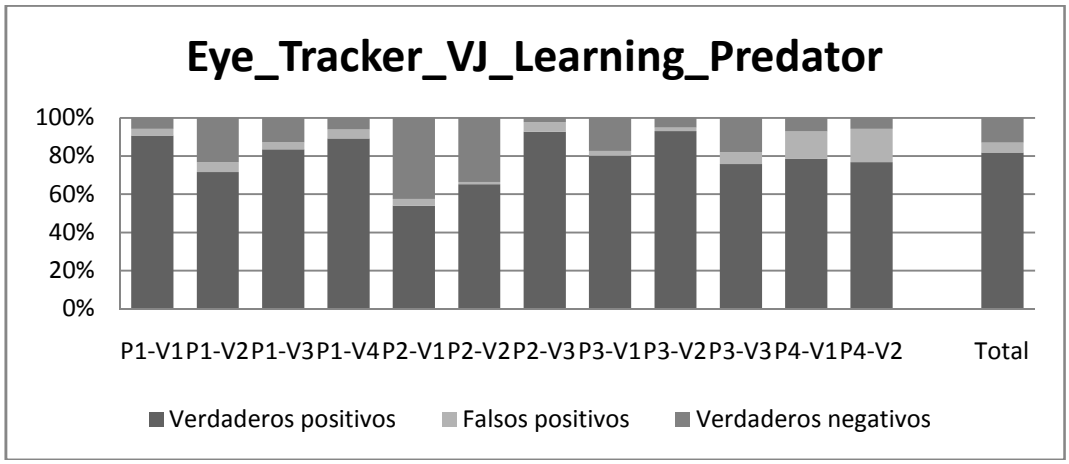


Figura 12.7 – Resultados de *Eye_Tracker_VJ_Learning_Predator* con filtrado de falsos positivos.

Resultados de Eye_Tracker_VJ_Predator con el filtrado de falsos positivos

	P1-V1	P1-V2	P1-V3	P1-V4	P2-V1	P2-V2	P2-V3
Detecciones correctas	1242	1374	2249	4673	413	1089	2387
Detecciones incorrectas	199	550	551	580	364	605	165
Total	1441	1924	2800	5253	777	1694	2552
Falsos positivos	35	69	121	309	16	19	100
Verdaderos positivos (%)	86%	71%	80%	89%	53%	64%	94%
Falsos positivos (%)	2%	4%	4%	6%	2%	1%	4%
Verdaderos negativos (%)	11%	25%	15%	5%	45%	35%	3%
Tiempo por imagen (s)	0,59	0,56	0,74	0,69	0,61	0,75	0,65

	P3-V1	P3-V2	P3-V3	P4-V1	P4-V2		**Total**
Detecciones correctas	2871	1815	2158	1143	1094		**22508**
Detecciones incorrectas	662	155	772	306	316		**5225**
Total	3533	1970	2930	1449	1410		**27733**
Falsos positivos	77	72	250	241	253		**1562**
Verdaderos positivos (%)	81%	92%	74%	79%	78%		**81%**
Falsos positivos (%)	2%	4%	9%	17%	18%		**6%**
Verdaderos negativos (%)	17%	4%	18%	4%	4%		**13%**
Tiempo por imagen (s)	0,96	0,81	0,69	0,86	0,85		**0,74**

Tabla 12.8 – Resultados de *Eye_Tracker_VJ_Predator* con filtrado de falsos positivos.

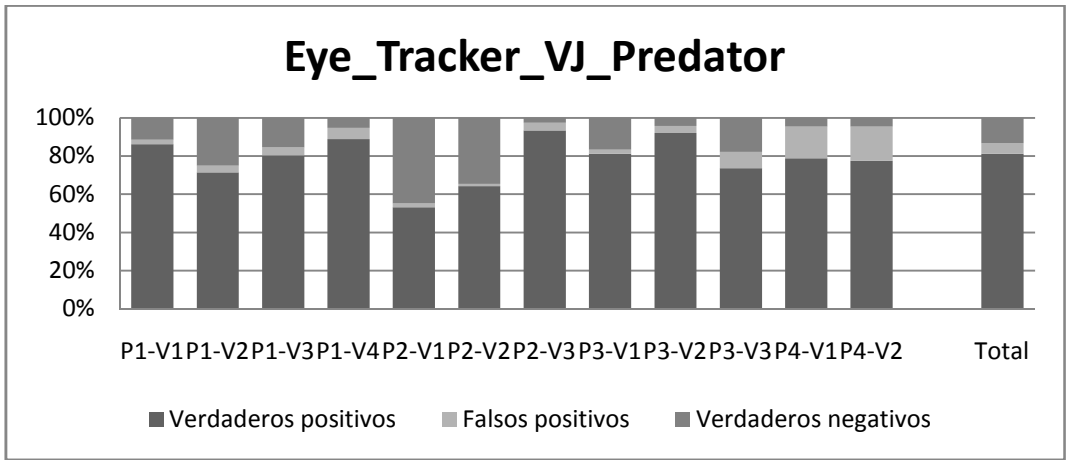


Figura 12.8 – Resultados de *Eye_Tracker_VJ_Predator* con filtrado de falsos positivos.

Tiempo promedio para cada imagen con y sin filtrado de falsos positivos

Se muestra a continuación una comparativa del tiempo de procesado de los algoritmos con los diferentes vídeos de la base de datos, tanto sin el filtrado de falsos positivos como con el filtrado.

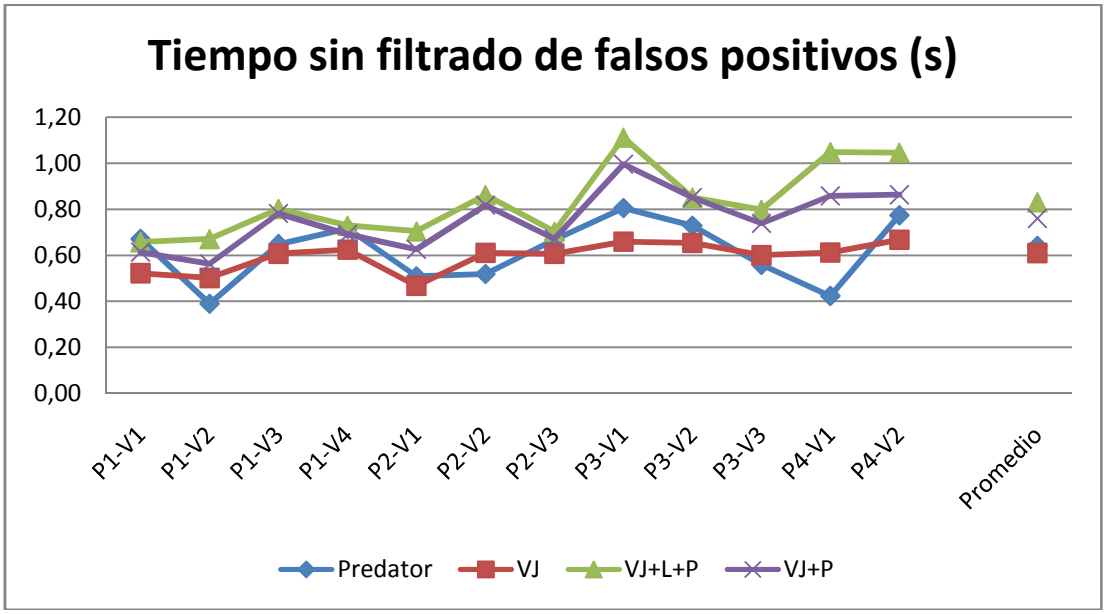


Figura 12.9 – Tiempo de ejecución sin filtrado de falsos positivos

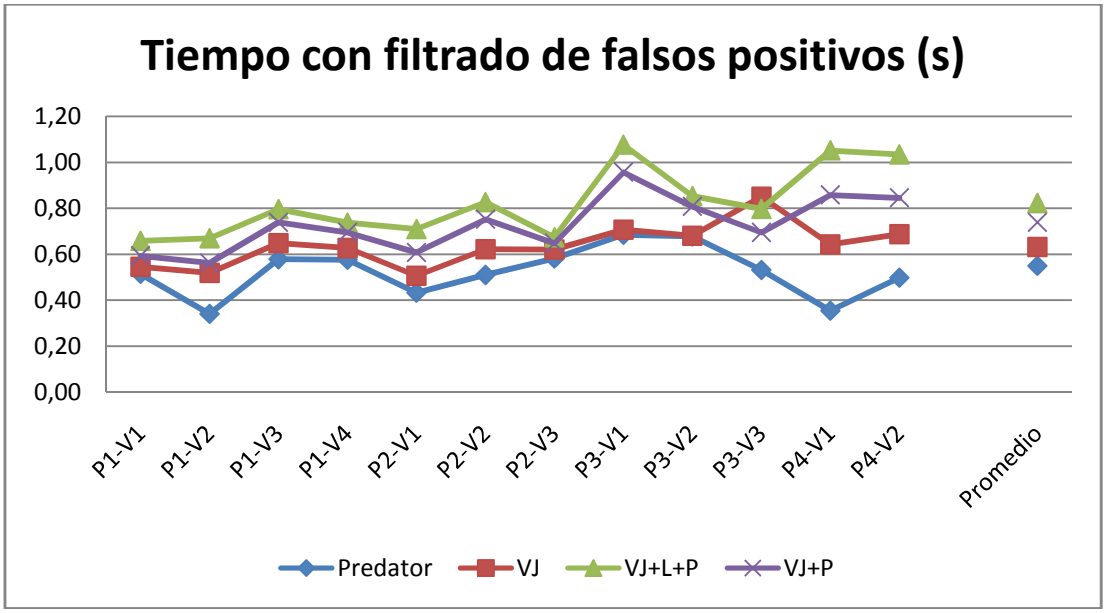


Figura 12.10 – Tiempo de ejecución con filtrado de falsos positivos

Resumen de los resultados

Se muestra a continuación los datos del procesado de los diferentes algoritmos con y sin el filtrado de falsos positivos sobre el total de imágenes procesadas.

	Eye_Tracker_ Predator		Eye_Tracker_VJ		Eye_Tracker_VJ_ Learning_Predator		Eye_Tracker_ VJ_Predator	
	No Filtrado	Filtrado	No Filtrado	Filtrado	No Filtrado	Filtrado	No Filtrado	Filtrado
	A1	A2	B1	B2	C1	C2	D1	D2
Detecciones correctas	15578	16514	17677	17651	23158	22720	23020	22508
Detecciones incorrectas	12155	11219	10056	10082	4575	5013	4713	5225
Total	27733	27733	27733	27733	27733	27733	27733	27733
Falsos positivos	8131	1822	528	433	2878	1444	2704	1562
Verdaderos positivos (%)	56%	60%	64%	64%	84%	82%	83%	81%
Falsos positivos (%)	29%	7%	2%	2%	10%	5%	10%	6%
Verdaderos negativos (%)	15%	34%	34%	35%	6%	13%	7%	13%
Tiempo por imagen (s)	0,64	0,55	0,61	0,63	0,83	0,82	0,76	0,74

Tabla 12.9 – Resumen de los resultados

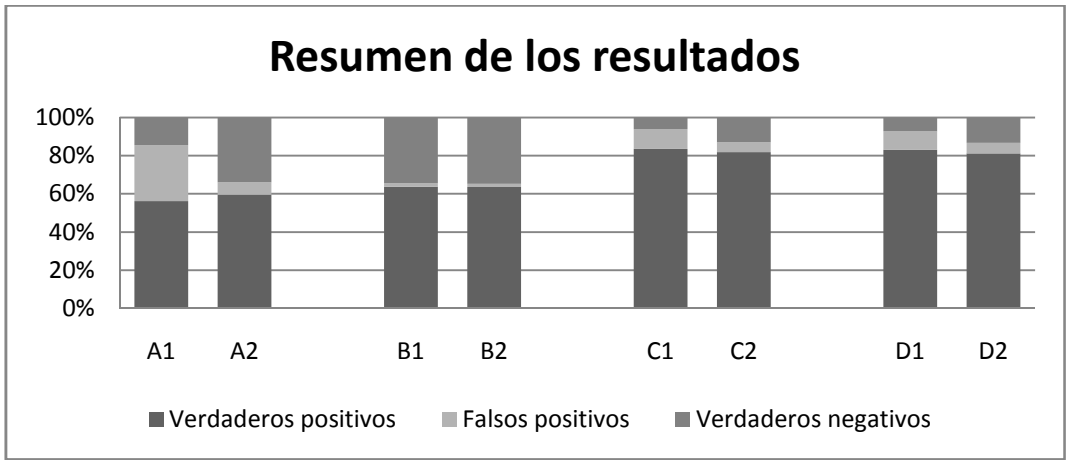


Figura 12.11 – Resumen de los resultados

13 – ANÁLISIS DE LOS RESULTADOS

13.1 – Análisis cuantitativo

Tras procesar las imágenes se ha cuantificado el número de imágenes procesadas correcta e incorrectamente en cada vídeo. Además se ha contabilizado el número de falsos positivos (resultados que el programa considera correctos pero que no lo son) y el tiempo de ejecución promedio de cada imagen.

13.1.1 – Eye_Tracker_Predator

Antes del filtrado de falsos positivos:
Eye_Tracker_Predator funciona correctamente en el 56% de los casos, e incorrectamente en el 44%.Esta proporción se mantiene en la mayoría de los casos. El índice de falsos positivos es del 29%. Del total de las imágenes *Eye_Tracker_Predator* genera resultados en el 85% de los casos, y en el restante 15% no devuelve nada. En ese 85% está incluido el 56% de verdaderos positivos y el 29% de falsos positivos. Es decir, *Eye_Tracker_Predator* genera resultados para el 85% de las imágenes pero aproximadamente uno de cada tres resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.64 segundos.

Después del filtrado de falsos positivos:
A pesar de que el filtrado tiene como objetivo la reducción de los falsos positivos, el índice de imágenes procesadas correctamente ha aumentado del 56% al 60%. Al eliminar parte de las detecciones erróneas, el programa tiene en la memoria de detecciones que considera correctas detecciones más fiables que sin el filtrado, por lo que la detección es mejor que en el caso anterior. El índice de falsos positivos ha descendido de 29% a 7%, una reducción muy significativa. Tras el filtrado, *Eye_Tracker_Predator* genera resultados para el 67% de las imágenes y aproximadamente uno de cada nueve resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.55 segundos, menor que sin el filtrado. Esta reducción de tiempo puede deberse a que la eliminación de falsos positivos supone un menor número de datos a tener en cuenta en cada procesado.

13.1.2 – Eye_Tracker_VJ

Antes del filtrado de falsos positivos:
Eye_Tracker_VJ funciona correctamente en el 64% de los casos, e incorrectamente en el 36%. Esta proporción no es constante en los diferentes vídeos. El número de falsos positivos es del 2%, un índice realmente bueno. Del total de las imágenes *Eye_Tracker_VJ* genera resultados en el 66% de los casos, y en el restante 34% no devuelve nada. En ese 66% está incluido el 64% de verdaderos positivos y el 2% de falsos positivos. Es decir, *Eye_Tracker_VJ* genera resultados para el 66% de las imágenes y aproximadamente uno de cada treinta resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.61 segundos, ligeramente menor que *Eye_Tracker_Predator*.

Después del filtrado de falsos positivos:
El índice de detecciones correctas e incorrectas y el índice de falsos positivos se mantiene constante tras el filtrado. Si bien es cierto que se ha reducido el número de falsos positivos, se trata de una reducción tan pequeña que no se ve reflejada en los índices. Es decir, *Eye_Tracker_VJ* genera resultados para el 66% de las imágenes y aproximadamente uno de cada cuarenta resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.63 segundos, 0.02 segundos mayor que sin el filtrado.

13.1.3 – Eye_Tracker_VJ_Learning_Predator

Antes del filtrado de falsos positivos:

Eye_Tracker_VJ_Learning_Predator funciona correctamente en el 84% de los casos, e incorrectamente en el 16%. Esta proporción no es constante en los diferentes vídeos. El número de falsos positivos es del 10%. Del total de las imágenes *Eye_Tracker_VJ_Learning_Predator* genera resultados en el 94% de los casos, y en el restante 6% no devuelve nada. En ese 94% está incluido el 84% de verdaderos positivos y el 10% de falsos positivos. Es decir, *Eye_Tracker_VJ_Learning_Predator* genera resultados para el 94% de las imágenes y aproximadamente uno de cada nueve resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.83 segundos, 0.2 segundos más que los anteriores. Este aumento se debe por un lado al *Learning*, y por otro a que las imágenes en las que *facedetector* no detecta la cara se vuelven a procesan con *Predator*, suponiendo esto dos procesados para la misma imagen.

Después del filtrado de falsos positivos:

El índice de verdaderos positivos se ha reducido del 84% al 82%. El índice de falsos positivos ha pasado del 10% al 5%. Tras el filtrado, *Eye_Tracker_VJ_Learning_Predator* genera resultados para el 87% de las imágenes y aproximadamente uno de cada dieciocho resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.82 segundos, prácticamente igual que sin el filtrado. A pesar de la reducción de un 2% de los verdaderos positivos en este método tras el filtrado, el filtrado supone un aumento de un 4% de verdaderos positivos en *Eye_Tracker_Predator* y una reducción significativa de falsos positivos en todos los casos, así que se opta por mantener el programa de filtrado.

13.1.4 – Eye_Tracker_VJ_Predator

Antes del filtrado de falsos positivos:

Eye_Tracker_VJ_Predator funciona correctamente en el 83% de los casos, e incorrectamente en el 17%. Esta proporción no es constante en los diferentes vídeos. El número de falsos positivos es del 10%. Del total de las imágenes *Eye_Tracker_VJ_Predator* genera resultados en el 93% de los casos, y en el restante 7% no devuelve nada. En ese 93% está incluido el 83% de verdaderos positivos y el 10% de falsos positivos. Es decir, *Eye_Tracker_VJ_Predator* genera resultados para el 93% de las imágenes y aproximadamente uno de cada nueve resultados es erróneo. El tiempo promedio de procesado de una imagen es 0.76 segundos, menor que *Eye_Tracker_VJ_Learning_Predator.* Se aprecia aquí la diferencia que supone el Learning en el tiempo de ejecución.

Después del filtrado de falsos positivos:

El índice de imágenes procesadas correctamente se ha reducido del 83% al 81%. El índice de falsos positivos ha pasado del 10% al 6%. Tras el filtrado, *Eye_Tracker_VJ_Predator* genera resultados para el 87% de las imágenes y aproximadamente uno de cada quince resultados es erróneo. *Eye_Tracker_VJ_Predator* genera resultados un 1% peor que *Eye_Tracker_VJ_Learning_Predator*, lo cual parece deberse a que éste último realiza el aprendizaje de todas las detecciones correctas. El tiempo promedio de procesado de una imagen es 0.74 segundos, ligeramente menor al tiempo sin el filtrado.

13.2 – Análisis cualitativo

Tras procesar los vídeos se han analizado los problemas que presenta cada programa. En general, el programa que más problemas presenta es *Eye_Tracker_Predator*, muy susceptible a los cambios de iluminación. *Eye_Tracker_VJ* presenta un índice de detecciones ligeramente mayor, y éstas son significativamente mejores. *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* mejoran mucho con respecto a los anteriores en cuanto a detecciones correctas, aunque a costa de un tiempo de procesado mayor. El programa de filtrado de falsos positivos cumple su cometido y reduce los falsos positivos en todos los vídeos procesados, de manera especial en los procesados con *Eye_Tracker_Predator*, que es el que más errores produce.

El principal problema de *Eye_Tracker_Predator* es que la detección de la cara se realiza únicamente en la primera imagen. Si esa detección no es correcta y define como zonas a seguir objetos que no son los ojos, en las sucesivas imágenes busca esos objetos que no son los ojos, resultando en una detección inútil. El resto de programas son más robustos en este aspecto, ya que en sucesivas imágenes vuelven a detectar la cara y pueden corregir el error.

Figura 13.1 – Ejemplo de detección facial errónea en la primera imagen (izquierda) y detección errónea en imagen sucesiva (derecha)

Otro gran problema que afecta a *Eye_Tracker_Predator* es la variación rápida de la iluminación. Ante un cambio brusco de iluminación el aspecto de los ojos varía significativamente, con la aparición de brillos y sombras que impiden a *predator* reconocer las características de las regiones que está siguiendo. Esto puede suponer tanto que no haya detección, lo que simplemente supone un menor número de verdaderos positivos, como que la detección sea errónea, aumentando la tasa de falsos positivos y reduciendo la fiabilidad del detector.

Figura 13.2 – Ejemplo de detección correcta (izquierda) y pérdida del objetivo por exceso de iluminación lateral (derecha)

El problema de la iluminación variable no es único de *Eye_Tracker_Predator*, también afecta al resto de programas. *Eye_Tracker_VJ*, si bien también se ve afectado, responde mejor ante la variación de iluminación siempre que ésta no afecte a la totalidad de la cara. Si la iluminación

aumenta hasta saturar una pequeña parte de la imagen de la cara, *Eye_Tracker_VJ* es capaz de reconocer las características faciales hasta cierto punto, aunque es cierto que si la saturación se produce en al menos media cara la detección falla.

Figura 13.3 – Ejemplo de detección correcta (izquierda) y pérdida del objetivo por exceso de iluminación frontal (derecha)

Eye_Tracker_VJ_Learning_Predator y *Eye_Tracker_VJ_Predator* mejoran significativamente en este aspecto, aunque es cierto que en condiciones extremas de iluminación no son infalibles. Por un lado realizan las mismas detecciones que *Eye_Tracker_VJ*, y por otro utilizan *predator* con muchas más imágenes de referencia correctas, derivando en una detección más fiable aunque más lenta.

Otro problema relacionado con la iluminación es el caso en el que el sol incide sobre la cámara. En este caso tanto *Eye_Tracker_Predator* como *Eye_Tracker_VJ* sufren fallos en la detección, aunque de manera bastante puntual. Tanto *Eye_Tracker_VJ_Learning_Predator* como *Eye_Tracker_VJ_Predator* corrigen este error con el doble procesado.

Figura 13.4 – Ejemplo de detección correcta (izquierda) y pérdida del objetivo por el sol incidente en la cámara (derecha)

Eye_Tracker_VJ utiliza como programa base el detector de caras entrenado para reconocer caras mirando de frente. Por ello, si la persona gira la cabeza un ángulo suficiente la detección no se realiza. Se estima que un ángulo de giro de 30 grados es suficiente para provocar el error en la detección. *Eye_Tracker_Predator* es capaz de seguir los ojos en caras con mayor inclinación siempre que el giro se realice de forma lenta. Tanto *Eye_Tracker_VJ_Learning_Predator* como *Eye_Tracker_VJ_Predator* combinan una buena detección de la cara con un buen tracking en movimientos lentos, por lo que nuevamente resultan más robustos que los anteriores. En cualquier caso, si el giro de la cabeza es tal que no se ven los ojos, ninguno de los programas los puede detectar.

Figura 13.5 – Ejemplo de detección correcta (izquierda) y pérdida por giro de cabeza (derecha)

En este caso extremo, se da tanto el caso de no obtener resultados como obtenerlos en un lugar erróneo.

Figura 13.6 – Ejemplo de detección correcta (izquierda) y detección incorrecta (derecha)

Un problema recurrente de *Eye_Tracker_Predator* en condiciones de iluminación no homogénea, como puede ser iluminación elevada en una mitad del ojo y sombra en la otra mitad, es una gran variación del tamaño de los bounding box. *Eye_Tracker_VJ* supera este problema al generar los dos bounding box del mismo tamaño. *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* utilizan *predator*, por lo tanto también son susceptibles de cometer este error, pero al utilizar *facedetector* como detector principal las probabilidades se reducen mucho.

Figura 13.7 – Ejemplo de detección correcta (izquierda) y reducción del tamaño de bounding box (derecha)

Eye_Tracker_VJ establece una disposición de los bounding box coherente, donde el bounding box del ojo izquierdo está a la izquierda del derecho. *Eye_Tracker_Predator* tiene una mayor libertad al procesar los dos ojos de manera independiente, y dado el gran parecido entre los dos ojos, en algunas de las ocasiones en las que un ojo está oculto el programa detecta el ojo que no le corresponde. *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* utilizan *predator* en menor medida, por lo que la probabilidad de que esto ocurra se reduce, aunque no se elimina completamente.

Figura 13.8 – Ejemplo de detección correcta (izquierda) y detección centrada en un mismo ojo (derecha)

Debido también a esa independencia en el procesado de los dos ojos, en ocasiones un bounding box se centra en un objeto que no es el ojo y que además está a una distancia muy grande del otro bounding box.

Figura 13.9 – Ejemplo de detección correcta (izquierda) y detección de un objeto que no es un ojo (derecha)

Un error que se ha dado de forma muy puntual es el de la detección de los ojos de la persona errónea. Este error se produce si la cara del usuario no se ve completa pero la de la segunda persona sí. Se puede evitar modificando el encuadre de la cámara.

Figura 13.10 – Ejemplo de detección correcta (izquierda) y detección en la persona incorrecta (derecha)

Finalmente, se ha observado que *Eye_Tracker_VJ* no detecta la cara en algunas ocasiones en las que el usuario parpadea. Este problema se evita con el uso de *predator*, como con *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator.*

Figura 13.11 – Ejemplo de detección correcta (izquierda) y no detección debida al parpadeo (derecha)

El programa de filtrado de falsos positivos resulta muy útil a la hora de detectar y eliminar los resultados erróneos. El programa que mejora más con el filtrado es *Eye_Tracker_Predator*, ya que la mayoría de los resultados erróneos que produce son los relativos a tamaños o posiciones entre bounding box incoherentes, o por detecciones de un único ojo.

Eye_Tracker_VJ mejora ligeramente tras el filtrado, eliminándose principalmente las detecciones que difieren significativamente de la primera, como puede ser la detección de la cara de la persona incorrecta.

Eye_Tracker_VJ_Learning_Predator y *Eye_Tracker_VJ_Predator* también mejoran sus índices de falsos positivos con el filtrado. Principalmente se eliminan las detecciones erróneas producidas por *predator*, relativas a tamaño o posición relativa de los bounding box, y en menor medida las relativas a *facedetector*, como la detección de la cara de la persona incorrecta.

En general, *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* mejoran significativamente con respecto a *Eye_Tracker_Predator* y *Eye_Tracker_VJ*, a pesar de necesitar un tiempo de procesado mayor. Si bien es cierto que sus índices de falsos positivos son superiores al de *Eye_Tracker_VJ*, son bastante bajos.

Eye_Tracker_VJ_Learning_Predator y *Eye_Tracker_VJ_Predator*, junto con la función de filtrado de falsos positivos, han resultado los más robustos en la detección de los ojos, con una cantidad de detecciones correctas muy elevada y una cantidad de detecciones incorrectas muy baja.

Ejemplos

Se muestra a continuación una serie de ejemplos en los que *Eye_Tracker_Predator* y *Eye_Tracker_VJ* son incapaces de detectar los ojos (columna izquierda) pero *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* lo hacen de manera correcta (columna derecha).

Figura 13.12 – Ejemplo de detecciones no realizadas por *Eye_Tracker_Predator* y *Eye_Tracker_VJ* (izquierda) y detecciones correctas por *Eye_Tracker_VJ_Learning_Predator* y *Eye_Tracker_VJ_Predator* (derecha)

Índice de confianza

El índice de confianza es un valor que genera *Predator* en cada detección que indica cuánto se parece el objeto detectado a los objetos que ha clasificado como válidos hasta ese momento. Tras el análisis se observa que en general las detecciones correctas tienen un índice de confianza medio-alto y las detecciones erróneas medio-bajo. Sin embargo, este índice no es fiable en todos los casos por lo que no podemos tomarlo como un criterio fiable para filtrar los resultados.

Figura 13.13 – Ejemplo de detección buena con un índice de confianza bajo (0.35 y 0.58)

Figura 13.14 – Ejemplo de detección errónea con un índice de confianza alto (0.78 y 0.95)

14 – DETECCIÓN DEL IRIS

14.1 – Introducción

Como complemento a la detección de la zona de los ojos se ha añadido una función de detección de iris previamente implementada. Permite la detección del iris tanto por el método de Timm-Barth [10] como por el de Wang [11].

14.2 – Funcionamiento

A partir de dos imágenes de ojos, estima la posición del iris en esos ojos.
El método de Timm-Barth estima la posición del centro del ojo mediante el cálculo de gradientes. El método de Wang localiza el centro del ojo mediante un filtro de varianza.

Figura 14.1 – Ejemplo de imagen a procesar (izquierda) y detección de los centros de los ojos (derecha)

14.3 – Análisis

Se ha aplicado la detección de iris a un conjunto de 1872 imágenes con los ojos correctamente visibles. La detección de la pupila ha sido correcta en 1639 imágenes por el método de Timm-Barth y en 1312 imágenes por el método de Wang. Con un índice de acierto de un 88% frente a un 70%, y un tiempo promedio por imagen de 0.31s frente a 0.36s, se opta por utilizar el método de Timm-Barth para la detección del centro del iris.

15 – APLICACIÓN DE SEGURIDAD

15.1 – Introducción

Como aplicación secundaria enfocada a la seguridad en la conducción, se ha implementado una función que alerta al usuario en caso de no detectarse los ojos durante un determinado tiempo.

15.2 – Funcionamiento

El programa alerta al usuario cuando ha pasado cierto tiempo sin detectar los ojos del usuario. Para ello calcula el número de imágenes tomadas en ese intervalo de tiempo y comprueba las detecciones de ojos en esas imágenes. En caso no haberse detectado los ojos en ninguna de esas imágenes genera una señal sonora de 1Khz de 0.1 segundos de duración. El hecho de analizar las detecciones en función del tiempo en lugar de un número fijo de imágenes implica un funcionamiento similar independientemente de la velocidad del procesador. Para el análisis visual del programa se muestra en la imagen un círculo rojo a la vez que se emite la alarma.

A modo de ejemplo, el programa emite el sonido de alerta si el usuario inclina la cabeza lo suficiente para no detectarle los ojos durante un tiempo superior al establecido, como en el caso de quedarse dormido.

Figura 15.1 – Ejemplo de alerta al usuario tras quedarse dormido al volante

16 – CONCLUSIONES Y LÍNEAS FUTURAS

16.1 – Conclusiones

El eye tracking es un concepto ampliamente desarrollado en laboratorios con dispositivos muy complejos, como luces de infrarrojos y cámaras de muy alta resolución. En este proyecto se han desarrollado cuatro funciones de eye tracking diseñadas para ser utilizadas en condiciones de luz natural y una cámara web de baja resolución, como en un entorno propio de la conducción en un vehículo.

Como punto de partida se ha analizado el funcionamiento del programa de tracking *OpenTLD / Predator*, programa capaz de realizar la detección, tracking y aprendizaje on-line de un objeto en una serie de imágenes.

También se ha analizado el programa de detección facial *facedetector*, basado en el algoritmo de detección de objetos de Viola-Jones.

A partir de estos dos programas se han implementado cuatro funciones de detección y tracking de los ojos, diferentes entre sí, utilizando cada una las dos funciones anteriores en mayor o menor medida.

La función *Eye_Tracker_Predator*, que realiza el tracking de los ojos a partir de la detección facial de la primera imagen, proporciona un índice de detecciones válidas del 60%, con un 7% de falsos positivos, y un tiempo promedio por imagen de 0.55 segundos. Es la función más rápida de las cuatro, pero su eficacia es la más baja.

La función *Eye_Tracker_VJ*, que realiza la detección facial de todas las imágenes, proporciona un índice de detecciones válidas del 64%, con un 2% de falsos positivos, y un tiempo promedio por imagen de 0.63 segundos. Es más lenta que la función anterior pero proporciona una mayor fiabilidad en la detección.

La función *Eye_Tracker_VJ_Learning_Predator*, que realiza la detección facial de todas las imágenes, aprende las detecciones válidas, y realiza el tracking en las incorrectas, proporciona un índice de detecciones válidas del 82%, con un 5% de falsos positivos, y un tiempo promedio por imagen de 0.82 segundos. Es la función más lenta, pero a su vez las más fiable de las cuatro.

La función *Eye_Tracker_VJ_Predator*, que realiza la detección facial de todas las imágenes y realiza el tracking en las detecciones incorrectas, proporciona un índice de detecciones válidas del 81%, con un 6% de falsos positivos, y un tiempo promedio por imagen de 0.74 segundos. Proporciona también unos resultados muy buenos y un tiempo de procesado algo menor que la anterior.

Para aumentar la fiabilidad de las funciones se ha implementado una función que filtra los resultados en base a las características faciales. Como se observa en el análisis de los resultados, esta función reduce significativamente los índices de falsos positivos.

Se ha analizado e incorporado una función de detección del centro del iris previamente implementada.

Como aplicación práctica orientada a la seguridad en la conducción, se ha implementado una función que alerta al usuario en caso de no detectar sus ojos durante un determinado tiempo.

16.2 – Líneas futuras

El presente proyecto ha sido programado utilizando MatLab por las ventajas que éste presenta para desarrollar prototipos. El principal problema de utilizar MatLab es su lentitud, que lo hace inapropiado para utilizarlo en tiempo real. Por ello, sería apropiado utilizar un lenguaje más rápido, como C o JAVA.

Por otro lado, en la tarea del eye tracking los programas de este proyecto terminan en la detección de los centros de los iris. Para completarlos, sería interesante incorporar una función de estimación de la mirada a partir de esos centros de iris, para obtener unas funciones de eye tracking completas.

17 – REQUISITOS DEL SISTEMA

17.1 – Programas

Se ha utilizado un ordenador de las siguientes características y software:

- Pentium (R) 4 CPU 3.4 GHz, 2.0 GB de RAM
- Microsoft Windows XP, Service Pack 3
- MatLab R2010a
- Microsoft Visual Studio 2010
- OpenCV 2.2
- Parche VS2010MEXSupport

17.2 – Instalación

Como primer paso se debe instalar MatLab 2010a o 2011a.Para el desarrollo de este proyecto se han utilizado ambas versiones con buenos resultados. No se ha probado la compatibilidad con versiones más antiguas o más recientes, así que en caso de utilizar una versión diferente a las anteriores es recomendable visitar el portal web de MatLab y *OpenTLD* para comprobar la compatibilidad.

Se debe instalar también Microsoft Visual Studio 2010. Esto es debido a que *OpenTLD* utiliza algunas funciones en lenguaje C++, que para ser ejecutadas por MatLab deben ser compiladas en formato mex.

Finalmente es necesario instalar OpenCV 2.2, que contiene una serie de funciones utilizadas por *facedetector*.

Para la compilación de los archivos mex se ejecuta en MatLab la instrucción "*mex -setup*". De entre los compiladores que se muestran se elige "Microsoft Visual C++ 2010". Es posible que no se muestre ese compilador en la versión 2010a, por lo que sería necesaria la instalación del parche *VS2010MEXSupport*, que proporciona la compatibilidad entre ambos programas. Tras la elección del compilador se ejecuta el archivo "*compile.m*", del directorio mex_2010 o mex_2011, dependiendo de si se utiliza la versión de MatLab 2010 ó 2011. Si la compilación es correcta se indicará con el mensaje "*Compilation finished*".

Finalizada la parte de instalación y comprobación de compatibilidades, llega el turno de configurar la cámara. Con la cámara conectada, ejecutar "*imaqhwinfo*" para ver los adaptadores instalados. Para obtener las características del adaptador de la cámara, por ejemplo "*winvideo*". Después ejecutar "*a=imaqhwinfo('nombre del adaptador')*", por ejemplo "*a=imaqhwinfo('winvideo)*". Finalmente, ejecutar "*a.DeviceInfo*", y se mostrará el formato del vídeo capturado. En el archivo "initcamera.m" elegir la fuente de vídeo que se corresponda con el formato del vídeo anterior.

Finalmente, ejecutar cualquiera de las cuatro funciones principales de este proyecto, indicando en el archivo de configuración "*opt_init.m*" que la fuente de vídeo es la cámara web, asignando el valor "1" al parámetro "*camera*".

En caso de utilizar un sistema operativo o versión de MatLab diferentes se debe consultar la página web de *OpenTLD* [6].

18 – REFERENCIAS BIBLIOGRÁFICAS

1 - http://en.wikipedia.org/wiki/Eye_tracking

2 - http://nosolousabilidad.com/articulos/eye-tracking.htm

3 - http://es.wikipedia.org/wiki/Detección_de_caras

4 - http://en.wikipedia.org/wiki/Viola–Jones_object_detection_framework

5 - http://es.wikipedia.org/wiki/Seguimiento_de_objetos

6 - https://github.com/zk00006/OpenTLD

7 - http://en.wikipedia.org/wiki/Lucas–Kanade_method

8 - http://en.wikipedia.org/wiki/Barnsley_fern

9 - http://en.wikipedia.org/wiki/OpenCV

10 - Fabian Timm and Erhardt Barth, Accurate eye centre localisation by means of Gradients

11 - Victoria Ponz, Arantxa Villanueva, Laura Sesma, Mikel Ariz, Rafael Cabeza, "Topography-Based Detection of the Iris Centre Using Multiple-Resolution Images," imvip, pp.32-37, 2011 Irish Machine Vision and Image Processing Conference, 2011

12 - Raquel Solá Asenjo, “Evaluación experimental de detectores de caras y características faciales”, enero 2011.

DESARROLLO DE APLICACIONES DE DETECCIÓN DE OJOS BASADAS EN ALGORITMOS DE DETECCIÓN FACIAL Y TRACKING

José Javier Bengoechea Irañeta

OBJETIVOS

- Análisis del programa de tracking *Predator/TLD*.
- Análisis del programa de detección facial *facedetector*.
- Implementación de 4 funciones de detección de ojos en entorno de iluminación variable.
- Implementación de una función de filtrado para la reducción de falsos positivos y aumentar la fiabilidad de las detecciones.
- Análisis e incorporación de un programa de detección del iris.
- Implementación de un programa de alerta orientado a la seguridad en la conducción.

PREDATOR / TLD

- Es un algoritmo de tracking (rastreo o seguimiento). Combina las funciones de tracking, detección y aprendizaje, es decir, además de detectar y seguir un objeto en una serie de imágenes es capaz de aprender online.
- Tracking: método de Lucas-Kanade
- Detección: ventana deslizante
- Learning: clasificación de objetos válidos y objetos no válidos
  aprendizaje online

PREDATOR / TLD

- Ejemplo:

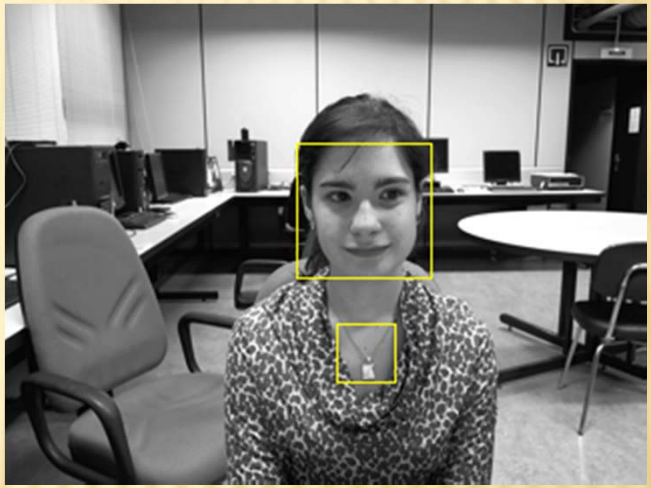
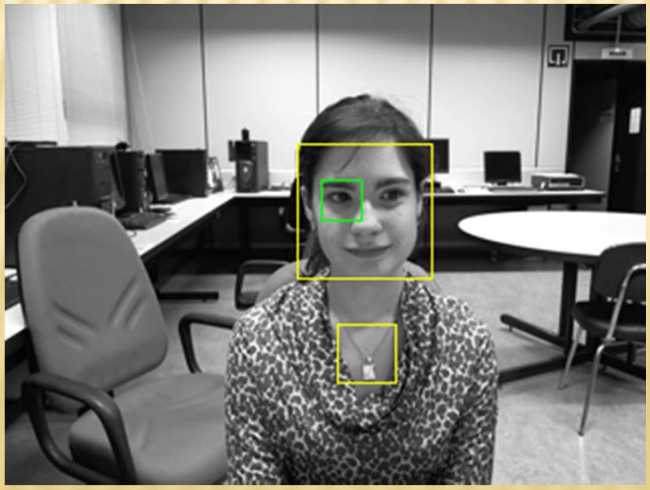
FACEDETECTOR

- Es un programa de detección facial basado en el algoritmo de detección de patrones de Viola-Jones. Localiza la cara en una imagen y recorta la zona de los ojos según conocimientos estructurales de la cara para devolver la zona de interés.

- Combina la detección de caras y de ojos para una mayor fiabilidad.

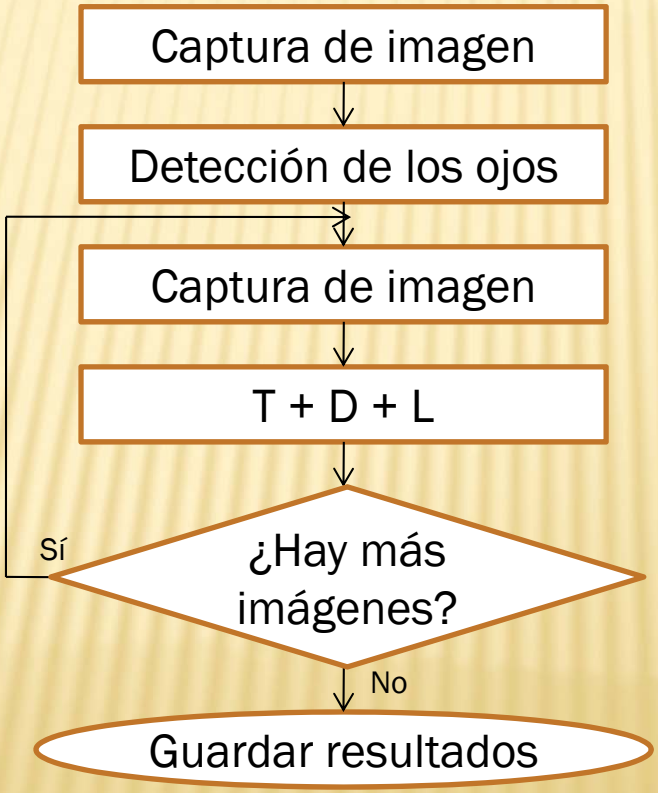
FACEDETECTOR

- Ejemplo:

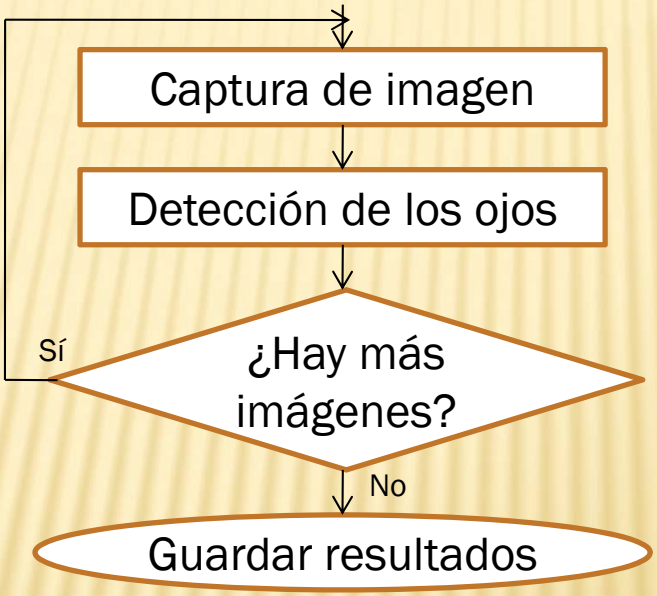

EYE_TRACKER_PREDATOR

- Calcula la posición de los ojos en la primera imagen con *facedetector* y realiza el tracking en las sucesivas imágenes con *Predator*.



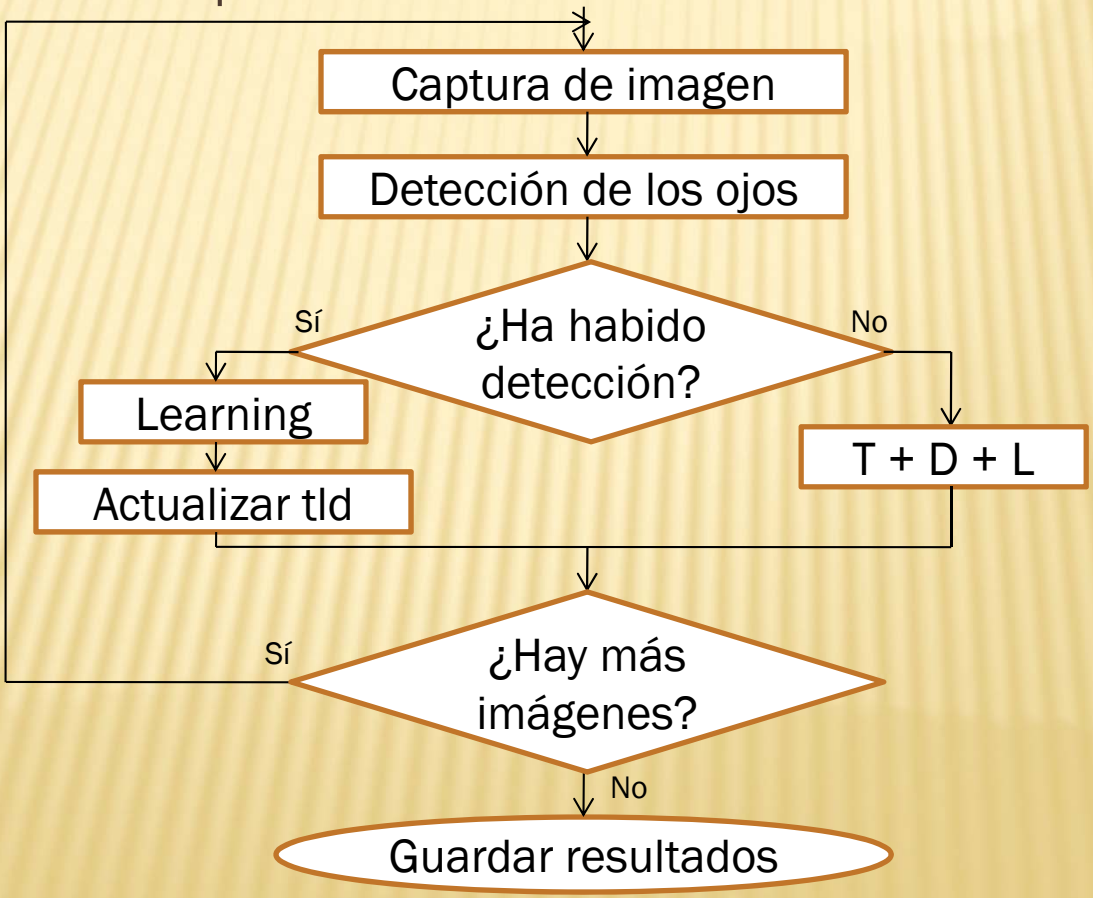
EYE_TRACKER_VJ

- Calcula la posición de los ojos utilizando *facedetector* en todas las imágenes.



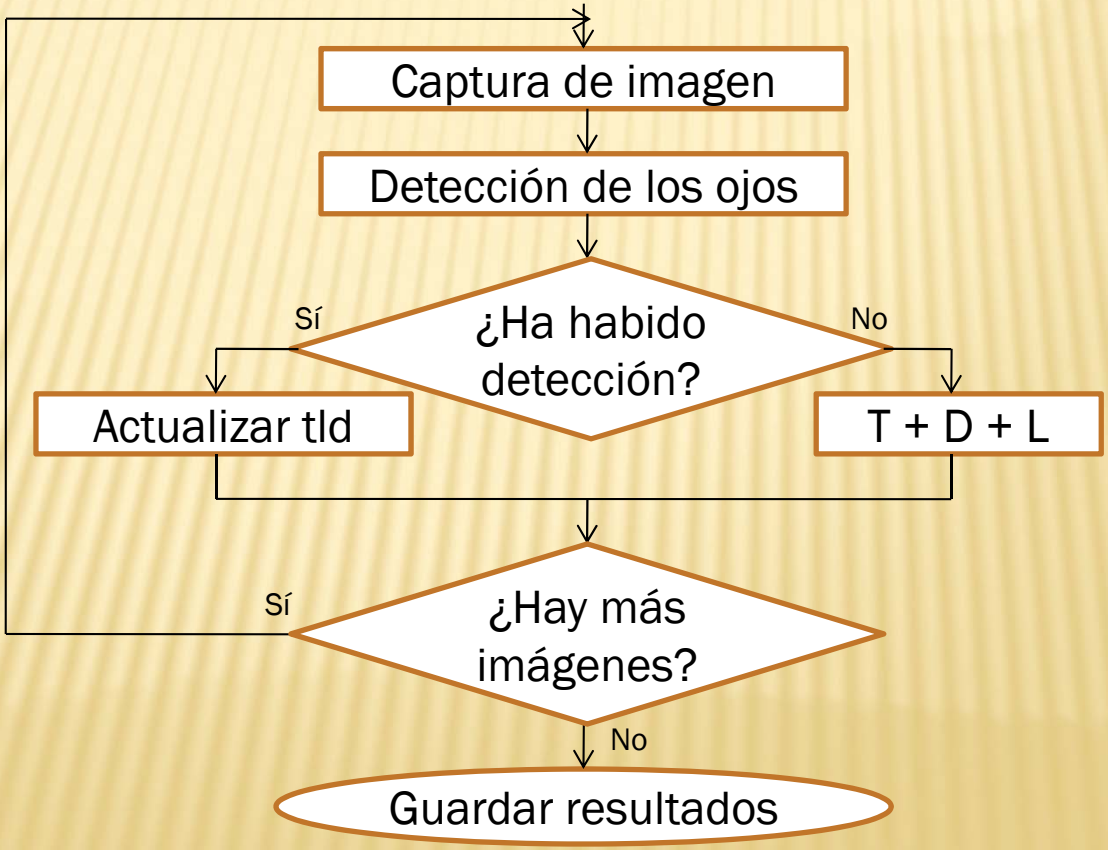
EYE_TRACKER_VJ_LEARNING_PREDATOR

- Calcula la posición de los ojos utilizando *facedetector* en todas las imágenes, aprende esas detecciones y utiliza *predator* en aquellas imágenes en las que *facedetector* no detecta la cara.



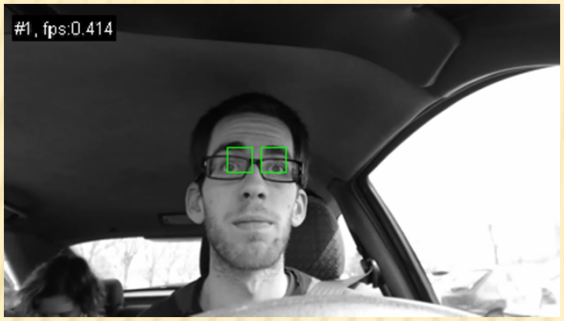
EYE_TRACKER_VJ_PREDATOR

- Calcula la posición de los ojos utilizando *facedetector* en todas las imágenes y utiliza *predator* en aquellas imágenes en las que *facedetector* no detecta la cara.



PROBLEMAS

- *Eye_Tracker_Predator*: Si la detección de los ojos en la primera imagen es incorrecta, el tracking en las sucesivas imágenes también es incorrecto:

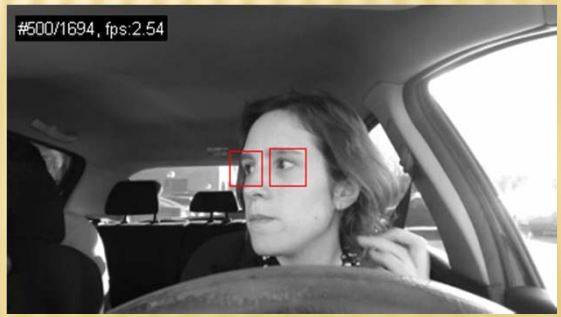


- Una iluminación elevada o variaciones rápidas provocan errores en la detección:

PROBLEMAS

- Cuando la persona no mira de frente, *facedetector* es incapaz de detectar la cara:

- Predator supera esa limitación siempre que el giro se produzca de manera lenta:

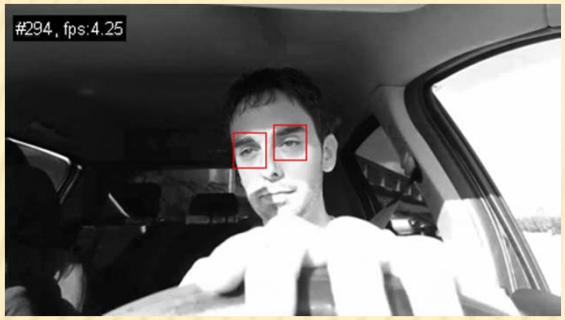
PROBLEMAS

- En el caso en el que la persona gira la cabeza completamente no es posible detectar los ojos:

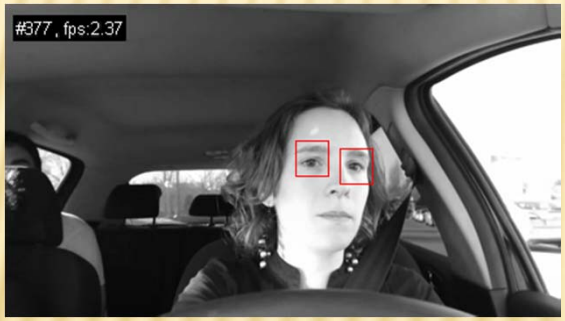
- Cuando el sol incide sobre la cámara se producen errores en la detección de modo intermitente:

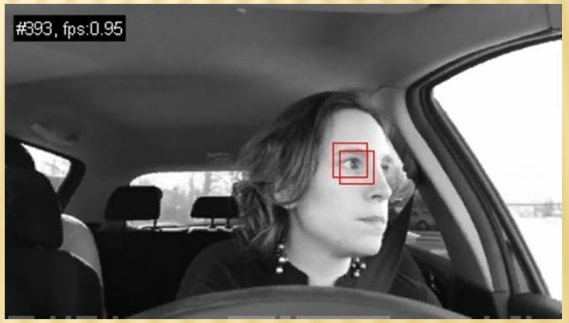
PROBLEMAS

- El bounding box tiene un tamaño incorrecto para contener un ojo:



- Los dos bounding box se centran en el mismo ojo:





PROBLEMAS

- El bounding box detecta un objeto incorrecto:

- Los bounding box se centran en los ojos equivocados:

PROBLEMAS

- Cuando la persona cierra los ojos, en ocasiones la detección falla:

- El índice de confianza de *Predator* no es representativo:

FILTRADO DE FALSOS POSITIVOS

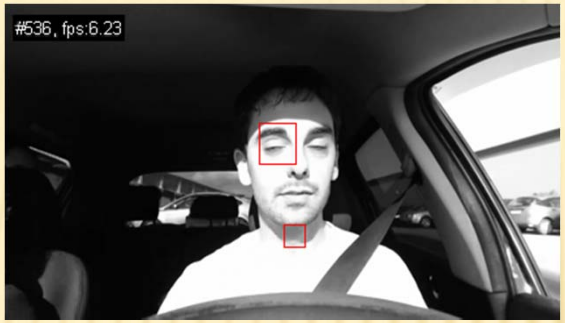
- Se ha implementado una función para desechar los resultados que con una alta probabilidad son erróneos. Se contemplan los siguientes casos:

- Diferencia significativa de tamaño de los bounding box:

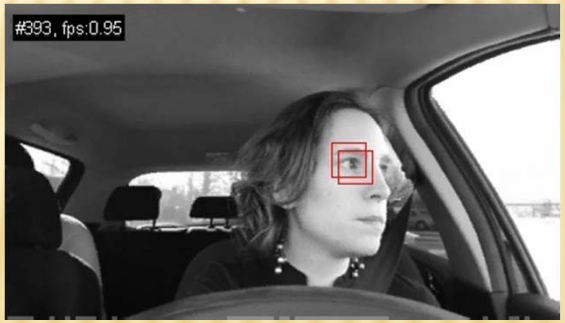
- Distancia horizontal entre los bounding box elevada:

FILTRADO DE FALSOS POSITIVOS

- Distancia vertical entre los bounding box elevada:



- Los dos bounding box se centran en el mismo ojo o se cruzan:



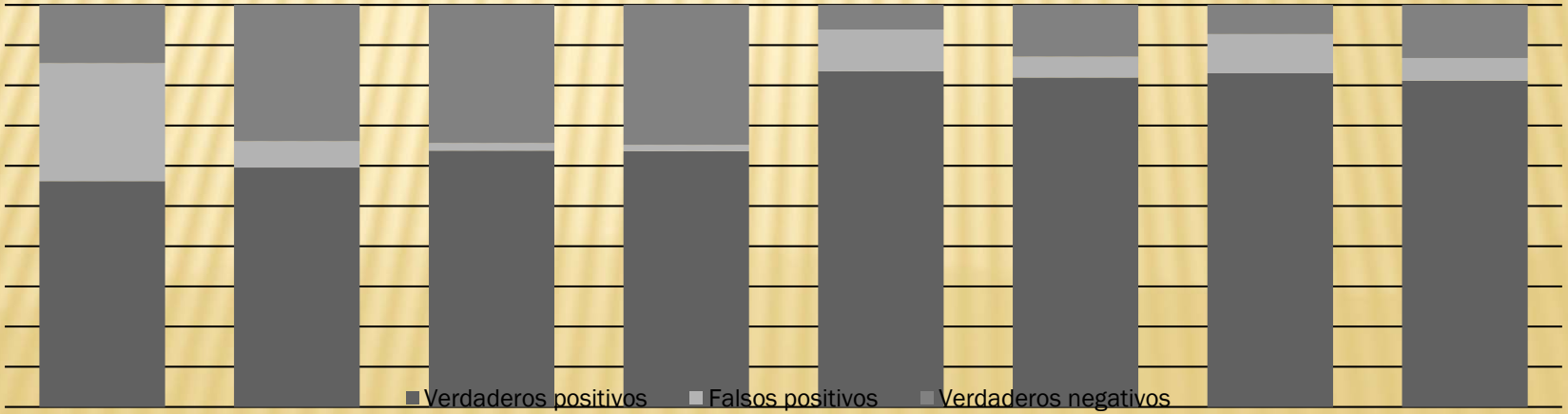
FILTRADO DE FALSOS POSITIVOS

- No se detectan los dos ojos:

- Resultados muy diferentes al primero:

RESULTADOS

	E_T_P		E_T_VJ		E_T_VJ_L_P		E_T_VJ_P	
	No filtrado	Filtrado	No filtrado	Filtrado	No filtrado	Filtrado	No filtrado	Filtrado
Verdaderos positivos	56%	60%	64%	64%	84%	82%	83%	81%
Falsos positivos	29%	7%	2%	2%	10%	5%	10%	6%
Verdaderos negativos	15%	34%	34%	34%	6%	13%	7%	13%
Tiempo	0.64	0.55	0.61	0.63	0.83	0.82	0.76	0.74



DETECCIÓN DEL IRIS

- Se ha incluido un programa de detección de iris ya implementado.

- Tras analizar el funcionamiento de los dos métodos de detección posibles (Timm y Wang) se opta por utilizar el método de Timm, debido a su mayor índice de detecciones correctas y similar tiempo de ejecución.

APLICACIÓN DE SEGURIDAD

- Se ha implementado un programa que alerta al usuario por medio de una señal sonora cuando ha transcurrido un determinado tiempo sin detectar los ojos.

- Ejemplo:

CONCLUSIONES

- *Eye_Tracker_Predator* es la función que peores resultados genera tanto de verdaderos positivos como de falsos positivos. Es muy sensible a cambios de iluminación y movimientos rápidos.

- *Eye_Tracker_VJ* genera un índice de verdaderos positivos ligeramente mejor que *Eye_Tracker_Predator* pero un índice de falsos positivos realmente bajo. Es muy sensible a giros de la cabeza y niveles de iluminación elevados.

- *Eye_Tracker_VJ_Learning_Predator y Eye_Tracker_VJ_Predator* superan las limitaciones y los errores de los anteriores en gran medida a costa de un tiempo de procesado mayor.

- El programa de filtrado reduce los falsos positivos en todos los programas aumentando su robustez y con un incremento del tiempo de procesado mínimo.

GRACIAS