Estudio de validación de software headtracker y facetracker para interacción web en JavaScript

Andrés Sánchez Caparrós

Resumen– En este proyecto se realiza un estudio del estado del arte de los software de seguimiento de la cabeza y cara en JavaScript para interacción web junto a su estudio cualitativo y cuantitativo. Primeramente se ha llevado a cabo un anàlisis de los software de código abierto para analizar su rendimiento y estudiar los algoritmos que implementan. Seguidamente se han desarrollado diferentes interfícies junto al artista digital Daniel Norton para interaccionar con objetos abstractos a partir del movimiento de la cabeza. Finalmente, se han mejorado algunos puntos de los algoritmos estudiados para mejorar el control y dar al usuario una interfície más robusta y que aporte mejor experiencia.

Palabras clave– Head tracker, Face tracker, interacción web, interfície web, objetos abstractos, visión por computador, estudio cualitativo, estudio cuantitativo.

Abstract– In this project, we do a study of the state-of-the-art of head and face tracking software in JavaScript for web interaction with their qualitative and quantitative study. First we work an analysis of open source software to analyze their performance and study the algorithms that implement code. Then, different interfaces have been developed by the digital artist Daniel Norton to interact with abstract objects from head movement. Finally, we have improved the algorithms studied to improve control and give the user a more robust interface and to provide better experience with us.

Keywords– Head tracker, Face tracker, web interaction, web interface, abstrackt objects, computer vision, qualitative study, quantitative study.

1 Introducción

En las últimas décadas, los avances respecto a la interacción computador-humano han evoluacionado mucho, hasta llegar a sistemas de interacción de cuerpo completo a partir de cámaras y sensores. La inquietud para desarrollar este proyecto es la interacción a partir de los movimientos de la cabeza con una cámara web en el contexto de la interacción con objetos abstractos en un equipo multidisciplinar junto al artista digital Dan Norton para definir los diferentes escenarios y funcionalidades de las interfícies gráficas. Este artículo está estructurado en 5 partes: En la primera parte se explican los objetivos que han permitido el avance del proyecto. En la siguiente parte se encuentra toda la información sobre el estado del arte del headtracker y facetracker para interacción web en JavaScript. En la tercera parte se explica la metodología aplicada en el proyecto y como ha permitido cumplir los objetivos planteados. Seguidamente, en la siguiente sección se describe cada uno de los algoritmos estudiados para la interacción web a partir de headtracker y factracker y seguidamente, los resultados del estudio cualitativo y cuantitativo a partir de los resultados obtenidos a partir del estado del arte y de las mejoras aplicadas a los algoritmos que implementan estos software. Finalmente, en la última sección se explican las conclusiones finales del proyecto junto a la bibliografía que ha facilitado el estudio de los algoritmos y el posterior desarrollo de las interfícies gráficas.

• E-mail de contacto: andres.sanchez.caparros@gmail.com
• Mención realizada: Computación
• Trebajo tutorizado por: Fernando Vilariño (Computación)
• Curs 2015/16

2 Objetivos

Los objetivos del proyecto derivan de un objetivo principal: Realizar un estudio de validación del estado del arte actual del software de interacción web a partir del movimiento de la cabeza. Para realizar esta validación, hemos

definido los siguientes sub-objetivos:

- Realizar un estudio de los algoritmos que implementan el headtracker y facetracker.
- Analizar el estado del arte actual en referencia a headtracker y facetracker en JavaScript.
- Extraer y analizar los parámetros que condicionan la robustez del headtracker y facetracker.
- Realizar mejoras con su respectivo estudio cualitativo sobre los algoritmos actuales para mejorar la robustez y el control del tracker.
- Desarrollar una interfície gráfica para interacción con objetos abstractos.

Una vez cumplidos estos sub-objetivos, se puede realizar una validación de este tipo de software para interacción web.

3 Estado del arte

3.1. Estado actual

Desde hace tiempo, para la captura de audio y video en desarrollo web, se ha dependido de complementos externos como es el caso de Adobe Flash o Silverlight. La aparición de Html5 nos permite un gran acceso a los dispositivos hardware instalados, como es el micrófono y la cámara web, por lo que varias empresas como Google, Firefox y Opera, entre otros, han trabajado en el desarrollo de diferentes APIs para facilitar este acceso. En este proyecto nos hemos centrado en getUserMedia[17], una API dedicada a la obtención de vídeo a partir de una webcam en tiempo real y sin necesidad de complementos externos. Actualmente, los nuevas tecnologías web que utilizan los componentes hardware como micrófonos y cámaras web, aprovechan las ventajas de Html5 y de estas APIs para gestionar los datos de entrada.

3.2. Principales referencias del estado del arte

Desde el inicio del proyecto, la base ha sido encontrar referencias de software dedicado al headtracker en JavaScript. La principal motivación para usar este lenguaje de programación es basicamente la próximidad a cualquier usuario habituado con el desarrollo web. Después de analizar varias librerías de código abierto, con el que poder estudiar sus algoritmos, hemos visto que todas usan el algoritmo de detección de objetos a tiempo real de Viola y Jones[2]. Sin embargo, este algoritmo no te devuelve la forma que tiene la cara, simplemente el marco que la envuelve, los ojos, boca y nariz. Con estos datos, algunos de estos software aplican cambios en el rostro del usuario, como es la alteración de las distancias entre estos rasgos, el color, etc. Algunas librerías, además de utilizar este algoritmo, van más allá y trabajan a partir de modelos de cara como es el caso del algortimo de Constrained Local Models[4] previamente aprendidos a partir de algoritmos de matching-learning e incluso algoritmos de búsqueda de puntos entre frames como el algoritmo de Lukas Kanade[7].

3.3. Referencias utilizadas en el proyecto

A lo largo del proyecto, se han probado en una misma interfície web, varias de estas técnicas de headtracker. Todas ellas de código abierto para poder realizar un estudio de los algoritmos y del funcionamiento que corre detrás de cada una de estas librerías. Hemos empezado por aquellas más simples y que dan mejores resultados como son los trabajos de Auduno, que nos facilita una librería de headtracker y otra de facetracker. La primera utiliza exclusivamente Viola y Jones y la segunda Constrained Local Models a partir de modelos aprendidos anteriormente. Seguidamente, se ha trabajado en combinar estos algoritmos con el algoritmo de Lukas Kanade para mejorar el control y la robustez del software.

4 Metodología

Por tal de conseguir los objetivos marcados, la metodología utilizada a sido la metodología en cascada, basada en definir los objetivos, anàlisis, diseño, construcción, pruebas e implementación. Al tener bien definidos los objetivos a asumir, el proceso ha realizar a sido lineal, sin embargo, a medida que avanzaba el proyecto y por problemas de mala gestión de planificación, la metodología utilizada no beneficiaba el desarrollo del proyecto, por lo que se decidió usar un método ágil que permitiera asumir objetivos más rápidamente. Para ello se decidió usar la metodología Scrum, basada en realizar varias iteraciones (sprints) para la realización de diferentes subojetivos y tener al final de cada sprint una pequeña parte del software final. Juntamente con el artista digital Dan Norton, se han definido los objetivos semanales para poder ver resultados y aproximar el desarrollo de las interfícies al estudio cualitativo de las tecnologías de headtracking.

4.1. Metodología en cascada

La metodología basada en cascada es una metodología líneal que permite que hasta que no acabe una etapa, no se pueda pasar a la siguiente. Esta metodología fué la escogida al inicio del proyecto ya que los requisitos iniciales estaban bien definidos y las tareas requerían que una tarea no empezara hasta acabar la anterior. En este caso, el estudio de los algoritmos es primordial antes que el desarrollo de las interfícies. En la Figura 1 podemos ver, a nivel general, las etapas de esta metodología para el desarrollo de software.

4.2. Metodología ágil - Scrum

La ventaja de cambiar de metodología se basa en que, teniendo bien definidos los objetivos del proyecto, simplemente se han desagregado y se ha trabajado semanalmente para cumplirlos.
La product backlog representa los objetivos a realizar del proyecto. Seguidamente con Dan Norton se han definido sprints de 2 semana en los que se ha desarrollado una pequeña interfície mostrando las ventajas de una y otra librería.

4.3. Planificación y seguimiento de cada metodología

En el inicio del proyecto, durante la metodología en cascada, se ha hecho el estudio del funcionamiento del algoritmo de Viola y Jones para el seguimiento de la cabeza dentro de vídeo seguidamente de una interfície web que permita la interacción con el movimiento de la cabeza. Seguidamente se siguió con esta metodología en el desarrollo y estudio del algoritmo de Contraint Local Models, en el que la planificación se alargó y finalmente se decidió cambiar la metodología. A partir de este punto, se han ido desarrollando pequeñas interfícies que cumplan pequeños subobjetivos de desarrollo, como es la interacción con el fondo de pantalla, sonidos a partir de Flocking hasta llegar a la inter-acción de objetos abstractos implementados a partir de Adobe Edge[18].

5 ALGORITMOS ESTUDIADOS

Para poder cumplir el objetivo principal del proyecto, se han estudiado diferentes algoritmos que implementan el estado del arte actual.

5.1. Algoritmo de Viola y Jones

El algoritmo de Viola y Jones[1] es un algoritmo diseñado para detectar cualquier tipo de objeto dentro de una imagen, en este proyecto, el objeto buscado es una cara dentro de una serie de imagenes que componen un video a tiempo real a partir de una webcam y de un navegador web. El algoritmo lo podemos dividir en 3 fases:

- Definición de las características que componen la imagen.
- Definición de los clasificadores fuertes a partir del algoritmo AdaBoost[12].
- Clasificación en cascada a partir de los clasificadores fuertes.

5.1.1. Definición de las características que componen la imagen

Para la primera fase, se utilizan características simples (características Haar) para calcular la diferencia de luminancia entre los píxeles, en su defecto, Viola y Jones evaluan imágenes de 24x24 píxeles para realizar el aprendizaje a partir de estas características simples. Por defecto, el uso de las 5 características Haar en imágenes de 24x24 píxeles, representa el cálculo de 45.396 regiones, lo que penaliza el tiempo de detección.

Para resolver y mejorar el tiempo de cálculo, utilizamos el concepto de imagen integral. El concepto de imagen integral se basa en definir regiones de píxeles como si se tratara de uno solo y reducir los cálculos considerablemente.

El valor de la imagen integral que agrupamos para el punto (x,y) es la suma de los píxeles que estan por encima y hacia la izquierda del punto, como se muestra en la ecuación 1

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y') \qquad (1)$$

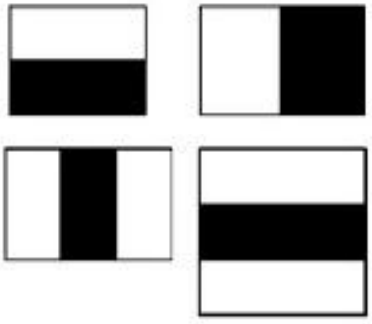

Fig. 1: Características Haar

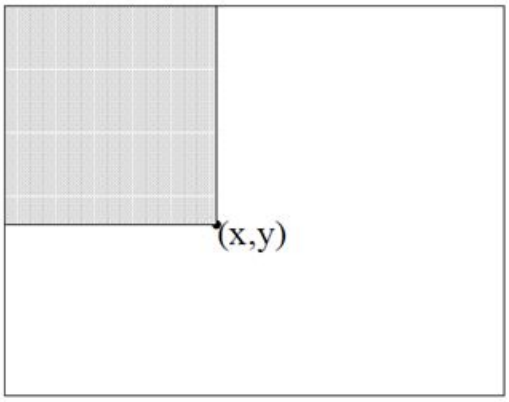

Fig. 2: Imagen integral

5.1.2. Definición de los clasificadores débiles a partir de AdaBoost

La siguiente fase del algoritmo es la que forma los clasificadores que servirán para detectar la cara en cualquier otra imagen. Se basa en la utilización del algoritmo AdaBoost definido por Freund y Shaphire para crear clasificadores fuertes a partir de muchos clasificadores débiles.
Un clasificador simple es una estructura que está formada por una característica y un umbral, y nos sirve para hacer una clasificación binaria (por encima o debajo del umbral). Estas clasificaciones son muy triviales y tienen un error muy bajo, por lo que el algoritmo de AdaBoost utiliza el voto ponderado de varios clasificadores débiles para crear un clasificador fuerte y alimentar una función cuadrática que haga una clasificación binaria con muy poca probabilidad de fallar.
Por lo tanto, esta fase del algoritmo de Viola y Jones se basa en definir clasificadores fuertes a partir de las caracteristicas obtenidas de la primera fase, utilizando caracteristicas positivas (rasgos faciales) y características negativas (rasgos no faciales).

5.1.3. Clasificación en cascada a partir de clasificadores fuertes

Finalmente, el algoritmo de Viola y Jones utiliza los clasificadores fuertes adquiridos en la fase anterior para clasificar. Viola y Jones plantearon la utilización de estos clasificadores para clasificar no caras en vez de caras, ya que es mucho más probable encontrar una característica no facial a lo largo de la imagen que un rasgo facial.
Entonces, la clasificación en cascada (Figura 3) se basa en

evaluar una subimagen con un clasificador fuerte, si el clasificador detecta que no es cara, descartamos esa subimagen, sin embargo, si lo clasifica como cara, realizamos la misma prueba con otro casificador.

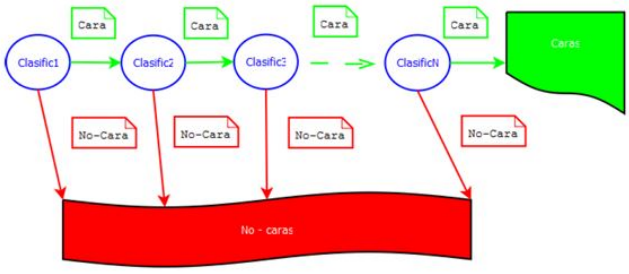

Fig. 3: Clasificación en cascada

De esta manera se clasificarán unicamente como caras, las subimágenes que sean evaluadas como caras por todos los clasificadores fuertes. Una vez obtenidas todas las características que forman parte de la cara, se obtiene el área que envuelve estas características, es decir, la cara dentro de la imagen.

5.2. Constraint Local Models (CLM)

Contraint Local Models es un método de localización de un conjunto de puntos, limitado por un modelo previamente aprendido. Este tipo de algoritmo aplicado a FaceTracker se basa en buscar en una imagen, la forma aprendida en un modelo de cara. Este modelo trata de ajustarse a la forma de la cara que encuentra, siempre respetando unas reglas que estan definidas dentro del modelo.
Este método se puede dividir en tres etapas:

- Creación del modelo inicial.
- Reducir la imagen a la zona de la cara.
- Aplicar modelo a la subimagen y obtener los rasgos de la cara.

5.2.1. Creación del modelo inicial

La primera fase del método CLM se basa en crear un modelo de cara 'estandar'. Para ello se realiza un marcado de los rasgos más representativos de la cara, como son los ojos, nariz, boca y contorno de la cara y después se realizan dos pasos:

- Crear un modelo para la forma de los rasgos y cara.
- Crear un modelo específico de cada rasgo representativo

Para crear el modelo de la forma, se aplica un PCA (Principal Component Analysis)[13] para reducir la dimensionalidad de los puntos marcados en los rostros de entrenamiento y obtener una mejor representación de estos puntos (de más importantes a menos). De esta manera obtenemos una distribución que permite valorar si otra distribución de puntos es parecida o no.
Para crear un modelo específico de cada rasgo, se aplica SVM (Support Vector Machines)[14] para realizar un aprendizaje supervisado a partir de muestras positivas y negativas de cada rasgo (ojos, nariz y boca) de manera que al final de esta parte, obtenemos un clasificador para cada rasgo.

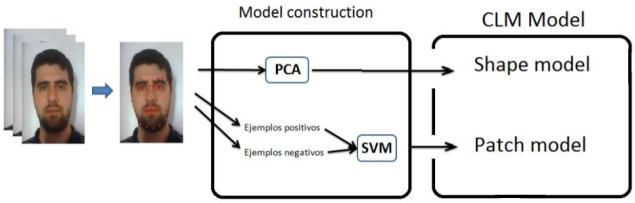

Fig. 4: Construcción del modelo

5.2.2. Reducción de la imagen

Esta parte del algoritmo se basa en aplicar el algoritmo de Viola y Jones en la imagen para encontrar un marco que englobe la cara junto a todos sus rasgos. De esta manera, obtenemos una subimagen con un tamaño perfecto donde aplicar el modelo aprendido anterior.

5.2.3. Aplicar modelo y obtención de los rasgos faciales

Una vez obtenida la subimagen que contiene la cara, se realiza una estimación de la cara dentro del rectángulo utilizando Mean Shift para ajustar los contornos a los puntos estimados de la cara.
Después de esta estimación inicial, se obtienen los puntos de la imagen que estan marcados por la estimación inicial, es decir, donde hemos estimado que hay un ojo, obtenemos la zona de la imagen que está en esa posición y la recortamos para clasificarla a partir del clasificador del modelo aprendido anteriormente. Una vez hecho para todas las características, se aplica el modelo de forma aprendido anteriormente para comprobar que los rasgos obtenidos respeten las limitiaciones de forma, es decir, dos ojos por encima de la nariz, esta por encima de la boca, etc. (Figura 5) Este paso se realiza N veces que, para tener un equilibrio entre robustez y velocidad,

$$N \in [10, 20]$$

o hasta que converja.

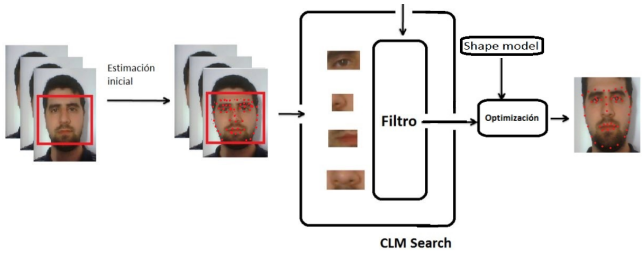

Fig. 5: Obtención de los rasgos faciales

5.3. Método Lucas Kanade

El método de Lucas Kanade define que el desplazamiento del contenido entre dos imágenes en dos instantes de tiempo muy cercanos es muy pequeño y aproximadamente

constante, por lo que, este método se puede utilizar para realizar un seguimiento de los rasgos de la cara entre frames, ya que la imagen entre frame y frame no varia mucho y varia a una velocidad casi constante.
Para utilizar este método, se utiliza un(os) punto(s) de referencia para buscarlo(s) en el siguiente frame, y un vector de velocidad que es el que define cuanto se mueven los píxeles entre frame y frame.
Este método empieza haciendo una estimación inicial en el frame inicial a partir de la disparidad entre los frames, y seguidamente por cada punto dispar, se aplica un gradiente[15] (Figura 6) para obtener su nueva posición en el siguiente frame.

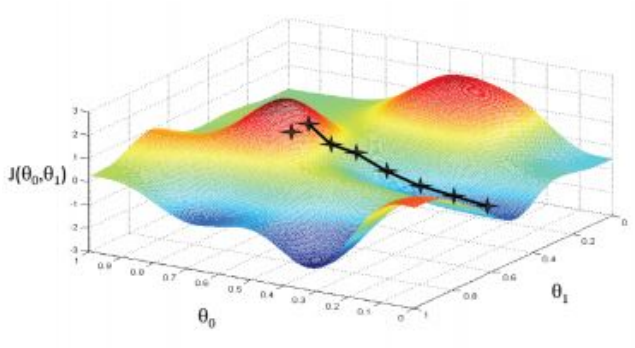

Fig. 6: Gradiente descendente

6 Estudio cuantitativo y cualitativo

6.1. Métricas para el estudio

Para realizar el estudio cuantitativo y cualitativo se han definido diferentes métricas para cada tipo y para cada implementación de los algoritmos anteriores que forman parte del estado del arte actual del Headtracker y Facetracker en JavaScript.
Para el estudio cuantitativo se han definido 3 métricas:

- Velocidad del algoritmo
- Frames por segundo (FPS)
- Ángulos límite de movimiento

Para el estudio cualitativo se han definido las siguientes métricas:

- Robustez con luz natural
- Robustez con luz artificial

Para realizar el análisis cuantitativo, primeramente se calcula el tiempo en milisegundos que requiere cada algoritmo en lenguaje JavaScript, seguidamente, se ha calculado los frames por segundo, es decir, la inversa del tiempo requerido por cada algoritmo en segundos (Fórmula 2)

$$FPS = \frac{1}{TiempoIteracion} \quad (2)$$

Finalmente, calculamos los ángulos a partir de los cuales, el algoritmo falla con las mejores condiciones de luz. Para esta parte del estudio, se calcula los ángulos para los ejes X e Y del plano 3D real, siendo el eje Z el eje perpendicular a la pantalla que va hacia el usuario y a una distancia real de 50cm (distancia a la pantalla de un ordenador portátil).
Para el cálculo utilizamos trigonometría básica para calcular el ángulo a partir de la regla del seno (Fórmula 3).

$$\alpha = \arcsin(\frac{a}{b}) \quad (3)$$

Donde a es la distancia entre la webcam y el punto de intersección de la cabeza y la pantalla; y b la distancia entre el usuario y la pantalla.

6.2. Estudio cuantitativo del estado del arte

Si probamos las métricas que evalúan cuantitativamente el software (apartado 6.1) con el estado del arte actual para interacción web en JavaScript, obtenemos los siguientes resultados:

TAULA 1: Resultados cuantitativos estado del arte

Resultados del estado del arte				
			Ángulos (grados)	
Algoritmo	Iteración (ms)	FPS	Eje X	Eje Y
Viola y Jones	15	66	65	42
CLM	7	142	35	42
L. Kanade*	16	62.5	70	57

6.2.1. Resultados Viola y Jones

Como podemos apreciar en la Tabla 1, el algoritmo de Viola y Jones trabaja a una velocidad de 15ms por iteración (66 frames por segundo). Esta velocidad se consigue aplicando unicamente el algoritmo y mostrando por pantalla la zona de la cara mediante un rectángulo. El número de fps es muy bueno ya que no deja que el ojo humano visualice paradas durante el vídeo, por lo que se considera que funciona a tiempo real. El ángulo límite de detección empieza a fallar a partir de los 65º en el eje horizontal, ya que a partir de ese ángulo, muchas características de la cara quedan ocultas, sin embargo, la rotación vertical de la cabeza está más limitada ya que por el mismo motivo anterior, los rasgos más relevantes desaparecen antes y se pierde la forma de la cara con más facilidad. Sin embargo, el algoritmo de Viola y Jones recupera muy rápidamente la cara en cuanto esta vuelve a una posición frontal.

6.2.2. Resultados CLM

En cuanto al método CLM, este algoritmo tiene una velocidad mucho más rápida que Viola y Jones debido a que solamente en las primeras iteraciones se aplican los cálculos más pesados, una vez detectada la cara, CLM se basa en la diferencia entre el frame anterior y el nuevo para seguir el tracking, por lo que la velocidad llega a ser el doble que la de Viola y Jones. Sin embargo, el límite de movimiento es mucho peor, CLM al trabajar con el contorno de la cara

además de los ojos, nariz y boca, al no ser una cara frontal, la forma varía y busca una nueva cara en el frame.

6.2.3. Resultados L.Kanade

Este es el método con más variedad de resultados. Puesto que el algoritmo trabaja a nivel de píxel, el seguimiento de un punto en el que los píxeles vecinos tienen valores similares provoca que el seguimiento falle.
Utilizando las métricas anteriores, la velocidad del algoritmo es muy rápida (tanto como Viola y Jones), y los angulos límite en ambos ejes son muy amplios, sin embargo, con movimientos rápidos, cambios de iluminación o seguimiento de puntos con poco contraste con los puntos adyacentes, provoca la pérdida del punto en frames posteriores.

6.3. Estudio cualitativo del estado del arte

Para evaluar cualitativamente los algoritmos, analizamos la robustez a partir de la iluminación del usuario. En este estudio separaremos la luz natural de la luz artificial.

6.3.1. Viola y Jones

Para el algoritmo de Viola y Jones apenas vemos diferencia, puesto que el algoritmo ya tiene aprendidos los clasificadores fuertes, solamente busca rasgos que formen parte de la cara independientemente entre ellos.
En la Figura 7 podemos ver que en ambos casos encuentra la cara perfectamente.

6.3.2. CLM

En este caso, el algoritmo tiene el problema que mean-shift no consigue recolocar las características de la cara entre frame y frame por el cambio de contraste a causa de la luz natural. Sin embargo, con luz artificial, el algoritmo es muy robusto.
En la Figura 8 podemos ver como el dibujo de los puntos de los rasgos para luz natural no se ajusta correctamente a la cara, sin embargo con luz artificial, los puntos se situan exactamente donde toca, coincidiendo con los rasgos faciales.

6.3.3. L.Kanade

Finalmente, el método de Lukas Kanade trae aspectos positivos para el seguimiento con luz natural. El seguimiento de puntos donde hay mucho contraste facilita que este método pueda trabajar correctamente. En la Figura 9 podemos ver como la zona con luz mantiene 2 puntos de los 4 que había en la imagen (simetricos en el lado opuesto).

6.4. Estudio cualitativo y cuantitativo de las mejoras aplicadas

6.4.1. Mejoras aplicadas

Para el proyecto, he definido e implementado varias mejoras en la utilización de los algoritmos en el seguimiento de la cara y sus rasgos en JavaScript:

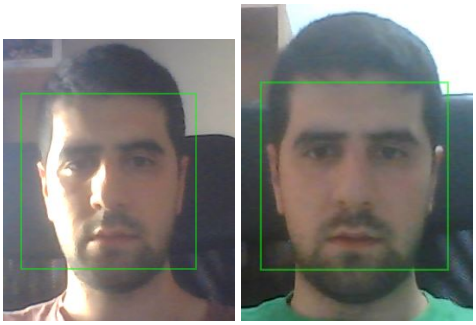

Fig. 7: Resultados Viola y Jones: Diferente iluminación

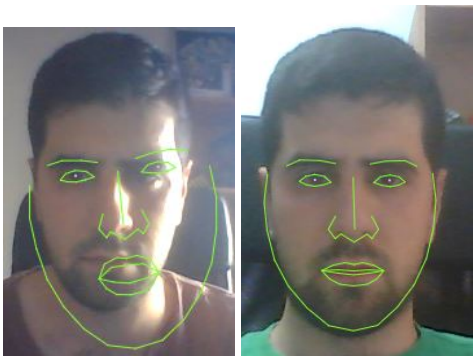

Fig. 8: Resultados CLM: Diferente iluminación

Fig. 9: Resultados L.Kanade: Luz natural

- Aplicar el método de L.Kanade junto a CLM para mejorar el control (M1)

  Esta mejora se basa en aplicar un seguimiento a través de los puntos claves que nos aporta CLM pero a través de los píxeles que lo forman. De esta manera, si pasamos el ángulo límite para CLM, podemos recuperar la posición exacta a partir de la posición de los ojos, nariz y boca seguidos por el método de L.Kanade.
  Esta mejora después de ser desarrollada, ha sido descartada. El motivo se basa en que el método de L.Kanade pierde el seguimiento muy facilmente y la pérdida no permite reposicionar los puntos obtenidos por CLM.

- Acumular el error en la fase del mean-shift de CLM para dar continuidad al seguimiento (M2).

  Visto que CLM aplica inicialmente realiza la detección de la cara y seguidamente realiza el seguimiento a partir de mean-shift, este método vuelve a

aplicar los primeros pasos cuando la diferencia entre el candidato del siguiente frame aplicando mean-shift tiene una diferencia grande con el frame anterior, vuelve a realizar la busqueda de la cara en toda la imagen.
La mejora se basa en no volver a aplicar esta segunda detección y dejar al usuario que vuelva a encontrar la cara, es decir, vuelva a la posición en la que la "máscara"de puntos se ha quedado para que mean-shift vuelva a coger los puntos correctos.

- Aplicar CLM junto a Viola y Jones para re-calibrar el seguimiento cuando este falle (M3).

  Esta mejora se basa en una ampliación de la anterior. Visto que Viola y Jones tiene una probabilidad muy baja de fallar, la mejora se basa en aplicar ambos algoritmos a la vez y mientras la detección a partir de CLM se mantenga dentro de la zona detectada por Viola y Jones, el seguimiento se mantiene a partir de mean-shift, en cambio, cuando los puntos obtenidos por CLM salen de la zona, volvemos a realizar la detección de la cara.

6.4.2. Estudio cuantitativo de las mejoras

Para realizar este estudio, aplicamos las mismas métricas que para el estado del arte actual. Para las 3 mejoras anteriores, se han obtenido los resultados mostrados en la tabla 2

TAULA 2: RESULTADOS CUANTITATIVOS DE LAS MEJORAS DEL ESTADO DEL ARTE

Resultados de las mejoras aplicadas				
			Ángulos (grados)	
Mejora	Iteración (ms)	FPS	Eje X	Eje Y
M1	17	58	–	–
M2	8	125	60	50
M3	155	6.45	60	60

La M1 no aporta más robustez al sistema, los ángulos límite són los mismos que para el CLM pero al no poder realizar el seguimiento con el método de L.Kanade, no es posible recuperar la posición correcta de la cara. El tiempo de ejecución no varía respecto al método de L.Kanade y se mantiene en los 58 frames por segundo.
Respecto a la M2, se aprecia una gran mejoría en los límites de movimiento, podemos realizar un movimiento de hasta 60^o horizontalmente y 50^o en el eje vertical sin perder el seguimiento. Sin embargo corremos el riesgo que una vez pasemos ese límite, podamos no volver a recuperar el seguimiento.
Finalmente, la M3 mejora la robustez de la M2, volviendo a aplicar CLM después de que el seguimiento no esté incluido en el marco definido por Viola y Jones. Los ángulos límite son los mismos que para la M2, sin embargo siempre recuperaremos el seguimiento cuando se pasen esos límites.
Por contra, este última mejora relantiza mucho la aplicación, el tiempo por iteración al tratar la imagen y realizar el algoritmo aumenta de de 17ms a 155ms, por lo que el frame rate cae hasta los 5-8 FPS.

6.4.3. Estudio cualitativo de las mejoras

Para estudiar la cualidad de las mejoras, nos centramos en su robustez, puesto que la comparación con luz natural y luz artificial es la misma (tratamiento de la imagen).
M1, como se ha explicado antes, queda descartada al no aportar nada al algoritmo CLM.
En la M2, se nota un gran mejoría puesto que los movimientos pueden ser muy amplios y la velocidad y el FPS son muy buenos. En cambio se corre el riesgo de que se pierda el seguimiento. Para evitar esta pérdida, se realizó la M3.
Esta mejora consigue la robustez que se buscaba, sin embargo penaliza mucho el frame rate. Esta penalización causa una mala experiencia con el usuario.

7 APLICACIÓN WEB

Una vez estudidos los algoritmos que componen el estado del arte en Headtracking y Facetracking, juntamente con el artista digital Dan Norton, se han definido dos interfícies para interacción web.

- Cambio de color
- Interacción con objetos abstractos

Para detectar hacia donde mira el usuario, se han utilizado M2 y M3 por separado para desarrollar estas dos interfícies. Ambas utilizan las distancias entre ciertos puntos de la cara para determinar el lado hacia donde mira.
Primeramente se valida el ángulo de la recta que forma la nariz, cuando este ángulo supera un pequeño lindar, podemos asegurar que está mirando a la derecha o izquierda, en ese caso miramos que distancia es más corta, si de la nariz a un costado de la cara o hacia el otro lado de la cara. En cambio, si el ángulo de la recta de la nariz tiene un valor pequeño, validamos si se está mirando hacia arriba, abajo o al centro. Esta validación se realiza de la misma manera que para validar derecha o izquierda, pero con los puntos de referencia el centro de la cara, frente y barbilla.

7.1. Cambio de color

Esta interfície esperaba ser una interfície básica en la que se pueda testear y validar hacia donde mueve la cabeza el usuario. En este caso, la interacción se basaba en cambiar los canales RGB del background de la pantalla a partir del movimiento.
Para hacer el cambio facil, se ha relacionado un canal de color cuando el usuario gire la cabeza hacia arriba, derecha o izquierda. En esta primera interfície, se define rotar la cabeza hacia la derecha aumenta el canal verde, izquierda el azul y arriba el rojo. Mantener la cabeza mirando al centro modifica los 3 canales para que vuelvan a 0 progresivamente.
Para mejorar la experiencia con el usuario, se han añadido botones para ocultar el vídeo y solamente se muestre la máscara de puntos que engloban los rasgos principales de la cara.
En la Figura **??** podemos ver un ejemplo de la interfície

básica de cambio de color para la M2 y la Figura 10 para la M3.

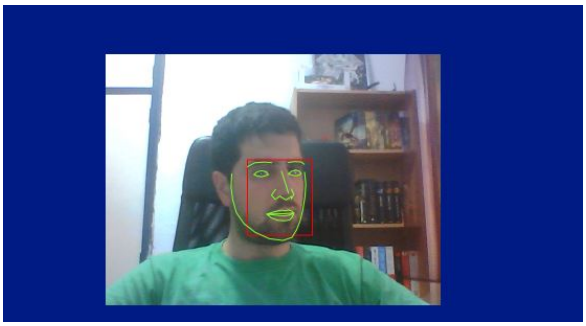

Fig. 10: Aplicación cambio de color: mejora 3

7.2. Interacción con objetos abstractos

Esta última interfície fué diseñada por el artista digital Dan Norton. Está compuesta por 4 objetos creados a partir de Adobe Edge, situados arriba, abajo, derecha e izquierda de la pantalla.
El objetivo de esta interfície es hacer reaccionar estos objetos cuando el usuario rota la cabeza hacia ellos, produciendo movimiento y sonido.
En la Figura 11 podemos ver un ejemplo de la interfície para la M2 para interacción con objetos abstractos y la Figura **??** para la M3:

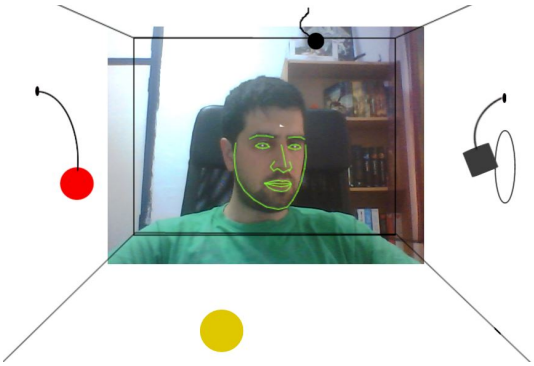

Fig. 11: Aplicación cambio de color: M2

8 Contribución al software de Headtracking y Facetracking

A partir del estado del arte actual, los algoritmos estudiados y las mejoras desarrolladas para obtener una interfície web robusta han contribuido a desarrollar una librería en JavaScript que utiliza de base el trabajo de Auduno [9] y que mejora los ángulos límite de rotación en 25º y 15º (ejes horizontal y vertical respectivamente) aportando una mejor robustez, además de detectar hacia donde está dirigida la cara del usuario (arriba, abajo, derecha e izquierda). Esta aportación permite futuros desarrollos para interacción web para cualquier persona que necesite detectar la posición de los rasgos faciales y conocer hacia donde está dirigida la cara.

9 Conclusiones

Después de realizar el estudio de las limitaciones del estado del arte actual y realizar mejoras para desarrollar una interfície gráfica de interacción web, se puede decir que se han asumido los objetivos principales del proyecto con éxito.
Además, una vez acabado el proyecto, podemos llegar a las siguientes conclusiones:

- A partir de los resultados obtenidos en el estudio, ell algoritmo de Viola y Jones es el mejor candidato en cuanto a detección de rostros a tiempo real.
- CLM y el método de L.Kanade són muy precisos cuando hay buena resolución y buen contraste entre los rasgos faciales, sin embargo, por el mismo motivo, su robustez no es muy eficiente a cambios de iluminación y movimientos muy rápidos.
- El comportamiento entre el algoritmo de Viola y Jones y CLM aporta resultados muy buenos, sin embargo la combinación de seguimiento sobre el mismo frame penaliza mucho el frame por segundo, por lo que hay que buscar un punto óptimo entre velocidad y robustez para aportar un buen feedback al usuario.
- Con las mejoras aportadas, M2 y M3, se ha conseguido mejorar considerablemente la robustez del software, consiguiendo mejorar el ángulo de rotación de la cabeza sobre ambos ejes y dar una buena experiencia de seguimiento al usuario. La mejora M3 consigue una robustez muy alta, sin embargo perjudica el frame rate que cae por debajo de 10 FPS. La mejora M1 tenia una base teórica muy buena, sin embargo depende mucho de la calidad de la imagen para su correcto funcionamiento.
- Se ha podido desarrollar una interfície de interacción web junto a Dan Norton en lenguaje JavaScript y de código abierto para que cualquier persona pueda visualizar el código y tener una base para futuros desarrollos.

Agradecimientos

Quisiera agradecer a mis amigos y compañeros de trabajo el apoyo que me han brindado a lo largo de todo el proyecto. A Onur Ferhat por su tiempo y su ayuda en la aportación de referencias y código, a Dan Norton y a mi tutor Fernando Vilariño por las ganas y motivación que me han aportado durante todo el transcurso del proyecto.

Referencias

[1] P.Viola, M.Jones, Robust Real-time Object Detection, Second international on statistical and computational threories of vision – modeling, learning", computing and sampling, Vancouver, Canada, 2001

[2] P. Viola, M.Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features", In Computer Vision and Pattern Recognition, Volume 1, 2001.

[3] The University of British Columbia, notes of Viola-Jones Face Detector, 2001

[4] David Cristinacce, Tim Cootes, ”Feature Detection and Tracking with Constrained Local Models”,Dept. Imaging Science and Biomedical Engineering, University of Manchester, Manchester, M13 9PT, U.K.

[5] Xiaoguang Yan, Çonstrained Local Model for Face Alignment, a Tutoria”l, Changchun University, 2001

[6] D. Cristinacce, T. Cootes, ”Feature Detection and Tracking with Constrained Local Models”, In BMVC

[7] Bruce D. Lucas, Takeo Kanade, .An Iterative Image Registration Technique with an Application to Stereo Vision”, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania

[8] Audun Mathias (Auduno),”Javascript library for head-tracking via webcam and WebRTC/getUserMedia”, https://github.com/auduno/clmtrackr

[9] Audun Mathias (Auduno), ”Javascript library for precise tracking of facial features via Constrained Local Models”, https://github.com/auduno/clmtrackr

[10] Eugene Zatepyakin (inspirit), ”JavaScript Computer Vision library. Lukas Kanade optical flow, haar object detector”, https://github.com/inspirit/jsfeat

[11] Colin Clark, Adam Tindale, ”Flocking: A Framework for Declarative Music-Making on the Web”, http://flockingjs.org/

[12] Jason Corso, ”Boosting and AdaBoost”, http://www.cse.buffalo.edu/jcorso/t/CSE555/files/lecture_boosting.pdf

[13] Victor Powell, Lewis Lehe, ”Principal Component Analysis Explained Visually”, http://setosa.io/ev/principal-component-analysis/

[14] Support Vector Machines, http://scikit-learn.org/stable/modules/svm.html

[15] Matt Nedrich, .An Introduction to Gradient Descent and Linear Regression”, https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

[16] ”Segmentation and Clustering”, EECS 442 Computer Vision, Fall 2012, http://vhosts.eecs.umich.edu/vision//teaching/EECS442_2012/lectures/seg_cluster.pdf

[17] W3C, ”Media Capture and Streams”, https://www.w3.org/TR/mediacapture-streams/

[18] Adobe Edge, ”Adobe Edge Animate CC”, http://www.adobe.com/es/products/edge-animate.html