Universidad Carlos III de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA
Ingeniería Industrial

Cálculo de la dirección de la mirada de una persona mediante análisis de imágenes

***Autor*:** Jorge Gonzalo Grande
Tutor: Arturo de la Escalera Hueso

Índice

1 Resumen

Posiblemente una de las prioridades de los desarrolladores de productos informáticos desde los comienzos de este campo ha sido la forma en la que los seres humanos pueden interactuar con estos sistemas. En la actualidad esta relación está sufriendo un cambio sustancial. Principalmente debido al uso de tecnologías más compactas, que permiten al usuario utilizar estos sistemas en nuevos entornos. Aunque también es debido al cambio de concepción al que se ven sometidos todos los sistemas de la casa, en los que se busca la integración de varios sistemas en un único equipo que los aglutina a todos. Consecuencia directa de estos avances son algunos sistemas que hoy día forman parte de nuestra vida diaria, pero que no hace tanto tiempo no eran más que proyectos en un laboratorio de investigación (teclados táctiles, control por voz, reconocimiento de figuras humanas). Este proyecto trata de adentrarse en ese campo, proponiendo un proceso de interacción entre humano y máquina, basado en la captación del movimiento de los ojos y la cara por medio de una imagen 2D tomada con una cámara.

Nuestro proyecto consiste en la creación de un sistema que va a permitir intuir la dirección de la mirada de una persona a través de una simple fotografía del rostro de dicha persona. El sistema está compuesto por un software de seguimiento de la mirada, y una serie de modelos, que buscan describir las características del entorno, como puede ser, el modelo de la cámara, o los modelos del rostro y de los ojos.

Los primeros sistemas de seguimiento del rostro, surgieron en la década de los 80, y permitían el reconocimiento de objetos rígidos, es decir, aquellos cuya forma no sufriera transformaciones. Más tarde fueron los sistemas *Active Shape Models* (o modelos activos de forma, conocidos por sus siglas en inglés ASMs) desarrollados por T.F. Cootes y C.J. Taylor en la década de los 90 los que permitieron el modelado flexible, o dicho de otra manera, el reconocimiento de objetos cuya forma pueda verse alterada en dos fotogramas distintos. Estos modelos ASM ajustaban los objetos basándose tan solo en las restricciones de forma dadas por el objeto en cuestión. Más avanzados que los ASM son los modelos *Active Appereance Models* (o modelos de apariencia activa, a los que nos referiremos de aquí en adelante por sus siglas en inglés AMM), también desarrollados por T.F. Cootes y C.J. Taylor en la misma década, que utilizarán la información de la textura, combinada con las restricciones de forma anteriores para crear y ajustar un modelo de la cara. Puesto que el ajuste de los sistemas AAM no es del todo eficiente, los siguientes sistemas que se desarrollaron ya a principios de siglo, fueron los *Constrained Local Models* (o modelos locales restringidos, a los que nos referiremos de aquí en adelante por sus siglas en inglés CLM) que permitían realizar un ajuste no lineal, basado en el uso de funciones cuadráticas.

Con esta perspectiva histórica este proyecto se ha enfocado en el uso y entendimiento de los modelos denominados CLMs, aunque por su gran similitud con los modelos AMMs muchos de las explicaciones versan también sobre estos modelos. Por tanto a partir de los sistemas CLMs de seguimiento del rostro, utilizaremos una serie de técnicas que permitirán extraer tanto la pose de la cabeza, como la posición del iris. Esto unido a un modelado correcto del entorno, permitirá extraer la dirección donde enfoca la mirada una persona.

2 Motivación y objetivos

2.1 Motivación

En los últimos años se ha experimentado un enorme crecimiento a nivel tanto de software, como de hardware, en el proceso de tratamiento de imágenes con ordenador. Se han mejorado los sistemas de adquisición de imágenes digitales, lo que permite capturar imágenes con mayores resoluciones y tiempos de muestreo. También se han mejorado los procesos de almacenamiento de las imágenes, por medio de *códec* de video que permiten trabajar con imágenes que contienen más información, a un coste computacional menor. Además hay diversos aspectos donde se continúa trabajando, y que serán la base de los futuros sistemas de tratamiento de imágenes, como la adquisición de imágenes 3D, o sistemas similares, que intentan emular la forma en que el ser humano percibe a través de sus ojos.

Estas mejoras han desencadenado una revolución en cuanto al uso del procesamiento de imágenes en la industria, y otros ámbitos en los que interactúan el hombre y la máquina. Dentro de la industria los sistemas de tratamiento de imágenes que intervienen en el proceso, tienen una función más focalizada en evitar las tareas repetitivas que producen desgaste a las personas y reducir así el factor de error humano. En otros ámbitos las aplicaciones son más diversas y existen multitud de funciones en las que se puede emplear un sistema de este tipo.
Algunos ejemplos que ilustran la cantidad de cometidos en los que se emplean el tratamiento de imágenes son:

Contabilización de objetos: Dentro de la industria es muy común la necesidad de conocer el número de objetos fabricados. Un sistema de tratamiento de imágenes puede realizar esta tarea, con unas buenas condiciones de iluminación.

Identificación de defectos: En la industria es muy importante identificar aquellas piezas que presentan algún defecto. Para ello se toman muestras de un determinado lote para estimar el número de piezas defectuosas en dicho lote. Con un sistema de tratamiento de imágenes se puede conseguir aumentar el tamaño de la muestra, permitiendo en algunos casos analizar la totalidad de los elementos fabricados.

Identificación de objetos: En un ámbito ya menos industrial, otro de las aplicaciones que se estudian, incluso dentro de nuestra Universidad, es la identificación de objetos. A través del procesamiento de imágenes, un sistema es capaz de diferenciar entre varios objetos que tiene en su base de datos. Este proceso puede ayudar en la detección de objetos peligrosos, por ejemplo, en los controles de seguridad en un aeropuerto.

Otra idea interesante es utilizar las imágenes para el reconocimiento de los rasgos característicos de una persona. Estos rasgos se pueden emplear posteriormente para diversas funciones. Este es uno de los propósitos de este proyecto, servirse de la identificación de las características del rostro para realizar un seguimiento de la mirada de una persona. Relacionado con este tipo de sistemas existen un gran número de ejemplos:

Alertas en automóviles: Numerosos fabricantes de coches han desarrollado sus propias aplicaciones basadas en el reconocimiento de las características del rostro, para aumentar la

seguridad no solo de las personas dentro del vehículo, sino también de las que rodean al vehículo. Por medio de parámetros como la mirada, el parpadeo, y otra serie de gestos, el sistema es capaz de determinar si el conductor está en disposición de conducir y si es consciente de los peligros potenciales que le rodean.

Reconocimiento de puntos de interés: Las empresas de marketing, o de desarrollo de páginas web, están interesadas en conocer cuáles son los puntos dónde una persona centra su atención en un momento determinado. Esto les proporciona una valiosa información para el desarrollo o diseño de sus páginas, campañas, o productos. Por medio de un sistema de seguimiento de la mirada y un software que procese esa información y la condense en gráficos que resalten la información relevante, es posible conocer esos puntos de interés.

Sistemas de autentificación: Son aplicaciones que a través de una detección de los rasgos faciales, autorizan o deniegan el acceso al sistema, en función de si sus rasgos coinciden con los de las personas autorizadas. Son sistemas que ofrecen una baja seguridad.

Además de todos estos ejemplos, uno de los objetivos más importantes en la concepción de este proyecto, era el de presentar una solución para personas que tienen una discapacidad física y motora, que permitiese mejorar su capacidad de comunicación entre usuario-máquina, y entre usuario-usuario utilizando como canal una máquina. En este sentido el avance de estas técnicas de reconocimiento facial ha supuesto un gran avance en el día a día de personas con este tipo de lesiones físicas, algunas de las soluciones factibles gracias a esta tecnología son:

Control del guiado de una silla de ruedas para personas tetrapléjicas: Este sistema permite al usuario guiar su silla de ruedas por medio de simples movimientos de su cabeza. Utiliza un sistema de captación de imágenes en tiempo real, constituido por la webcam de un portátil, que detecta la posición de la cabeza en un determinado instante, la procesa extrayendo información tanto de la posición de su cabeza cómo de los ojos, y genera una señal de control en función del movimiento que se desee hacer.

Control de un ordenador por medio de gestos: Del mismo modo que en la aplicación anterior, se puede utilizar el seguimiento de la posición de la cabeza, así como el de los ojos, para controlar las distintas funciones de un ordenador. Este sistema podría sustituir al ratón y teclado del ordenador, en caso de conseguir una precisión adecuada. Nuestro proyecto pretende implementar este tipo de sistema.

En el futuro, cuando estos sistemas mejoren los problemas de exactitud y precisión, estas nuevas formas de interactuar con las máquinas sustituirán a los sistemas utilizados en la actualidad. Puesto que suponen una mejora en lo que se refiere a rapidez y libertad de movimientos, y eso se traduce en una mayor eficiencia en el control de estos sistemas.

2.2 Objetivos

Por tanto el objetivo principal de este proyecto es el de construir un sistema que permita la detección de la mirada de una persona a partir de una imagen 2D. Este objetivo lo vamos a desglosar en tres objetivos concretos.

En primer lugar desarrollaremos un *software* que nos permita detectar tanto la pose de la cabeza, como la posición exacta de los ojos, a partir del uso de los ya mencionados CLMs. En este caso el software CLM ha sido cedido por Jason Mora Saragih y es la base de nuestro posterior algoritmo de seguimiento del rostro.

El segundo de los objetivos consiste en la creación de varios modelos (un modelo del rostro, un modelo de la cámara, un modelo del entorno) de forma que podamos relacionar las variables que caracterizan a cada uno de estos modelos, y conseguir así transformar la captura de una imagen tomada por una cámara, en una serie de medidas que identifican dónde está dirigiendo la mirada esa persona.

El tercer objetivo tiene que ver con el aprendizaje obtenido de estos sistemas de seguimiento del rostro humano, y la documentación los mismos, gracias a toda la información obtenida durante la realización de este proyecto en lo que se refiere a los sistemas comúnmente utilizados para este propósito. De esta forma se pretende crear un punto de partida para que en futuros trabajos en los que se empleen estos sistemas, se tengan un conocimiento mayor de los mismos.

3 Fundamentos teóricos

3.1 Sistemas para el seguimiento del rostro.

3.1.1 Introducción

El cerebro humano tiene una capacidad congénita para simplificar la información que recibe. De manera que si a nuestro cerebro llega una determinada señal, ya sea a través de la vista, o de cualquiera de nuestros sentidos, él es capaz de clasificar la información que recibe relacionando a gran velocidad las características percibidas, y gracias a un aprendizaje anterior, almacenar la información ya no como características aisladas, sino como un ente conocido. A partir de la información clasificada y priorizada el cerebro emite, casi de forma inmediata, una respuesta. Por ejemplo el cerebro es capaz de relacionar un círculo naranja con la forma y textura de una naranja, y a su vez con comida, en un breve lapso de tiempo. Este es el motivo por el que las personas somos capaces de identificar la información contenida en una imagen, y no tanto por el nivel de detalle con el que el ojo la capta. Ésta es una de las características más interesantes de nuestro cerebro, puesto que nos permite trabajar con imágenes de forma que estas no resulten "pesadas", al permanecer en nuestro cerebro solo la información que consideramos relevante.

Sin embargo, toda persona que haya realizado un estudio de una imagen utilizando un ordenador, se habrá dado cuenta de que los datos adquiridos, ya sea por medio de una cámara de fotos, de video, escáner, o a través de cualquier otra técnica, son útiles para su representación, pero no nos aportan ninguna información adicional que nos permita trabajar con ella para simplificarla. Esto es consecuencia directa de la forma en la que se almacena la información de dichas imágenes en los ordenadores. Este es el reto al que se han enfrentado todas aquellas personas interesadas en obtener información de una imagen por medio de un ordenador. Condensar la información de una imagen, y centrarse en lo que en ese momento nos interesa, ya sea un coche, su matrícula, una persona, o una cara.

Por este motivo desde hace un par de décadas, los investigadores trabajan en técnicas que permiten a un ordenador extraer una serie de características, relevantes para la persona, de una imagen. A este grupo de técnicas pertenecen los denominados modelos flexibles, que son aquellos que buscan modelar la apariencia de un objeto determinado que se encuentra dentro de una imagen (ya sea una fotografía, un escáner, una resonancia electromagnética…) y cuya forma y apariencia varían a lo largo del tiempo dentro de esa imagen, según un patrón que podemos estudiar por medio de la experiencia. Este tipo de sistemas nos ayudan a simplificar la información de la imagen, basándose en modelos de objetos de lo que se quiere conocer, tales como su posición o la variación de alguna de sus características a lo largo del tiempo en una imagen. Este tipo de sistemas, aunque no permita reconstruir la información completa de la imagen, si permiten simplificar toda la información contenida en ella, y centrarnos en aquellas características de mayor interés para cada aplicación. Lo que ayudará en la construcción de nuestro modelo, así como en el posterior procesado que se realice con la ayuda de este modelo. En el caso de este proyecto, se va a modelar la forma de la cara, para poder hacer un seguimiento de la posición de la misma, que permitirá conocer la orientación de la cabeza, así como de los ojos, para finalmente realizar un seguimiento de la mirada. El proceso se divide en dos partes. Por un lado, se crea el modelo de la cara a partir de las imágenes de entrenamiento. En segundo lugar, se ajusta dicho modelo a nuevas imágenes que presentan diferentes posiciones de la cabeza y de los ojos.

El primer paso para la creación de un modelo, será la identificación del objeto a modelar y de las características que permiten diferenciar este objeto de otro cualquiera. Esto es lo que se conoce como entrenamiento, y es la base de los sistemas de modelado flexible. Es importante destacar que sin un correcto entrenamiento los modelos resultantes no serán fiables o contendrán defectos. Para realizar dicho entrenamiento se requiere de un usuario que indique al ordenador cuáles son las características del objeto de interés a considerar en el modelo y dónde aparecen en la imagen. Veamos un ejemplo. En este proyecto, se busca modelar una cara. Para crear este modelo se nos plantean varias preguntas ¿Qué es lo que define una cara?, ¿Qué la hace parecer diferente de otros objetos que podemos encontrar en la misma imagen? La respuesta serán las características del objeto que implementaremos en nuestro modelo, en este caso la forma o silueta y la textura o color. Es para saber qué forma y qué textura tiene un rostro humano para lo que necesitaremos de la experiencia del ser humano. Es decir, será el usuario el que indique que debemos buscar una forma elíptica, dos ojos en las esquinas superior derecha e izquierda, una nariz con determinada textura, etc…Con estas características relacionadas entre sí, veremos en los siguientes capítulos cómo somos capaces de construir un modelo bastante aproximado de la cara de una persona, en diferentes posiciones, para una imagen cualquiera.
Una vez se extraiga dicho modelo seremos capaces, mediante el uso combinado de la información de la imagen y del modelo, de identificar y acotar una serie de rasgos de interés. Estos rasgos pese a que son parte de la información contenida en la imagen, no podríamos procesarlos por medio de un ordenador sin la ayuda del modelo. En nuestro caso, aunque realicemos un modelo para toda la cara, nuestro interés se va a centrar tan solo en la posición de los ojos en la imagen, y en la orientación de la cabeza, respecto de un sistema de coordenadas concreto.

3.1.2 Modelos de reconocimiento facial: AAM y 3DMM

Los modelos más importantes para el reconocimiento y seguimiento de la cara son los *Active Appearance Models* (AAMs) y los *3D Morphable Models* (3DMMs). Ambos modelos han sido diseñados para trabajar con las características mencionadas en el apartado anterior, la forma y la textura. Si bien AAM y 3DMM son modelos muy similares en algunos aspectos (permiten variaciones lineales en la forma y en la textura), presentan algunas diferencias. La principal diferencia entre ambos, aunque no la única, es que la componente que caracteriza la forma de la cara en el modelo AAM, está formado por una malla triangular de puntos bidimensional, mientras que en el caso de los modelos 3DMM esta malla triangular es tridimensional.
En nuestro proyecto vamos a utilizar los AAM. El hecho de que un modelo AAM tenga una malla triangular bidimensional, no significa que esta no contenga información tridimensional, ni que no pueda representar un sistema tridimensional.
Puesto que es interesante desde el punto de vista de este proyecto tener información tridimensional del modelo, vamos a intentar formar dicho modelo tridimensional a partir del modelo bidimensional que proporcionan los modelos AAM. Una de las ventajas que nos aportará este método frente al modelo 3DMM es la mayor velocidad en el ajuste entre el modelo y la imagen, actualmente los algoritmos que hacen uso de los modelos AAM suelen operar a velocidades superiores a los 200 *fps*. Por otro lado otra de las ventajas que se nos presenta es la facilidad en la construcción del modelo, los algoritmos AAM son capaces de crear un modelo a partir de una imagen 2D, mientras que los modelos 3DMM son necesarios en ocasiones información 3D para construir el modelo.
Existen discrepancias en lo que al uso de algoritmos AMM para la construcción de modelos 3D se refiere. Principalmente porque estos modelos buscan construir modelos 3D a partir de modelos 2D, para ello necesitan un mayor número de parámetros para conseguir dicha reconstrucción. Esto en sí mismo no supone una desventaja, puesto que, como ya hemos comentado anteriormente,

consiguen ajustar dicho modelo a una velocidad mayor. Pero sí lo es el hecho de que al contener un mayor número de parámetros, estos modelos puedan generar un mayor número de instancias, que en el caso de los modelos 3DMM no son capaces de generar. Ya que se piensa que estas instancias no deberían existir en caso de que queramos ajustarnos estrictamente a las posibles variaciones que se pueden permitir en una cara humana. El motivo por el que no se eliminan del modelo AMM es que estas instancias pueden ayudar a hacer que el ajuste de los algoritmos sea más preciso y eficaz.
En cualquier caso, en una aplicación como la nuestra en la que el tiempo de cómputo debe ser lo más ajustado posible y no buscamos representar los rasgos fáciles de manera fidedigna, sino que buscamos una aproximación para la orientación de la cabeza y la posición que ocupan los ojos en la imagen, existen claramente suficientes argumentos para que la elección del modelo AAM quede justificada.

3.1.3 *Active Appearance Models* (AAMs)

Como ya hemos explicado los AAM son sistemas para generar modelos del rostro a partir de imágenes bidimensionales, basados en dos componentes la forma y la textura. La representación 2D de los AAM se define por medio de una malla triangular, en concreto por la localización de los vértices de dicha malla, que viene dado por las coordenadas (u, v) de cada punto en la imagen:

$$s = \begin{pmatrix} u_1 & u_2 & \dots & u_n \\ v_1 & v_2 & \dots & v_n \end{pmatrix} \quad (1)$$

Los modelos AAMs permiten variaciones lineales en la forma. Esto significa que la matriz de forma s se puede representar como una base de forma s_0 más la combinación lineal de m matrices de forma s_i:

$$s = s_0 + \sum_{i=1}^{m} p_i \, s_i \quad (2)$$

donde los coeficientes p_i serán los parámetros de forma. Para la creación de los modelos de forma que requieren los AAM se marcarán determinados puntos de interés en varias imágenes, este proceso se conoce como entrenamiento. Este proceso requiere de un conjunto de imágenes (a las que denominaremos imágenes de entrenamiento), que irán acompañadas de un fichero de datos donde se almacenan las posiciones de cada vértice de la malla para cada imagen. Para conformar el fichero de datos se requiere de la interacción de un usuario que señale la forma del rostro para cada una de esas imágenes de entrenamiento (normalmente marcando la posición de cada uno de los vértices de la malla manualmente para cada una de las imágenes). Para obtener la base de forma s_0 y las posibles variaciones de forma s_i aplicaremos a cada una de las imágenes de entrenamiento: *Iterative Procrustes Algorithm* (conocido como análisis de Procrustes y cuyas siglas en inglés son IPA) y a continuación Principal *Component Analasys* (o análisis de componentes principales y cuyas siglas en inglés son PCA), estas dos técnicas se explicarán más adelante en el capítulo relativo a la creación del modelo. Con lo que el modelo de forma quedará definido.

La textura en un modelo AAM estará definida por aquellos puntos que se encuentren dentro de la malla base s_0. Simplificaremos la notación llamando también s_0 al conjunto de pixeles $\mathrm{u} =$

$(\mathrm{u},\mathrm{v})^{\mathrm{T}}$que contiene la base s_0 en su interior. La textura de un modelo AMM es entonces una imagen $A(u)$, donde los pixeles u ϵ s_0. Los modelos AAM también permiten variaciones lineales de la textura. Esto significa que la textura $A(u)$ puede ser representada por una base de textura $A_0(u)$ más una combinación lineal de l imágenes de textura $A_i(u)$:

$$A(u) = A_0(u) + \sum_{i=1}^{l} \lambda_i A_i(u) \tag{3}$$

donde λ_i son los parámetros de textura. Al igual que en la forma, para obtener los *patches* de la imagen que definirán el modelo de textura, aplicaremos PCA a al conjunto de las imágenes de entrenamiento (previa normalización de la forma del modelo). Posteriormente para cada uno de los vértices de la malla de forma, extraeremos un descriptor de la textura de dicha imagen A_i, ya sea extrayendo un *patche* pequeño de la imagen de entrenamiento u por medio de otras técnicas. Así el modelo de textura quedará definido.

Pese a que las ecuaciones (2) y (3) describen la variación de la forma y la textura del modelo AAM, no describen como crear una posible pose de dicho modelo. La pose del modelo AAM con parámetros de forma p_i y parámetros de textura λ_i se crea mediante el *warping* (se puede traducir esta palabra por deformación, aunque el concepto de *warp* es mucho más amplio, y por ello se continuará usando en el proyecto) de la textura A de la malla base s_0 hasta hacerla coincidir con la malla de forma del modelo s. En concreto el conjunto de mallas s_0 y s define un *warp* afín a trozos entre s_0 y s que llamaremos $W(u;p)$. Con esto queda definido nuestro modelo completo, formado por la malla de forma y de textura.

Una vez definido nuestro modelo, el objetivo es usar este para realizar el seguimiento de la cara de una persona. Para ello vamos a ajustar el modelo AAM secuencialmente, *frame* por *frame*, al video que deseemos analizar. Lo que significa que la imagen de entrada $I(x)$ y nuestra aproximación dada por el modelo $M\big(W(u;p)\big) = A(x)$ deben ser lo más parecidas posibles. Es importante elegir las coordenadas de pixeles donde es conveniente realizar este cálculo: las coordenadas de la imagen o las coordenadas del modelo. Para conseguir un algoritmo de ajuste más eficiente, se realiza el cálculo en las coordenadas del modelo, así pues u ϵ s_0. Por tanto el nivel de intensidad de un pixel en la imagen vendrá dado por $I\big(W(u;p)\big)$.

Además de manera análoga a (2), podemos calcular nuestra plantilla como $A(u) = A_0(u) + \sum_{i=1}^{l} \lambda_i A_i(u)$. Por tanto dada una imagen $I(u')$ cualquiera, el criterio que se va a utilizar para realizar este ajuste es minimizar:

$$\sum_{u \epsilon s_0} \left[A_0(u) + \sum_{i=1}^{l} \lambda_i A_i(u) - I\big(W(u;p)\big) \right]^2 \quad (4)$$

simultáneamente con respecto a los parámetros 2D del AAM, los de forma p_i y los de textura λ_i, donde la suma se realiza para todo pixel u ϵ s_0. En general esta optimización es no lineal para los parámetros de forma p, aunque sí lo es para los parámetros de textura λ. Pese a que esta no sea la ecuación que finalmente vayamos a minimizar, puesto que no contiene ninguna información acerca de la malla 3D, se van a explicar los pasos que hay que seguir para encontrar los parámetros λ_i y p_i que minimizan la ecuación (4), puesto que el procedimiento se utilizará posteriormente cuando resolvamos la ecuación completa 3D.

Para simplificar la notación vamos a definir el error de una imagen para un pixel, en coordenadas del AAM, como:

$$E(X) = A_0(u) + \sum_{i=1}^{l} \lambda_i A_i(u) - I\big(W(u;p)\big) \quad (5)$$

este error $E(u)$ se puede calcular de la siguiente forma. Para cada pixel u ϵ s_0, calculamos su correspondiente pixel $W(u;p)$ en la imagen de entrada transformando u por medio del *warp* afín a trozos definida por W. La imagen de entrada se evalúa en el pixel $W(u;p)$, típicamente se utiliza interpolación bilineal para extraer el valor. El resultado de esta operación se substrae de la apariencia del modelo $A_0(u) + \sum_{i=1}^{l} \lambda_i A_i(u)$ y este a su vez se almacena en E. Es decir, se aplica un *warp* a la imagen de entrada I hasta que la malla base s_0 coincide con s y a continuación se substrae su valor de la textura del modelo AAM actual.

Existen dos tipos de algoritmos que permitan minimizar la ecuación (5), vamos a explicar brevemente cada uno de ellos y hablaremos de las ventajas e inconvenientes que puedan presentar:

1. *Gradient Descent algorithms*. Estos métodos encuentran el mínimo local de una función acercándose al mismo tomando pequeños incrementos proporcionales al gradiente negativo de dicha función en ese punto. Tienen como ventaja que la convergencia está asegurada si se cumplen una serie de premisas conocidas. La mayor desventaja que tienen es su lentitud. El cálculo de derivadas parciales, matriz Hessiana, y el gradiente direccional deben realizarse para cada iteración.

2. *Ad-Hoc Fiiting Algorithms*. Debido a la lentitud que tienen los métodos del apartado anterior, se han realizado numerosos estudios para simplificar los procesos para minimizar la ecuación (5). En todos ellos se parte de la suposición de que existe una relación

constante lineal entre el error de la imagen $E(u)$ y los incrementos sumados a los parámetros de forma y textura del modelo de forma que:

$$\Delta p_i = \sum_{u \in s_0} R_i(u)E(u) \qquad y \qquad \Delta \lambda_i = \sum_{u \in s_0} S_i(u)E(u) \tag{6}$$

donde $R_i(u)$ y $S_i(u)$ son imágenes constantes definidas en la base s_0, y por tanto no dependen de los parámetros del modelo. Esta simplificación se realiza basándose en que todos los métodos anteriores de *Gradient Descent* se reducen a calcular Δp_i y $\Delta \lambda_i$, como funciones lineales del error de la imagen $E(u)$, y después actualizaban la estimación de los parámetros de la siguiente forma $p_i \leftarrow \Delta p_i + p_i$ y $\lambda_i \leftarrow \Delta \lambda_i + \lambda_i$. Aunque en estos métodos anteriores de *Gradient Descent* no se consideraba a $R_i(u)$ ni a $S_i(u)$ constantes, sino que dependían de los parámetros p y λ. Es por esto que para cada iteración se recalculaban los parámetros. Y esto, es en esencia por lo que estos métodos eran tan lentos. Para evitarlo algunos algoritmos asumen que $R_i(u)$ y $S_i(u)$ son constantes y no dependen de los parámetros p y λ. Esta simplificación es una aproximación, aunque no es correcta.

Como se ha descrito arriba los algoritmos existentes para el ajuste del modelo AAM se encuentran en una de las dos categorías. O bien utilizan la aproximación de los métodos *Gradient Descent* con las ventajas de utilizar un algoritmos basado en principios, pero muy lento, o hacen una suposición que probablemente sea incorrecta para obtener eficiencia, y en el proceso perder precisión en el ajuste. Tras lo visto en el párrafo anterior acerca de cómo se actualizan los parámetros del modelo, podemos llegar a pensar que no existe ningún algoritmo *Gradient Descent* que se resuelva para Δp, y nos permita actualizar los parámetros $p \leftarrow \Delta p + p$. Afortunadamente existen formas alternativas de actualizar estos parámetros. Una de ellas es actualizando el *warp* conjunto componiendo el *warp* actual con un *warp* incremental de parámetros Δp. De esta forma la actualización sería:

$$W(u;p) \leftarrow W(u;p) \circ W(u;\Delta p) \tag{7}$$

Esta forma de actualizar los parámetros se conoce como *Compositional Approach*, que es otra forma equivalente al *Additive Approach* (con este término hacemos referencia al algoritmo creado por Lucas-Kanade para la alineación de imágenes). A continuación se va a tratar de describir ambos métodos, *Compositional* y *Additive Approach* desde el punto de vista del problema de la alienación de imágenes. Esto tiene una gran importancia, como veremos a continuación, puesto que resolver el problema de alienación de las imágenes, es una parte importante del problema de ajuste del modelo AAM.

Alineación de las imágenes utilizando Lucas-Kanade

El objetivo de la alineación de imágenes es encontrar en qué posición se encuentra una plantilla concreta dentro de una imagen. Este algoritmo de *Gradient Descent*, fue descrito en Lucas y Kanade [1] por primera vez, y tiene como objetivo encontrar el óptimo local de las imágenes alineadas minimizando la diferencia de cuadrados entre una plantilla que no varía, llamémosla $A_0(u)$, y una imagen dada $I(u)$ con respecto a los parámetros de *warp* p:

$$\sum_u \left[A_0(u) - I\big(W(u;p)\big)\right]^2 \tag{8}$$

donde $W(u;p)$ será el *warp* que relaciona los pixeles u de la plantilla del modelo con los de la imagen de entrada. $I\big(W(u;p)\big)$ debe ser por tanto una imagen de las mismas dimensiones que la plantilla del modelo, a la que se la aplica un *warp* para que coincidan las coordenadas de la imagen con las de la plantilla $A_0(u)$.

El problema de minimización de la ecuación (8) en función de p es no lineal. Aunque $W(u;p)$ sea una transformación lineal para p, los valores de los pixeles $I(u)$ son no lineales en las coordenadas de u. Para linealizar el problema Lucas-Kanade usaron una estimación de p inicialmente, para luego resolver iterativamente el problema para incrementos de los parámetros Δp. La minimización quedará por tanto:

$$\sum_u \left[A_0(u) - I\big(W(u;p+\Delta p)\big)\right]^2 \tag{9}$$

Δp será la incógnita, y en cada iteración se actualizará p de la siguiente forma $p \leftarrow \Delta p + p$. Es posible linealizar está expresión entorno a p utilizando su desarrollo en serie de Taylor:

$$\sum_u \left[A_0(u) - I\big(W(u;p)\big) - \nabla I \frac{\partial W}{\partial p}\Delta p\right]^2 \tag{10}$$

donde ∇I es el gradiente de la imagen I evaluado en p, y $\frac{\partial W}{\partial p}$ es el Jacobiano del *warp* también evaluado en p. De este método solo se necesita saber que existe solución para la ecuación (10) que es de la forma:

$$\Delta p = H^{-1} \sum_{u} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[A_0(u) - I\big(W(u;p)\big)\right] \tag{11}$$

en la que H es la aproximación de *Gauss-Newton* de la matriz *Hessiana*:

$$H = \sum_{u} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[\nabla I \frac{\partial W}{\partial p}\right] \tag{12}$$

Se trata de un método muy lento, puesto que tanto ∇I como $\frac{\partial W}{\partial p}$ dependen de p, y por lo tanto H y $\nabla I \frac{\partial W}{\partial p}$ deben ser recalculados en cada iteración, siendo ambos operaciones muy costosas. A este algoritmo le denominaremos *Forward Additive* porque en cada iteración se actualiza p, sumándole el correspondiente Δp, de ahí el concepto de *Additive*. El concepto de *Forward* indica el sentido de la estimación del parámetro del *warp*: el *warp* proyecta todo a las coordenadas de la imagen.

Alineación de Imágenes *Forwards Compositional*

En el algoritmo anterior los parámetros del *warp* se calculan añadiendo un offset Δp a los parámetros actuales p. El punto de partida del algoritmo *Forwards Compositional* está basado en el método anterior, y consiste en calcular un *warp* incremental $W(u;\Delta p)$ que en cada iteración se compondrá con el *warp* actual $W(u;p)$. La minimización en este caso será:

$$\sum_{u} \left[A_0(u) - I\big(W(W(u;\Delta p);p)\big)\right]^2 \tag{13}$$

Y la actualización se realiza mediante la composición del *warp* actual con el incremental:

$$W(u;p) \leftarrow W(u;p) \circ W(u;\Delta p) \tag{14}$$

Si calculamos la solución de la ecuación (13) se obtiene el *warp* incremental proyectado en el sentido de la imagen. Si después componemos el *warp* incremental con el actual utilizando (14) para actualizar el *warp* y repetimos este proceso hasta la convergencia obtendremos los parámetros p por el método *Forwards Compositional*. Para obtener la solución de la ecuación (13) se realiza el desarrollo en serie de Taylor de la ecuación (13:

$$\sum_{u} \left[A_0(u) - I\big(W(W(u;0);p)\big) - \nabla I(W(u;p)) \frac{\partial W}{\partial p} \Delta p\right]^2 \tag{15}$$

y suponemos que $p = 0$ es el warp identidad, de forma que $W(u;0) = u$. Existen dos diferencias entre la ecuación (15) y la ecuación (10). La primera es que el gradiente se calcula en $I(W(u;p))$. La segunda es que el Jacobiano es evaluado en $(u;0)$ y por lo tanto es una constante que se puede calcular previamente. Aunque la composición de warp es más costosa que la actualización llevada

a cabo en el algoritmo *Forward Addtive*, es un pequeño coste por no tener que calcular el Jacobiano en cada iteración.

La clave de este algoritmo es que en cada iteración la actualización se realiza entorno a $p = 0$ cada vez. Es por eso que el Jacobiano es constante.

Alineación de imágenes *Inverse Compositional*

Este algoritmo es una modificación del algoritmo *Forwards Compositional*, donde los roles de la plantilla y la imagen se han invertido. En lugar de calcular el *warp* incremental con respecto a $I(W(u;p))$ se calcula con respecto a la plantilla $A_0(u)$. La explicación de que es equivalente realizar una u otra operación se puede encontrar en [3]. De manera más intuitiva se puede entender cómo que al invertir los roles de la imagen y la plantilla, estaremos calculando el *warp* incremental de manera que resulte ser el inverso del *warp* incremental calculado en el caso anterior.

Por tanto invirtiendo esas variables, podremos minimizar la expresión de la ecuación (13):

$$\sum_u \left[I\big(W(u;p)\big) - A_0(W(u;\Delta p))\right]^2 \tag{16}$$

respecto de Δp, y actualizar el warp utilizando:

$$W(u;p) \leftarrow W(u;p) \circ W(u;\Delta p) \tag{17}$$

Si utilizamos de nuevo el desarrollo en serie de Taylor en la ecuación (16) se tiene:

$$\sum_u \left[I\big(W(x;p)\big) - A_0(W(u;0)) - \nabla A_0 \frac{\partial W}{\partial p}\Delta p\right]^2 \tag{18}$$

considerando que $W(u;0)$ es el warp identidad, la solución de mínimos cuadrados de este problema es de la forma:

$$\Delta p = H^{-1} \sum_u \left[\nabla A_0 \frac{\partial W}{\partial p}\right]^T \left[I\big(W(u;p)\big) - A_0(u)\right] \tag{19}$$

donde H es la matriz Hessiana con $A_0(u)$ en lugar de I:

$$H = \sum_{u} \left[\nabla A_0 \frac{\partial W}{\partial p}\right]^T \left[\nabla A_0 \frac{\partial W}{\partial p}\right] \quad (20)$$

puesto que A_0 es una constante, y el Jacobiano $\frac{\partial W}{\partial p}$ siempre se evalúa entorno a $p = 0$, la mayoría de los cálculos serán realizados una única vez en una fase de procesado previa. Como consecuencia se consigue un algoritmo de alineación de las imágenes muy eficiente.

A continuación se explicará cómo se aplica el algoritmo de alineación de las imágenes *Inverse Compositional* anterior en el contexto del AAM, más concretamente cómo se va a resolver el problema de minimización planteado en la ecuación (4).

En primer lugar vamos a considerar que no existen variaciones en la textura, o lo que es lo mismo, el valor de m en la ecuación (4) es 0. Si se realiza esta simplificación podemos observar como las ecuaciones (4) y (8) son idénticas y por lo tanto resolver el problema de ajuste de un AAM se convierte en resolver el problema de alineación de las imágenes. El algoritmo de ajuste de AMM se puede dividir por tanto en los siguientes pasos:

Algoritmo *Inverse Compositional* sin variación de textura

Cálculos previos:

(3) Evaluar el gradiente ∇A_0 de la plantilla $A_0\ (u)$

(4) Evaluar el Jacobiano $\frac{\partial W}{\partial p}$ en $(u; 0)$

(5) Calcular las imágenes *Steppest Descent*: $\nabla A_0\ \cdot \frac{\partial W}{\partial p}$

(6) Calcular la matriz Hessiana utilizando la ecuación (20)

Iteramos:

(1) Aplicar $W(x; p)$ a I para calcular $I(W(u; p))$

(2) Calcular el error de la imagen $I\big(w(u; p)\big) -\ A_0\ (u)$

(7) Calcular $\sum_u [\nabla A_0 \cdot \frac{\partial W}{\partial p}]^T \big[I\big(W(u; p)\big) - A_0\ (u)\ \big]$

(8) Calcular Δp utilizando la ecuación (19)

(9) Actualizar el *warp* $W(u; p) \longleftarrow W(u; p) \circ W(u; \Delta p)^{-1}$

Imagen 1: *Algoritmo Inverse Compositional sin variación de textura.*

Donde la mayoría de los pasos de dicho algoritmo son operaciones normales con vectores, matrices e imágenes, tales como, calcular el gradiente de una imagen o la diferencia entre dos imágenes. Se van a explicar aquellos pasos que son más inusuales, como pueden ser: el paso 1 en el que se realiza el *warp* de I con warp $W(u; p)$, el paso 4 que calcula el Jacobiano de la *warp* afín a trozos, y el paso 9 en el que se invierte el *warp* incremental afín a trozos, y se compone con la estimación actual del *warp* afín a trozos.

Warp afín a trozos

La imagen $I\big(W(u;p)\big)$ se calcula por warp inverso de la imagen I con el warp $W(u;p)$. Por ejemplo, para cada pixel u de la base s_0, se calcula $W(u;p)$ y se evalúa el valor de dicho punto en la imagen I. Todo pixel u de la base s_0 se encuentra dentro de uno de los triángulos formados por los vértices de la malla 2D distribuidos a los largo de la imagen. Por tanto se pueden calcular a partir de la ecuación (2) y los parámetros actuales p. De forma que la posición de un pixel $u = (x,y)^T$ que se encuentra dentro del triángulo formado por los vértices $\big(x_i^0, y_i^0\big)^T, \big(x_j^0, y_j^0\big)^T$ y $(x_k^0, y_k^0)^T$ vendrá dada por:

$$u = (x,y)^T = \big(x_i^0, y_i^0\big)^T + \alpha\left[\big(x_j^0, y_j^0\big)^T - \big(x_i^0, y_i^0\big)^T\right] + \beta\left[(x_k^0, y_k^0)^T - \big(x_i^0, y_i^0\big)^T\right] \qquad (21)$$

Donde:

$$\alpha = \frac{\big(x - x_i^0\big)\big(y_k^0 - y_i^0\big) - \big(y - y_i^0\big)\big(x_k^0 - x_i^0\big)}{\big(x_j^0 - x_i^0\big)\big(y_k^0 - y_i^0\big) - \big(y_j^0 - y_i^0\big)\big(x_k^0 - x_i^0\big)} \qquad (22)$$

$$\beta = \frac{\big(y - y_i^0\big)\big(x_j^0 - x_i^0\big) - \big(x - x_i^0\big)\big(y_j^0 - y_i^0\big)}{\big(x_j^0 - x_i^0\big)\big(y_k^0 - y_i^0\big) - \big(y_j^0 - y_i^0\big)\big(x_k^0 - x_i^0\big)} \qquad (23)$$

El resultado de aplicar un *warp* a dicho punto sería:

$$W(u;p) = (x,y)^T = (x_i, y_i)^T + \alpha\left[\big(x_j, y_j\big)^T - (x_i, y_i)^T\right] + \beta\,[x_k, y_k)^T - (x_i, y_i)^T] \qquad (24)$$

Donde $(x_i, y_i)^T$, $\big(x_j, y_j\big)^T$ y $(x_k, y_k)^T$ son los vértices del triángulo en la base s. Las ecuaciones (22), (23) y (24) constituyen un *warp* afín simple:

$$W(u;p) = (a_1 + a_2 \cdot x + a_3 \cdot y, a_4 + a_5 \cdot x + a_6 \cdot y)^T \qquad (25)$$

Los 6 parámetros $(a_1, a_2, a_3, a_4, a_5, a_6)$ pueden ser fácilmente calculados a partir de los parámetros de forma p, combinando las ecuaciones (2), (22), (23) y (24). Éste cálculo sólo se bebe realizar una vez por cada triángulo de la malla, no una vez por pixel.

Cálculo del *Warp* Jacobiano

La transformación $W(u;p)$ reubica la posición de los pixeles de una imagen en la base $s = (x_1, y_1, x_2, y_2, \dots, x_v, y_v)$ de forma acorde a un warp afín a trozos con parámetros p. Si se aplica la regla de la cadena a $W(u;p)$ se obtiene:

$$\frac{\partial W(u;p)}{\partial p} = \sum_{i=1}^{v} \frac{\partial W}{\partial x_i}\frac{\partial x_i}{\partial p} + \frac{\partial W}{\partial y_i}\frac{\partial y_i}{\partial p} \tag{26}$$

dónde los primeros componentes del Jacobiano son $\frac{\partial W}{\partial x_i}$ y $\frac{\partial W}{\partial y_i}$ los Jacobianos del warp con respecto a los vértices de la malla s. A partir de la ecuación (24) se pueden escribir como:

$$\frac{\partial W}{\partial x_i} = (1-\alpha-\beta, 0)^T \;\; y \;\; \frac{\partial W}{\partial y_i} = (0, 1-\alpha-\beta)^T \tag{27}$$

Estas componentes del Jacobiano son imágenes del tamaño de la malla s_0. Se pueden ver algunos ejemplos en la imagen (2).

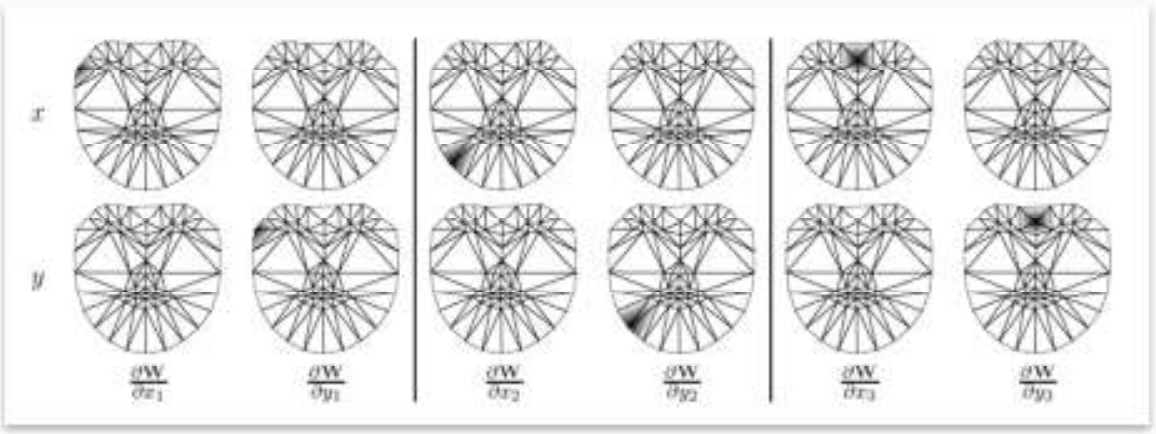

Imagen 2: *Imágenes del Jacobiano.* Imágenes de las componentes Jacobianos $\frac{\partial W}{\partial x_i}$ y $\frac{\partial W}{\partial y_i}$ calculados para tres triángulos diferentes. En la fila de arriba está la componente x del Jacobiano, la componente y está en la fila de abajo.

La segunda de las componentes del Jacobiano son $\frac{\partial x_i}{\partial p}$, $\frac{\partial y_i}{\partial p}$. Si se deriva la ecuación (2):

$$\frac{\partial x_i}{\partial p} = \left(s_1^{x_i}, s_2^{x_i}, \dots, s_v^{x_i}\right) \;\; y \;\; \frac{\partial y_i}{\partial p} = (s_1^{y_i}, s_2^{y_i}, \dots, s_v^{y_i}) \tag{28}$$

donde denominamos $s_j^{x_i}$ a los componentes de s_j que se corresponden con x_i y de forma similar con y_i. Estas componentes $\frac{\partial x_i}{\partial p}$ y $\frac{\partial y_i}{\partial p}$ son por tanto los vectores de forma s_i reordenados de forma adecuada.

Invertir el *Warp*

Para realizar el algoritmo *Inverse Compositional* debemos invertir el *warp* afín a trozos incremental $W(u;\Delta p)$ para calcular $W(u;\Delta p)^{-1}$. Puesto que:

$$W(u;\Delta p) = W(u;0) + \frac{\partial W}{\partial p}\Delta p + O(\Delta p^2) \tag{29}$$

Teniendo en cuenta siempre que $W(u;0) = x$ es el warp identidad, se tiene:

$$W(u;\Delta p) \circ W(u;-\Delta p) = u - \frac{\partial W}{\partial p}\Delta p + \frac{\partial W}{\partial p}\Delta p = x + O(\Delta p^2) \tag{30}$$

Es por eso que si consideramos una aproximación de primer orden podemos decir que:

$$W(u;\Delta p)^{-1} = W(u;-\Delta p) \tag{31}$$

Conviene aclarar que los Jacobianos en la ecuación (30) no se han calculado exactamente en el mismo punto, pero puesto que han sido evaluados en puntos separados por una distancia $O(\Delta p)$, serán iguales para orden cero en Δp. Puesto que la diferencia está multiplicada por Δp podemos considerar cero para órdenes mayores de cero. Además la composición de dos warps no ha sido definida estrictamente, y por tanto la ecuación (30) es algo informal. Sin embargo la esencia del argumento es correcta. Una vez se conocen las aproximaciones de primer orden de un *warp* afín a trozos, se puede usar la composición. Dando como resultado que el *warp* de $W(u;-\Delta p)$ seguido por el warp $W(u;\Delta p)$ es igual al warp identidad para la aproximación de orden uno en Δp.

Composición del *warp* incremental con el *warp* actual

Tras haber invertido el *warp* afín a trozos $W(u;\Delta p)$ para calcular $W(u;\Delta p)^{-1}$ se compone éste con el *warp* actual $W(u;p)$ para obtener $W(u;p) \circ W(u;\Delta p)^{-1}$. Dada la estimación actual de los parámetros p, la posición de los vértices de la malla $s = (x_1, y_1, x_2, y_2, \dots, x_v, y_v)^T$ se puede obtener a partir de la ecuación (2). Del apartado anterior sabemos que los parámetros del *warp* $W(u;\Delta p)^{-1}$ son $-\Delta p$. Se calcula entonces las posiciones de los vértices de la malla utilizando la ecuación (2) para estimar sus variaciones:

$$\Delta s_0 = -\sum_{i=1}^{n} \Delta p_i s_i \tag{32}$$

donde $\Delta s_0 = (\Delta x_1^0, \Delta y_1^0, \dots, \Delta x_n^0, \Delta y_n^0)^T$ son las variaciones aplicadas a la posición de los vértices de la malla base por la aplicación de $W(u;\Delta p)^{-1}$.

Para componer $W(u;\Delta p)^{-1}$ con $W(u;p)$, se calcula los correspondientes cambios de posición en los vértices de la malla actual $\Delta s = (\Delta x_1, \Delta y_1, \dots, \Delta x_n, \Delta y_n)^T$. Dadas las posiciones, podemos calcular los parámetros de $W(u;p) \circ W(u;\Delta p)^{-1}$ resolviendo la ecuación (2) para los nuevos parámetros:

$$p_i' = s_i(s + \Delta s - s_0) \tag{33}$$

donde p_i' es el parámetro i-ésimo de $W(u;p) \circ W(u;\Delta p)^{-1}$. Resta entonces calcular Δs a partir de Δs_0. Si consideramos el vértice i-ésimo de la malla, se calcula $(\Delta x_i, \Delta y_i)^T$ en la malla actual a partir de $(\Delta x_i^0, \Delta y_i^0)^T$ en la malla base s_0. Como anteriormente se ha descrito, este warp se calcula por medio del *warp* afín definido entre los dos triángulos, el de la malla s y el de la malla s_0. Puesto que existen varios triángulos que comparten el vértice i-ésimo, existen también diferentes *warp* afines, y por tanto el vértice puede tomar diferentes posiciones, dependiendo si se elige uno u otro. Es por eso que se decide utilizar la media de todas esas posiciones como la posición correcta para la malla incremental Δs.

Como último paso para explicar cómo resolver la ecuación (4) al completo, debemos considerar las variaciones en la textura. Para ello se utiliza la técnica propuesta en [4]. Así la ecuación (4) se rescribe como:

$$\sum_{u\epsilon s_0}\left[A_0(u)+\sum_{i=1}^{l}\lambda_i A_i(u)-I\big(W(u;p)\big)\right]^2=\left\|A_0(u)+\sum_{i=1}^{l}\lambda_i A_i(u)-I\big(W(u;p)\big)\right\|^2 \quad (34)$$

donde $\|\cdot\|$ es la norma L2. Esta ecuación se debe minimizar simultáneamente respecto de p y $\lambda=(\lambda_1\lambda_2,\dots,\lambda_n)^T$. Si denominamos al subespacio generado por los vectores A_i como $span(A_i)$ y a su complemento ortogonal como $span(A_i)^{\perp}$. La ecuación (34) queda de la siguiente forma:

$$\left\|A_0(u)+\sum_{i=1}^{l}\lambda_i A_i(u)-I\big(W(u;p)\big)\right\|^2_{span(A_i)^{\perp}}+\left\|A_0(u)+\sum_{i=1}^{l}\lambda_i A_i(u)-I\big(W(u;p)\big)\right\|^2_{span(A_i)} \quad (35)$$

donde $\|\cdot\|^2_L$es la norma L2 al cuadrado del vector proyectado sobre el subespacio L. El primero de los dos términos se simplifica directamente, puesto que la norma solo considera las componentes ortogonales del vector en el subespacio $span(A_i)$, cualquier término que se encuentre en el subespacio $span(A_i)$ se puede simplificar. Por tanto querremos minimizar:

$$\left\|A_0(u)-I\big(W(u;p)\big)\right\|^2_{span(A_i)^{\perp}}+\left\|A_0(u)+\sum_{i=1}^{l}\lambda_i A_i(u)-I\big(W(u;p)\big)\right\|^2_{span(A_i)} \quad (36)$$

El primero de estos dos términos no depende de λ_i. Además para cualquier valor de p, el mínimo valor del segundo término es siempre 0. Por tanto se minimiza (36) resolviendo primero un problema de minimización del primer término solo con respecto a p, y utilizando después ese valor óptimo de p como una constante para minimizar el segundo término solo respecto de λ_i. Considerando que los vectores de la base A_i son ortonormales entre sí, la minimización del segundo término tiene una solución del tipo:

$$\lambda_i=\sum_{u\in s_0}A_i(u)\cdot\left[I\big(W(u;p)\big)-A_0(u)\right] \quad (37)$$

Que corresponde con el producto escalar de $A_i(u)$ con el error de la imagen obtenido tras haber realizado la primera minimización.

Minimizar el primer término de la ecuación (36) es similar a resolver el problema de alineación de imágenes con el algoritmo *Inverse Compositional* sin tener en cuenta las variaciones de la textura, cuyo algoritmo se esquematizó en la imagen (1). La única diferencia es que se trabaja en el subespacio $span(A_i)^{\perp}$ en lugar de trabajar en el espacio vectorial completo sobre los pixeles de s_0. Pero ni siquiera se necesita proyectar el error de la imagen en este subespacio. Solo se proyectará $\nabla A_0 \frac{\partial W}{\partial p}$ en el subespacio $span(A_i)^{\perp}$. La razón por la que no es necesario proyectar el error de la imagen en este subespacio es que se va a calcular posteriormente el producto escalar de éste con $\nabla A_0 \frac{\partial W}{\partial p}$. Con que uno de los dos términos del producto escalar se proyecte en un subespacio lineal, el resultado será como si ambos lo estuviesen. Así el error de la imagen será "proyectado".

Denominamos a $\nabla A_0 \frac{\partial W}{\partial p}$ como la imagen *Steepest Descent* porque este producto escalar describe el camino más corto para reducir el error, si ignoramos la normalización provocada por el Hessiano. Llamaremos SD_j a cada elemento de la imagen $\nabla A_0 \frac{\partial W}{\partial p}$, siendo $j = 1, \dots, n$. Las imágenes del *Steepest Descent* se proyectan en el subespacio $span(A_i)^{\perp}$ de la siguiente manera:

$$SD_j(u) = \nabla A_0 \frac{\partial W}{\partial p_j} - \sum_{i=1}^{m} \left[\sum_{u \in s_0} A_i(u) \cdot \nabla A_0 \frac{\partial W}{\partial p_j} \right] A_i(u) \qquad (38)$$

El algoritmo completo por tanto será el que se muestra en la imagen (3):

Algoritmo *Inverse Compositional* con variación de textura

Cálculos previos:

(3) Evaluar el gradiente ∇A_0 de la plantilla $A_0\ (u)$

(4) Evaluar el Jacobiano $\frac{\partial W}{\partial p}$ en $(u; 0)$

(5) Calcular las imágenes *Steppest Descent* utilizando la ecuación (38)

(6) Calcular la matriz Hessiana utilizando las variaciones de las imágenes *Steepest Descent*

Iteramos:

(1) Warp I aplicando $W(x; p)$ para calcular $I(W(u; p))$

(2) Calcular el error de la imagen $I\big(w(u; p)\big) - A_0\ (u)$

(7) Calcular el producto escalar de las variaciones de las imágenes *Steepest Descent* con el error de la imagen

(8) Calcular Δp multiplicándolo por la inversa de la matriz Hessiana

(9) Actualizar el *warp* $W(u; p) \longleftarrow W(u; p) \circ W(u; \Delta p)^{-1}$

Cálculos finales:

(10) Calcular λ_i utilizando la ecuación (37)

Imagen 3: *Algoritmo Inverse Compositional con variación de textura.*

Una vez se ha resuelto el problema bidimensional, es momento de definir el modelo 3D. Como en el caso anterior del modelo 2D, también se construye un modelo 3D lineal de la forma:

$$\bar{s} = \bar{s}_0 + \sum_{i=1}^{\bar{m}} \bar{p}_i \, \bar{s}_i \tag{39}$$

donde los coeficientes $\bar{p}_i$ son los parámetros 3D de forma y $\bar{s}$, etc, son las coordenadas 3D de la forma:

$$\bar{s} = \begin{pmatrix} x_1 \, x_2 \cdots \, x_n \\ y_1 \, y_2 \cdots \, y_n \\ z_1 \, z_2 \cdots \, z_n \end{pmatrix} \tag{40}$$

El cálculo de estas coordenadas 3D se realiza utilizando las variaciones que tienen lugar en la forma del modelo 2D en distintas imágenes consecutivas. Para conseguir que los resultados se ajusten a un modelo tridimensional correcto, es necesario imponer dos restricciones métricas: las restricciones de ortonormalidad en las matrices de rotación, y las restricciones en las bases s_i del modelo. Cómo queremos combinar este modelo 3D con el modelo 2D necesitaremos un modelo que nos permita relacionar estas coordenadas 3D o coordenadas del mundo, y las coordenadas 2D o coordenadas de la imagen. En nuestro caso se ha elegido el modelo de la perspectiva débil, que se define cómo:

$$u = Px = \begin{pmatrix} i_x & i_y & i_z \\ j_x & j_y & j_z \end{pmatrix} x + \begin{pmatrix} a \\ b \end{pmatrix} \tag{41}$$

donde (a, b) hacen referencia a una traslación con respecto al origen de la imagen y los ejes de proyección i y j son del mismo tamaño y ortogonales. Como vemos la matriz de proyección P, tiene 6 grados de libertad, que podemos relacionar con la orientación 3D de la cabeza (*pitch*, *yaw*, *roll*), una traslación 2D y un factor de escalado. Este modelo de proyección es válido en el caso de que las variaciones de posición de la cara con respecto al eje Z, sean pequeñas frente a su posición Z_0. Una vez se ha definido P se relacionan las variaciones de la forma 2D con las variaciones de forma 3D, por medio de:

$$s = P\left(\bar{s}_0 + \sum_{i=1}^{\bar{m}} \bar{p}_i \, \bar{s}_i\right) \tag{42}$$

donde P y $\bar{p} = (\bar{p}_i, \ldots, \bar{p}_n)$ toman cualquiera de sus posibles valores.

Las restricciones que se a imponen a los parámetros de forma del modelo AAM p son que existan valores de P y $\bar{p}$ tales que la proyección 2D del modelo de forma 3D sea igual al modelo de forma 2D, estas restricciones se pueden escribir de la forma:

$$\min_{P,\bar{p}} \left\| N\left(s_0 + \sum_{i=1}^{m} p_i\, s_i\,; q\right) - P\left(\bar{s}_0 + \sum_{i=1}^{\bar{m}} \bar{p}_i\, \bar{s}_i\right)\right\|^2 = 0 \tag{43}$$

donde $N(u;q)$ es una transformación global que utilizamos para normalizar las posiciones de los píxeles de la malla de forma, $\|\cdot\|^2$ es la suma de cuadrados de los elementos de la matriz. Los únicos valores de esta ecuación que no son conocidos $(m, \bar{m},\ s_i,\ \bar{s}_i)$ u optimizados $(P, \bar{p}\,)$ son p, q.

Una vez construido el modelo 3D, lo se busca ajustar dicho modelo a una serie de imágenes de la cara, con diferentes poses de la cabeza. Para se parte del algoritmo de ajuste que utilizado para el ajuste AAM sin parámetros de normalización de la malla de forma q. El objetivo será minimizar:

$$\sum_{u \epsilon s_0} \left[A_0(u) + \sum_{i=1}^{l} \lambda_i\, A_i(u) - I\big(W(u;p;q)\big)\right]^2 \tag{44}$$

simultáneamente respecto de los parámetros de forma p, los parámetros de textura λ, y los parámetros de normalización q. Para combinar las ecuaciones (43) y (44), y relacionar así los parámetros de forma del modelo con los de textura, se introduce la ecuación (43) como restricciones suaves en la ecuación (44) con un constante K que ejerce de peso en esta ecuación. Así pues el objetivo es minimizar:

$$\left\| A_0(u) + \sum_{i=1}^{l} \lambda_i\, A_i(u) - I\big(W(u;p;q)\big)\right\|^2 + K \left\| N\left(s_0 + \sum_{i=1}^{m} p_i\, s_i; q\right) - P\left(\bar{s}_0 + \sum_{i=1}^{\bar{m}} \bar{p}_i\, \bar{s}_i\right)\right\|^2 \tag{45}$$

respecto de p, q, λ, P. Como vemos las constante K es quien define la importancia de las restricciones de forma, en el límite cuando $K \to \infty$, las restricciones de forma se convertirán en restricciones firmes, es decir, no se tendrá en cuenta la textura del rostro. Es interesante destacar que en la fijación de esta matriz de constantes K reside en gran parte el éxito o el fracaso de un ajuste correcto del modelo.

Se aplica a la ecuación (45) la técnica vista en el capítulo anterior para optimizar secuencialmente los parámetros de forma p, q y los parámetros de textura λ. De esta manera se quiere optimizar:

$$\left\|A_0(u) - I\big(W(u;p;q)\big)\right\|^2_{span(A_i)^\perp} + K\left\|N\left(s_0 + \sum_{i=1}^{m} p_i\, s_i; q\right) - P\left(\bar{s}_0 + \sum_{i=1}^{\bar{m}} \bar{p}_i\, \bar{s}_i\right)\right\|^2 \quad (46)$$

respecto de p, q, P y $\bar{p}$, donde $\|\cdot\|^2_{span(A_i)^\perp}$ es la norma al cuadrado L2 de los vectores proyectados en el complemento ortogonal del subespacio lineal extendido por los vectores $A_1, \ldots, A_l$. Tras lo que se calcularán los parámetros de textura utilizando la ecuación:

$$\lambda_i = \sum_{u \in s_0} A_i(u) \cdot \left[I\big(W(u;p;q)\big) - A_0(u)\right] \quad (47)$$

donde los parámetros p y q son solución de una optimización previa. El criterio de optimización para la ecuación (46) será de la forma:

$$\left\|A_0(u) - I\big(W(u;p;q)\big)\right\|^2_{span(A_i)^\perp} + F(p;q;P;\bar{p}) \quad (48)$$

para minimizar la ecuación (48) podemos utilizar el algoritmo *Inverse Compositional*, minimizando de forma iterativa:

$$\left\|A_0(W(u;\Delta p;\Delta q)) - I\big(W(u;p;q)\big)\right\|^2_{span(A_i)^\perp} \quad (49)$$

Anteriormente hemos visto cómo podemos minimizar $\left\|A_0(u) - I\big(W(u;p)\big)\right\|^2_{span(A_i)^\perp}$ utilizando el algoritmo *Inverse Compositional*. Para resolver $\left\|A_0(W(u;\Delta p;\Delta q)) - I\big(W(u;p;q)\big)\right\|^2_{span(A_i)^\perp}$ se procede de forma similar, aunque aparece una nueva transformación de semejanza 2D $N(u;q)$ que complica la resolución del problema. A continuación se describen aquellos pasos del algoritmo de ajuste del AAM, que sufren alguna modificación con respecto al ajuste 2D del modelo, debido a la transformación global de forma $N(u;q)$.

Resolución del ajuste AAM teniendo en cuenta transformaciones globales en la forma

Para ello se empleará el algoritmo *Inverse Compositional* utilizado y explicado anteriormente. El nuevo warp con el que vamos a trabajar será:

$$N \circ W(u;p,q) = N(W(u;p);q) \quad (50)$$

Donde definimos $N(u;q)$ como la transformación para la normalización de la forma global.

Por ejemplo, $N(u;q)$ puede ser el conjunto de transformaciones de semejanza 2D dadas por:

$$N(u;q) = \begin{pmatrix} (1+a) & -b \\ b & 1+a \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{51}$$

Donde los cuatro parámetros $q = (a, b, t_x, t_y)^T$ se pueden interpretar de la siguiente manera. Los dos primeros (a, b) tienen relación con la escala k y la rotación θ: $a = k\cos(\theta) - 1$ y $b = k\sin(\theta)$. Los otros dos (t_x, t_y) son las traslaciones según los ejes x e y. La ecuación (50) está parametrizada para que la transformación identidad sea $q = 0$. Cabe destacar que la de arriba no es la única forma en la que se puede parametrizar una transformación de semejanza 2D. Otra forma es definir un conjunto como un subconjunto especial de warp afines a trozos utilizados en el AAM. La malla base es $s_0 = (x_1^0, y_1^0, \dots, x_v^0, y_v^0)^T$. Si se escoge $s_1^* = s_0 = (x_1^0, y_1^0, \dots, x_v^0, y_v^0)^T$, $s_2^* = (-y_1^0, x_1^0, \dots, -y_v^0, x_v^0)^T$, $s_3^* = (1, 0, \dots, 1, 0)^T$, $s_4^* = (0, 1, \dots, 0, 1)^T$ entonces el conjunto de variaciones lineales de forma permitidas por el AAM serán exactamente igual al conjunto de transformaciones de semejanza 2D:

$$N(u;q) = s_0 + \sum_{i=1}^{4} q_i s_i^* \tag{52}$$

donde $q = (q_1, q_2, q_3, q_4)$. DE forma que la transformación entre los parámetros de ésta transformación y los de la transformación más común (transformación no lineal) de la ecuación (50) es muy sencilla:

$$\begin{aligned} a &= q_1 & t_x &= q_3 \\ b &= q_2 & t_y &= q_4 \end{aligned} \tag{53}$$

Se utilizará la representación de N dada por la ecuación (51), dado que al ser similar a W, muchas de los procesos que utilizados para W se podrán emplear de la misma manera con N, en especial la derivada del Jacobiano.

En lugar de $W(u;p)$. El ajuste de una imagen $I(u)$ a través del AAM consiste en este caso en la minimización de:

$$\sum_{u \epsilon s_0} \left[A_0(u) + \sum_{i=1}^{l} \lambda_i A_i(u) - I\big(N(W(u;p);q)\big) \right]^2 \tag{54}$$

simultáneamente respecto de los parámetros de textura λ, los parámetros lineales de forma p, y los parámetros globales del warp de forma q.

Para resolver esta minimización aplicando los mismos pasos que en el caso sin warp global de forma, utilizando como warp $N \circ W$ con parámetros (q, p) en lugar del warp W con parámetros p.

Para realizar el warp afín a trozos necesario para calcular $I\big(N(W(u;p);q)\big)$ necesitamos conocer la posición que tomarán los puntos de la malla base s_0 al aplicar el warp $N \circ W$. Se denomina al conjunto de pixeles resultantes de aplicar la transformación $W(u;p)$ a la malla s cómo $W(s;p)$, se trata de un abuso de terminología que simplificará la notación del problema. De la ecuación (2) se obtiene:

$$W(s_0;p) = s_0 + \sum_{i=1}^{n} p_i \, s_i \tag{55}$$

Ahora queremos calcular $N(W(s_0;p);q)$, por tanto debemos aplicar el warp a cada vértice en $W(s_0;p)$ con la transformación de semejanza $N(s_0;q)$, esto se puede realizar de la siguiente manera. Puesto que:

$$N(s_0;q) = s_0 + \sum_{i=1}^{4} q_i s_i^* \tag{56}$$

podemos calcular la posición de los pixeles s_0 cuando se les aplica N, en concreto, de todos los triángulos. Como ya se hizo con anterioridad, podemos calcular el warp afín para estos triángulos. Puesto que el conjunto de las transformaciones de semejanza es un subconjunto del conjunto de warp afines, se puede aplicar el warp afín N a cada vértice en $W(s_0;p)$ para calcular el cambio de la posición de la malla base producida por $N \circ W$. Por tanto este warp afín a trozos se realizará idénticamente a como se realizó considerando tan solo los parámetros locales.

Puesto que se ha cambiado el warp, también se verá modificado el Jacobiano del mismo. En este caso el Jacobiano de $N \circ W$ se define como $(\frac{\partial}{\partial q} N \circ W \, , \frac{\partial}{\partial p} N \circ W)$. Puesto que $W(u;0) = N(u;0) = u$ es el warp identidad, y se evalúa el Jacobiano en $p = 0$, $q = 0$, con lo que:

$$\frac{\partial}{\partial q} N \circ W = \frac{\partial N}{\partial q} \tag{57}$$

y

$$\frac{\partial}{\partial p} N \circ W = \frac{\partial W}{\partial p} \tag{58}$$

puesto que ambos *warps* se definen de la misma manera (los dos utilizan variaciones lineales de la forma en la malla base), el cálculo del Jacobiano $\left(\frac{\partial}{\partial q} N \circ W \, , \frac{\partial}{\partial p} N \circ W\right) = (\frac{\partial N}{\partial q}, \frac{\partial W}{\partial p})$ se realizará de forma idéntica al descrito en el caso que solo se tengan en cuentan los parámetros locales, solo que para el cálculo de $\frac{\partial N}{\partial q}$ se utilizará s_i^*, y para el cálculo de $\frac{\partial W}{\partial p}$ se utilizará s_i.

En cuanto a la inversión del warp ya demostramos anteriormente que $W(u; \Delta p)^{-1} = W(u; -\Delta p)$ para aproximación de primer orden en Δp. Si se cambia W por $N \circ W$ y los parámetros Δp por $(\Delta p, \Delta q)$ se tiene:

$$N \circ W(u; \Delta p, \Delta q)^{-1} = N \circ W(u; -\Delta p, -\Delta q) \tag{59}$$

para aproximación de primer orden en Δp y Δq.

Para la composición de warps, lo primero que se calcula es el cambio de posición de los pixeles de la malla s_0 cuando se le aplica:

$$(N \circ W)(u; p, q) \circ (N \circ W)(u; \Delta p, \Delta q)^{-1} \approx (N \circ W)(u; p, q) \circ (N \circ W)(u; -\Delta p, -\Delta q) \tag{60}$$

El cambio de las coordenadas para s_0 cuando se le aplica $N \circ W(u; -\Delta p, -\Delta q)$ se puede calcular de manera similar a como lo hemos realizado cuando no había transformaciones globales de forma. Primero se calcula:

$$W(s_0; -\Delta p) = s_0 - \sum_{i=1}^{n} \Delta p_i \, s_i \tag{61}$$

para después calcular:

$$N(s_0; -\Delta q) = s_0 - \sum_{i=1}^{4} \Delta q_i s_i^* \tag{62}$$

y el warp afín $N(s_0; -\Delta q)$ utilizando la técnica descrita por las ecuaciones (22), (23) y (24). Después se aplica este warp afín a los puntos de $W(s_0; -\Delta p)$ para obtener $(N \circ W)(s_0; -\Delta p, -\Delta q)$. Previamente ya se ha calculado la posición de los pixeles de la malla base s_0 cuando se le aplica $(N \circ W)(u; p, q)$.

Se usará la misma técnica utilizada para el caso en el que no se tienen en cuenta los parámetros locales para combinar ambos warps, y calcular así las posiciones de los pixeles de s_0 cuando se les aplica el warp de la ecuación (56). Denominaremos a los puntos resultantes como $s^{\dagger}$. Se calcula el nuevo conjunto de parámetros p y q tales que:

$$(N \circ W)(s_0; p, q) = s^{\dagger} \tag{63}$$

Por lo general resolver esta ecuación para p y q es un problema de optimización no lineal. Pero siendo N una transformación de semejanza, el problema se puede resolver de manera bastante sencilla. Primero obsérvese que:

$$(N \circ W)(s_0; q, p) = N(s_0 + \sum_{i=1}^{n} p_i \, s_i \, ; q) \tag{64}$$

puesto que N se puede escribir como la ecuación (51), esta expresión es igual a:

$$N(s_0; q) + \begin{pmatrix} (1+a) & -b \\ b & (1+a) \end{pmatrix} \sum_{i=1}^{n} p_i \, s_i \tag{65}$$

El producto entre la matriz 2×2 y con los vectores de forma de dimensión $2v$, se realiza extrayendo cada uno de los pares de coordenadas de los vértices xy correspondientes, multiplicándolos por la matriz 2×2, y reemplazándolos después. La ecuación (65) se rescribe de la forma:

$$N(s_0; q) + \left[(1+a) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \sum_{i=1}^{n} p_i \, s_i \right] + \left[b \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \sum_{i=1}^{n} p_i \, s_i \right] \tag{66}$$

El segundo término de la ecuación (66) es ortogonal a s_i^* porque s_i es ortogonal a s_i^*. El tercer término de la ecuación (66) es ortogonal a s_i^* porque para todo vector perteneciente a s_i^* si se cambia el papel de x e y y cambiamos el signo de uno de ellos, se obtiene un vector que es múltiplo uno del otro de s_i^*. Puesto que $N(s_0; q) = s_0 + \sum_{i=1}^{4} q_i s_i^*$ finalmente se va a resolver:

$$s_0 + \sum_{i=1}^{4} q_i s_i^* + \left[(1+a) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \sum_{i=1}^{n} p_i \, s_i \right] + \left[b \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \sum_{i=1}^{n} p_i \, s_i \right] = s^{\dagger} \tag{67}$$

donde la solución para q_i será:

$$q_i = s_i^*(s^{\dagger} - s_0) \tag{68}$$

Teniendo en cuenta las relaciones de ortonormalidad discutidas previamente. Una vez se conocen los parámetros q, podemos calcular:

$$p_i = s_i \cdot (N(s^{\dagger}; q)^{-1} - s_0) \tag{69}$$

Finalmente el resto de operaciones permanecerán invariantes con respecto al caso en el que no se tienen en cuenta los parámetros globales, puesto que usar el warp $N \circ W$ en lugar de W no cambia nada, salvo en el cálculo de las imágenes *Steepest Descent* para p y q, que se necesita proyectarlas en el subespacio $span(A_i)^{\perp}$ utilizando el equivalente de la ecuación (38). Calculamos por tanto las imágenes *Steeppest Descent*:

$$SD_j(u) = \nabla A_0 \frac{\partial N}{\partial q_j} - \sum_{i=1}^{m} \left[\sum_{u \in s_0} A_i(u) \cdot \nabla A_0 \frac{\partial N}{\partial q_j} \right] A_i(u) \tag{70}$$

para cada uno de los cuatro parámetros de semejanza normalizados (q_1, q_2, q_3, q_4) y:

$$SD_{j+4}(u) = \nabla A_0 \frac{\partial W}{\partial p_j} - \sum_{i=1}^{m} \left[\sum_{u \in s_0} A_i(u) \cdot \nabla A_0 \frac{\partial W}{\partial p_j} \right] A_i(u) \tag{71}$$

para cada p donde $j = 1, \dots n$. La concatenación de las imágenes *Steepest Descent* constituyen un único vector con cuatro imágenes para q seguidas de n imágenes para p. Si llamamos a los elementos de este vector $SD_j(u)$, el elemento $(j,k)^{th}$ de la matriz Hessiana de tamaño $(n+4) \times (n+4)$ se calculará como:

$$H_{j,k} = SD_j(u) \cdot SD_k(u) \tag{72}$$

En la imagen (4) podemos encontrar un resumen del proceso a seguir para resolver el ajuste del AAM teniendo en cuenta los parámetros globales q:

Algoritmo *Inverse Compositional* con variación de textura y parámetros globales

Cálculos previos:

(3) Evaluar el gradiente ∇A_0 de la plantilla $A_0\ (u)$

(4) Evaluar el Jacobiano $\frac{\partial W}{\partial p}$ y $\frac{\partial N}{\partial q}$ en $(u; 0)$

(5) Calcular las variaciones de las imágenes *Steppest Descent* utilizando las ecuaciones (70) y (71)

(6) Calcular la matriz Hessiana utilizando la ecuación (72)

Iteramos:

(1) *Warp* I aplicando $W(x; p)$ seguido de $N(u; q)$ para calcular $I(N(W(u; p); q))$

(2) Calcular el error de la imagen $I(N(W(u; p); q)) - A_0\ (u)$

(7) Calcular $\sum_{u \epsilon s_0} SD_i(u) \cdot I(N(W(u; p); q)) - A_0\ (u)$ para $i = 1, \dots, n + 4$

(8) Calcular $(\Delta p, \Delta q)$ multiplicándolo (7) por la inversa de la matriz Hessiana

(9) Actualizar el *warp* $(N \circ W)(u; p; q) \longleftarrow (N \circ W)(u; p; q) \circ (N \circ W)(u; \Delta p; \Delta q)^{-1}$

Cálculos finales:

(10) Calcular λ_i utilizando la ecuación (47)

Imagen 4: *Algoritmo Inverse Compositional con variación de textura y parametros globales q.*

Una vez se ha minimizado:

$$\left\|A_0(u) - I\big(W(u;p;q)\big)\right\|^2_{span(A_i)^\perp} \tag{73}$$

respecto de Δp y Δq, y después actualizado el warp $W(u;p;q) \leftarrow W(u;p;q) \circ W(u;\Delta p;\Delta q)^{-1}$. Al utilizar el algoritmo Inverse Compositional hemos realizado un cambio en la forma de actualizar los parámetros incrementales, de $(\Delta p, \Delta q)$ a $J(\Delta p, \Delta q)$ donde:

$$W\big(u;(p,q) + J(\Delta p, \Delta q)\big) \approx W(u;p;q) \circ W(u;\Delta p;\Delta q) \tag{74}$$

en aproximación de primer orden, y J es una matriz $(m+4) \times (m+4)$. En general, la matriz J depende de los parámetros (p,q), pero se puede calcular fácilmente. La ecuación (73) quiere decir que para minimizar la expresión de la ecuación (48) utilizando el algoritmo *Inverse Compositional*, debemos minimizar de forma iterativa:

$$G(\Delta p, \Delta q) + F((p,q) + J(\Delta p, \Delta q); P + \Delta P; \bar{p} + \Delta \bar{p}) \tag{75}$$

simultáneamente respecto de Δp, Δq, ΔP, y $\Delta \bar{p}$, donde $G(\Delta p, \Delta q)$ es la expresión descrita en la ecuación (49). La razón por la que se utiliza el algoritmo *Inverse Compositional*, es que la Hessiana de Gauss-Newton de la expresión de la ecuación (75) es una constante y por tanto se puede calcular en un paso previo. Es fácil demostrar que la Hessiana de Gauss-Newton de la expresión de la ecuación (75) es la suma de la Hessiana para G y la Hessiana para F. De manera parecida, la actualización de los parámetros del *Steepest Descent* de Gauss-Newton para la expresión completa es igual a la suma de los parámetros para cada una de las expresiones por separado. Y en consecuencia se puede crear un algoritmo de optimización eficiente a partir del algoritmo *Inverse Compositional*.

Para entender mejor todo este proceso de creación y ajuste del sistema de seguimiento de la cara, vamos a intentar explicarlo desde un punto de vista menos matemático. Vamos a explicar uno de los sistemas de seguimiento del cual ya habíamos hablado, el *Constrained Local Model* (CLM), y que es el sistema que empleará en nuestro proyecto para la creación de nuestro algoritmo.

3.1.4 Constrained Local Model (CLM)

Se va a definir el proceso completo necesario para realizar el seguimiento de la cara de una persona utilizando CLM. El sistema tiene cómo entrada una imagen, en la cual, hay un rostro humano, pero no conocemos su posición. Para conocer su posición se aplica un detector facial, como puede ser el detector de Viola-Jones, que indica en qué parte de la imagen debemos centrar nuestra búsqueda. Una vez estamos en este punto vamos a considerar lo siguiente: dada una parte de una imagen, la cual contiene una cara (y en ocasiones también parte del fondo), ¿Qué proceso permite a una persona encontrar las diferentes partes que componen la cara (nariz, la boca, etc…)?

Existen diferentes maneras para resolver esto. Una de ellas sería: puesto que se tiene una idea clara en nuestra cabeza de qué apariencia tiene una cara, por ejemplo, se conoce que su perímetro tiene forma elíptica, sabemos dónde se encuentra la posición de los ojos y la boca. Además se conoce la apariencia que tienen los ojos, de manera que seríamos capaces de reconocerlos en una imagen.

El proceso que se describe en el párrafo anterior consta de dos partes:

a. Se tiene una idea de cómo son los ojos y la nariz de una persona normal. Hablando con propiedad esto se traduce, en que existe un modelo de la nariz, los ojos, etc... Y se utiliza este modelo para buscarlos en los distintas imágenes de video que queramos ajustar.

b. También se conoce la disposición de los ojos y la nariz: los ojos se encuentran en la parte superior de la cabeza, y la nariz debajo de estos, entre ambos. Esto se utilizará como si se tratase de una serie de restricciones. De manera que existen una serie de posiciones viables para cada una de los elementos que componen nuestra cara. Estas restricciones facilitarán el trabajo, puesto que cuando se busque una de estas partes en la imagen, se centrarán en aquellas partes que no incumplan dichas restricciones.

Recapitulando, se utilizan los modelos de los elementos de la cara para buscar en una parte concreta de la imagen donde es posible que se encuentre nuestro objeto. Y gracias al conocimiento previo que tenemos podemos restringir esta búsqueda. Este es el razonamiento que está detrás del CLM. Se utilizan modelos locales para encontrar elementos concretos, y se combinan con el conocimiento previo de la forma de la cara, para restringir la búsqueda.

Una vez descrita la mecánica que hay detrás de estos modelos, se necesitan dos tipos de información diferentes. La primera es un modelo del aspecto que tiene cada parte de la cara por separado, como boca, ojos, nariz, etc. A estos lo denominaremos *patches*. Se puede considerar como un modelo de la apariencia concreta de una zona. Y se utilizará para buscar partes concretas de la cara por toda la imagen. El otro tipo de información que se necesita es dónde se va a encontrar cada uno de estos *patches*, esto será el modelo de forma. Que se utiliza para restringir la búsqueda de cada *patche* individual.

Una vez el modelo se ha construido a partir de esa información necesaria que se ha descrito en el párrafo anterior, ya se puede buscar los elementos de la cara (contorno, nariz, boca, etc…) en una imagen. La idea para esta búsqueda es muy simple: puesto que nuestro modelo nos dice la posible forma y disposición de cada una de estos *patches*, se va a buscar dónde es más probable que aparezca cada uno de estos *patches*. También se asegurará que no buscamos fuera de los límites marcados por nuestro modelo de forma.

Por tanto, conceptualmente la implementación de CLM consta de dos fases:

a. Construir un modelo CLM a partir de las imágenes de entrenamiento.

b. Usar el modelo CLM para buscar en nuevas imágenes.

Cada una de estas dos fases se explicarán por separado:

Creación del modelo

Antes de usar el modelo CLM para buscar caras en las imágenes, se necesita construir un modelo. Construir un modelo tiene que ver con aprender qué apariencia tiene una cara y los objetos que hay en ella, a veces se conoce a esta fase como proceso de aprendizaje. También se considera este proceso de construcción del modelo como una fase de entrenamiento, en la que se construye un modelo a partir de imágenes de prueba. Este modelo consta de dos partes, una que describe la variación de la forma de los puntos característicos, el denominado modelo de forma, y otra que describe la apariencia que tiene la zona que rodea a estos puntos característicos, lo que anteriormente se ha llamado modelo de *patch*.

El modelo de forma describe cómo puede variar la forma de la cara, y normalmente se construye aplicando *Principal Component Analysis* (PCA) sobre las imágenes de entrenamiento.

PCA es una técnica matemática que utiliza transformaciones ortogonales para transformar un conjunto de observaciones de variables con posible correlación en un conjunto de variables linealmente no relacionadas denominadas componentes principales. El número de componentes principales siempre va a ser menor que el número de variables originales. Esta transformación está concebida para que la primera de las componentes principales, sea siempre aquella que tenga la varianza más grande posible (es decir, representa la mayor variabilidad en los datos como le sea posible), y cada componente principal que la siga tendrá la mayor varianza posible, teniendo en cuenta que tiene la restricción de que tiene que ser ortogonal a todas las componentes anteriores.

De manera más intuitiva, lo que se busca cuando aplicamos PCA para construir nuestro modelo, son las posibles variaciones que puede presentar una cara, de manera que si se suma una combinación de estas posibles variaciones, a una media de todas las caras que podamos encontrar, nos permita reconstruir en cierto modo la cara completa. Matemáticamente lo que aquí se desarrolla, tiene que ver con el cálculo de los autovalores asociados a cada autovector, para una variable cualquiera, llamémosla forma de la cara, textura de la misma, etc. Esto va a permitir simplificar de manera importante nuestro modelo, puesto que lo que se tiene una base lineal y ortogonal del espacio para la variable a la que apliquemos PCA. Hay que destacar que este proceso es sensible a la escala de las variables de entrada. Es por eso que se utiliza un método para garantizar la invariabilidad, en lo que a la posición de la cara se refiere, como el *Procrustes Analysis*. Ya que tanto la escala, como la posición, así como la orientación de la cabeza, es posible que no sean las mismas para todas las imágenes de prueba.

El método de Procrustes Analysis compara dos objetos que se encuentran en el espacio. Para ello intentara por medio de la traslación, rotación y escalado uniforme de los objetos, que ambos

ocupen una posición parecida y tengan un tamaño acorde, minimizando lo que se conoce como la distancia de Procrustes. En nuestro caso, se conoce que todos los objetos que estamos procesando son caras, y que son las caras de una misma persona, que en teoría son idénticas unas con respecto a otras en diferentes posiciones, es coherente buscar un método que nos permita eliminar dichas variaciones, para después utilizar PCA sobre la imagen trasladada y rotada.

El modelo de *patch* describe cómo es la apariencia alrededor de los puntos de interés. Para conformar este modelo se va a utilizar los *Support Vector Machine* (SVM), aunque este no es el único tipo de descriptor que se puede utilizar.

Los SVM son una herramienta que se utiliza para el reconocimiento de patrones en cada uno de los puntos de interés de nuestro modelo. Para ello se requiere que para cada punto se haga un entrenamiento con diferentes imágenes de dimensiones iguales, en las que se encontrarán imágenes de entrenamiento que correspondan con el *patch* concreto, e imágenes que no pertenezcan a ese *patch*. De esta manera se considera a los SVM como una función lineal de la intensidad que tienen los pixeles alrededor de los puntos de interés. Solo se necesita calcular los pesos de dicha función para cada pixel en las imágenes de entrenamiento, y se usarán estas después en nuevas imágenes para comprobar si contienen el *patch* en un lugar concreto de la imagen.

Proceso de búsqueda

Una vez construido nuestro modelo CLM, puede comenzar la búsqueda de la posición de la cara, boca, nariz, ojos, etc. Este proceso se puede dividir en los siguientes pasos:

1. Búsqueda aproximada de la zona de la imagen donde se encuentra la cara utilizando algún tipo de detector como Viola-Jones.

2. Para cada punto de interés, se aplica SVM para obtener una respuesta al comparar el *patch* con la zona señalada por el detector, y obtener así una determinada imagen respuesta.

3. Se ajusta cada imagen respuesta con una función cuadrática.

4. Optimización de dicha función cuadrática, introduciendo las restricciones de forma en esta, para encontrar la posición optima en esa iteración.

5. Se repiten los pasos del 2 al 4 hasta que el algoritmo converja o se superen un número *n* de iteraciones.

A continuación se explicará cada uno de estos pasos por separado.

Pasos 1 y 2. Para comenzar la búsqueda se utiliza una aproximación inicial de los puntos de interés, por ejemplo inicializando dichos puntos como los de la base de forma s_0, que serán una media de todas las posiciones que los puntos de interés puedan adoptar. Después se aplicará el método de *Template Matching* en toda la imagen para comparar los niveles de intensidad de los pixeles de la imagen que rodean a cada punto de interés con los obtenidos durante el entrenamiento. Obtiene una imagen respuesta para cada punto de interés correspondiente con la zona concreta de búsqueda. Si el valor en la imagen respuesta es alto, significa que la probabilidad de coincidencia es alta, mientras que una probabilidad baja se traduce en un valor bajo en la imagen respuesta. Con estas imágenes se determinará la posición óptima de cada punto de interés. Se Podría pensar que lo óptimo es situar cada punto de interés allí donde tenga un mayor valor de la imagen respuesta, pero esto nos conducirá a la obtención de formas extrañas, debido a que no se cumplen las restricciones de forma. Por tanto el objetivo es encontrar la mejor posición para cada punto de interés, utilizando la información de la imagen respuesta, teniendo en cuenta las variaciones de forma permitidas por el modelo de forma.

Pasos 3 y 4. En este paso se busca encontrar en primer lugar todas las posiciones con la mayor respuesta posible, y verificar si violan las restricciones de forma, si lo hacen repetir el proceso para una posición que tenga el siguiente valor de imagen respuesta, y continuar así hasta que encontremos una que cumplan con las restricciones de forma. Con esto se obtiene la solución de nuestro problema y el problema estaría resuelto. Sin embargo el método anterior es imposible realizarlo por su alto coste computacional, por lo que se recurrirá a una simplificación. Las imágenes respuesta del problema se expresarán como funciones cuadráticas 2D, y trataremos el problema como un problema de mínimos cuadrados, en el que se busca minimizar el error cuadrático medio entre la imagen respuesta y la función cuadrática. Si se añaden las restricciones de forma a dicha función, se obtiene por tanto el problema anterior, con variables que serán la posición de cada punto característico. Optimizando esta función obtendremos la mejor posición para cada punto de interés.

Paso 5. Una vez se tiene la posición de cada punto de interés, se podría pensar que nuestro objetivo está cumplido. Pero ya que se ha realizado una búsqueda con SVM en una zona concreta alrededor de los puntos de interés, y no con respecto a toda la imagen, puede que nuestro resultado sea un máximo local. Si queremos encontrar el máximo absoluto, debemos repetir los pasos de 2 al 4, hasta que los puntos de interés alcancen una posición estable.

3.2 Detección del Iris

En este capítulo se va a detallar las técnicas empleadas para la detección del iris de forma teórica. El orden en el que se presentan es función de su orden de aparición en el algoritmo final.

3.2.1 Umbralización

Dada una imagen representada por sus niveles de gris, se busca segmentar la imagen, de forma que se permita separar aquellos objetos que nos interesan del resto de la imagen o fondo. La

umbralización consiste por tanto en determinar un valor de intensidad, al que se denomina valor umbral (k), que binariza la imagen, es decir, asigna al fondo un valor de intensidad cero, y a los objetos segmentados otro valor conocido distinto de cero. Si denotamos nuestra imagen como $f(x, y)$, aquellos puntos que cumplan $f(x, y) > k$ pasarán a ser puntos objeto siendo el resto puntos del fondo.

La umbralización puede ser multivariable como sucede con una imagen en color, pero en nuestro caso, el interés se centra en imágenes que trabajan en escala de grises, porque permiten simplificar la búsqueda del objeto, consiguiendo así una mejora en la velocidad del procesado.

El problema de la umbralización está en encontrar el valor umbral que consigue segmentar nuestra imagen correctamente. Para explicar este concepto necesitamos conocer qué es el histograma de una imagen y qué representa.

El histograma es una representación gráfica del número de píxeles que contienen el mismo nivel de intensidad dentro de la imagen. Puede entenderse como la probabilidad de que un valor de intensidad determinado aparezca en la imagen. Para conseguir que la umbralización sea lo más sencilla posible, el histograma unidimensional debe ser de tipo bimodal, esto permite separar ambas clases cómo se indica en la imagen (5):

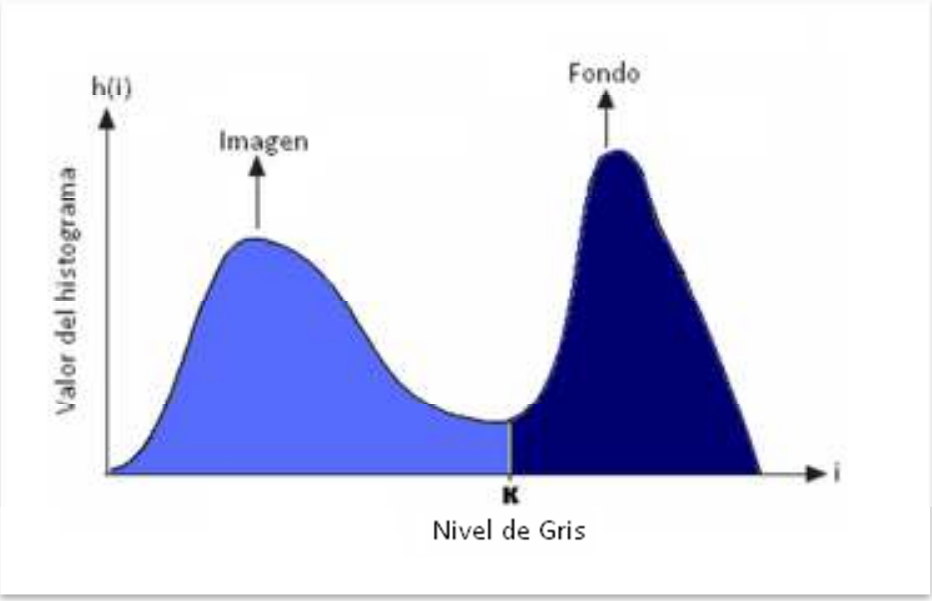

Imagen 5: *Histograma de una imagen.* La imagen muestra el histograma de una imagen donde se pueden distinguir las dos clases (imagen y fondo) y el valor umbral (k) que las divide.

Una de los problemas de la umbralización es que solo se considera la intensidad de los pixeles, y no las relaciones entre estos. Por ello no existe ninguna garantía de que los pixeles identificados como objeto sean continuos o formen parte del mismo grupo u objeto. Los errores cometidos durante la umbralización se dividen en dos grupos. Falso-positivo, puede ocurrir que se incluyan pixeles del fondo en el objeto. Falso-negativo, es decir, que no se consideren pixeles del objeto puesto que se consideren parte del fondo (ocurre sobre todo en los bordes de los objetos). Estos problemas empeoran a media que aumenta el ruido en la imagen.

3.2.2 Método de Otsu

Para ayudarnos con la elección del valor umbral (k) que separe el fondo del objeto, y minimizar así los efectos provocados por los problemas que hemos mencionado anteriormente, se utilizarán los métodos del valor umbral. En concreto se usará el método de Otsu, llamado así por Nobuyuki Otsu, quien publicó este método en el año 1979. Este método automático (se trata de un gran avance puesto que no requiere ayuda humana alguna) utiliza una búsqueda exhaustiva para encontrar el valor umbral (k) que minimiza la varianza entre clases. Se define varianza entre clases como la suma ponderada de la varianza de cada clase:

$$\sigma_w^2(k) = \ w_1(k)\sigma_1^2(k) + \ w_2(k)\sigma_2^2(k) \tag{76}$$

donde w_i representa las probabilidades de cada una de las clases separadas por un valor umbral k, y σ_i^2 representa la varianza de cada una de las clases. Calcular la varianza entre clases para cada una de las dos clases en función del valor umbral (k) requiere muchos cálculos, pero existe una forma más sencilla de resolver esto.

Se puede comprobar que minimizar la varianza dentro de cada clase, es lo mismo que maximizar la varianza entre clases:

$$\sigma_b^2(k) = \ \sigma^2 w_2(k)[\mu_1(k) - \mu_2(k)]^2 \tag{77}$$

La probabilidad de cada clase $w_i(t)$ extrae del histograma en función de k:

$$w_1(k) = \sum_0^k p\ (i) \tag{78}$$

Mientras que la media de cada clase $\mu_i(k)$ se calcula cómo:

$$\mu_1(k) = \sum_0^k p(i)x(i) \tag{79}$$

Este método tiene una serie de hipótesis que es importante conocer, puesto que el no conocerlas puede ocasionar problemas en la búsqueda del valor umbral (k):

1. Se considera iluminación uniforme.
2. El histograma ha de ser de tipo bimodal.
3. No se considera ningún tipo de estructura entre objetos, ni de coherencia espacial.

Las ventajas que nos aporta este método son su sencillez en cuanto a programación se refiere, y que opera relativamente rápido gracias al uso de histogramas para binarizar la imagen.

3.2.3 Transformada de Hough

La transformada de Hough es un método que permite encontrar objetos que tienen una determinada forma dentro de una imagen. Suele tratarse de formas sencillas, tales como líneas rectas, circunferencias o elipses. El método define una serie de parámetros de la forma a encontrar, y se crea un espacio de memoria, conocido como espacio acumulador, en el que se tienen en cuenta todas las posibles combinaciones de estos parámetros. Mediante un proceso de votación, se valoran los puntos donde es más probable encontrar la forma deseada, aquellos que contengan un mayor número de nominaciones serán los mejores candidatos a ser la forma que se busca. Para entender este proceso mejor se van a explicar los diferentes métodos que podemos encontrar, comenzando desde el más básico:

Transformada de Hough para rectas.

Este método se apoya en la idea de que todo pixel de una imagen puede formar parte de un conjunto mayor de puntos, que a su vez formen una recta. Si se parametriza cada línea, por ejemplo, por su pendiente m y su corte con el eje de ordenadas b, se obtiene que un punto (x_0, y_0) en una imagen, se corresponde con un conjunto de puntos en el nuevo espacio de parámetros (m, b), en concreto, cada punto del nuevo espacio representa una recta que pasa por el punto (x_0, y_0). Si se toman todos los puntos de la imagen cuyo nivel de gris sea mayor que cero, y los transformamos al espacio (m, b), y sumamos las contribuciones de cada uno de ellos, entonces las rectas en el espacio (x, y) aparecerán como máximos locales en el espacio (m, b). Al espacio (m, b) se le conoce como espacio acumulador, porque es donde se suman las contribuciones de todos los puntos.

Es posible que la representación con parámetros pendiente y corte con el eje de ordenadas, no sea la más adecuada por dos motivos:

1. Debido a la gran densidad de líneas que tienen una pendiente similar.
2. El hecho de que la pendiente tiene un rango demasiado amplio $(-\infty, +\infty)$.

Por estos motivos se prefiere la representación que parametriza cada recta como un punto en coordenadas polares (ρ, θ). La recta representada pasa por dicho punto pero será parametrizada por la distancia y el ángulo del radio que va perpendicular desde el origen hasta el corte con la recta, tal y como se pude apreciar en la imagen (6b):

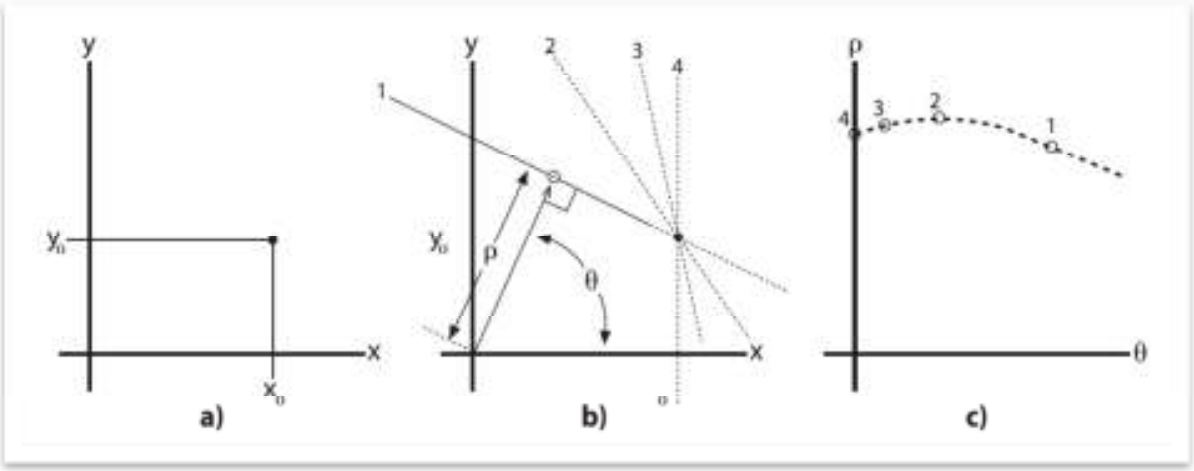

Imagen 6: *Esquema transforma de Hough.* Un punto $(\mathrm{x}_0, \mathrm{y}_0)$ en la imagen (figura a) genera muchas posibles rectas, cada una de ellas parametrizada por un ρ y un θ distintos (figura b); estas rectas en el espacio (ρ, θ) se transforman en puntos, los cuales forman juntos una determinada forma característica.

Transformada de Hough para circunferencias.

La transformada de Hough para circunferencias trabaja de manera similar a la descrita anteriormente para rectas. La razón por la que no trabaja exactamente de la misma manera, es porque si tratamos se aplica el mismo proceso, el espacio acumulador se transforma en un espacio 3D: Dos dimensiones para las coordenadas del centro (x, y), y otra más para el radio R. Repercutiendo en unos mayores requisitos de memoria y una velocidad de procesado mucho menor. Para evitar este problema se utilizará un método más complicado denominado *método del gradiente de Hough.*

El *método del gradiente de Hough* sigue los siguientes pasos. Primero la imagen pasa por una fase de detección de bordes, utilizando por ejemplo *Canny*. Después para cada punto de la imagen con nivel de gris distinto de cero, se calcula su gradiente, utilizando en este caso *Sobel*. Usando este gradiente, a cada punto de la línea recta que viene dada por esa pendiente desde una distancia mínima dada, hasta una máxima, se le incrementa su valor en el espacio acumulador. Al mismo tiempo la posición de los puntos con valores de gris distintos de cero es anotada. Los candidatos serán por tanto aquellos puntos de esta lista, que estén por encima de un valor umbral determinado y sean mayores que sus vecinos más próximos. Estos candidatos se clasificarán en orden descendiente de su valor en el acumulador, para que los centros de las circunferencias más votadas, aparezcan primero. Después para cada posible centro se consideraran todos los puntos con valor de gris distinto de cero. Teniendo en cuenta la distancia del radio de menor a mayor, se irán probando diferentes radios y viendo cuál es el óptimo en función del número de puntos distintos de cero que encajen con el mismo. Se considerará una circunferencia válida, si tiene un

número mínimo de puntos distintos de cero, y está a una distancia suficiente de otros centros seleccionados anteriormente.

Este método consigue trabajar a una mayor velocidad que si se considerase un acumulador 3D, pero quizá lo más importante, es que consigue evitar el problema de las escasas muestras posibles que tendría lugar en caso de usar el acumulador 3D, que es más sensible al ruido y hace que los resultados sean inestables. Sin embargo este método tiene algunas carencias, que debemos tener en cuenta.

En primer lugar, el uso de *Sobel* para calcular la derivada local, que se utilizará como la tangente en ese punto, no es un argumento matemático estable. Puede funcionar "la mayor parte de las veces", pero nos generará un ruido en nuestros resultados.

En segundo lugar, todos los puntos de valor de gris distinto de cero son considerados por cada candidato a centro, por lo que si el valor umbral de esos puntos es demasiado bajo, puede que el cálculo sea muy lento. En tercer lugar, puesto que solo se considera una única circunferencia para cada centro, si se tienen varios círculos concéntricos, solo se obtendrá uno de ellos.

Por último, puesto que cada centro se considera en orden ascendente respecto de su valor en el acumulador, y puesto que no se guardan nuevas circunferencias que estén demasiado cerca de circunferencias elegidas previamente, hay una tendencia a que las circunferencias de mayor radio sean las que se obtengan en caso de que haya círculos concéntricos o casi concéntricos.

4 Algoritmos desarrollados

Después de desarrollar desde un punto de vista teórico los aspectos más relevantes de este proyecto, vamos a explicar de manera detallada el proyecto desde un punto de vista práctico, centrarnos en mayor medida en los posibles problemas que han aparecido a lo largo de la consecución del proyecto. El esquema a seguir en esta sección será similar al del apartado teórico. Partiremos de una imagen inicial a la que se aplicarán algoritmos, algunos de ellos basados en los algoritmos de las librerías OpenCV y otros de la propia librería, para conseguir el objetivo final que será la obtención tanto de la orientación de la cabeza, como del ángulo de la mirada. Se explicarán por separado cada una de las siguientes fases: detección de la cara (*Facetracking*), detección del iris (*Eyetracking*).

4.1 FaceTracking

Este es un capítulo que abarca una gran cantidad de procesos, y aunque no todos pertenecen directamente al modelado de la imagen por medio de AAM, debido a su estrecha relación con la obtención del mismo, se ha decidido no separarlos en diferentes capítulos.

También es necesario resaltar que el código AAM que aquí se va a explicar, no ha sido creado por el autor de este proyecto, sino que ha sido cedido por Jason Mora Saragih, bajo unos términos de una licencia académica, que no permite su uso para fines comerciales.

Al tratarse de un programa que va a realizar un seguimiento de la cara de una persona, la entrada de este será un archivo de video, en lugar de una imagen. Puesto que sabemos que hay diferentes tipos de códec de video, resoluciones, etc… Comenzaremos verificando si las características de la imagen son compatibles con nuestras funciones, en caso de no serlo, se comprobará si existe la posibilidad de transformarlos para que sí lo sean. La primera comprobación por tanto que se realizará consiste en asegurarnos de que el archivo de video es compatible. Tras eso se verifica que la resolución de nuestro archivo de video es la correcta, en caso contrario transformaremos la imagen al tamaño deseado. La última comprobación antes de comenzar el proceso de identificación de la cara, es verificar que la imagen es de un solo canal, sino es así, tal y como se hizo en el caso anterior se procesará la imagen para que lo sea.

Tras asegurarse que nuestra imagen tiene las propiedades que se quieren, seguiremos con el proceso de ajuste e identificación de las características del rostro, cuyos pasos están esquematizados en la imagen (7):

1 Obtener imagen.
2 Detectar cara usando Viola-Jones
3 Inicializar malla de forma 2D
4 Calcular parámetros del modelo 3D
    4.1 Calcular malla de forma 3D
    4.2 Alinear mallas 2D y 3D
    4.3 Actualizar malla 3D
    4.4 Calcular parámetros
    4.5 Repetir estos pasos hasta que tol<tolmax o n>nmax
5 Ajustes para optimizar el modelo
    5.1 Actualizar malla de forma 2D con los últimos parámetros calculados
    5.2 Alinear esta malla con las de entrenamiento
    5.3 Obtener *patches* de la imagen y comparación con imágenes de entrenamiento
        5.3.1 Seleccionar la vista más apropiada en función de las imágenes entrenadas
        5.3.2 Calcular el tamaño de imagen a extraer
        5.3.3 Obtener *patche* teniendo en cuenta la normalización de la imagen
        5.3.4 Obtener la imagen respuesta de cada *patche*
6 Normalización de la malla con parámetros de semejanza
7 Optimización de malla de forma y de textura
    7.1 Optimización de parámetros globales del modelo
        7.1.1 Combinar información de la malla de forma global y de textura en una función
        7.1.2 Cálculo de los parámetros obteniendo el mínimo de dicha función
        7.1.3 Se repiten los pasos anteriores hasta que tol<tolmax o n>nmax
    7.2 Optimización de parámetros locales del modelo
        7.2.1 Combinar información de la malla de forma local y de textura en una función
        7.2.2 Cálculo de los parámetros obteniendo el mínimo de dicha función
        7.2.3 Se repiten los pasos anteriores hasta que tol<tolmax o n>nmax
8 Actualizamos la posición de la cara que será utilizada en el siguiente ajuste

***Imagen 7**: Algoritmo de ajuste del modelo CLM.*

La descripción del algoritmo comienza, siguiendo el orden del diagrama de flujo anterior. Considerando que la imagen de entrada es nueva y por lo tanto no ha sido detectada todavía ninguna cara en ella, la primera etapa consiste en realizar una búsqueda rápida de qué objetos se encuentran en la imagen. Para lo cual vamos a utilizar un detector de tipo Viola-Jones. A continuación se va a describir brevemente cómo funciona este detector, enfocándolo especialmente en aquellos aspectos que son interesantes para nuestro proyecto:

Viola-Jones

El detector de caras utilizado, es una versión del famoso detector creado por Paul Viola y Michael Jones conocido como "Viola-Jones detector", aunque OpenCV ha incorporado una serie de mejoras a este y le ha dado el nombre de "Haar Classifier" porque utiliza *Haar features* como "materia prima" para los clasificadores empleados para identificar objetos en de la imagen. Los *Haar features* se pueden describir como filtros rectangulares, que aplicados a la imagen crean una segmentación para un determinado trozo de imagen como se aprecia en (8):

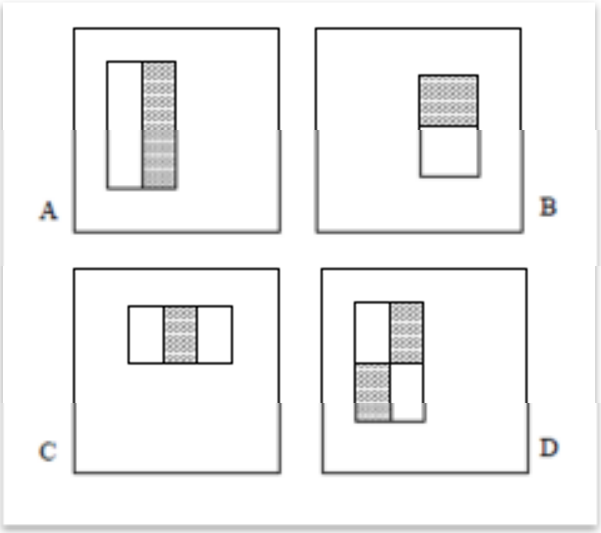

Imagen 8: *Haar features.* Algunos ejemplos de Haar features utilizados para extraer las características de la imagen.

La imagen (8) muestra dos zonas diferenciadas en cada rectángulo, dependiendo de qué zona se trate, se restará o sumará el nivel de gris de los pixeles correspondientes de la imagen, creando así un descriptor de una zona determinada. Como paso previo se requiere la construcción de estos identificadores mediante un entrenamiento. A cada parche de la imagen que contenga una cara se le etiquetara como cara, y en caso de no contener una cara se la etiquetará como fondo. Esto va a permitir utilizar posteriormente un sistema de clasificación supervisado para reconocer las posibles caras dentro de la imagen. Se trata de un sistema de clasificación supervisado, porque etiquetan cada parche de la imagen para ampliar la información de los datos (es una cara o no es una cara). No vamos a explicar el tipo de clasificador que utiliza, porque para nosotros es suficiente con saber que gracias al uso del mismo se consigue una tasa de detección grande (tiene pocos falsos negativos o caras sin detectar), al coste de que la tasa de rechazo también sea baja (tiene muchos falsos positivos o caras detectadas erróneamente).

Los pasos que sigue el programa para la detección de los posibles objetos son:

1. Calcula las *Haar features* para utilizarlas como variables de entrada, se aplicara un valor umbral a estos para decidir si son o no parte de una cara.

2. Para acelerar el proceso de cálculo de las *Haar features* se utiliza una técnica conocida como cálculo integral. Tampoco vamos a explicar en qué consiste porque lo único que nos interesa es saber que facilita el cálculo de estas características.

3. Utiliza *stadistical boosting* para crear una clasificación binaria (cara o no cara) que se caracteriza por tener una alta tasa de detección y una baja tasa de rechazo. Este paso se encarga de relacionar cada parche de la imagen (o características de la misma) con el objeto que deseamos encontrar. Lo hace por medio de una función a la que se asignan diferentes pesos para cada característica de la imagen en función de la importancia que tengan dentro del objeto. El resultado de la función tiene 3 posibles valores, o bien un -1 si esta es menor que 0, 0 si es igual a 0 y 1 si es mayor que 0.

4. Por último se utiliza una clasificación de nodos en cascada. Donde los primeros nodos serán aquellos que identifiquen un mayor número de objetos aunque permitan muchas detecciones erróneas, el siguiente nodo será el segundo mejor en cuanto a detección con menor tasa de rechazo, y así hasta el final. El algoritmo considerará como objeto aquel que haya superado la cascada completa.

Puesto que nuestro interés se centra solo en encontrar una zona determinada de la imagen en la que luego realizar nuestra búsqueda de la cara, y no es crítico determinarla de manera exacta, antes de comenzar a usar el detector de Viola-Jones vamos a reducir el tamaño de la imagen. Esto supondrá un ahorro en lo que se refiere a tiempo computacional. También previo a la ejecución de este algoritmo es conveniente realizar un ecualizado del histograma, que extiende los valores de intensidad de la imagen, lo cual es necesario porque las características de las *Integral Image* se basan en diferencias entre regiones rectangulares, por eso si el histograma no está ecualizado, las diferencias podrían estar sesgadas por la iluminación general o por la exposición de las imágenes de prueba.

Tras detectar la posición de la imagen donde se encuentra nuestra cara, comienza el algoritmo AAM propiamente dicho. Se inicializará la malla de forma 2D, de tal forma que los puntos de interés de la malla se reorganicen en las nuevas posiciones que les corresponden teniendo en cuenta el cambio de sistema de coordenadas dado por nuestro detector de cara. En este paso además de utilizar la información del detector de Viola-Jones, los puntos de la malla de forma se inicializarán a unos valores arbitrarios, llamados valores de referencia.

Como se explicó en la introducción teórica, para poder realizar el seguimiento de la cara, una vez se ha completado la fase de entrenamiento, se necesita calcular los parámetros de forma de nuestro modelo 3D que viene determinado por la ecuación (39).

Para ello se realiza un proceso iterativo al que denominaremos cálculo de parámetros 3D. El cual comienza con el cálculo de la malla de forma 3D. Considérese que se trata de una primera

iteración, con lo que dicha malla se inicializa al valor medio de la malla de forma, este dato se ha obtenido gracias a la información recopilada de las imágenes de entrenamiento. En el caso de que no sea la primera iteración la posición de la malla 3D se puede calcular directamente utilizando la ecuación (39), donde los parámetros p_i serán los obtenidos al final de este proceso iterativo.

Una vez ambas mallas han sido inicializadas, el siguiente paso consiste en alinear la malla 3D, con la malla 2D de nuestra imagen. Este alineamiento nos proporciona los parámetros del modelo de la perspectiva débil que usamos para relacionar las coordenadas del mundo con las de la imagen tal y como se muestra en la ecuación (41). Para conseguir alinear ambas mallas, se utiliza la descomposición en valores singulares (DVS) de (41), para poder utilizar la matriz pseudoinversa.

El proceso iterativo continua recalculando la posición de los puntos de la malla 3D en función de los giros y traslaciones que se han obtenido en nuestro modelo de la cámara. Una vez los puntos de la malla 3D se encuentran en una posición no arbitraria, sino estimada, se calculan los parámetros de nuestro modelo por primera vez:

$$\bar{p}_i = \bar{s}_i \cdot (\bar{s} - \bar{s}_0) \tag{80}$$

Donde $\cdot$ representa el producto escalar entre dos vectores, e i es el número de puntos de interés de los que consta nuestra malla. Se observa además que esta ecuación es simplemente la ecuación (39) despejada. Para terminar se comprueba la diferencia entre los parámetros $\bar{p}$ hallados con los de la iteración previa (en caso que exista la iteración previa), si esta diferencia es pequeña (en término de la norma entre ambos vectores) el cálculo de los parámetros 3D habrá terminado y nuestros parámetros $\bar{p}$ se habrán hallado con suficiente exactitud, en caso contrario repetiremos el mismo proceso, utilizando ahora los nuevos parámetros calculados para la siguiente iteración. En ocasiones los parámetros no podrán ser determinados con la exactitud deseada, en ese caso tras un numero n de iteraciones la rutina terminará. Al acabar esta rutina se habrán determinado, por tanto, los parámetros $\bar{p}$ de nuestro modelo 3D, además de los parámetros de la cámara, que serán los que relacionen las coordenadas del mundo con las de la imagen.

Una vez se ha terminado con la búsqueda de los parámetros del modelo, se realizará el ajuste y optimización tanto de la malla de forma cómo de los *patches* de textura. El proceso comienza actualizando nuestra malla de forma 2D a la nueva posición dada por los parámetros $\bar{p}$ obtenidos. Con nuestra nueva malla de puntos 2D se aplica el método de Procrustes, con lo que se encontrará una transformación de semejanza entre la malla de forma obtenida actualmente, y aquellas que han sido obtenidas durante el entrenamiento. Este proceso evita tener añadir un offset debido al descentrado de nuestra malla con respecto al centro de la imagen.

Ahora que las mallas de entrenamiento se encuentran alineadas con las actuales (o lo que es lo mimo, ambas caras se encuentran alineadas) comenzamos el proceso de búsqueda de los distintos *patches* de la imagen (se calcularán 3 *patches* de distintos tamaños para cada punto de interés de la malla), para posteriormente compararlos con los *patches* de entrenamiento. Entonces para cada uno de los puntos de la malla 2D, se crea un *patche* de la imagen, o lo que es lo mismo, conjunto

de pixeles alrededor de los puntos de interés de la imagen, los cuales se comparan siguiendo los siguientes pasos:

1. Se selecciona el *patche* correspondiente a la vista que mejor se corresponda con nuestra orientación de la cabeza actual de entre todos los entrenados.

2. Se calcula el tamaño de la imagen a comparar, que será función de la orientación actual y el tamaño del *patche*.

3. Se extrae esta imagen a partir de la imagen original, teniendo en cuenta los parámetros de semejanza obtenidos (se normaliza la posición de la cara para compararla con la textura de las imágenes de entrenamiento), y el tamaño calculado anteriormente. Para ello se empleará la función *cvGetCuadrangleSubPixel()* de OpenCV, que ofrece una serie de ventajas durante la adquisición de imágenes en caso de encontrarnos en los bordes de la imagen original, o en el caso de extraer pixeles de la imagen que no estén en coordenadas enteras, para lo que utilizará la interpolación bilineal.

4. Se calculará la imagen respuesta de cada punto y para cada tamaño de *patche*.

Antes de seguir avanzando, se va a explicar en profundidad la manera de calcular la imagen respuesta. Para comparar la respuesta del *patche* de la imagen se debe comparar el *patch* con las plantillas de entrenamiento. Para comparar los distintos *patches* utilizaremos la función *cvMatchTemplate()* de OpenCV, que obtiene una imagen respuesta superponiendo una plantilla a cada una de las áreas de la imagen. Esta imagen respuesta representa la probabilidad de encontrar el objeto que estamos buscando (*patche*) centrado en ese punto de la imagen. Puesto que las probabilidades representadas en la imagen no suman uno, se procede a escalarlas para que lo hagan. El proceso en el caso de que dispongamos de varios *patches* para comparar no varía salvo que una vez se calcule la respuesta de cada *patche* por separado, se normalizará para que las probabilidades sumen uno, y finalmente se multiplicarán todas ellas. Por tanto después de este se obtiene una matriz respuesta para cada punto perteneciente a la malla 2D, y para cada uno de los tamaños de *patche* concretos.

A partir de cada una de estas imágenes respuesta, y de las restricciones del modelo, se va a realizar un proceso de optimización. El proceso de optimización consta de dos partes:

- Un primer proceso de optimización de los parámetros globales, donde se considera que la cara es un objeto rígido y por tanto solo modificaremos las variables que parametrizan los movimientos de la cara como un objeto indivisible (Parámetros q).

- Un segundo proceso de optimización que permite posibles variaciones en los parámetros locales del algoritmo, por tanto utilizaremos los modos de variación del modelo, es decir, sus autovectores, para encontrar la posición concreta de cada punto de interés de nuestra malla.

Tenemos que tener en cuenta que previamente a este proceso de optimización se ha de normalizar la posición de la malla para compararlas con los puntos de interés de la malla de entrenamiento. En ambos proceso de optimización se pretende ajustar la imagen respuesta a una función cuadrática, y a su vez añadir las restricciones de forma a dicha función. Por lo tanto si se trata de minimizar de forma iterativa esta función, que contiene información tanto de la imagen respuesta como de la variación en la distancia que sufren los puntos de interés, podremos encontrar así los parámetros del modelo (tanto globales, como locales). Al tratarse de un proceso iterativo la solución se puede alcanzar tras un número n de iteraciones si se cumple que:

$$\|s^n - s^{n-1}\| \leq tol \tag{81}$$

Donde s es la matriz de puntos de interés, y tol es una tolerancia elegida por el usuario. O bien si se ha superado el número máximo de iteraciones el proceso se da por terminado. Puesto que hemos acabado con el proceso de optimización de los parámetros, podemos entonces calcular nuestra malla de forma definitiva con parámetros locales p y globales q. Para terminar con el proceso de ajuste, se actualiza la posición en la imagen donde se encuentra la cara en la imagen siguiente, obteniendo un rectángulo de la imagen dónde comenzar dicha búsqueda. Con esto se da por concluido el capítulo que describe el funcionamiento del *Facetracker*, pese a que no se ha descrito la función que permite al programa representar el modelo en la imagen, debido a que no se ha considerado una parte esencial del algoritmo y su complejidad es mucho menor.

4.2 EyeTracking

El sistema de seguimiento de la mirada (*EyeTracking*) utiliza la información del modelo AAM para conocer la posición de los ojos en la imagen. Una vez se conoce la posición de ambos ojos se realizan una serie de transformaciones sobre la imagen de entrada, para obtener una imagen en blanco y negro de la zona de los ojos.

Esta etapa de preparación comienza definiendo una zona de interés (conocida por sus siglas en inglés ROI) dentro de la imagen sobre la que se va a trabajar. Esta ROI está definida por un rectángulo que tendrá como coordenadas la zona que abarca cada uno de los ojos. Se considera a una ROI cómo una máscara, la cual estará presente en todas las funciones que apliquemos a partir del momento en que esté activa. En OpenCV la función que activa una ROI es *cvSetImageROI()*. A continuación se modifica la zona de la imagen, donde la imagen pasará a representarse en escala de grises en caso de tratarse de una imagen a color.

El siguiente punto consiste en la creación de una máscara que nos permita obtener la forma exacta del ojo, en lugar del rectángulo calculado con el ROI. Este paso es importante puesto que se

necesita eliminar aquellos pixeles de la zona del ojo que no formen parte del iris o de la córnea. Este proceso empleará, de nuevo, la información de la posición de los puntos de interés de ambos ojos. A partir de ellos, se puede dividir nuestra imagen en dos zonas, en función de si los puntos se encuentran dentro de la región delimitada por los párpados o no, tal y como se puede ver en la imagen (9):

Imagen 9: *Máscaras del ojo.* Se presentan varias imágenes de máscaras de ambos ojos, donde observamos como se delimita el ojo con la información obtenida por el Facetracker. Nos ayuda a eliminar pixeles que pueden influir durante el cálculo del valor umbral (k).

Una vez se ha calculado tanto la región de interés como la máscara, se va a realizar una copia de la imagen en escala de gris, utilizando la máscara como plantilla para la copia. La función cvCopy() permite realizar este proceso, obteniendo como resultado la imagen que buscábamos de la zona del ojo.

La siguiente fase del código consiste en umbralizar la imagen obtenida de la zona del ojo en el apartado anterior. Para ello se ha de considerar el histograma de la imagen en primer lugar. En OpenCV no existe una función que realice el histograma. Por tanto partiendo de un vector $hist = (h_1, h_2, \dots, h_{256})$, se recorrerá la imagen y acumulará el número de pixeles que contiene un determinado nivel de gris en $hist$.

El último paso antes de realizar la umbralización de la imagen, será calcular el valor umbral que nos permita diferenciar el fondo de la imagen. Para ello se aplicará el método de Otsu, que fue detallado en un capítulo previo durante la introducción teórica. Para llevar a cabo este proceso tampoco existe una función concreta en OpenCV que lo implemente, por lo que se ha tenido que escribir un código que lo implemente. Finalmente, y una vez se conoce el valor umbral óptimo a partir del cual vamos a umbralizar, se realizará dicha operación utilizando la función de OpenCV *cvThreshold().* El resultado de esta operación será la imagen (10), donde tanto el iris como los párpados estarán a nivel de gris 0, mientras que el fondo del ojo tendrá un nivel de gris 255. A continuación podremos substraer de dicha imagen la imagen (9) creada previamente, utilizando la función *cvSub(),* e invertir los niveles de gris de la imagen por medio de la función *cvNot().* Con esto conseguiremos obtener una imagen donde el iris está definido por el nivel de gris 0, mientras que el resto de la imagen es el fondo con nivel de gris 255.

Imagen 10: *Imágenes umbralizadas.* Conjunto de imágenes umbralizadas de ambos ojos. Se obsera cómo la detección del iris será mas sencilla en caso que el mismo esté centrado. Además se aprecia la importancia de una correcta iluminación, ya que los reflejos producidos en el iris pueden provocar fallos en la detección.

Tras la obtención de la imagen del iris umbralizada, se procede a encontrar el centro de dicho iris, que se considerará la pupila, y el radio del mismo. Aunque el iris tiene una forma ovalada, en este proyecto vamos a considerar que podemos aproximar la forma de éste a un círculo sin que lo acuse de forma significativa al realizar el cálculo posterior de la dirección de la mirada. Para conseguir ajustar la forma del iris a la de un círculo, previamente debemos obtener el perímetro externo de dicha forma. Se puede conseguir extraer el perímetro de una forma cualquiera por medio de una serie de operaciones morfológicas aplicadas a la imagen. En este caso en primer lugar se aplica una erosión, seguido a su vez de una apertura (formada por una erosión seguida de una dilatación), para a continuación diferenciar la imagen del iris completo, con la imagen del iris al que se le han aplicado las transformaciones anteriores.

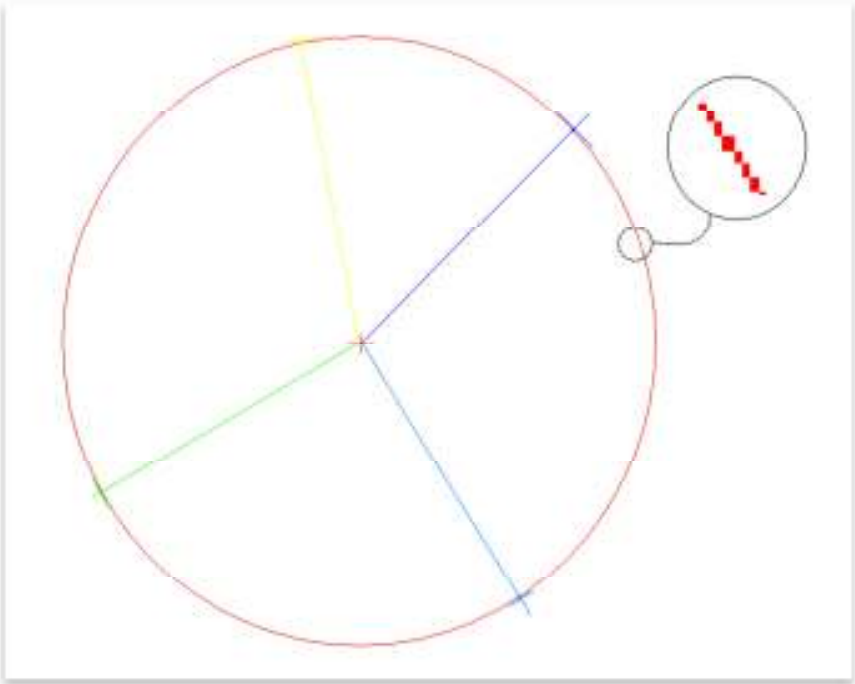

Imagen 11:*Dientes de sierra.* En el esquema superior se puede observar como a partir de un cículo que a simple vista parece continuo, si ampliásemos la vista aparecería ese perfil de dientes de sierra, que provoca que la detección no sea correcta utilizando cvHoughCircles().

Con la imagen dada de la circunferencia del iris se buscará el centro de la circunferencia. Para ello se podría utilizar la transformada de Hough para círculos. Como ya se explicó en la introducción teórica, el algoritmo de Hough es un método que elige el centro de una circunferencia atendiendo al número de votos que ha recibido por parte de los puntos que forman el perímetro del iris en nuestra imagen (puntos con nivel de gris mayor que 0 en la imagen). Sin embargo y puesto que nuestro perímetro no es idealmente continuo, sino que existe un escalonamiento debido a los pixeles que lo definen, este perímetro tendrá en muchas zonas lo que se denominan como *dientes de sierra*, que hacen imposible encontrar el centro usando este sistema, tal y como se puede observar en la imagen (11).

El método que se empleará para sustituir al implementado por la función cvHoughCircle() es el de votación tradicional, utilizado para cvHoughLines(), donde el espacio acumulador tendrá 3 variables (x, y, R). Se utilizará esta opción por tres motivos:

- En primer lugar, porque el tamaño de la imagen al que estamos aplicando la transformada de Hough es pequeño (nunca superior a los 20×20 pixeles).

- En segundo lugar, tras haber realizado la umbralización y haber obtenido el perímetro, el número de puntos con valores de gris mayores que 0 (que son aquellos que se van a considerar en las votaciones) es muy reducido.

- En tercer y último lugar, el algoritmo de Hough para circunferencias, analiza únicamente el radio más votado en una única imagen. En nuestro caso también estábamos interesados en encontrar el radio del iris que más se repite para el conjunto total de imágenes. Esta información es muy útil puesto que se puede considerar que nuestro radio del iris es prácticamente constante a lo largo del tiempo. Por medio de esta característica se extraerá un radio medio con un suficiente número de imágenes. A partir del cual, se buscará la posición de dicho radio medio en el espacio acumulador. Esto nos proporciona por tanto la posición de la circunferencia para el tamaño de ojo que más veces se repite.

Por consiguiente la velocidad del método no se incrementará de forma sustancial y además se realiza una estimación del radio medio para todas las imágenes. No existe ninguna función dentro de las librerías de OpenCV que implemente este algoritmo, por lo que se ha tenido que crear un algoritmo propio. Este algoritmo se puede dividir en tres etapas:

La primera etapa consiste en la creación y actualización de la base de datos de los radios y puntos más votados. Así pues se tiene una matriz tridimensional que almacena los votos para cada uno de los puntos de la imagen (x, y) y para cada radio posible R. De esta forma una vez se haya completado de comprobar todos los posibles círculos de la imagen, podremos extraer el radio más votado en la segunda etapa de este algoritmo. En el tercer paso se actualizará la lista para los radios más votados para todas las imágenes y se extraerá la posición más probable (la más votada) para encontrar el centro de este radio alternativo. El resultado de este proceso lo podemos observar en la imagen (12), donde el círculo verde corresponde al radio medio de todas las imágenes, y el rojo corresponde al radio más votado en una única imagen.

Imagen 12:*Fotogramas detección color.* En la imagen superior se observa la detección del ojo con la malla 2D también representada. Observesé como el circulo detectado puede superar los límites de la mallla. Abajo se obsera la detección del radio medio entre todas las imágenes en verde, donde en este caso coincide con la detección para esa imagen.

4.3 Cálculo de la mirada

La última parte del algoritmo se centra en el cálculo final del ángulo de los ojos y de la posición de la cabeza. En primer lugar se explicará el proceso de cálculo correspondiente al ángulo conseguido por medio del movimiento de los ojos. Para ello vamos a considerar el globo ocular como una esfera de radio R_e. Además vamos a suponer que esta esfera sobresale del ojo también R_e. Como se puede ver en la imagen (13). El ángulo b que corresponde con el ángulo de *yaw* se calcula por medio de la ecuación (82):

$$b = \text{atan}(\frac{x}{R_e}) \tag{82}$$

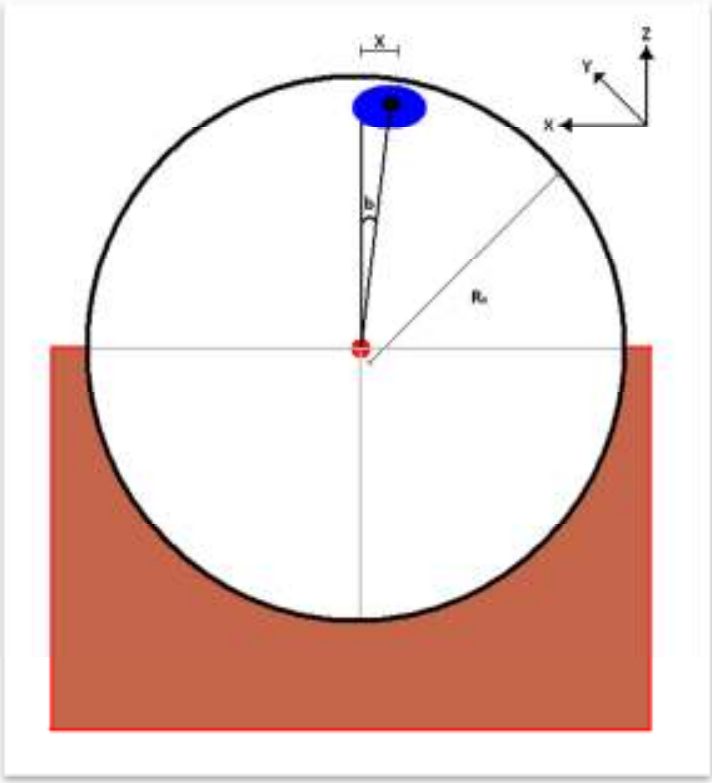

Imagen 13: Esquema representativo del ojo visto desde arriba. Observese como para este modelo las posiciones cercanas al centro del ojo son más verosímiles que aquellas alejadas del mismo.

Este proceso se puede utilizar para el cálculo del ángulo de pitch (c). Tan solo se necesita modificar la distancia x por y, tal y como se ve en la ecuación (83):

$$c = \text{atan}(\frac{y}{R_e}) \tag{83}$$

El último de los cálculos corresponde al ángulo de giro de la cabeza. Este ángulo es un dato del modelo tridimensional del CLM. El modelo permite conocer estos ángulos, ya que conocemos los parámetros del modelo de la perspectiva débil, que han sido calculados durante el proceso de ajuste del modelo. Por tanto el ángulo final de la mirada será la suma del ángulo obtenido de los parámetros del modelo de perspectiva débil de la cámara y el ángulo b (o c) obtenido con la ecuación (82).

Podemos ver en la imagen (14) el resultado final de nuestro seguimiento del rostro, para diferentes fotogramas de un mismo video.

Imagen 14: *Fotogramas de detección final.* Una serie de fotogramas que muestran el mallado del Facetracker tras el ajuste, y la detección de los ojos realizada por el eyetracker. Se pueden apreciar los errores en la detección del ojo en algunos de los fotogramas.

5 Resultados

Tras la descripción de todos los procesos que se han utilizado para conseguir realizar el seguimiento de la mirada y de la pose de la cabeza, pasaremos a describir el montaje y los resultados obtenidos en el proyecto.

5.1 Descripción del montaje

En primer lugar explicaremos brevemente la forma en la que se han obtenido las imágenes para la toma de resultados. La posición de los distintos objetos se muestra en el esquema representado en la imagen (15).

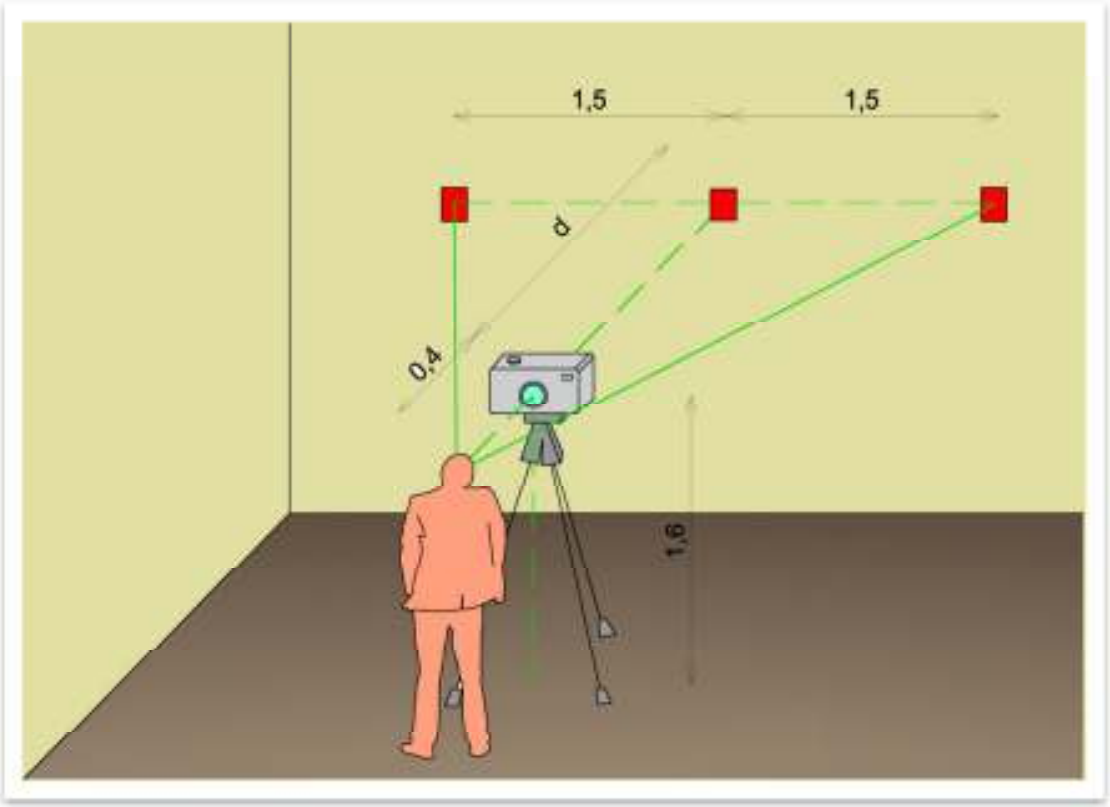

***Imagen 15**: Esquema de la instalación para la obtención de las imágenes.* Para la toma de imágenes el usuario se sitúa a una distancia de la cámara de 0.4 m, estando ojos y cámara en el mismo plano horizontal. La cámara a su vez se separa una distancia d (que toma los siguientes valores dependiendo de la secuencia: 1.5 m, 3 m, 4.5 m) del objetivo central. El usuario dirigirá su mirada a tres puntos de interés, separados entre sí por 1,5 m.

Como se puede observar en la imagen (15) el sistema consta de tres elementos principales: el usuario, la cámara, y por último los puntos objetivo. Vamos a explicar que precauciones hemos tenido a la hora de posicionar cada uno de ellos en su lugar correspondiente.

En el caso del usuario, es muy importante conseguir que la iluminación sobre la cara fuese lo más homogénea posible, por ello se colocó a la persona frente a una amplia ventana y no se emplearon

focos de luz artificiales sobre ella. Otro de los factores a tener en cuenta cuando se sitúa a la persona, es el del fondo de la habitación. Conviene que el fondo de la habitación tenga colores que sean diferentes a los de la cara de la persona, ya que en caso contrario puede ocurrir que la detección de la cara sea defectuosa.

En cuanto a la colocación de la cámara, hay que destacar sobre todo la gran importancia que tiene la distancia a la cara para que la detección sea correcta. En nuestro caso no se han realizado medidas complementarias que ayuden a constatar este hecho, pero durante la realización del proyecto sí que se ha observado que uno de los factores que más afecta a la detección correcta de la cara es la distancia entre esta y la cámara. Por otro lado recordar que la cámara ha sido calibrada previamente a la toma de imágenes.

Por último los puntos objetivo, que en nuestro caso son 3, y se encuentran separados entre sí una distancia de $1{,}5\ m$ en la recta horizontal que los une. La razón por la que sólo se han tomado 3 puntos objetivos es por una cuestión de espacio. Se podrían haber introducido nuevos puntos objetivo entre medias de estos, pero debido a la exactitud del sistema, se ha optado por una configuración más simple, que permita observar los errores cometidos durante la detección. Finalmente señalar que la distancia d que aparece en el esquema es una variable que tomará 3 valores distintos $d = (1{,}5\ m, 3\ m, 4{,}5\ m)$, de nuevo el motivo por la que solo se obtienen 3 medidas es la falta de espacio.

5.2 Resultados experimentales

Una vez descrito el montaje se procede a presentar los resultados experimentales. Los resultados presentados son medidas de la dirección de la mirada del usuario (representado por el ángulo de *yaw*), cuando este dirige la mirada a cualquiera de los puntos de interés. La secuencia utilizada para tomar los datos es la siguiente: en primer lugar se sitúa la cámara y la persona e la primera de las distancias d, para posteriormente pedirle al usuario que fije la posición de la cabeza en uno de los tres posibles ángulos para los que vamos a tomar medidas. El usuario dirigirá su mirada a los puntos objetivos, una vez haya acabado de recorrer con su mirada todos los puntos se repetirá el proceso para las distintas posiciones de la cabeza. Este proceso se repite para todas las posibles distancias d. El número de medidas tomadas es superior a 30 imágenes para cada uno de los puntos objetivo.

- Caso 1: Pose de la cabeza 1 y mirada dirigida al punto objetivo situado en la izquierda

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	9,24	8,07	7,49	16,73	15,55	45,00
3	14,89	7,57	-4,03	10,86	3,54	26,57
4,5	14,93	3,72	-8,56	6,37	-4,84	18,43

***Tabla** 1: Medidas obtenidas para el caso 1.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 1 y el punto objetivo está situado a la izquierda.

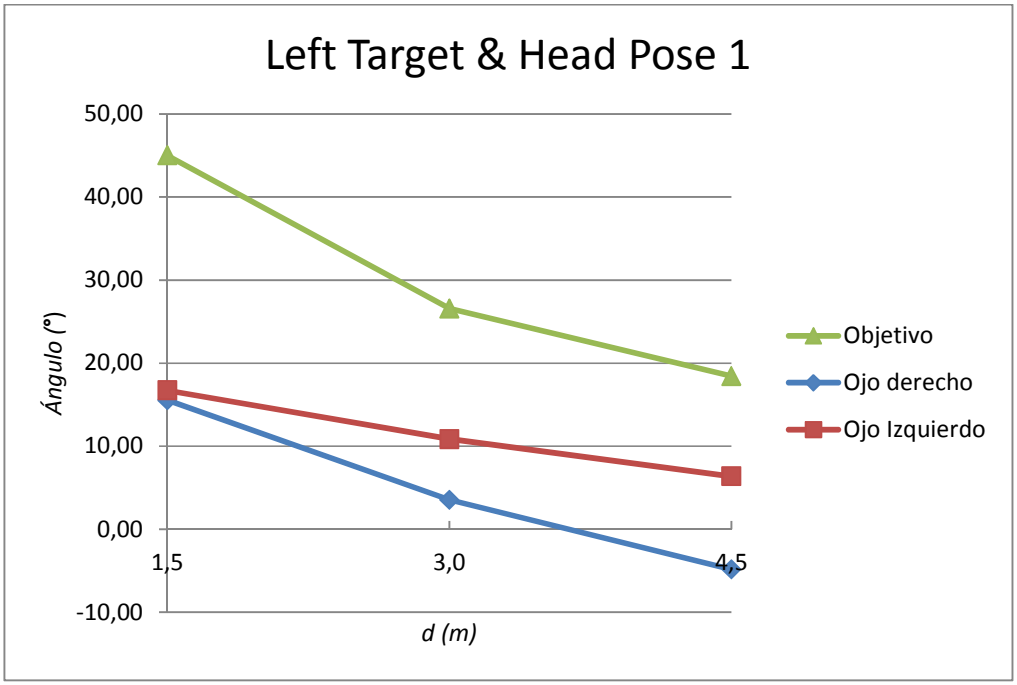

Figura 1: *Representación gráfica resultados caso 1.* Se observa la diferencia entre los valores medidos con la cabeza mirando con el objetivo situado a la izquierda, y los valores objetivo. A mayores distancias, decrece la diferencia entre las medidas y el objetivo.

- Caso 2: Pose de la cabeza 2 y mirada dirigida al punto objetivo situado en la izquierda

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	8,01	10,89	8,15	16,16	19,05	45,00
3	14,50	7,20	-3,64	10,86	3,56	26,57
4,5	5,32	4,07	7,64	12,96	11,70	18,43

Tabla 2: *Medidas obtenidas para el caso 2.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 2 y el punto objetivo está situado a la izquierda.

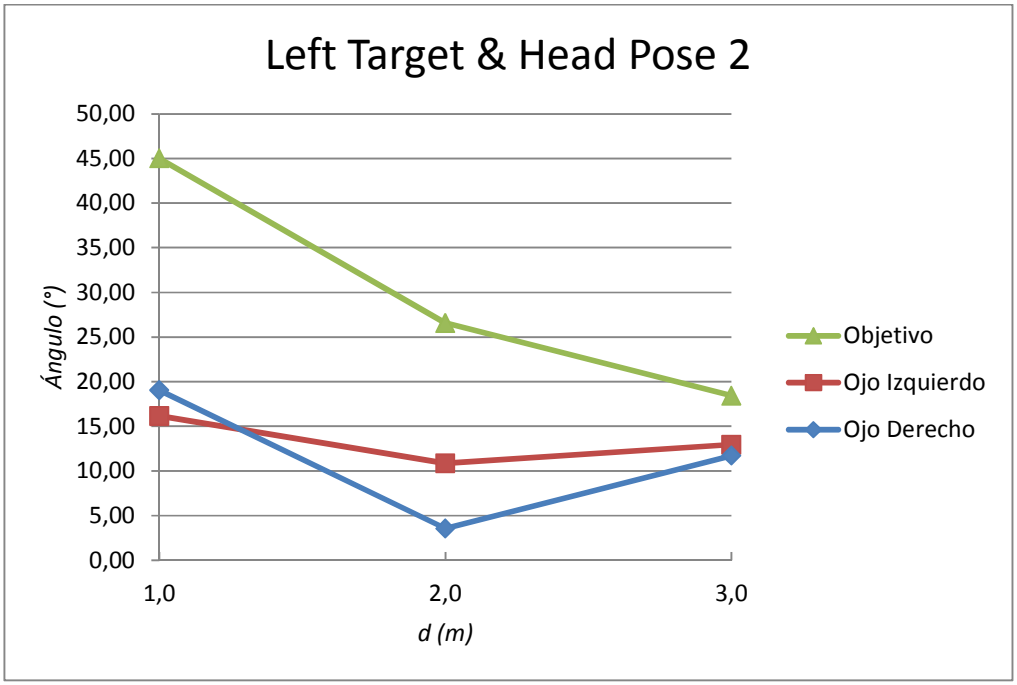

Figura 2: *Representación gráfica resultados caso 2.* Se observa la diferencia entre los valores medidos con la cabeza centrada con el objetivo situado a la izquierda, y los valores objetivo. A mayores distancias, vuelve a decrecer la diferencia entre las medidas y el objetivo.

- Caso 3: Pose de la cabeza 3 y mirada dirigida al punto objetivo situado en la izquierda

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	13,40	9,34	-2,21	11,18	7,12	45,00
3	9,31	11,72	6,28	15,59	18,01	26,57
4,5	0,94	9,52	12,32	13,26	21,84	18,43

Tabla 3: *Medidas obtenidas para el caso 3.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 3 y el punto objetivo está situado a la izquierda.

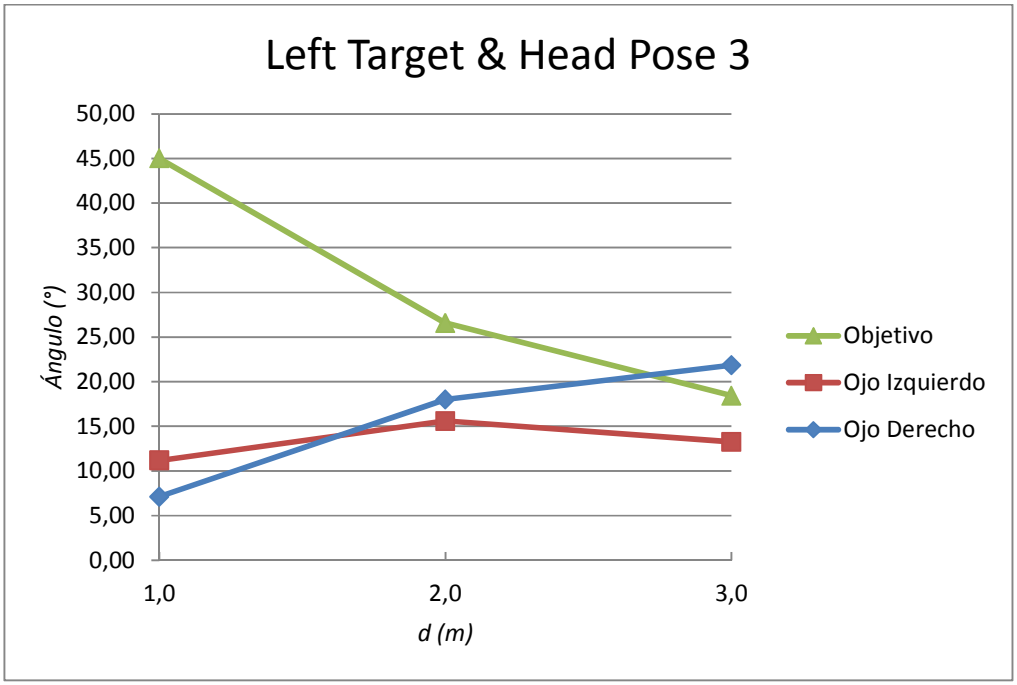

Figura 3: *Representación gráfica resultados caso 3.* Se observa la diferencia entre los valores medidos con la cabeza mirando a la derecha con el objetivo situado a la izquierda, y los valores objetivo. A mayores distancias, también decrece la diferencia entre las medidas y el objetivo.

- Caso 4: Pose de la cabeza 1 y mirada dirigida al punto objetivo situado en el centro

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	-4,73	-3,88	6,44	1,72	2,56	0,00
3	4,85	-1,26	-3,32	1,53	-4,58	0,00
4,5	7,33	-4,99	-10,74	-3,41	-15,73	0,00

Tabla 4*: Medidas obtenidas para el caso 4.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 1 y el punto objetivo está situado en el centro.

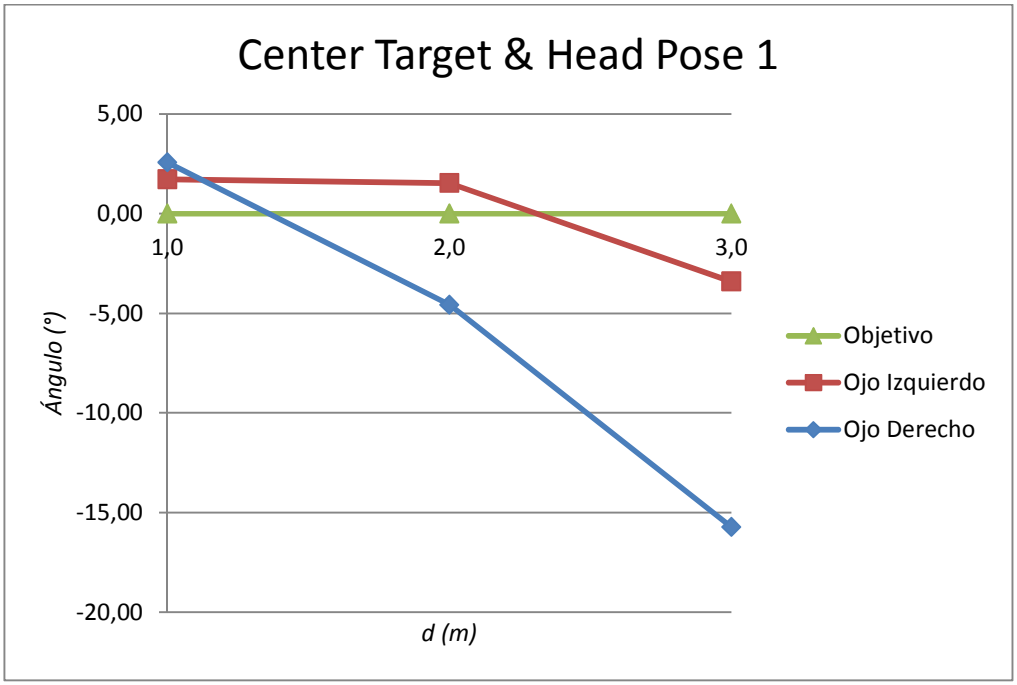

Figura 4: *Representación gráfica resultados caso 4.* Se observa la diferencia entre los valores medidos con la cabeza mirando a la izquierda con el objetivo situado en el centro, y los valores objetivo. En este caso se observan menores diferencias entre las medidas y el objetivo.

- Caso 5: Pose de la cabeza 2 y mirada dirigida al punto objetivo situado en el centro

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	-3,34	-0,89	6,31	2,96	5,42	0,00
3	4,46	-1,85	-4,00	0,47	-5,85	0,00
4,5	1,89	-6,45	4,68	6,57	-1,77	0,00

Tabla 5: *Medidas obtenidas para el caso 5.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 2 y el punto objetivo está situado en el centro.

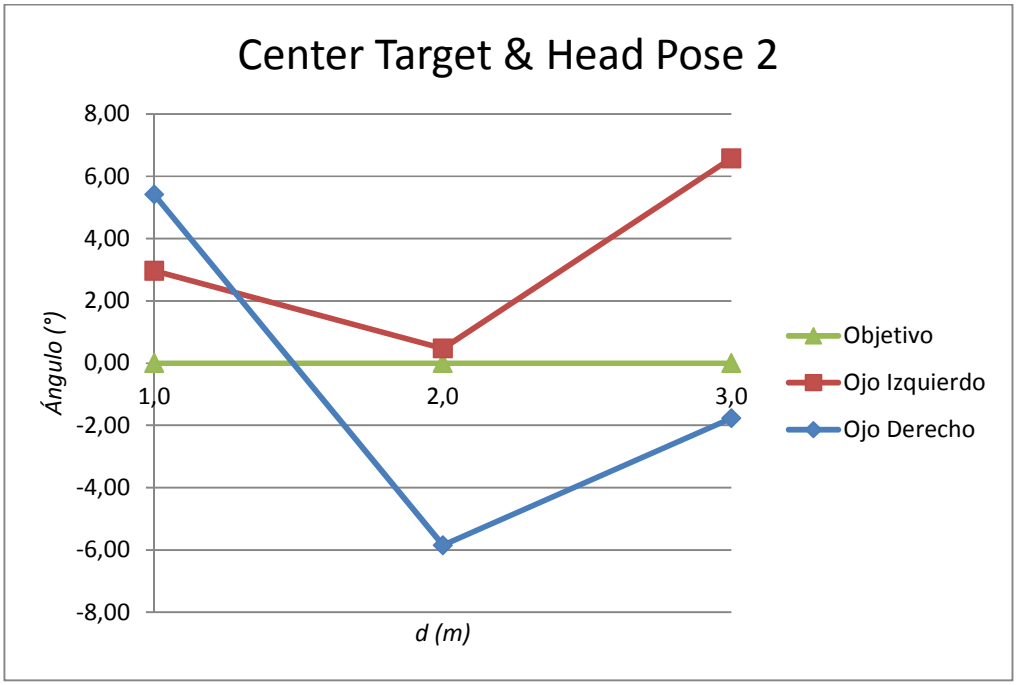

Figura 5: *Representación gráfica resultados caso 5.* Se observa la diferencia entre los valores medidos con la cabeza mirando al centro con el objetivo situado en el centro, y los valores objetivo. Como en el caso anterior se observan menores diferencias entre las medidas y el objetivo.

- Caso 6: Pose de la cabeza 3 y mirada dirigida al punto objetivo situado en el centro

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	2,42	-0,59	-5,52	-3,10	-6,11	0,00
3	0,89	-0,90	4,90	5,80	4,00	0,00
4,5	-4,74	1,26	11,71	6,97	12,97	0,00

Tabla 6: *Medidas obtenidas para el caso 6.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 3 y el punto objetivo está situado en el centro.

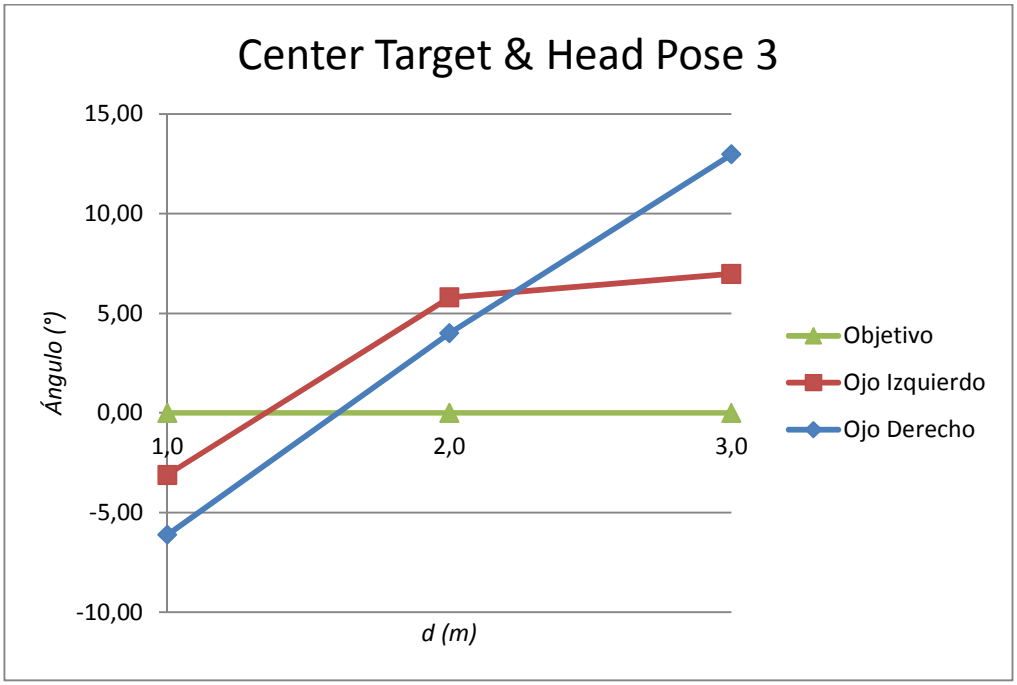

Figura 6: *Representación gráfica resultados caso 6.* Se observa la diferencia entre los valores medidos con la cabeza mirando a la derecha con el objetivo situado en el centro, y los valores objetivo. Una vez más se observan menores diferencias entre las medidas y el objetivo.

- Caso 7: Pose de la cabeza 1 y mirada dirigida al punto objetivo situado la derecha

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	-14,97	-12,46	1,24	-13,73	-11,22	-45,00
3	-3,82	-10,45	-3,86	-7,68	-14,31	-26,57
4,5	0,94	-10,05	-11,61	-10,67	-21,66	-18,43

Tabla 7*: Medidas obtenidas para el caso 7.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 1 y el punto objetivo está situado a la derecha.

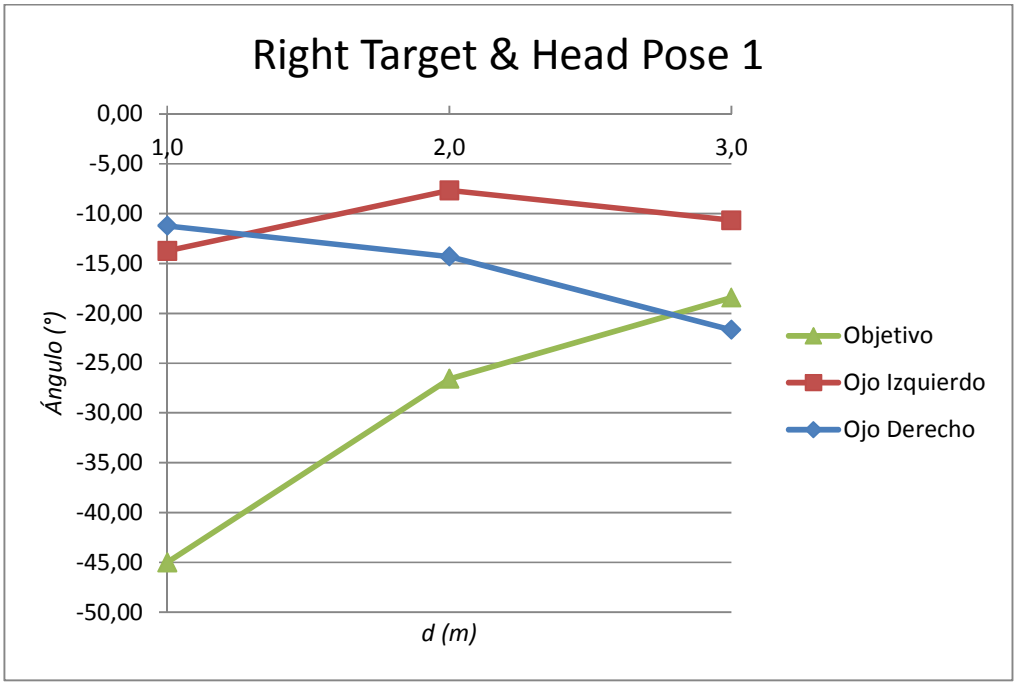

Figura 7: *Representación gráfica resultados caso 4.* Se observa la diferencia entre los valores medidos con la cabeza mirando a la izquierda con el objetivo situado en el centro, y los valores objetivo. En este caso se observan menores diferencias entre las medidas y el objetivo.

- Caso 8: Pose de la cabeza 2 y mirada dirigida al punto objetivo situado la derecha

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	-8,76	-8,99	3,40	-5,36	-5,59	-45,00
3	-1,06	-8,39	-8,96	-10,02	-17,35	-26,57
4,5	-4,12	-5,77	-2,18	-6,30	-7,94	-18,43

Tabla 8: *Medidas obtenidas para el caso 8.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 2 y el punto objetivo está situado a la derecha.

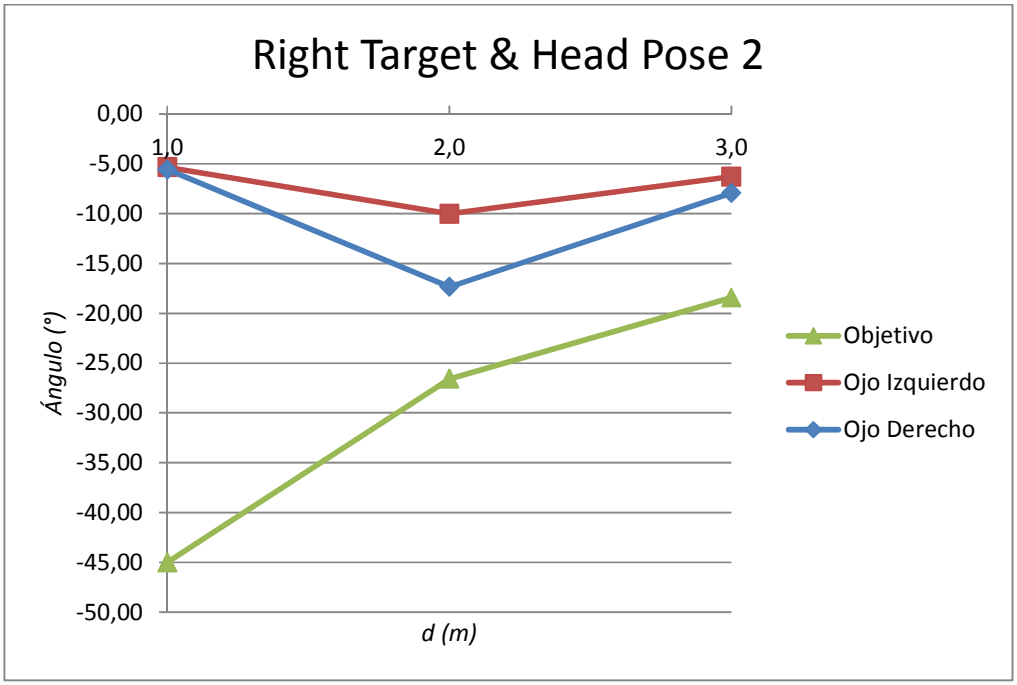

Figura 8: *Representación gráfica resultados caso 8.* Se observa la diferencia entre los valores medidos con la cabeza mirando al centro con el objetivo situado a la derecha, y los valores objetivo. Se aprecia la misma tendencia de disminución de las diferencias al aumentar la distancia.

- Caso 9: Pose de la cabeza 3 y mirada dirigida al punto objetivo situado la derecha

	Ángulo (°)			Ángulo mirada (°)		Punto
d(m)	Leye	Reye	Yaw	Leye	Reye	Objetivo (°)
1,5	-9,18	-10,48	-6,09	-15,27	-16,57	-45,00
3	-13,05	-8,65	6,22	-6,82	-2,43	-26,57
4,5	-7,57	-6,03	14,07	6,50	8,04	-18,43

Tabla 9: *Medidas obtenidas para el caso 9.* Medidas de los ángulos obtenidos (Ángulo y Ángulo mirada) y ángulos objetivo (Punto Objetivo) para el caso en el que la cabeza se encuentra en posición 3 y el punto objetivo está situado a la derecha.

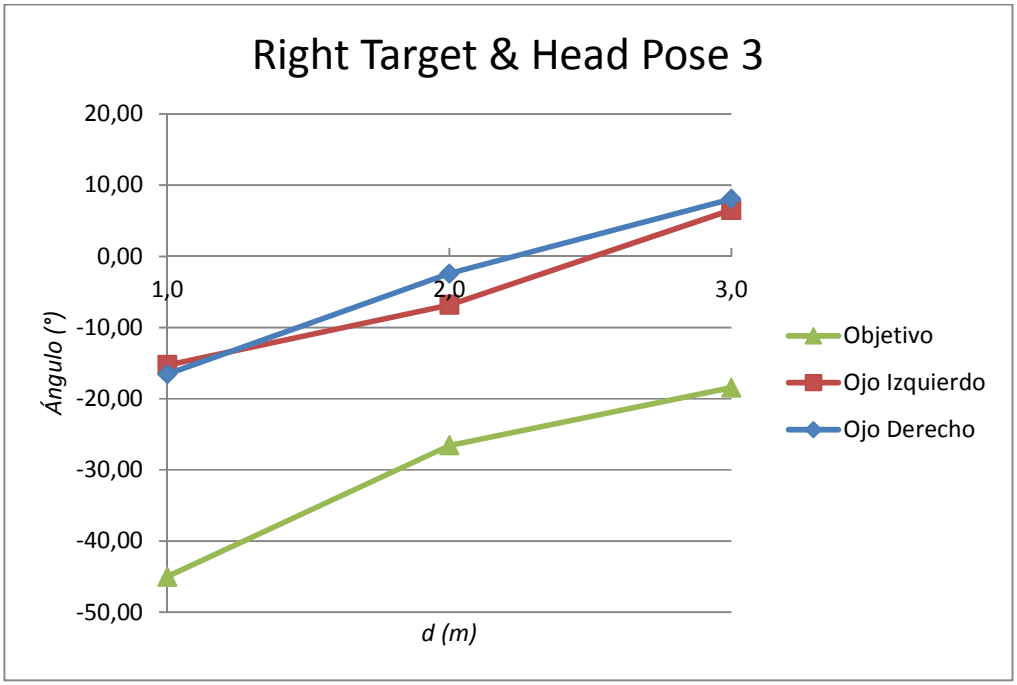

Figura 9: *Representación gráfica resultados caso 9.* Se observa la diferencia entre los valores medidos con la cabeza mirando a la derecha con el objetivo situado a la derecha, y los valores objetivo.

5.3 Discusión de resultados

De los resultados anteriores podemos extraer varias conclusiones. La principal conclusión que se extrae es que con nuestro algoritmo de seguimiento de la mirada no tiene una gran precisión ni exactitud a la hora de detectar la mirada, pero sí se puede distinguir a qué punto se está dirigiendo la mirada de la persona. El sistema presenta grandes diferencias entre las medidas realizas y sus valores teóricos, incluso cuando la distancia al objetivo es menor ($d = 1{,}5\ m$) Ahora bien, a través de las medidas angulares tomadas para cada uno de los puntos objetivos, para una distancia dada, sí que observamos la diferencia que expresábamos entre los distintos puntos. Podemos explicar esta falta de exactitud debido a varias causas:

- En primer lugar el error puede deberse en parte a un mal ajuste por parte del programa a la hora de analizar tanto la posición de las pupilas, cómo la pose de la cara. Dado que se han tomado más de 30 valores para cada una de las medidas parte de este error debería estar compensado por la media de estos valores.

- En segundo lugar hay que destacar que el objetivo al que se esté mirando es también muy importante. Cuando el usuario dirige su mirada al punto objetivo situado en el centro (aquel en el que tanto su cara cómo sus ojos tienen un menor ángulo) el sistema mejora su exactitud considerablemente. Esto se puede explicar de manera sencilla, ya que tanto el cálculo de la pose de la cara, como el cálculo del centro de nuestro ojo, se realizan mejor cuando los ángulos no son muy pronunciados.

- Tanto la iluminación como la elección del entorno donde se va a realizar el seguimiento de la mirada son factores a tener muy en cuenta. Una mala iluminación de la cara, como por ejemplo, que la zona derecha de la cara esté más iluminada que la izquierda, o que la zona inferior de la cara esté más iluminada que la superior, puede producir un mal ajuste del contorno de la cara debido a la diferencia de texturas. Además si se ilumina con una luz demasiado directa el reflejo sobre los ojos también puede provocar una mala detección del iris de los mismos.

- Por último, un aspecto teórico al que no hemos dado gran relevancia y podría tenerla. En la elección de nuestro modelo vimos cómo cuanto más simple sea el modelo mejor es para nuestras posibilidades de realizar un seguimiento eficiente. Pero veíamos también como un modelo excesivamente simple puede que no refleje completamente la realidad. Durante todo el proyecto se han ido realizando una serie de aproximaciones para simplificar el modelo, y algunas de ellas tienen un gran peso sobre el resultado final. Como ejemplo podemos citar la diferencia que existe entre considerar la región del ojo como un plano, en lugar de tratarlo cómo parte de una circunferencia, tal y cómo se puede ver en la imagen (). Eso hubiese complicado el modelo, pero a su vez nos hubiese permitido reflejar los ángulos de la mirada con una mayor exactitud.

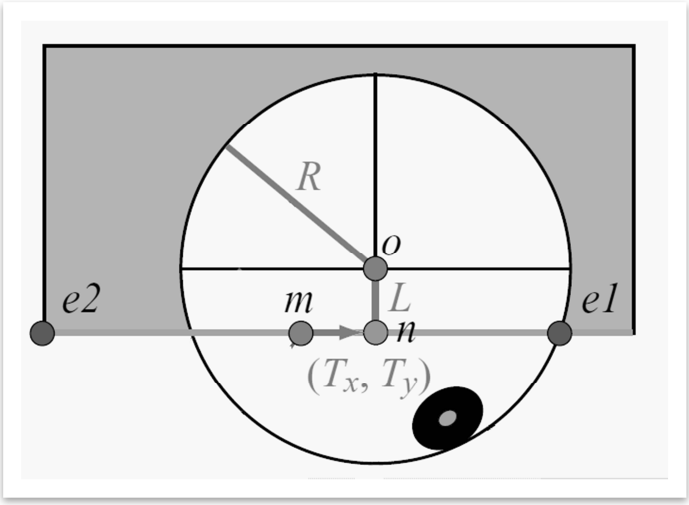

Imagen 16

6 Conclusiones y trabajos futuros

6.1 Conclusiones

Del conjunto del proyecto podemos extraer diversas conclusiones. En primer lugar es importante destacar que ha sido posible realizar tanto el seguimiento del rostro, como el de los ojos, de forma satisfactoria para un conjunto de imágenes tomadas en diferentes posiciones y condiciones de iluminación. Por tanto, en lo que respecta al primero de los objetivos se considera satisfecho, puesto que en las imágenes se puede apreciar como la detección del iris es correcta en una amplia mayoría de las imágenes.

Por otro lado en el capítulo anterior hemos podido observar que los problemas de exactitud en lo que respecta al cálculo de la dirección de la mirada son muy importantes. Por eso consideramos que el segundo de los objetivos de este proyecto no se ha podido cumplir. Consideramos que los errores obtenidos se deben casi exclusivamente al modelado realizado para conseguir transformar la información 2D de la imagen en información acerca de la dirección en la mirada. Hemos concluido eso porque si bien los ángulos que hemos obtenido son incorrectos, sí que podemos diferenciar unos puntos de otros, por tanto consideramos que el primero de los objetivos se ha cumplido, mientras que el segundo no.

En cuanto al último de los objetivos, que básicamente trataba de ofrecer una guía para futuros proyectos que estén interesados en trabajar con sistemas de seguimiento de rostro, consideramos que gracias a esta memoria, las personas interesadas podrán encontrar documentación suficiente para tener una idea claro del funcionamiento y posibilidades que ofrecen estos proyectos.

Por último queremos realizar un análisis acerca de viabilidad para que futuros proyectos que utilicen esta tipo de tecnología, sean capaces de realizar un seguimiento en tiempo real de la mirada de una persona. En ese sentido podemos decir que con el código utilizado, el tratamiento de cada una de las imágenes lleva demasiado tiempo cómo para poder realizar un seguimiento en tiempo real en un ordenador convencional de 2 núcleos. Sin embargo sí que se pueden introducir mejoras que harían acelerar el procesado de las imágenes, como el procesamiento en paralelo, y de esta forma conseguir una tasa de tratamiento de imágenes en torno a 20 *frames/seg* que sería lo necesario para poder realizar un seguimiento en tiempo real.

6.2 Trabajos futuros

El hecho de que los resultados obtenidos no hayan sido los esperados en cuanto a lo que exactitud y tiempo de cálculo se refiere, nos da pie a introducir este capítulo en el que vamos a desarrollar las posibles mejoras que podríamos introducir en este sentido, así como algunas que intentan avanzar en el desarrollo del mismo.

La primera de las mejoras tiene relación con los problemas sufridos en cuanto a lo que exactitud se refiere. Si el modelo de la cámara que hemos obtenido es correcto, y nuestro seguimiento de la

pupila también, debemos considerar que el modelo elegido para representar los ojos de la persona no es correcto. Por tanto, se debe trabajar en construir un modelo más sólido de los ojos de la persona, empezando por considerar curva la esfera del mismo, y evaluando los resultados para comprobar la mejoría.

Además Durante el cálculo de la posición del iris, decidimos aprovechar el hecho de que el tamaño de este iba a ser prácticamente constante. De esta forma una vez hubiésemos detectado correctamente una serie de radios, podríamos acotar la búsqueda del radio óptimo con los radios detectados previamente. Al implementar este sistema no hemos conseguido una mejoría durante la detección del iris. Sin embargo otra mejora que se podría implementar es el uso de técnicas estadísticas, para mejorar el proceso anterior, y conseguir simultáneamente una mejora en la exactitud del sistema y en el tiempo de cómputo de este.

Otro punto interesante es el del tiempo de cómputo de cada una de las imágenes. Para mejorarlo es interesante la aplicación del procesado en paralelo durante el cálculo de las imágenes respuesta en el ajuste de las posiciones de los puntos de interés, y también durante el proceso de optimización de dichas posiciones.

Otro de los objetivos de este proyecto era el de conseguir un aprendizaje acerca de los sistemas de seguimiento de rostro. En ese sentido otro de los pasos que se podría dar para mejorar la información de este proyecto, sería ocuparse de la creación del código de uno de estos sistemas. Ya sea creando parte del mismo, como puede ser realizar el entrenamiento de imágenes, o bien ocuparse de la creación completa (entrenamiento, búsqueda y detección final) de un sistemas de detección de objetos más sencillos que una cara, como puede ser el de una mano humana.

Por último consideramos que los sistemas de seguimiento del rostro humano han sido estudiados en profundidad, aunque solo desde un punto de vista en concreto, utilizando modelos de puntos dados por imágenes entrenadas, lo que se conoce como *Point Distribution Model* (PDM). Aunque estos sistemas ofrecen una serie de ventajas importantes, otro enfoque es posible. Si en lugar de utilizar un modelo creado a partir de imágenes, tratamos de crear un modelo virtual que se ajuste a las imágenes, podríamos incluir en estos modelos una serie de parámetros diferentes, que tengan una mayor información en cuanto a lo que a la forma del objeto se refiere. Si, por ejemplo, tratamos de modelar los movimientos de una mano, utilizando un sistema virtual, compuesto por tendones, huesos y músculos, desde luego la información que contienen estos en lo que a variaciones de forma se refiere es mucho mayor, aunque tendría como inconveniente la incapacidad para tratar tanta información. Por tanto consideramos que sería interesante el estudio de estos enfoques en casos simples, para constatar si es viable o no su desarrollo.

7 Presupuesto

Descripción	Cantidad	Coste unitario	Coste total
Horas Ingeniero Industrial	1920	7,00 €	13.440,00 €
Licencia software CLM	1	2.000,00 €	2.000,00 €
Licencia Visual C++	1	2.500,00 €	2.500,00 €
Ordenador con cámara frontal	1	3.000,00 €	3.000,00 €
Total			20.940,00 €

8 Referencias

[1] *B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 674–679, 1981.*

[2] *S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, volume 1*,pages 1090–1097, 2001.

[3] *S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. International Journal of Computer Vision, 56(3):221–255, Feb. 2004. Previously appeared as CMU Robotics Institute Technical Report* CMU-RI-TR-02-16.

[4] *G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. PAMI, 20:1025–1039, 1998.*

[5] *T. Cootes, G. Edwards, and C. Taylor. Active appearance models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(6):681–685, June 2001*.

[6] *J. Xiao, S. Baker, I. Matthews, and T. Kanade. Real-time combined 2D+3D active appearance models. In Submitted to the IEEE Conference on Computer Vision and Pattern Recognition, 2004.*

[7] J. Xiao, J. Chai, and T. Kanade. A closed-form solution to non-rigid shape and motion recovery. In *Proceedings of the European Conference on Computer Vision*, 2004.

[8] *Takahiro Ishikawa, Simon Baker, Iain Matthews, and Takeo Kanade, "Passive driver gaze tracking with active appearance models," Tech. Rep. CMU-RI-TR-04-08, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2004*.