UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Carrera

"Interfaz hombre-máquina basado en el análisis de los gestos faciales mediante visión artificial"

Nombre del alumno: D. Jose Francisco Velasco Cerpa

Año: 2010

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Carrera

"Interfaz hombre-máquina basado en el análisis de los gestos faciales mediante visión artificial"

Alumno: D. Jose Francisco Velasco Cerpa

Director: D. Daniel Pizarro Pérez

Tribunal:

Presidente:	D. Luis Miguel Bergasa Pascual
Vocal 1º:	D. Eduardo Sebastián Martinez
Vocal 2º:	D. Daniel Pizarro Pérez

Calificación: .....

Fecha: .....

“Cada fracaso le enseña al hombre
algo que necesitaba aprender”

-Charles Dickens-

A mi abuela Ana

Agradecimientos

En primer lugar, quiero agradecerle toda su ayuda, apoyo, dedicación y paciencia a Daniel Pizarro, tutor de este proyecto. Así mismo, quiero agradecer a mis compañeros y al resto de profesores del "espacio inteligente" toda su ayuda a lo largo de estos meses y el haberme acogido entre ellos, como uno más, desde el primer momento.

Gracias a mis amigos y compañeros de la universidad, por haber compartido conmigo todos esos buenos y malos momentos que nos han hecho crecer en todos los sentidos. Sin vosotros hubiera sido todo mucho más difícil.

Le agradezco también a mis amigos, con los que he crecido desde la niñez, su cariño y el ejemplo y la influencia que cada uno de ellos han tenido en mi.

Gracias Eva, por haber sido mi compañera y amiga durante mas de ocho años. Tu amor y tu forma de ser me han hecho ganar humildad para reconocer mis errores y aprender de ellos, y seguridad en mi mismo. Siempre he considerado que hay algo especial en ti.

Por último, deseo dar las gracias a toda mi familia. Gracias a mis padres que por su tesón y esfuerzo en hacerme crecer y desarrollarme como una persona independiente, con criterio, pensamientos e ideas propios. Yo no lo hubiera hecho mejor.

En especial, quiero agradecer a mi abuela Ana, a quien va dedicado este proyecto, el haber sido para mí como una segunda madre. Por toda su ternura, afecto y cariño mostrado durante todos estos años, te mereces que este proyecto vaya dirigido a ti.

Índice de contenido

Índice de ilustraciones

I. Resumen

1.- Resumen

El presente proyecto muestra como, a partir de un Modelo de Apariencia Activa (AAM), es posible crear una aplicación que es capaz de reconocer los gestos faciales de un usuario y actuar en concordancia. El objetivo final es conocer de que herramientas se dispone actualmente que permitan mejorar la capacidad de comunicación de un usuario con una máquina, primordialmente centrando la atención en la ayuda a la discapacidad.

Los Modelos de Apariencia Activa caracterizan el rostro humano mediante una malla formada por triángulos, cuyos vértices coinciden con puntos característicos de la cara, y mediante la apariencia (conjunto de píxeles y sus intensidades) encerrada en cada uno de esos triángulos. En un AAM, las variaciones de forma y apariencia están definidas de forma lineal. La forma define una función de transformación o Warp desde una imagen a otra mediante un conjunto de transformaciones geométricas afines definidas para cada triángulo. En conjunto y debido a la discretización y cuantificación de la imagen, la función de Warp es una transformación no lineal.

En este proyecto se presta especial atención a los algoritmos de ajuste de un AAM para representar una imagen de entrada, permitiendo así el reconocimiento de getos. Debido a las necesidades de tiempo real requeridas por tratarse de una aplicación interactiva, es necesario que dicho algoritmo sea de baja complejidad computacional, lo que permitirá obtener tiempos breves de respuesta sin que esto afecte, en la medida de lo posible, a la exactitud del resultado. Son muchos los algoritmos de ajustes propuestos en la literatura. El presente proyecto se centra en algoritmos de ajuste basados en la minimización de una función de coste, en la que se buscan los parámetros que definen un AAM que minimizan dicho coste. En el proyecto se detallan los siguientes:

- Lucas Kanade: Es el algoritmo originalmente propuesto para el ajuste de un AAM y se basa en el método iterativo de Gauss-Newton para la optimización. Es un algoritmo preciso pero complejo computacionalmente.
- Composicional Inverso: Algoritmo más óptimo computacionalmente que el anterior. Se basa en considerar una actualización iterativa de los parámetros que definen el AAM diferente. Mediante una aproximación denominada como “Project out” se puede conseguir un método que, aunque de manera aproximada, puede ser implementado en tiempo real.

Los distintos algoritmos considerados en el proyecto han sido evaluados mediante pruebas experimentales.

A parte del ajuste del AAM, en este proyecto, también se aporta una solución al problema de inicialización, ya que los algoritmos mencionados necesitan partir de una posición de inicio cercana a la solución final. Para ello se propone el uso del popular algoritmo de detección de caras de Viola & Jones.

Se ha realizado una implementación completa del algoritmo haciendo uso de las librería OpenCV, resolviendo aspectos prácticos ajenos al desarrollo teórico como la realización de la función de 'warp' o las estructuras de datos necesarias para llevar a cabo la comunicación entre las diferentes partes del programa.

Por último se propone una prueba de concepto, en la que, mediante un AAM, se puede controlar un cursor de ratón y hacer click mediante gestos faciales.

II. Memoria

1.- Introducción

En la actualidad, el uso de herramientas, tales como los ordenadores, en las que son necesarios comandos complejos para su uso y en las que se desarrollan entornos gráficos cada vez mas interactivos e intuitivos, se ha convertido en habitual en nuestra sociedad.

Para la mayoría de las personas resulta sencillo manejar este tipo de herramientas mediante diversos dispositivos hardware estandarizados como pueden ser ratones, touchpad, pantallas táctiles, etc. Sin embargo, cabe destacar que existen personas que, por diversas razones, no son capaces de manejar este tipo de dispositivos teniendo, por consiguiente, serios problemas a la hora de manejar un ordenador.

Como alternativa, en el presente proyecto final de carrera se propone el uso de la visión artificial para la detección de determinados gestos faciales que, mediante su posterior procesado, permitan al usuario realizar comandos ya habituales como hacer "click" o mover un puntero, así como comandos más complejos y específicos de las necesidades del usuario.

El uso de la visión artificial tiene grandes ventajas como es el uso de dispositivos de entrada estándar como lo son las cámaras web USB o las cámaras FireWire (IEEE 1394), caracterizadas por la disponibilidad de controladores en diversos sistemas operativos, además de por haber experimentado una importante reducción de su precio en los últimos años, principalmente, debido a la popularización de su uso en diversos dispositivos electrónicos de consumo masivo. Otra de las ventajas que presenta la visión por computador es que resulta un **método no intrusivo**, es decir, no requiere de la colocación de diodos u otro tipo de dispositivos en el cuerpo del usuario, lo que afectaría negativamente a la usabilidad.

Se ha elegido el uso de la cara debido a que, hoy en día, es una de las partes del cuerpo menos utilizadas en HMI. Además de tener un amplio abanico de movimientos y posturas posibles (lo cual puede facilitar la comunicación en el humano y la máquina), tiene la virtud de conservar la movilidad en ciertas circunstancias en las que se pierde esa capacidad en las manos, como por ejemplo en tetraplejias o amputaciones.

La técnica en la que se basa este proyecto utiliza los denominados Modelos de Apariencia Activa (Active Appearance Models, en inglés). Propuesta originalmente por [Cootes et al, 1998], esta técnica ha sido objeto de diversas y profundas revisiones en los últimos años, entre ellas destaca [Baker et al,2002].

Estado del Arte.

Las interfaces de usuario han experimentado un gran desarrollo debido a su continua evolución desde interfaces en formato texto basadas en teclado a interfaces gráficas basadas en el uso del ratón.

El uso de movimiento humanos, especialmente realizados con las manos, es lo que más se ha popularizado desde los primeros ordenadores. Por ejemplo con el uso de teclados, ratones, joysticks, etc.

Sin embargo, en los últimos tiempos ha crecido el interés en la denominada Interacción inteligente Humano máquina (**HMII**). Lo que ha propiciado numerosos estudios en diversos aspectos tales como el modelado y análisis del movimiento, reconocimiento de patrones, 'machine learning' e incluso estudios psicolingüísticos.

En este sentido, el uso de periféricos de entrada basados en la captura de imágenes tales como los escáneres o las cámaras web vienen siendo ampliamente empleados desde hace mucho tiempo. Sin embargo, no ha sido hasta los últimos años cuando este tipo de dispositivos (cámaras) han sido utilizados de forma interactiva para el control de la máquina.

Por ejemplo, en el campo de los videojuegos existe un elevado número de aplicaciones en donde se utilizan cámaras como medio para controlar el juego. El EyeToy para la PlayStation2 de Sony o Kinect (proyecto Natal) par Xbox360 de Microsoft son algunos ejemplos.

La razón fundamental para el reciente auge de las cámaras como dispositivos de entrada y de control son los siguientes:

- Reducción de los costes de los sensores CMOS y CCD, así como su mejor integración. Esto ha hecho posible que cada vez más dispositivos como pantallas o ordenadores portátiles incorporen pequeñas cámaras de mediana calidad.
- Aumento de la potencia y reducción de los precios de CPU y DSP, que permiten un elevado número de operaciones por segundo, lo que posibilita llevar acabo satisfactoriamente algoritmos complejos que anteriormente no resultaban eficientes.
- Nuevos avances en visión artificial. Como, por ejemplo, los modelos de Apariencia Activa, en los que esta basado este proyecto, que fueron propuestos por primera vez en 1998.

Las interfaces basadas en visión artificial ofrecen como principal ventaja, con respecto a otro tipo de tecnologías, el ser una alternativa no intrusiva y de coste reducido, capaces de utilizar información generada naturalmente por el usuario, lo cual los convierte en sistemas muy versátiles.

En el presente proyecto se plantea el uso de los denominados Modelos de Apariencia Activa (Active Appearance Models AAM en inglés) para el análisis de la imagen, ya que éstos proporcionan un método muy preciso que permite reconocer gestos faciales de un usuario.

Modelos de apariencia Activa

Los modelos de apariencia activa (AAM) permiten reproducir de manera sintética imágenes de superficies que incluyen deformaciones no rígidas y cambios de apariencia. Se basan en la obtención de un modelo estadístico, en una fase de entrenamiento, de la forma (determinada por un conjunto de puntos característicos) y la apariencia (mapa de bits) del objeto de interés [Cootes et al, 1998]. En el presente proyecto, al tratar con caras, la forma se define por los puntos que forman una malla parecida a la mostrada en la Ilustración 1, la cual varía con los diferentes gestos del usuario. Así mismo, la apariencia esta definida por el mapa de bits que se encuentran en el interior de los triángulos que forman los puntos de la malla.

Ilustración 1: Ejemplo de malla en un AAM

En consecuencia, un AAM queda definido por dos conjuntos de parámetros: parámetros de forma y parámetros de apariencia. Durante la fase de entrenamiento se aprenden las relaciones entre los parámetros y los cambios que estos producen en el modelo. El proceso de ajuste de un AAM trata de aprovechar esta información para ajustar el modelo a una imagen de entrada reduciendo el error entre ambas.

Los modelos de apariencia activa (AAM's) están relacionados estrechamente con conceptos tales como Active Blobs [Sclaroff & Isidoro, 1998][Sclaroff & Isidoro, 2003] y Morphable models [Blanz & Vetter, 1999][Jones & Poggio, 1998], que son modelos paramétricos, no lineales y generativos empleados para ciertos fenómenos visuales. La aplicación mas frecuente de los modelos de apariencia activa hasta la fecha es la de modelado de caras. Sin embargo, los AAM's también pueden ser útiles en otras aplicaciones [Jones & Poggio, 1998][Sclaroff & Isidoro, 2003].

En una aplicación típica, el primer paso es ajustar el AAM al la imagen de entrada, por ejemplo eligiendo aquellos parámetros del modelo que maximicen el ajuste entre la imagen sintética generada por el AAM y la imagen de entrada. A partir de ahí, los parámetros del modelo son utilizados para cualquiera que sea el fin de la aplicación. Por ejemplo, tal y como se propone en el presente proyecto, los parámetros puede analizarse mediante un clasificador para reconocer gestos faciales.

Sin embargo, el uso de clasificadores en este tipo de modelos, permiten la posibilidad de extraer mucha más información que la necesaria en este proyecto, por ejemplo en [Lanitis et al, 1997] el mismo modelo era empleado para el reconocimiento facial, la estimación de la pose y para reconocer el gesto.

El ajuste o 'fitting' de un modelo de apariencia activa a una imagen es un problema no lineal. El enfoque habitual propuesto por [Cootes et al, 1998] consiste en la actualización aditiva e iterativa de los parámetros del modelo.

Dada la estimación actual de los parámetros de forma, es posible realizar la transformación de la imagen de entrada desde su forma original a una forma común donde es posible calcular una imagen de error entre la imagen generada por el modelo (usando los parámetros calculados en ese momento) y la imagen a la que el modelo trata de ajustarse. Se define esa transformación como warp de la imagen, ya que es esta la nomenclatura utilizada en la litaratura. La función warp definida en este proyecto es una transformación afín definida a trozos.

En los primeros algoritmos [Cootes et al, 1998], simplemente se asumía la existencia de una relación lineal y constante entre la imagen de error mencionada y la actualización aditiva de los parámetros. Los coeficientes de la relación lineal y constante eran calculados o bien por regresión lineal o bien por métodos numéricos.

Desafortunadamente, Asumir que existe una relación tan simple entre el error y la actualización apropiada del modelo es en general incorrecta. El resultado de realizar esta presunción se traduce en unos resultados del algoritmo de 'fitting' pobres, tanto en número de iteraciones requeridas para converger, como en la precisión del ajuste.

Con el objetivo de mejorar el rendimiento, en [Baker et al,2002] se propone utilizar el algoritmo de Lucas-Kanade, que utiliza la aproximación de Gauss-Newton para la minimización del error cuadrático entre la imagen de entrada y la imagen generada por el modelo AAM. La actualización de los parámetros en este algoritmo es incremental y aditiva. Esta característica, que proporciona simplicidad y exactitud, hace que sea necesario recalcular el Hessiano en cada iteración, lo que convierte a este algoritmo en ineficiente para una aplicación en tiempo real.

El algoritmo composicional [Backer & Matthews, 2001][Baker et al,2002] se contrapone al enfoque de los algoritmos utilizados normalmente para el proceso de alineación de imágenes explicados anteriormente, ya que utiliza la regla de la composición, en lugar de la aditiva, para la actualización de los parámetros. Dependiendo de si la actualización mediante composición se realiza sobre la instanciación del modelo o sobre el warp de la imágen se obtiene o bien composicional inverso aditivo, o bien el composicional inverso. Este algoritmo consigue que el Hessiano sea contante, lo que ahorra mucho tiempo en cada iteración (al poder precalcularlo).

Una diferencia del composicional inverso es que únicamente puede ser aplicado en funciones de warp que formen *semigrupos,* ya que el resultado de la composición del "warp" con otro "warp" incremental debe seguir siendo descrita por los mismos parámetros. En el segundo caso (que es referido posteriormente en este proyecto como composiciconal inverso). Es necesario que el warp utilizado forme un *grupo* ya que debe existir el inverso del warp.

Aunque casi la mayoría de warps utilizados en visión artificial forman grupo, hay un conjunto importante de ellos que no lo son, tales como los usados en los *Modelos Flexibles de Apariencia* (FAMs), en *Active Blob*s, o los inicialmente usado en [Cootes et al, 1998] para AAMs.

Objetivos.

El objetivo principal perseguido en la realización de este proyecto, es la aplicación de técnicas de registro de imágenes basadas en Modelos de Apariencia Activa, para el desarrollo e implementación de una interfaz hombre-máquina que posibilite interpretar los comandos que el usuario ejecute con gestos faciales. En la Ilustración 2, se muestra una visión de conjunto de la aplicación que se pretende desarrollar

El proyecto tiene un enfoque teórico-práctico, de modo que sea posible estudiar el comportamiento de los AAMs y evaluar su aplicabilidad a la detección de gestos faciales en un caso real, donde se pretende desarrollar, mediante visión por computador, una forma de comunicación entre una persona y una máquina utilizando gestos faciales.

A partir del objetivo principal mencionado, en el que se basa el proyecto, surgen una serie de objetivos secundarios que se especifican a continuación:

1. Modelado paramétrico del rostro de una persona. Se considera necesario encontrar alguna forma de 'traducir' la expresión de una persona a un conjunto limitado de parámetros. Las características deseadas para esta parametrización son las siguientes:
   a) El número de parámetros debe ser lo más pequeño posible. Sólo de esta forma se conseguirá reducir la dimensión del problema, haciéndolo, solo así, viable computacionalmente.
   b) Es deseable que la parametrización contenga la mayor información posible. Es decir, que pueda ser representado de forma inequívoca, por medio de los parámetros disponibles, la mayor cantidad de muecas y gestos realizables por una persona. Además se valorará que puedan aportar información añadida como pueda ser la identificación de la persona (Aunque esto último queda fuera del objetivo principal de este proyecto)
   c) Indepencia entre el gesto y la posición, el tamaño o la rotación de la cara. Esto facilitará en gran medida la distinción de gestos a partir de cada parámetro.
2. Implementación de un algoritmo de ajuste del modelo elegido. Será necesario encontrar una algoritmo que, a partir de una imagen de entrada, sea capaz de ajustar los parámetros del modelo para que “encaje”, lo mejor posible, con la imagen de entrada. En este objetivo destaca la necesidad de que el algoritmo usado sea eficiente computacionalmente, de forma que permita ser utilizado en una aplicación en tiempo real. Estos son los aspectos importantes a la hora de elegir un algoritmo:
   a) Tiempo de convergencia reducido. Tanto el número de iteraciones como el tiempo necesario en ejecutar cada iteración debe de ser pequeño y acotado si se desea ajustarse a la **limitación de tiempo real** que este tipo de aplicación, normalmente interactiva, requiere.

b) Precisión elevada. Evaluada tanto en términos de mínimo error del ajuste, como en numero de imágenes en las que el algoritmo converge convenientemente.

3. Inicialización automática del algoritmo de ajuste. La mayoría de los algoritmos disponibles necesitan ser inicializados 'cerca' de la solución deseada. Es por ello que se tratará de encontrar alguna solución a esta limitación.
4. Creación de un método de clasificación de gestos.
5. Implementación práctica de cada uno de los aspectos necesarios para la realización del HMI mediante visión artificial. Se considera fundamental el hecho de ser capaces de plasmar en un programa funcional todos los aspectos teóricos utilizados en este desarrollo. De esta forma se podrá cuantificar de una forma fidedigna las ventajas o inconvenientes de cada una de las posibilidades.

   Adicionalmente, en este punto se tendrán en cuenta aspectos como la usabilidad del programa. Tratando proponer un lenguaje sencillo que permita al usuario dar órdenes básicas y que, a la vez, permita una interpretación precisa por parte del programa.

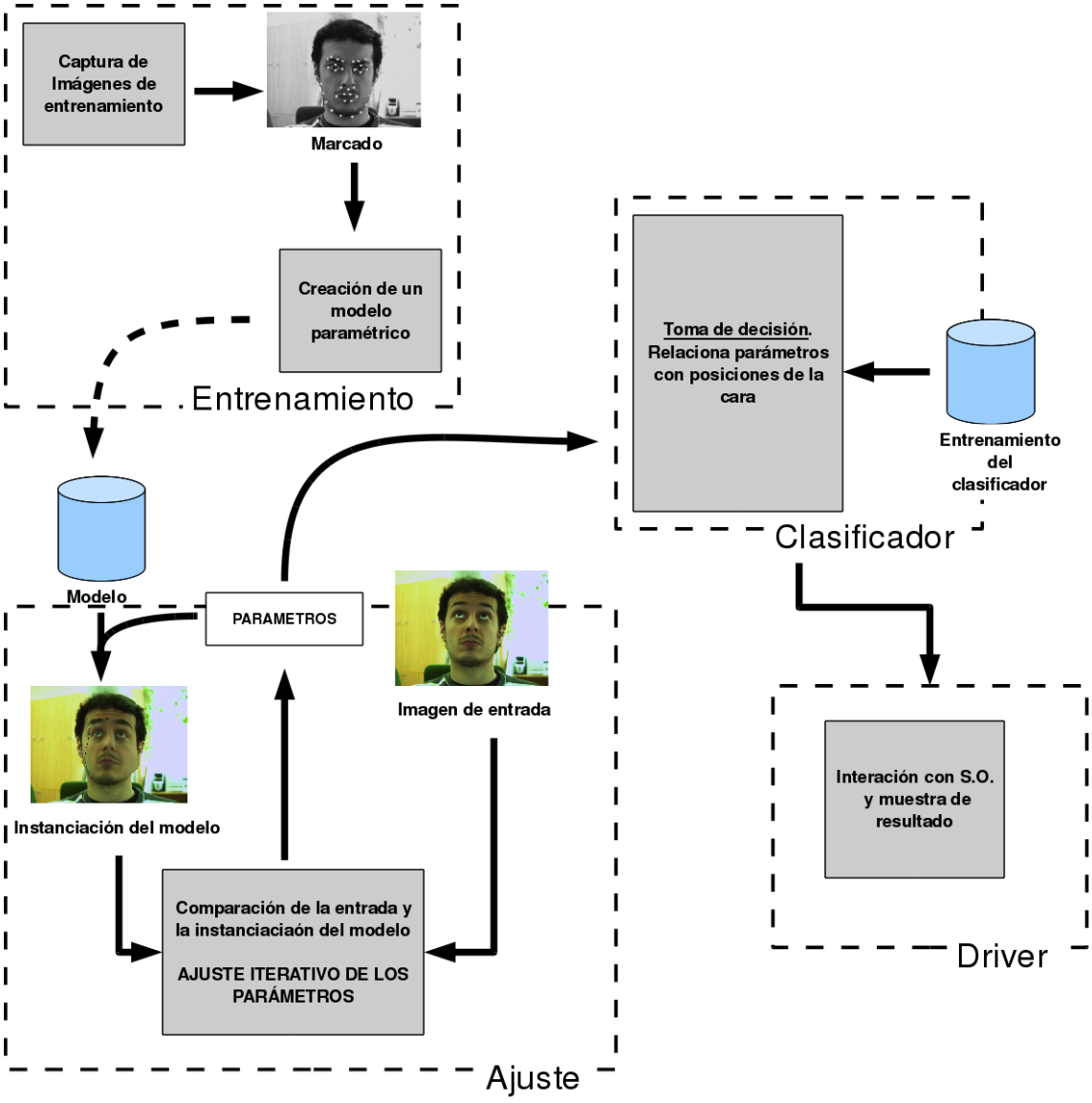


Ilustración 2: Vision general del proyecto

2.- Conceptos teóricos relacionados.

Introducción

El objetivo en este capítulo es tratar con rigor matemático cada una de las herramientas teóricas utilizadas,. De esta manera, se pretende recorrer cada uno de los conceptos teóricos necesarios para llevar a cabo el desarrollo de este proyecto.

En primer lugar se tratará el Análisis de componentes principales (PCA), una potente herramienta, imprescindible para llevar a cabo la fase de entrenamiento, que permite discriminar que dimensiones del problema contienen mayor información y cuáles son prácticamente ruido.

Posteriormente, se introduce el concepto de Modelos de Apariencia Activa (AAM's), técnica en la que esta basado el desarrollo del proyecto. En este apartado se incluyen definiciones fundamentales como las de forma y apariencia de un AAM.

En el tercer apartado, se explica el algoritmo de Procrustes, necesario para eliminar toda información relativa a rotaciones de la cara en las imágenes de entrenamiento. Este algoritmo será necesario cuando se añada a la función warp una transformada global que 'absorba' los cambios de tamaño, posición y giro.

Para finalizar, se presentará el algoritmo de Viola&Jones, ampliamente utilizado en los últimos tiempos para la detección de caras en tiempo real. El uso que posteriormente se le dará al algoritmo de Viola&Jones, es el de poder inicializar el algoritmo de fitting cada vez que sea necesario.

Con todo esto se presentan la bases sobre las que se apoyará el resto de conceptos utilizados en este proyecto.

Análisis de componentes principales (PCA)

El análisis de componentes principales (PCA) en una herramienta ampliamente utilizada en el análisis de datos moderno, en campos tan diversos como neurociencia o visión por computador. Su éxito es debido a que es un método simple y no paramétrico.

Con un mínimo esfuerzo computacional, PCA permite encontrar el camino para reducir la dimensión de un conjunto de datos, a priori complejo, encontrando estructuras simplificadas que podían estar ocultas y que a menudo se encuentra implícitas en este tipo de problemas.

Motivación: un ejemplo.

Para entender mejor que es PCA y como utilizarlo, se usará el ejemplo utilizado en [Jonathon Shlens, 2009].

Sirva el ejemplo hipotético de un investigador que está estudiando el movimiento oscilatorio, para ello engancha una pelota a un muelle ideal y se evita todo tipo de rozamiento.

Conociéndose las leyes de la física, que explican este tipo de movimiento, se podría relacionar la posición en x con cada instante de tiempo ($x(t)=\sin(\omega \cdot t + \phi)$). Sin embargo, como dicho investigador desconoce estas leyes, se decide por poner varias cámaras en distintas posiciones para registrar el movimiento de la bola. Incluso, debido a la ignorancia del investigador, ni siquiera sabe donde están los ejes (x,y,z) y el ángulo formado por las cámaras no es de 90º (Ilustración 3).

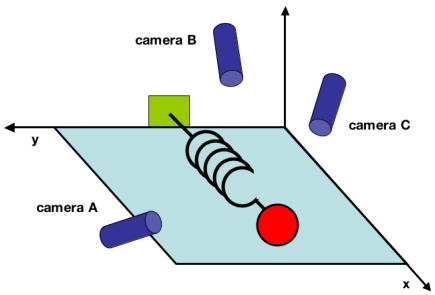


Ilustración 3: Ejemplo PCA

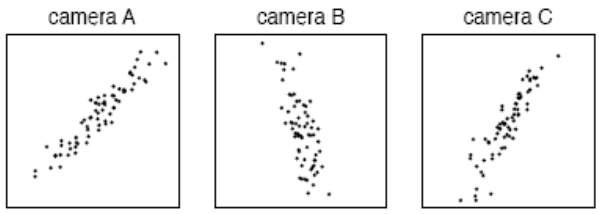


Ilustración 4: Resultado obtenido en el experimento del ejemplo

En la ilustración 4, se muestra un posible resultado que se obtendría del experimento mencionado. En él se puede apreciar que en el registro de la posición realizado por las cámara existe cierto ruido que aleja la observación de la ideal.

En este caso tan simple, parece fácil llegar a la conclusión que todas las cámaras están observando la misma linea y que esta se refiere al eje 'x', pero en general llegar a este tipo de conclusiones no es trivial. Es por ello que PCA resulta tremendamente útil en problemas en los que se desconocen su dimensión real y se disponen de medidas con ruido.

Fundamentos: Cambio de base

El análisis PCA se fundamenta en encontrar el cambio de coordenadas más apropiado para un conjunto de datos de entrada, de modo que la estructura de los mismos quede alineada con las nuevas coordenadas. Como se ha visto anteriormente, de este cambio de bases se espera que sea capaz de filtrar el ruido y de desvelar cualquier posible estructura oculta.

Base origen

Para el análisis, cada muestra temporal del experimento se trata como una muestra individual en el conjunto de datos.

En el ejemplo anterior, cada dato podría ser expresado como un vector de 6 dimensiones (2 dimensiones por cada cámara para determinar su posición en la imagen). En general, el vector constará de *m dimensiones.*

$$x = \begin{bmatrix} X_A \\ Y_A \\ X_B \\ Y_B \\ X_C \\ Y_C \end{bmatrix} \tag{1}$$

Normalmente, la base origen elegida está determinada por el propio experimento y se suele elegir de tal forma que refleje fácilmente el resultado de este. Es muy común que, por simplicidad, la base elegida sea una base ortonormal, representado por la matriz identidad de dimensión *mxm*.

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \vdots \\ \boldsymbol{b_m} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2}$$

Cambio de base

PCA pretende responde la siguiente pregunta: *"¿Existe otra base, combinación lineal de la base de origen, en la que se pueden expresar mejor el conjunto de datos?"*.

La linealidad simplifica enormemente el problema restringiendo las posibilidades. Además permite emplear todo lo que conocido de álgebra lineal, en especial los cambios de base.

$$\boldsymbol{P} \cdot \boldsymbol{X} = \boldsymbol{Y} \tag{3}$$

Donde son definidos los siguientes elementos:

- $\mathbf{p_i}$ son las filas de **P.**
- $\mathbf{x_i}$ son las columnas de **X.**
- $\mathbf{y_i}$ son las columnas de **Y.**

La matriz de covarianza

La varianza de conjunto de datos de **media cero** se define como $\sigma_X^2 = \frac{1}{n} \cdot \sum_i x_i^2$. De forma parecida, se define la covarianza entre dos conjuntos de datos de media nula como $\sigma_{AB}^2 = \frac{1}{n} \cdot \sum_i a_i \cdot b_i$ y se cumple lo siguiente:

- σ_{AB} es cero si y solo si A y B son incorrelados
- $\sigma_{AB}^2 = \sigma_A^2$ si $A=B$

Con estos principios, se define la matriz de covarianza para cualquier matriz cuyas filas sean vectores que representen un conjunto de datos de la siguiente forma:

$$\boldsymbol{C}_X \equiv \frac{1}{n} \cdot \boldsymbol{X} \cdot \boldsymbol{X}^T \tag{4}$$

La matriz de covarianza tiene las siguientes propiedades:

- C_X es una matriz cuadrada *mxm* y simétrica
- Los términos de la diagonal C_X coinciden con la varianza de los vectores que formaban X.
- Los términos fuera de la diagonal son la covarianza entre dos de los vectores de X.

Volviendo al ejemplo del principio, los datos que caen fuera de la diagonal, estarán relacionados con la redundancia lógica de haber grabado con tres cámaras una escena. Sin duda, lo deseable sería que todos los datos fueran incorrelados, es decir, que no existiera redundancia y, por lo tanto, la matriz de covarianza fuese diagonal.

Componentes principales

Partiendo de la idea anterior, se puede resumir el objetivo de PCA como "*encontrar una matriz* $\boldsymbol{P}$ *que cumpla (3) tal que la* $\boldsymbol{C_Y}$ *sea diagonal*". Para lo cual, se seguirá el siguiente desarrollo:

$$\begin{aligned} \boldsymbol{C_Y} &= \frac{1}{n}\cdot\boldsymbol{Y}\cdot\boldsymbol{Y^T} = \frac{1}{n}\cdot(\boldsymbol{P}\cdot\boldsymbol{X})\cdot(\boldsymbol{P}\cdot\boldsymbol{X})^T \\ &= \frac{1}{n}\cdot\boldsymbol{P}\cdot\boldsymbol{X}\cdot\boldsymbol{X^T}\cdot\boldsymbol{P^T} = \boldsymbol{P}\cdot(\frac{1}{n}\cdot\boldsymbol{X}\cdot\boldsymbol{X^T})\cdot\boldsymbol{P^T} \\ \boldsymbol{C_Y} &= \boldsymbol{P}\cdot\boldsymbol{C_X}\cdot\boldsymbol{P^T} \end{aligned} \tag{5}$$

En este momento, es necesario recordar que cualquier matriz **A** simétrica es diagonalizable, y se cumple que $\boldsymbol{A} = \boldsymbol{E}\cdot\boldsymbol{D}\cdot\boldsymbol{E^T}$ donde **D** es una matriz diagonal cuyos valores de la diagonal coinciden con los autovalores de **A**; y **E** es una matriz de paso, cuyas columnas son los autovectores de **A**.

Si se elige inteligentemente **P** tal que $\boldsymbol{P} = \boldsymbol{E^T}$ y sabiendo que $\boldsymbol{P^T} = \boldsymbol{P^{-1}}$ se obtiene el siguiente resultado:

$$\begin{aligned} \boldsymbol{C_Y} &= \boldsymbol{P}\cdot\boldsymbol{C_X}\cdot\boldsymbol{P^T} = \boldsymbol{P}\cdot\left(\boldsymbol{E}\cdot\boldsymbol{D}\cdot\boldsymbol{E^T}\right)\cdot\boldsymbol{P^T} \\ &= \boldsymbol{P}\cdot\left(\boldsymbol{P^T}\cdot\boldsymbol{D}\cdot\boldsymbol{P}\right)\cdot\boldsymbol{P^T} \\ \boldsymbol{C_Y} &= \boldsymbol{D} \end{aligned} \tag{6}$$

De este resultado se pueden sacar las siguientes conclusiones:

- Las componentes principales de **X** son los autovectores de $\mathbf{C_X}$.
- Cada autovalor de **X** coincide con la varianza de cada componente principal. Se asumirá que las componentes de mayor varianza son de mayor importancia, los datos con menor varianza, más incorrelados, se suelen achacar al ruido

En la práctica, como X debe tener media cero, se calcula la media del conjunto de datos de entrada y se resta.

Tras el análisis con PCA, es posible reducir la dimensión de un problema sin perder información importante, para ello se deberá eliminar del espacio vectorial de destino aquellas componentes con menor importancia (menor autovalor asociado) . Esto implica eliminar de la matriz de paso **P** aquellos vectores asociados a los autovalores mas pequeños.

Modelos de Apariencia Activa

Los Modelos de Apariencia Activa (AAM's) fueron propuestos inicialmente por [Cootes et al, 1998] y están íntimamente relacionados con los conceptos de *"Active Blobs"* y *"morphable models"*, ya que todos ellos, son modelos generadores (es decir, que a partir de ellos se pueden reconstruir objetos, que jamás han sido utilizados en la construcción del modelo) , paramétricos y no lineales, Estos modelos se engloban dentro de los *Modelos de Apariencia y forma lineales (lineal shape and appearance models)* que surgieron, en su mayoría, de forma independiente entre 1997 y 1998.

Para simplificar la terminología, en este proyecto, se tratarán de diferenciar el modelo del algoritmo de ajuste (*fitting*). Se utiliza el término AAM para referir únicamente al modelo, ya que se considera que existen varios algoritmos aplicables con el objetivo de reducir el error entre una imagen de entrada y otra imagen generada a partir del modelo (ver apartado 4),.

Tipos de AAM's

Fundamentalmente, dos son las formas de caracterizar un AAM. Los que tratan de forma separada los modelos de forma y los de apariencia, y los que lo hacen de forma conjunta mediante un conjunto de parámetros que engloban tanto forma como apariencia. A los primeros, se les suele denominar *AAM's independientes* y a los segundos *AAM's combinados.* En este proyecto solamente se tratarán los primeros.

AAM's independientes

Forma:

La forma en una AAM está definida por una malla y en particular por los vértices de esa malla. Matemáticamente se define la forma s como un vector formado por las coordenadas de los vértices que forman la maya.

$$s = (x_1, y_1, x_2, y_2, \ldots, x_v, y_v)^T \tag{7}$$

Los AAM permiten variaciones lineales en la forma. Esto significa que la forma puede ser expresada como la suma de una base de forma s_0 y n vectores de forma s_i .

$$s = s_0 + \sum_{i=1}^{n} p_i \cdot s_i \tag{8}$$

Siendo p_i los **parámetros de forma**. Se considerará que los vectores s_i son ortonormales. En caso contrario siempre se puede aplicar un proceso de ortonormalización a los mismos y su consecuente parametrización.

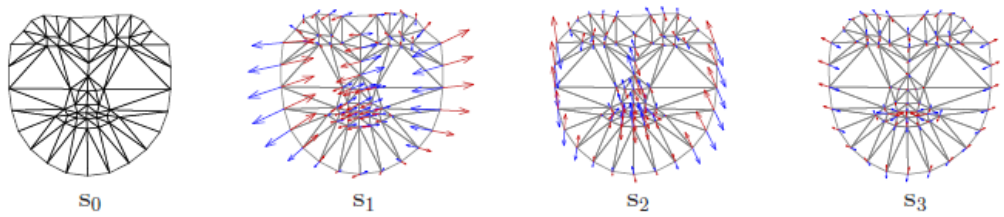


Ilustración 5: Modelo lineal de forma en un AAM lineal.
El modelo consiste en una base s_0 y una combinación de n vectores de forma s_i

Los AAM's son normalmente calculados a partir de q imágenes, a las que se les relacionan q vectores que definen la malla previamente fue etiquetada a mano para cada imagen.

Posteriormente, esas imágenes de entrenamiento son sometidas a un análisis de componentes principales (PCA). Este proceso dará como resultado s_0, que se corresponde con la media de todas las mallas de entrenamiento, y s_i, que coincide con los n autovectores asociados a los n autovalores mayores.

En Ilustración 5, se puede ver un ejemplo de base de forma en la que aparecen la malla media s_0 y una serie de vectores s_1 que representan las posibles transformaciones que pueden afectar a la media.

Normalmente, antes del análisis PCA, se suelen normalizar todas las mallas de entrenamiento utilizando el algoritmo Procrustes. Este paso elimina toda la información correspondiente a transformaciones globales de la malla, es decir, cambios de tamaño, desplazamientos y giros. Permaneciendo, únicamente, los cambios debidos a transformaciones no rígidas de la malla.

Apariencia

La apariencia, de un AAM independiente se define dentro de la malla base s_0. Al igual que en la bibliografía utilizada, abusando de notación, se utiliza s_0 para denotar a todos los píxeles $\boldsymbol{x} = (x, y)^T$ que caen dentro de la malla s_0. Por consiguiente, la apariencia $A(x)$ es una imagen definida sobre los píxeles $x \in s_0$. Los AAM's permiten cambios lineales en la apariencia, lo que quiere decir que $A(x)$ puede ser expresado como una base de apariencia $A_0(x)$ y una combinación de m imágenes de apariencia $A_i(x)$:

$$A(x) = A_0(x) + \sum_{i=1}^{m} \lambda_i \cdot A_i(x) \quad \forall x \in s_0 \tag{9}$$

Donde λ_i se corresponde con los parámetros de apariencia.

Al igual que con la forma, se considerará que A_i son ortonormales, ya que en el caso de que no lo fueran, ortonormalizarlos solo implicaría un cambio en la parametrización.

El entrenamiento se realiza, como con la forma, aplicando PCA sobre las q imágenes de entrenamiento una vez modificadas con una función warp que las normaliza, transformándolas desde la malla de entrenamiento a la malla base s_0. En el caso de este proyecto, como se estudiará posteriormente, para realizar la función warp, la malla es triangularizada y se crea una relación afín

entre los triángulos correspondientes de la malla origen (en la imagen de entrenamiento) y la de destino (la malla base).

Como resultado del análisis PCA, tal y como pasaba con la forma, se obtiene A_0 de la media de todas las imágenes normalizadas y A_i de los m autovalores con mayor autovalor asociado. Tal y como se muesta en la Ilustración 6.

Debido a haber realizado previamente la normalización mencionada, se obtiene un autoespacio de apariencia más compacto que el obtenido originalmemte.

Ilustración 6: Modelo lineal de apariencia en un AAM lineal. El modelo consiste en una base A_0 y una combinación de n vectores de apariencia A_i

Función de warp

Esta función definida a trozos permite relacionar cada punto en el interior de la malla s_0 con un punto en el interior de cualquier malla s definida por los parámetros p. Como ya se ha mencionado en el apartado referido a la apariencia, el propósito de emplear esta función es el de poder comparar la apariencia del modelo en un espacio común, independiente a los parámetros de forma.

La función de warp se expresa matemáticamente como $W(x,p)$, siendo x un punto en s_0 y p los parámetros que definen las variaciones de forma. El valor devuelto por esta función es el punto correspondiente en s.

Evidentemente, según la anterior definición, cada uno de los v puntos que forman la malla s_0 deben transformarse en su correspondiente en s mediante (8). El resto de puntos en el interior de la malla se encuentran situados dentro de un triángulo cuyos vértices son tres puntos de s_0: (x_i^0, y_i^0), (x_j^0, y_j^0) y (x_k^0, y_k^0), al tratarse de una transformación afín, el resultado de calcular el warp de cada uno de esos puntos permanecerá en el interior del mismo triangulo, esta vez definido con tres puntos de s: (x_i, y_i), (x_j, y_j) y (x_k, y_k). (Ver la siguiente ilustración)

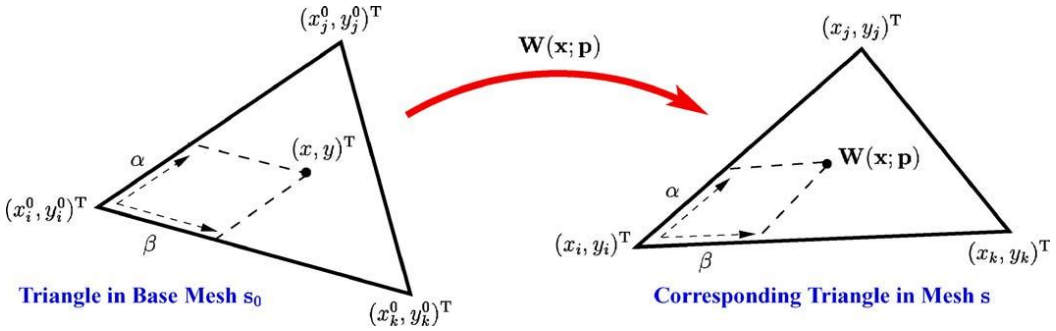


Ilustración 7: Función warp

Instanciación de modelo

Las ecuaciones (8) y (9) describen las variaciones en forma y apariencia. Sin embargo, no describen como generar una instancia del modelo, es decir, como combinar forma y apariencia para sintetizar una cara.

Dados los parámetros de forma $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)^T$ se puede utilizar la ecuación (8) para generar la forma del AAM ***s***. De manera similar, dados los parámetros de apariencia $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)^T$, se puede generar la apariencia ***A(x)*** definida en el interior de la malla base s_0. Finalmente, a partir de ***s*** y ***A(x)***, se genera la instancia del modelo AAM ***M(x)*** haciendo el warp de la ***A(x)*** desde la malla base, en la que está definido, a la malla del modelo ***s***. La siguiente ecuación y la Ilustración 8 que le sigue resume lo explicado en este párrafo:

$$M(W(x;p)) = A(x) = A_0(x) + \sum_{i=1}^{m} \lambda_i \cdot A_i(x) \quad \forall\, x \in s_0 \tag{10}$$

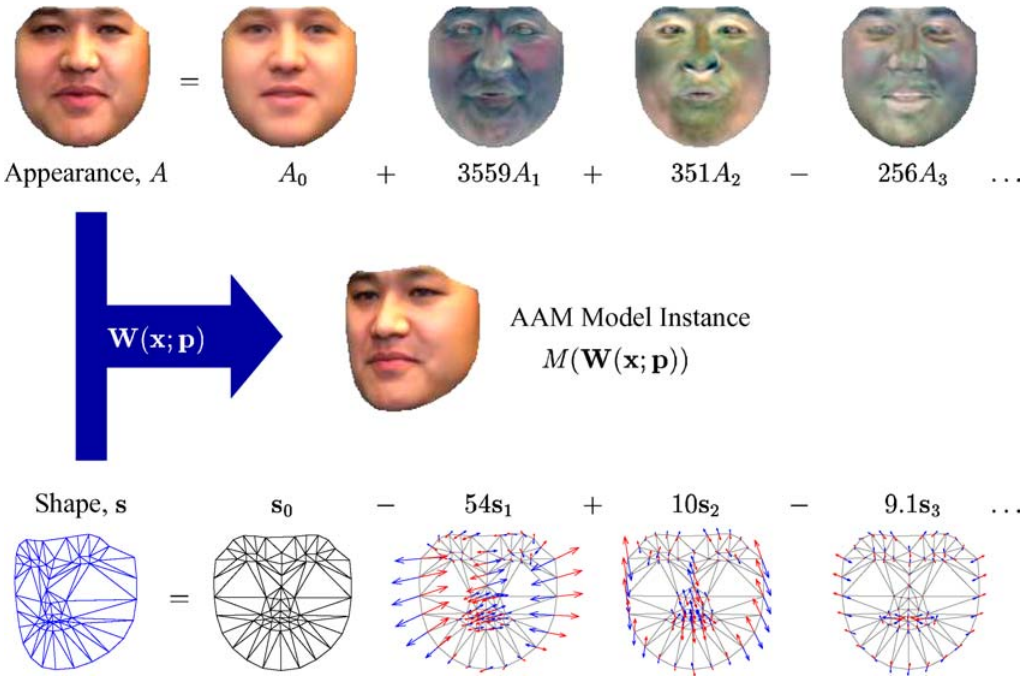


Ilustración 8: Ejemplo de Instanciación

Procrustes

Introducción

El análisis de Procrustes es un tipo de análisis estadístico que hace que un conjunto de distintas formas tengan el mismo tamaño, orientación y posición. De esta manera, resulta más sencilla y/o eficaz la comparación entre ellas. El análisis de Procrustes es ampliamente utilizado en campos como la bioingeniería.

El nombre de Procrustes tiene su origen en la mitología griega. Procrusto también llamado Damastes, era un bandido y posadero que tumbaba a sus víctimas en una cama de hierro para posteriormente, después de atados, estirarlos o cortarlos para hacerlos coincidir con el tamaño de la cama.

Objetivo

El motivo por el que se necesitará el algoritmo de Procrustes es el de eliminar toda aquella información relativa al tamaño, rotación y translación contenida en la malla de las imágenes de entrenamiento.

A pesar de que esta información es útil y siempre estará presente, no aporta dato alguno sobre el gesto de la cara sino de su posición. Es por ello que resulta muy conveniente el tener identificados estos parámetros y no mezclarlos con los parámetros de forma, que sirven para modelar cambios no rígidos en la malla.

De esta manera, aplicando procrustes antes de PCA, se conseguirá evitar que el análisis PCA "aprenda" a reproducir estos cambios que, por otra parte, son fácilmente reproducibles mediante transformaciones afines como se verá más adelante.

Distancia de Procrustes

De la misma forma que con la distancia Euclídea entre dos vectores se puede conocer la proximidad entre ellos, la distancia de Procustres da idea de la rotación que hay que aplicar a un conjunto de puntos para llegar a parecerse a otro conjunto.

Para poder calcular esta distancia se debe, en primer lugar, eliminar la información de translación y de tamaño del conjunto de puntos. Esto es algo bastante común: basta con restarle a cada vector su media en **x** y en **y** y normalizarlo. Así sea el vector *s* definido en (7) se definirá su traslación en **x** y en **y**, respectivamente, como:

$$\bar{x} = \frac{\sum_{i=1}^{v} x_i}{v} \tag{11}$$

$$\bar{y} = \frac{\sum_{i=1}^{v} y_i}{v} \tag{12}$$

En donde *v* es el numero total de puntos que componen la malla del entrenamiento.

Por su parte, la norma de la malla se calcula de la siguiente manera:

$$\|s\| = \sqrt{\sum_{i=1}^{v} (x_i - \bar{x})^2 + (y_i - \bar{y})^2} \tag{13}$$

A partir de las ecuaciones (11), (12) y (13) se puede calcular una nueva instancia de la malla en la que ha sido eliminada la traslación y el tamaño:

$$\hat{s} = \frac{(x_1 - \bar{x}, y_1 - \bar{y}, \ldots, x_v - \bar{x}, y_v - \bar{y})}{\|s\|} \tag{14}$$

Sean dos vectores $\boldsymbol{a} = (x_1, y_1, \ldots, x_v, y_v)$ y $\boldsymbol{b} = (w_1, z_1, \ldots, w_v, z_v)$ que describen una malla, se define como norma de Procrustes como la raíz cuadrada de la suma de la distancia que hay entre el mismo punto de las dos malla al cuadrado:

$$d_{procrustes} = \sqrt{\sum_{i}^{n} (x_i - w_i)^2 + (y_i - z_i)^2} \tag{15}$$

El objetivo del algoritmo que se plantea a continuación es reducir todo lo posible esta distancia, lo que conllevará a reducir la rotación entre las dos mallas.

Análisis de Procrustes ortogonal y generalizado.

El análisis de Procrustes ortogonal es una aproximación del problema mencionado usando algebra lineal [Ross].

Se partiŕa de una matriz X que se desea rotar mediante una matriz de rotación Q para que se alinee lo más posible a una matriz $\bar{X}$ usando el criterio de distancia de Procrustes. Esto es lo mismo que minimizar la siguiente ecuación.

$$\|X \cdot Q - \bar{X}\| \tag{16}$$

Donde $\|\cdot\|$ representa la norma de Frobenius, es decir, la traza (suma de los elementos de la diagonal principal) de la matriz resultante de multiplicar la matriz por su transpuesta, esto es:

$$\|A\| = traza(A^T \cdot A) \tag{17}$$

Atendiendo a la definición de norma anterior se puede expresar (16) de la siguiente manera:

$$\|X \cdot Q - \bar{X}\| = traza([X \cdot Q - \bar{X}]^T \cdot [X \cdot Q - \bar{X}]) = traza(X^T \cdot X + \bar{X}^T \cdot \bar{X}) - 2 \cdot traza(\bar{X}^T \cdot X \cdot Q \tag{18}$$

Como ambos término son positivo y en el primero no aparece Q, la tarea de minimizar la expresión (16) se puede conmutar por la de maximizar el segundo término.

Usando la descomposición en valores singulares $\bar{X}^T \cdot X = U \cdot S \cdot V^T$ y la propiedad cíclica de la traza se obtiene:

$$traza(X^T \cdot X \cdot Q) = traza(U \cdot S \cdot V^T \cdot Q) = traza(S \cdot V^T \cdot Q \cdot U) = traza(S \cdot H) \tag{19}$$

Como S, por definición, es una matriz diagonal semi-definida positiva y H es una matriz ortogonal ya que es producto de otras matrices ortogonales, la expresión (19) se hace máxima cuando la matriz H es la identidad. Por lo tanto:

$$H = I = V^T \cdot Q \cdot U \tag{20}$$

Por lo que la Q que minimiza la distancia de Procrustes es:

$$Q = V \cdot U^T \tag{21}$$

Conclusión

Según lo analizado hasta ahora, se utilizará la descomposición en valores singulares para implementar el algoritmo de *Procrustes Ortogonal.* Sin embargo, todavía queda por definir cuál será el proceso tomado para el total de todas las imágenes de entrenamiento, cómo sera definido $\bar{X}$ al comienzo del algoritmo, como se procede para actualizarlo y cuándo se dará por concluido el algoritmo, asumiendo que todos los vectores están alineados.

Con ese objetivo, se plantea el siguiente resumen del algoritmo tomado:

***Resumen Algoritmo* Procrustes**

Iterar:

1) Calcular $\bar{x}$ e $\bar{y}$
2) Calcular la media: $\|s\| = \sqrt{\sum_{i=1}^{v} x_i^2 + y_i^2}$
3) Hallar $\hat{s}$ mediante (14).

Con todos los vectores *s* del entrenamiento

4) Inicializar $\bar{s}$ con el primer $\hat{s}$

Iterar:

Iterar:

a) Hallar la descomposición en valores simples de $\bar{s}^T \cdot \hat{s} \rightarrow U \cdot S \cdot V^T$
b) Calcular la matriz de rotacion Q usando (21)
c) Calcular la matriz H usando la ecuación (30)
d) Actualizar $\hat{s}$ multiplicándolo por la matriz de rotación Q.

Con todos los vectores $\hat{s}$

5) Calcular la media de los nuevos $\hat{s}$, $\bar{s}_2$
6) Realizar procrustes (pasos a,b,c y d) con $\bar{s}$ y $\bar{s}_2$. (Actualizando $\bar{s}$)

Hasta que $\Delta \bar{s} < \varepsilon$

Algoritmo de Viola & Jones

Introducción

El algoritmo de Viola & Jones [Viola & Jones, 2003] describe un algoritmo de **detección de caras** que es capaz de procesar imágenes rápidamente manteniendo un alto ratio de aciertos. Viola & Jones se fundamenta en 3 puntos principales que se estudiarán a continuación.

El motivo de incluir este algoritmo en este proyecto es la necesidad que tiene los Modelos de Apariencia Activa de ser inicializados en un entorno cercano a la solución para que converjan correctamente. Gracias a Viola & Jones se obtendrá la posición de la cara en las imagen y su tamaño, con lo que se podrá realizar una primera aproximación de los parámetros (sobre todos los relativos a la transformación global, ver pág: 73) que servirá de punto de partida del Algoritmo de Minimización de AAM's

"Integral Image"

El análisis de Viola & Jones se fundamenta en el uso de 'features' que, a su vez, están basadas en las transformadas wavelets de Haar. La ventaja principal de analizar estas 'features' en lugar de hacerlo diréctamente con los píxeles de la imagen es que con estas 'features' se puede codificar información útil que resultaría difícil aprender usando un conjunto finito de imágenes de entrenamiento.

Las 'features' propuestas orginalmente es un pequeño conjunto de 'features' rectangulares que permiten la localización de bordes, esquinas y puntos aislados. Hay datos que indican que se puede realizar una detección adecuada utilizando únicamente 2 features.

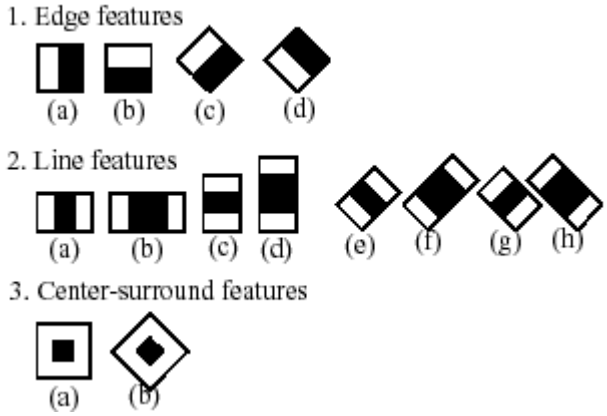


Ilustración 9: Ejemplo de Features utilizadas por la bibloteca OpenCV

'Integral Image' es un método propuesto para optimizar el cálculo de estas 'features'. Dicho método consiste en una nueva forma de representación de la imagen donde cada píxel contiene la suma de todos los píxeles situados por encima y a la derecha de él, con él incluido:

$$ii(x, y) = \sum_{x' \leq x,\ y' \leq y} i(x', y') \tag{22}$$

Donde *ii(x , y)* es la 'Integral Image' y *i(x , y)* es la imagen original.

Usando esta representación, el cálculo de 'features' rectangulares se puede simplificar muchísimo. Por ejemplo, para el cálculo de la primera 'feature' mostrada en la Ilustración 9 se utilizarían únicamente 6 accesos al array *ii(x , y).*

Clasificador basado en AdaBoost

AdaBoost es una algoritmo de aprendizaje adaptativo que permite estimular el rendimiento de cualquier algoritmo de aprendizaje simple.

Utilizando un clasificador basado en AdaBoost sobre unas imágenes de entrenamiento, el algoritmo de Viola & Jones es capaz de "aprender" que features indican la presencia de una cara en una subventana y cuales no.

A continuación se presenta el pseudoalgoritmo del algoritmo del clasificador:

***Resumen Algoritmo* Aprendizaje**

Sean $(x_1, y_1), \ldots, (x_n, y_n)$ imágenes de ejemplo, donde $y_i = 0, 1$ para puntos donde hay fallo o acierto respectivamente.

- Inicializar los pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0, 1$ respectivamente, siendo m es el número de fallos y l el numero de aciertos.
- Iterar desde t=1 hasta t=T (siendo T el número máx de iteraciones):

  1) Normalizar los pesos: $w_{1,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

  2) Para cada 'feature' fj se entrena un clasificador hj el cual está restringido a usar una sola 'feature', el error esta ponderado con respecto al peso: $\varepsilon_t = \sum_i w_i \cdot \left| h_j(x_i) - y_i \right|$.

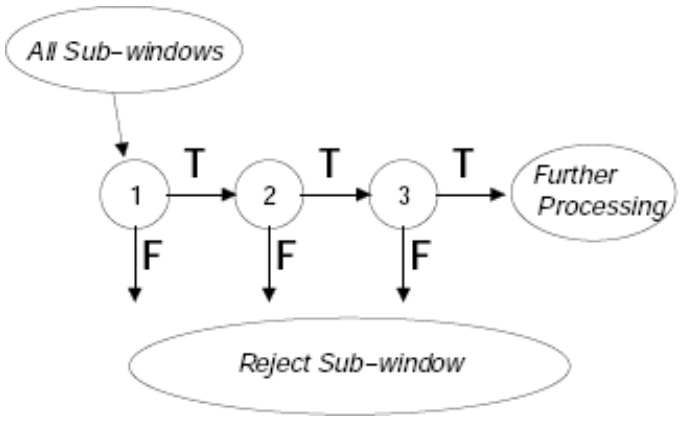
  3) Elegir el clasificador ht que obtenga el menor error εt.

  4) Actualizar los pesos: $w_{t+1,i} = w_{t,i} \beta_t^{(1-e_i)}$. Donde $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ e_i=0 si el ejemplo x_i ha sido clasificado correctamente y e_i=1 en el caso contrario.
- El clasificador final sera:

$$c(x) = \begin{cases} 1 & Si \ \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & En\ otro\ caso \end{cases} \text{ Donde } \alpha_t = \log \frac{1}{\beta_t}$$

Método en cascada

La última de las 3 mejoras que propone Viola & Jones para la mejora del rendimiento es un método que combina sucesivamente clasificadores más complejos con un procesado en cascada, desechando regiones donde los primeros clasificadores han fallado y profundizando más en aquellas regiones que proporcionan aciertos del clasificador.



3.- Entrenamiento

Introducción

El entrenamiento es una parte principal del sistema desarrollado en el presente proyecto. Gracias a un buen entrenamiento es posible distinguir entre que variaciones de la forma y la apariencia son posibles en un rostro humano y cuales no. De esta forma, mediante un análisis PCA, ser reducirán considerablemente las dimensiones del problema.

Debido a que el entrenamiento es el primer paso en el que se establecen los primeros límites, se debe tener en consideración que un entrenamiento demasiado pequeño resultará una base con pocas componentes, incapaz de representar movimientos complejos o sutiles de la cara. Por el contrario, un sobreentrenamiento provocará que las dimensiones del problema crezcan, hasta el punto que el fitting se haga ineficiente, e incluso inviable.

Por todo esto, es fundamental un cuidadoso y exhaustivo estudio de como va a realizarse el proceso de entrenado del modelo.

Para diseñar un buen entrenamiento, resulta de gran importancia identificar aspectos del modelo que son independientes y, en consecuencia, entrenarlos separadamente. De esta forma, facilita el ajuste del modelo, e incluso su posterior interpretación. En este caso, se tratará de independizar tanto en el modelo como en el entrenamiento las deformaciones de la malla, la apariencia, y las transformaciones globales que afectan a la malla.

El uso de PCA implica que el entrenamiento será, en su mayoría, un desarrollo lineal. Esto reducirá la complejidad del mismo y permitirá utilizar las numerosas librerías y funciones de Álgebra lineal existentes (incluso dentro de OpenCV), lo que facilitará su posterior desarrollo.

Captura de datos.

Este primer paso del entrenamiento es el único en el que se requerirá de la actuación humana. En él un usuario experimentado debe de seleccionar las imágenes convenientes y marcar la malla correctamente, tal y como debería ser posicionada por el sistema.

Es esencial que, en este primer paso, dicho usuario sea exhaustivo y trate de colocar los puntos de forma que siempre este en la misma posición dentro de la cara, evitando en todo caso la conmutación entre diversos puntos. Ya que este tipo de errores harían "aprender" movimientos que no son propios de la cara sino de un error en el entrenamiento.

En todo caso, en este primer paso, se presenta un problema fundamental: las autooclusiones, es decir partes de la cara que son ocultadas por otras partes de la propia cara. Por ejemplo, un ojo ocultado por el tabique nasal. Este fenómeno provocará que la base de apariencia crezca, ya que debe de absorber cambios que son relativos a la forma pero que no han podido ser caracterizados por esta.

La implementación de la captura de datos se llevará acabo a partir de varios pasos que se muestra en el siguiente diagrama.

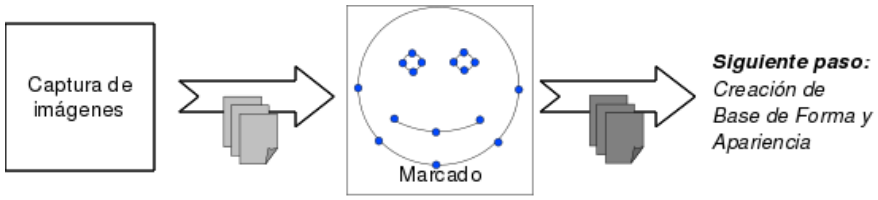


Ilustración 10: Captura de datos

- **Captura de imágenes:** Se trata de obtener imágenes por medio de una cámara. Es importante que la iluminación en todas las imágenes tomadas sea aproximadamente igual.

Por otra parte, en principio, es conveniente no utilizar muchas personas distintas ya que esto podría provocar que se disparasen el número de dimensiones de tanto la base de forma como la de apariencia.

- **Marcado:** Como se ha comentado con anterioridad, esta es la única función en todo el desarrollo que no se puede automatizar. Para su acometido, es necesario que alguien coloque, "a mano", uno por uno los puntos representativos de la malla que define la forma para cada una de las imágenes que conformarán el entrenamiento.

Tras estos dos procedimientos, se obtendrán tantos mapas de bits como imágenes tenga el entrenamiento, y asociado a cada imagen un conjunto de $2v$ duplas {x,y}, siendo v el número de vértices de la malla, que representan las coordenadas de cada uno de los puntos señalados. Se puede ver un ejemplo del marcado utilizado en la implementación y la malla correspondiente a ese configuración de puntos en la Ilustración 11

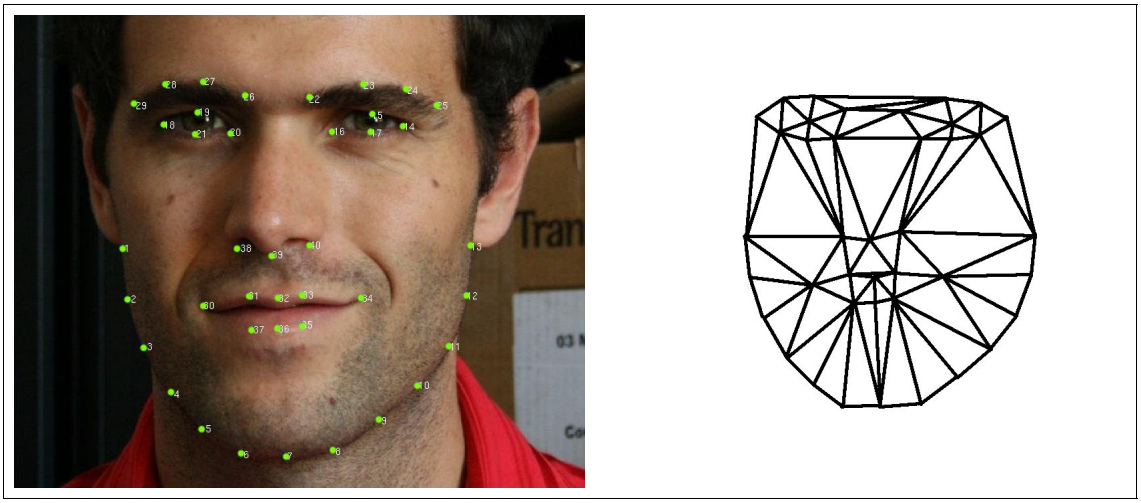


Ilustración 11: Ejemplo de marcado utilizado y correspondiente malla

Estas $2v$ duplas ordenadas y dispuestas en forma de vector se corresponden directamente con el vector de forma mencionado en la ecuación (7).

A partir de esta paso inicial, se dispone de todo lo necesario para realizar una base PCA, tal y como mse muestra a continuación.

Creación de bases de forma y apariencia

Una vez obtenidas y marcadas posteriormente las imágenes que forman el entrenamiento, se hace necesario su procesado. Básicamente, interesaría conocer cuál y/o cuánta de la información obtenida es redundante, cuál aporta información y cuál es ruido que se ha producido en el proceso de captura de datos, por ejemplo debido al ruido de la cámara o introducido por la persona que marcó las imágenes.

Todo este análisis es posible realizarlo mediante el uso de PCA, una poderosa herramienta explicada anteriormente en este proyecto (ver pág 32) y que se utilizada ampliamente en estudios multivariable.

Al estar utilizando modelos de apariencia activa independientes, se analizarán por separado forma y apariencia, obteniendo por separado dos bases independientes. Lo ideal sería que forma y apariencia fueran independientes entre si, sin embargo, esto no ocurre en general. Aunque, como se explica posteriormente, se puede asumir como una buena aproximación la independencia entre ambas.

Base de forma.

El primer paso al construir una base con PCA es eliminar todo aquello que sea redundante en el conjunto de datos que se dispone. Para ello se calcula la media de todos vectores de forma **s** asociados a la imagen de entrenamiento para posteriormente restársela a cada uno. De esta for-

ma, se conseguirá que todos los vectores de forma tengan media nula. Al vector media se le denominará, a partir de ahora, $\mathbf{s_0}$.

Posteriormente, se hallarán la componentes principales del conjunto de vectores con media nula obtenidos en el paso anterior.

Debido a que cada componente principales se corresponde con un autovalor de la matriz de autocorrelación de los datos, cada una de ellas tendrá asociada un autovalor. Es de interes ordenar cada componente principal en orden descendiente dependiendo de su autovalor ya que de esta forma se sabrá cuales aportan más información, haciendo posible incluso el filtrado de las que menos información aportan para reducir la complejidad del problema.

Para el filtrado se utilizará el porcentaje de energía (autovalor al cuadrado) que cada vector representa del total. El procedimiento es el siguiente:

1. Se calcula la suma de los autovalores al cuadrado para conocer la 'energía' total del sistema. $E_{total} = \sum_{i=1}^{n} \alpha_i^2$ donde α es el autovalor y n es el numero de autovectores devueltos por la función PCA
2. Se establece un umbral (en el caso de este proyecto ha sido establecido a 0.99)

Se añaden vectores a la base hasta que $\frac{\sum \alpha_i^2}{E_{total}} > Umbral$.

Dichas componentes principales ordenadas y filtradas serán los vectores que compondrán la base y serán referidas a partir de ahora como $\mathbf{s_i}$. Cabe reseñar, que el número de vectores de la base será como máximo igual al número de imágenes de entrenamiento, reduciéndose este número, tanto más cuanto más restrictivo sea el criterio de filtrado.

En la siguiente ilustración se resume todo lo explicado hasta ahora.

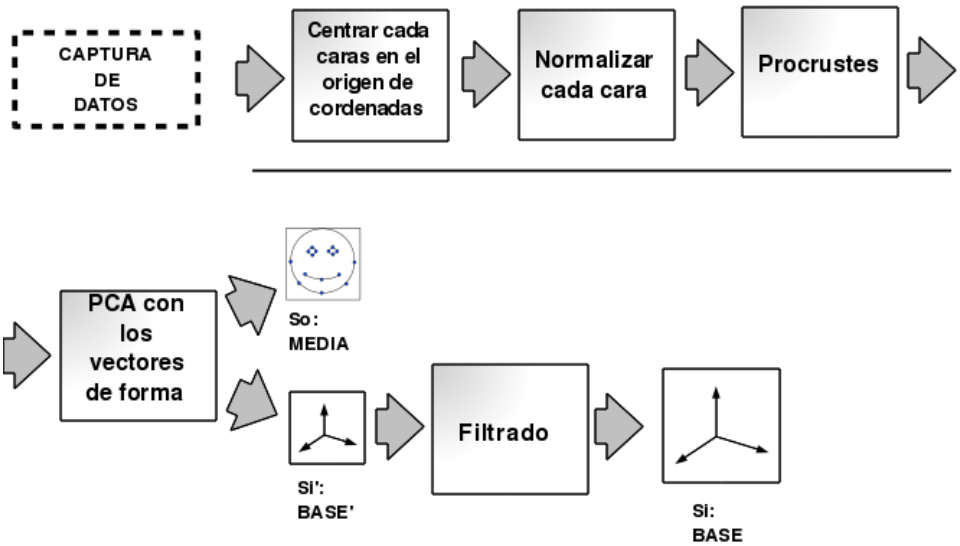


Ilustración 12: Creación de la base de forma

Base de apariencia.

Como se ha comentado anteriormente, el procedimiento para crear la base de apariencia es prácticamente el mismo que el utilizado para la base de forma.

Al igual que en la forma, este proceso también está fundamentado en PCA. Sin embargo, en esta ocasión, los vectores analizados será los constituidos por el valor de cada uno canales de todos los píxeles que forman la imagen una vez transformada esta, mediante la función warp a una forma de referencia común para todas las imágenes analizadas. En la Ilustración 13 se muestra un ejemplo de 6 imágenes del entrenamiento tras haber sido transformadas mediante la función warp.

Dicha forma de destino, que en principio puede ser cualquiera, en la bibliografía consultada se suele hacer coincidir con $\mathbf{s_0}$. Particularmente, en este proyecto, se tratará de respetar este criterio, no obstante habrá situaciones, como por ejemplo para aplicar alguna mejora del algoritmo, en las que será de mayor utilidad cambiarla.

Ilustración 13: Subconjunto de las imágenes utilizadas para crear la base de Apariencia.

En ella se puede ver ,de izquierda a derecha y de arriba a abajo: 1.- De frente, 2.- Mirando a la izquierda, 3.-Mirando a la derecha, 4.- Mirando hacia arriba, 5.- Sonriendo, 6.- Cara de Sorpresa.

Como ya se hizo con la forma, se ordenan los autovectores de apariencia atendiendo a los autovalores asociados. Posteriormente se realizará un filtrado de los autovectores que representan menos energía (menos información) del total utilizando el mismo método que el empleado al crear la base de forma. Es decir, atendiendo al porcentaje de energía (autovalor al cuadrado) de cada autovector en relación con la energía total.

Resumen

En la siguiente ilustración se resume el proceso de creación de bases de forma y aparienecia.

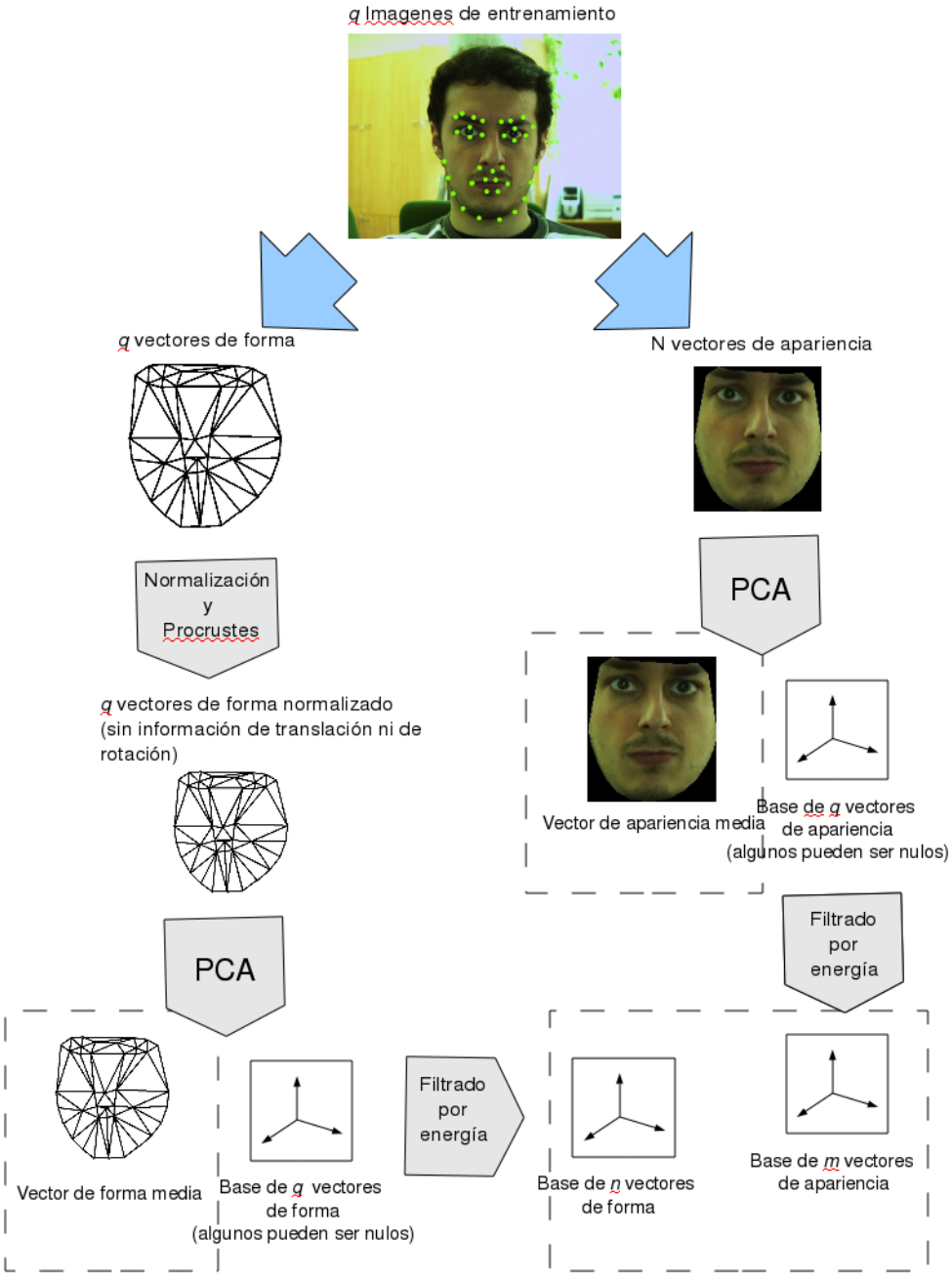


Ilustración 14: Resumen de la creación de bases

4.- Ajuste o "*fitting*"

Objetivo

Se deasean encontrar los parámetros óptimos de forma y apariencia para ajustar un AAM a una imagen de entrada ***I(x')***. Esto significa que la imagen y la instanciación del modelo (10) deben ser similares. Evidentemente, para que el proceso de ajuste esté bien definido, se debe establecer un criterio sirva para identificar cuando dos imágenes son similares, para posteriormente optimizarlo.

Un buen criterio sería utilizar el error cuadrático entre el modelo y la imagen. Sin embargo, existen dos variables independientes en las que este error podría ser calculado: x' en el marco de la imagen $I(x')$ o x en el marco donde esta definida la apariencia del AAM $A(x)$, ambas relacionadas entre si por la función warp.

En términos de algoritmia es mucho mejor elegir la variable definida dentro de $A(x)$, ya que esta se define en la malla de destino del warp (en principio, por convenio, s_0) que es común para todo el proceso y para todas la imágenes de entrada. Por lo tanto, en resumen, la siguiente expresión muestra el error cuadrático que se desea minimizar.

$$\sum_{x \in s_0} E(x) = \sum_{x \in s_0} [A(x) - I(W(x;p))]^2 = \sum_{x \in s_0} [A_0(x) + \sum_{i=1}^{m} \lambda_i \cdot A_i(x) - I(W(x;p))]^2 \qquad (23)$$

Minimizar esta expresión, simultáneamente para los parámetros p y los parámetros λ, será el objetivo de todos los algoritmos de ajuste analizados en este proyecto. En general, en un AAM, la variación de la forma (parámetros p) será no lineal, por el contrario, la apariencia (parámetros λ) tiene un comportamiento lineal.

Algoritmos de Optimización

Los algoritmos de ajuste de AMM's existentes se dividen en dos categorías: Los que toman la aproximación analítica basada en el descenso por el gradiente, con la ventaja de ser algoritmos ampliamente documentados y basados en principios matemáticos pero con la contrapartida de ser “muy lentos”. Por otra parte, existen algoritmos que basan su funcionamiento en suposiciones de partida aproximadas, que pretenden conseguir eficiencia en el ajuste a costa de la precisión del resultado, alguno de estos algoritmos se explican en [Matthews & Baker, 2004].

Este último tipo de algoritmos fueron descartados desde un principio por no proporcionar la robustez ni la precisión requerida para el objetivo del proyecto.

Algoritmo de Lucas-Kanade (Forwards Additive).

En este algoritmo se asume que se conoce una estimación inicial de los parámetros $q = (p, \lambda)^T$ y se calcula, iterativamente, el incremento de esos parámetros $\Delta q = (\Delta p, \Delta \lambda)^T$ que minimiza el error,para posteriormente actualizar los parámetros ($q \leftarrow q + \Delta q$).

$$\sum_{x \in s_0} [A(x) + \sum_{i=1}^{m} (\Delta \lambda_i \cdot A_i(x)) - I(W(x; p + \Delta p))]^2 \tag{24}$$

La ecuación (24) puede ser linealizada en el entorno de p y λ usando el teorema de Taylor:

$$\sum_{x \in s_0} [A(x) + \sum_{i=1}^{m} (\Delta \lambda_i \cdot A_i(x)) - I(W(x; p)) - \nabla I(W(x; p)) \cdot \frac{\partial W(x, p)}{\partial p} \cdot \Delta p]^2 \tag{25}$$

Esta última ecuación puede ser reescrita de forma matricial de la siguiente manera:

$$\sum_{x \in s_0} [E(x) + SD \cdot \Delta q]^2 \tag{26}$$

Donde E(x) coincide con la definición de la ecuación (23), por su parte SD se define de forma similar a la jacobiana del error:

$$SD = (-\nabla I(W(x; p)) \cdot \frac{\partial W(x; p)}{\partial p}, A_1(x), \ldots, A_m(x)) \tag{27}$$

Nótese que tanto el gradiente del warp de la imagen y la derivada de la función warp han sido particularizados para el valor de parámetros *p* actual (es decir, el obtenido en la iteración anterior)

Para encontrar los mínimos locales, se detiva con respecto a nuestras variables $\Delta q = (\Delta p, \Delta \lambda)^T$ y se iguala a cero, obteniendo:

$$2 \cdot \sum_{x \in s_0} SD^T \cdot [E(X) + SD \cdot \Delta q] = 0 \tag{28}$$

De donde se despeja el incremento de los parámetros:

$$\Delta q = -H^{-1} \cdot \sum_{x \in s_0} [SD^T \cdot E(x)] \tag{29}$$

Estando H definida como:

$$H = \sum_{c \in s_0} SD^T \cdot SD \tag{30}$$

En la Ilustración 15 se muestra un resumen del algoritmo de Lukas-Kanade.

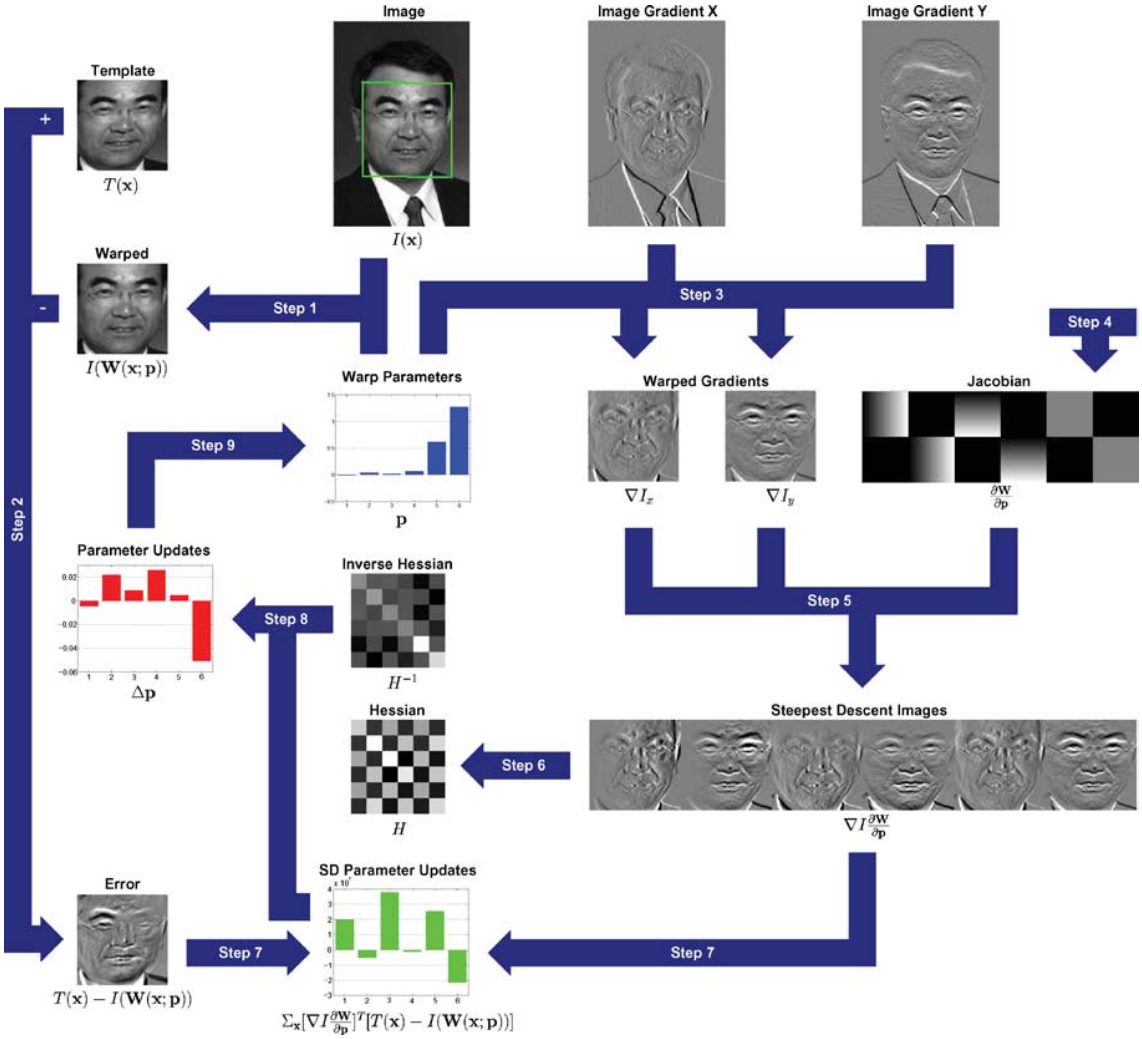


Ilustración 15: Resumen Algoritmo Lucas-Kanade

Resumen Algoritmo Lucas-Kanade

Iterar:

1) Calcular el warp de la imagen de entrada. Para obtener $E(x)=A(x)-I(W(x;p))$
2) Calcular la imagen de error. $I(W(x;p))$
3) Realizar el warp $W(x;p)$ del gradiente de I. $\nabla I(W(x;p))$.
4) Evaluar $E(x)=A(x)-I(W(x;p))$ en $(x;p)$.
5) Calcular la matriz de paso. Ecuación (27)
6) Calcular la matriz H usando la ecuación (30)
7) Calcular Δq . $\Delta q = -H^{-1}\cdot\sum_{x\in s_0}[SD^T\cdot E(x)]$.
8) Actualizar los parámetros. $q\leftarrow q+\Delta q$.

Hasta que $\Delta q<\varepsilon$

Este algoritmo es muy exacto y ampliamente utilizado en visión artificial para la alineación de dos imágenes, estando una de ellas parametrizadas. Por este motivo, será considerado en este proyecto como **límite teórico de precisión** de todos los algoritmos usados.

Sin embargo, el principal inconveniente de este algoritmo es el gran peso computacional de cada iteración. Como se puede apreciar en el extracto anterior del algoritmo, para cada iteración ya que se ha de recalcular el warp de la imagen, su gradiente y el Jacobiano de la función warp.

Además, se necesita mucho tiempo y capacidad de proceso para calcular la matriz SD y el Hessiano, en cada iteración. Todo esto hace a el Algoritmo de Lucas-Kanade inadecuado para la aplicación en tiempo real que se persigue.

Composicional inverso.

Como se ha visto hasta ahora, el algoritmo de alineación de imágenes de Lucas-Kanade utiliza demasiado tiempo y capacidad de cálculo al tener que recalcular tanto la matriz SD como el Hessiano para cada iteración.

El objetivo es hacer que las matrices SD y H sean constantes, para así poder precalcularlas reduciéndose así el tiempo en cada iteración. Para ello, este algoritmo realiza los siguientes cambios con respecto al algoritmo de Lucas-Kanade.

1. Nueva forma de actualización de los parámetros: $W(x;p) \leftarrow W(x;p) \circ W(x;\Delta p)^{-1}$
2. Cambio de roles entre I(x) y A(x).

Debido a los cambios anteriores, la ecuación (24), que era la expresión que se trataba de minimizar, será rescrita de la siguiente forma:

$$\sum_{x \in s_0} [A_0(W(x;\Delta p)) - I(W(x;p))]^2 \tag{31}$$

Tal y como ocurría en el algoritmo de Lucas-Kanade, se aplicará el teorema de Taylor en el entorno de p , es decir $\Delta p = 0$.

$$\sum_{x \in s_0} [A_0(W(x;0)) + \nabla A_0(W(x,0)) \cdot \frac{\partial W}{\partial p} \cdot \Delta p - I(W(x;p))]^2 \tag{32}$$

En este punto se debe notar que siempre se cumple la igualdad $W(x;0) = x$. Con lo que Δp , asumiendo que las ecuaciones (32) y (25) son prácticamente iguales, será:

$$\Delta p = -H_{ci}(x)^{-1} \cdot \sum_{x \in s_0} SD_{ci}(x)^T \cdot [A_0(x) - I(W(x;p))] \tag{33}$$

Donde SD_{ci} es:

$$SD_{ci}(x) = \nabla A_0(W(x,0)) \cdot \frac{\partial W(x;0)}{\partial p} \tag{34}$$

Y H_{ci} :

$$H_{ic}(X) = \sum_{x \in s_0} SD_{ci}(x)^T \cdot SD_{ci}(x) \tag{35}$$

Debido a que SD_{ci} siempre se evalúa para p=0, esta matriz y por extensión H_{ic} siempre van a ser constantes. Por lo que, es factible precalcularlas, tal y como se pretendía, con el consiguiente ahorro de tiempo de proceso en cada iteración.

La siguiente Ilustración muestra un resumen del algoritmo Composicional Inverso

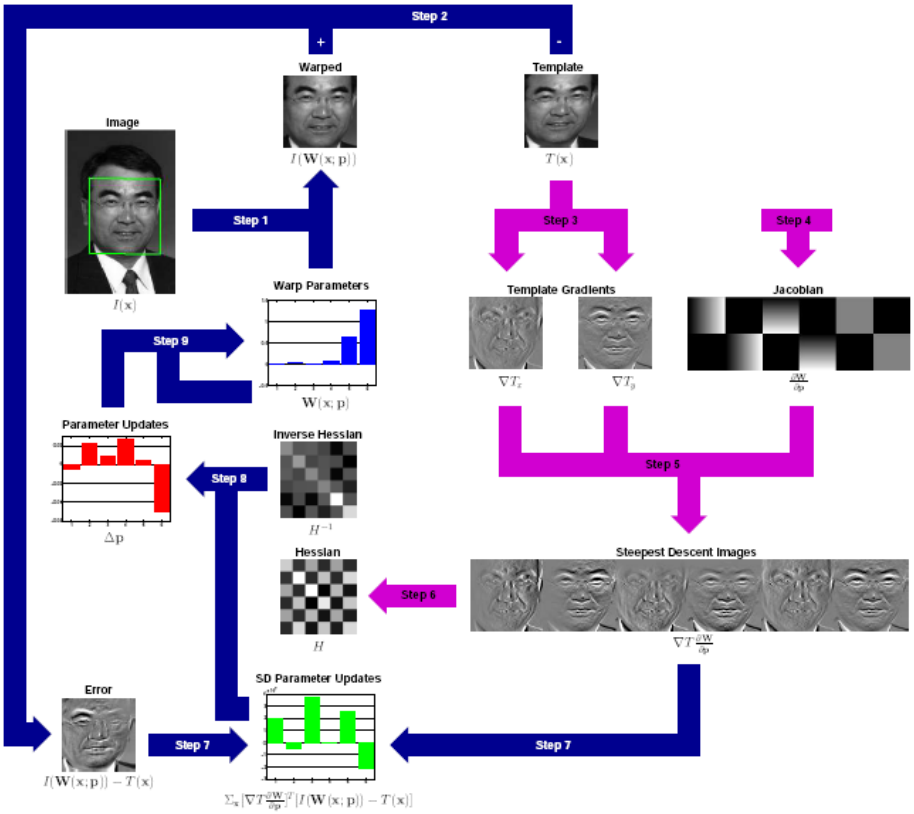


Ilustración 16: Resumen del Algoritmo Composicional Inverso

Resumen Algoritmo Composicional Inverso

Precalcular:

1) Calcular el gradiente A_0. ∇A_0
2) Calcular y evaluar el Jacobiano $\frac{\partial W}{\partial p}$ en $p = 0$.
3) Calcular la matriz de paso modificada usando la ecuación (34).
4) Calcular la matriz del Hessiano a partir de la matriz de paso modificada. Ecuación (35).

Iterar:

5) Realizar el warp de I con $W(x;p)$ para calcular $I(W(x;p))$.
6) Calcular la imagen de error. $E(x)=I(W(x;p))-A_0$
7) Hacer el producto escalar de la matriz de paso con la imagen de error.
8) Calcular Δp multiplicando el resultado de el paso anterior por la inversa del Hessiano.
9) Actualizar los parámetros: $W(x;p)\leftarrow W(x;p)\circ W(x;\Delta p)^{-1}$.

Nótese que en el anterior desarrollo no se ha tenido en cuenta variaciones de apariencia. Esto es debido a que la regla de actualización de la apariencia no cambia ($\lambda\leftarrow\Delta\lambda+\lambda$). Esto hace que al tratarse conjuntamente, tal y como se hace en el algoritmo de Lucas-Kanade, no se aprecien bien los beneficios del algoritmo de composicional inverso, siendo menos eficiente incluso que Lucas-Kanade [Baker et al, 2003b]

"Project Out" del composicional inverso

Debido al problema expresado anteriormente se deben buscar un algoritmo que prtmita combinar la rapidez del composicional inverso con el cálculo de los cambios en la base de apariencia.

El objetivo es minimizar la siguiente expresión que incluye los parámetros de apariencia:

$$\begin{aligned}\sum_{x\in s_0}[A(W(x;\Delta p))+\sum_{i=1}^{m}(\Delta\lambda_i\cdot A_i(W(x;\Delta p)))-I(W(x;p))]^2=\\ =\|A(W(x;\Delta p))+\sum_{i=1}^{m}(\Delta\lambda_i\cdot A_i(W(x;\Delta p)))-I(W(x;p))\|^2\end{aligned} \tag{36}$$

Este algoritmo propone expresar el módulo al cuadrado de la imagen de error ($A(x)-I(W(x;p))$) como la suma de los módulos al cuadrado de sus proyecciones en dos subespacios ortogonales: el generado por los vectores de apariencia A_i, y su ortogonal.

$$\|A(x)-I(W(x;p))\|^2=\|A(x)-I(W(x;p))\|^2_{span(A_i)}+\|A_0(x)-I(W(x;p))\|^2_{span(A_i)^\perp} \tag{37}$$

En este segundo subespacio (segundo término) el error es independiente de los parámetros de apariencia. La idea de este algoritmo es asumir que el error en el subespacio mencionado solo se debe a los parámetros de forma, de tal manera que serán calculados minimizando ese término de forma prácticamente idéntica a la expuesta en algoritmo original. Por otra parte, ya que ,en el primer término, se trabaja en el espacio generado por $A_{i,}$, para cualquier valor de p siempre existirá un conjunto de λ_i, para el que este segundo término se haga nulo. Así pues, para minimizar la expresión (36) basta con minimizar únicamente el primer término, que sólo depende de los parámetros p, para posteriormente calcular, si fuera necesario, los parámetros λ que anulan el segundo término.

Todo lo que se necesita hacer es proyectar SD_{ci} en el segundo subespacio vectorial ($SD_{po}=(SD_{ci})_{span(A_i)^\perp}$).

$$SD_{po}=(SD_{ci})_{span(A_i)^\perp}=SD_{ci}-\sum_{i=0}^{m}\left[\sum_{x\in s_0}A_i(x)\cdot SD_{ci}(x)\right]\cdot A_i(x) \tag{38}$$

Esto hará que no sea necesario siquiera proyectar el error, ya que al estar en producto escalar con , SD_{po} que sí que está en el espacio ortogonal al generado por los $A_{i,}$, el resultado también estará dentro del citado espacio. El resto de pasos son idénticos al algoritmo de composicional inverso original, con la salvedad que el cálculo del Hessiano se realiza con SD_{po} en lugar de con SD_{ci}

$$H_{po}(X)=\sum_{x\in s_0} SD_{po}(x)^T\cdot SD_{po}(x) \tag{39}$$

Una vez calculados los parámetros *p* iterativamente, se pueden calcular los parámetros de apariencia de forma muy sencilla mediante la siguiente ecuación:

$$\lambda_i=\sum_{x\in s_0} A_i(x)\cdot[A_0(x)-I(W(x;p))] \tag{40}$$

Resumen Algoritmo Composicional Inverso con “Proyect Out”

Precalcular:

1) Calcular el gradiente A_0. ∇A_0
2) Calcular y evaluar el Jacobiano $\frac{\partial W}{\partial p}$ en $p = 0$.
3) Calcular la matriz de paso modificada usando la ecuación (38).
4) Calcular la matriz del Hessiano a partir de la matriz de paso modificada. Ecuación (39).

Iterar:

5) Realizar el warp de I con $W(x;p)$ para calcular $I(W(x;p))$.
6) Calcular la imagen de error. $E(x)=I(W(x;p))-A_0$
7) Hacer el producto escalar de la matriz de paso con la imagen de error.
8) Calcular Δp multiplicando el resultado de el paso anterior por la inversa del Hessiano.
9) Actualizar los parámetros: $W(x;p)\leftarrow W(x;p)\circ W(x;\Delta p)^{-1}$.

Posteriormente:

10) Calcular los parámetros de apariencia usando la ecuación (40)

Este algoritmo, en principio, proporciona la suficiente simplicidad para que el tiempo de cómputo sea adecuado para una aplicación en tiempo real. Sin embargo se le aprecian algunos leves inconvenientes:

- Durante la iteración se desconoce la evolución de los parámetros de apariencia. Esto obliga a tener que prescindir de información importante para saber si el algoritmo esta convergiendo adecuadamente. Esto es importante en los casos que el algoritmo degenera y se quiere reinicializar el algoritmo (ver capítulo Viola & Jones, pág: 45).
- Este algoritmo tiene resultados 'pobres' al tratar de modelar ganancias y offset en la apariencia. Como se verá en el siguiente apartado.

Modelando Ganancia y Offset en la apariencia: Normalización del composicional inverso.

Una primera aproximación para modelar cambios de iluminación es utilizar un modelo de ganancia y de offset. Sin embargo , nótese que los cambios de iluminación no se traducen de forma lineal en la imagen, más aún en este caso en el que se trabaja con volúmenes complejos.

La forma más sencilla de modelar cambios en la iluminación es introducir en la base de apariencia un vector que coincida con la media de apariencia $A_1=A_0$, para modelar la ganancia, y un vector todo-unos por canal de color, para modelar el offset. De esta forma cualquier combinación de ganancia y offset podrá ser generada siendo su ganancia $(1+\lambda_1)$ y su offset $\lambda_{2..\,.\,.\,2+n\_canales\,-1}$ respectivamente para cada canal.

Sin embargo, se presenta el siguiente inconveniente:

Si la imagen de entrada coincide con la media de apariencia $I(x)=A_0(x)$, el cambio en los parámetros *p* que se debe producir, debería ser el mismo que si la imagen con la media de apariencia multiplicada por un factor de ganancia $I(x)=g\cdot A_0(x)$.

Sin embargo, el incremento de los parámetros calculado será, para el primer caso:

$$\begin{aligned}\Delta p &= -H_{po}(x)^{-1}\cdot\sum_{x\in s_0} SD_{po}(x)^T\cdot[A_0(x)-I(W(x;p))] = \\ &= H_{po}(x)^{-1}\cdot\sum_{x\in s_0} SD_{po}(x)^T\cdot I(W(x;p))\end{aligned} \tag{41}$$

Y para el segundo:

$$\begin{aligned}\Delta p &= -H_{po}(x)^{-1}\cdot\sum_{x\in s_0} SD_{po}(x)^T\cdot[A_0(x)-g\cdot I(W(x;p))] = \\ &= g\cdot H_{po}(x)^{-1}\cdot\sum_{x\in s_0} SD_{po}(x)^T\cdot I(W(x;p))\end{aligned} \tag{42}$$

Nótese que el término de A_0 desaparece debido a que es ortogonal a SD_{po}.

En definitiva, a pesar de que ambas imágenes de entrada debían converger en los mismos parámetros de forma, el algoritmo toma pasos más grandes o más pequeños dependiendo de la ganancia utilizada.

Una forma sencilla de solucionar esto es normalizando Δp con la ganancia de la imagen:

$$\gamma = 1 + \lambda_1 \tag{43}$$

El algoritmo que se ha considerado mejor para realizar estas labores de forma conjunta ha sido el de Normalización del composicional inverso, descrito en [Baker et al, 2003b]. En él se calcula los parámetros de apariencia dentro de la iteración, permitiéndo conocer λ_1, y así actualizar correctamente Δp. Además, se obtendrán el resto de parámetros de apariencia que servirán para conocer con mayor exactitud si el algoritmo está degenerando.

Para ello, en lugar de proyectar SD sobre el espacio ortogonal al generado a los vectores de la base de apariencia, como se proponía en el resultado anterior, se proyecta directamente el error. Además, se realiza la siguiente estimación.

$$\lambda_i = \sum_{x \in s_0} A_i(x) \cdot E(x) \tag{44}$$

Por lo que la normalización del error se puede expresar como:

$$E_{norm}(x) \leftarrow E(x) - \sum_{i}^{m} \lambda_i \cdot A_i(X) \tag{45}$$

Estos dos últimos pasos hacen crecer bastante la complejidad del algoritmo con respecto al “Project Out”, sobre todo en lo casos en que m>>n.

Así, con todo lo mencionado hasta ahora, la actualización de parámetros calculada con este algoritmo sería:

$$\Delta p = -\frac{1}{\gamma} \cdot H_{ci}(x)^{-1} \cdot \sum_{x \in s_0} SD_{ci}(x)^T \cdot E_{norm}(x) \tag{46}$$

El siguiente resumen trata de clarificar este algoritmo, que se parece en gran parte al anterior, y que ha sido el **definitivamente elegido para el desarrollo de este proyecto**.

Resumen Algoritmo Normalización del Composicional Inverso

Precalcular:

1) Calcular el gradiente A_0. ∇A_0
2) Calcular y evaluar el Jacobiano $\frac{\partial W}{\partial p}$ en $p = 0$.
3) Calcular la matriz de paso modificada usando la ecuación (34).
4) Calcular la matriz del Hessiano a partir de la matriz de paso modificada. Ecuación (35).

Iterar:

5) Realizar el warp de I con $W(x;p)$ para calcular $I(W(x;p))$.
6) Calcular la imagen de error. $E(x)=I(W(x;p))-A_0$
7) Estimar λ calcular $E_{norm}(x)$ usando la ecuaciones (44) y (45).
8) Hacer el producto escalar de la matriz de paso con la imagen de error.
9) Calcular Δp multiplicando el resultado de el paso anterior por la inversa del Hessiano.
10) Normalizar Δp dividiendo entre la ganancia de Apariencia (γ)
11) Actualizar los parámetros: $W(x;p) \leftarrow W(x;p) \circ W(x;\Delta p)^{-1}$.

Función warp

Definición e Implementación

A lo largo de este proyecto se ha mencionado en varias ocasiones la función de warp $W(x\,;p)$. Esta función definida a trozos permite relacionar cada punto en el interior de la malla s_0 con un punto en el interior de cualquier instancia de la malla s definida por los parámetros p. Como ya se mencionó en el apartado referido al entrenamiento (pág: 49) el propósito de emplear esta función es el de poder comparar la apariencia del modelo en un espacio común e independiente a los parámetros de forma.

Evidentemente, según la anterior definición, cada uno de los v puntos que forman la malla s_0 deben transformarse en su correspondiente en s mediante (8). El resto de puntos en el interior de la malla se encuentra situado dentro de un triángulo cuyos vértices son tres puntos de s_0: (x_i^0, y_i^0), (x_j^0, y_j^0) y (x_k^0, y_k^0). Al tratarse de una transformación afín, el resultado de calcular el warp de cada uno de esos puntos permanecerá en el interior del mismo triangulo, esta vez definido con tres puntos de s: (x_i, y_i), (x_j, y_j) y (x_k, y_k).

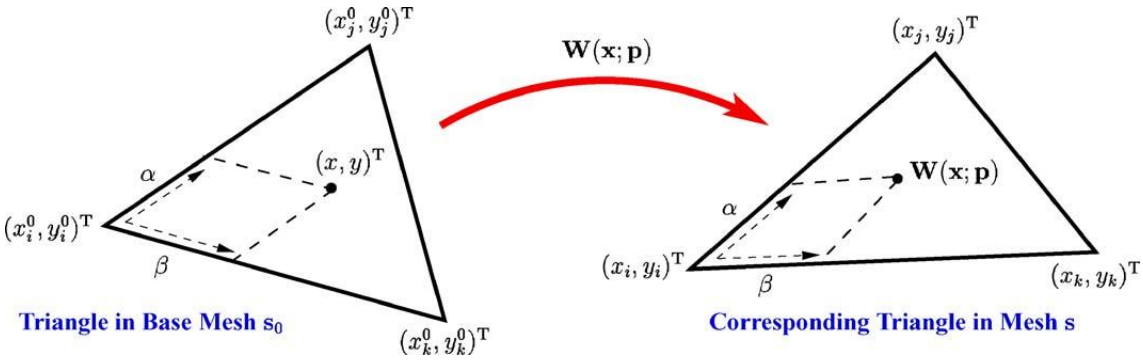


Ilustración 17: Función warp

Según lo mostrado en la Ilustración 17, cada píxel $\mathbf{x} = (x\,, y)$ en el triángulo (x_i^0, y_i^0), (x_j^0, y_j^0) y (x_k^0, y_k^0) puede ser expresado de la siguiente forma.

$$\boldsymbol{x} = (x\,, y) = (x_i^0, y_i^0) + \alpha\left[(x_j^0 + y_j^0) - (x_i^0 + y_i^0)\right] + \beta\left[(x_k^0 + y_k^0) - (x_i^0 + y_i^0)\right] \tag{47}$$

Donde:

$$\alpha = \frac{(x - x_i^0)(y_k^0 - y_i^0) - (y - y_i^0)(x_k^0 - x_i^0)}{(x_j^0 - x_i^0)(y_k^0 - y_i^0) - (y_j^0 - y_i^0)(x_k^0 - x_i^0)} \tag{48}$$

Y:

$$\beta = \frac{(y-y_i^0)(x_j^0-x_i^0)-(x-x_i^0)(y_j^0-y_i^0)}{(x_j^0-x_i^0)(y_k^0-y_i^0)-(y_j^0-y_i^0)(x_k^0-x_i^0)} \tag{49}$$

El resultado de aplicar la función warp será:

$$W(x\,;p) = (x_i, y_i)+\alpha\left[(x_j+y_j)-(x_i+y_i)\right]+\beta\left[(x_k+y_k)-(x_i+y_i)\right] \tag{50}$$

Sustituyendo en la ecuación (50) la expresiones de (49) y (48) se obtiene una ecuación con la siguiente forma:

$$W(x\,;p) = (a_1+a_2\cdot x+a_3\cdot y\,, a_4+a_5\cdot x+a_6\cdot y) \tag{51}$$

Es importante reseñar que en la expresión anterior, los parámetros a_i sólo deben ser calculados para cada triángulo y no para cada píxel.

En resumen, el proceso para el cálculo del warp quedaría de la siguiente manera:

1. Dado unos parámetros *p* calcular (x_i , y_i) para todos los vértices de *s.*
2. Calcular (a_1 , a_2 , a_3 , a_4 , a_5 , a_6) para cada triángulo.
3. Para cada píxel **x** buscar los valores (a_1 , a_2 , a_3 , a_4 , a_5 , a_6) correspondientes al triángulo que pertenecen.
4. Calcular W(x;p) utilizando (51).

En la implementación propuesta en este proyecto, se ha creado en el entrenamiento una tabla de búsqueda para poder efectuar, con mayor rapidez, la búsqueda del triángulo al que pertenece cada píxel

Derivación.

En todos los algoritmos de ajuste analizados en este proyecto es necesario el cálculo de la derivada de la función warp con respecto a los parámetros *p*. Sin embargo, la diferencia principal entre ambas metodologías radica en cuales son los parámetros *p* para los que se particulariza esa derivada.

Según lo explicado en el apartado anterior, el resultado de la función warp depende de los parámetros *p* que determinan la posición de los vértices de la malla *s*. Aplicando la regla de la cadena sobre W(x ;p) se obtiene:

$$\frac{\partial W}{\partial p} = \sum_{i=1}^{v}\left[\frac{\partial W}{\partial x_i}\frac{\partial x_i}{\partial p}+\frac{\partial W}{\partial y_i}\frac{\partial y_i}{\partial p}\right] \tag{52}$$

Los Jacobianos de la función con respecto a los vértices de *s*, $\frac{\partial W}{\partial x_i}$ y $\frac{\partial W}{\partial y_i}$, son fácilmente calculables a partir de la ecuación (50):

$$\frac{\partial W}{\partial x_i}=(1-\alpha-\beta, 0) \text{ y } \frac{\partial W}{\partial x_i}=(0,1-\alpha-\beta) \tag{53}$$

De lo anterior se puede deducir que $\frac{\partial W}{\partial x_i}$ y $\frac{\partial W}{\partial y_i}$ tienen valor máximo 1 en el vértice x_i y va reduciéndose según el píxel analizado se aleja de él y se acerca a los otros vértices del triángulo, los cuales van ganando protagonismo. El tamaño de estos jacobianos será el mismo que s_0.

La otra parte, la que se refiere a la variación de los vértices con respecto a los parámetros *p*, se obtiene directamente de la ecuación (8), siendo:

$$\frac{\partial x_i}{\partial p}=(s_1^{x_i}, s_2^{x_i}, \ldots, s_n^{x_i}) \text{ y } \frac{\partial y_i}{\partial p}=(s_1^{y_i}, s_2^{y_i}, \ldots, s_n^{y_i}) \tag{54}$$

Donde $s_j^{x_i}$ representa la componente de s_j que corresponde a x_i y de forma similar para y_i.

El resultado de calcular la ecuación (52) es una matriz con tantas filas como el doble de los píxeles de s_0 y el mismo número de columnas que de parámetros *p*. Como el resultado es constante para cualquier valor de *p* este método de cálculo de la derivada será el mismo para los dos algoritmos de 'fitting'. La anterior circunstancia, como se verá posteriormente, cambiará cuando se añada una transformación global para independizar la función warp del tamaño y/o la posición de la maya.

Inversa de la función warp:

En el composicional inverso y todos los demás algoritmos basados en él, se necesita el cálculo de la inversa del warp particularizado para un incremento de *p* pequeño $W(x;\Delta p)^{-1}$.

Partiendo del desarrollo de Taylor de primer orden de la función warp:

$$W(x;\Delta p)=W(x;0)+\frac{\partial W}{\partial p}\Delta p+O(\Delta p^2)=x+\frac{\partial W}{\partial p}\Delta p+O(\Delta p^2) \tag{55}$$

Se puede demostrar que $W(x;\Delta p)^{-1} \simeq W(x;-\Delta p)$ de la siguiente manera:

$$W(x;\Delta p)\circ W(x;-\Delta p) = x-\frac{\partial W}{\partial p}\Delta p+\frac{\partial W}{\partial p}\Delta p+O(\Delta p^2) = x+O(\Delta p^2) \tag{56}$$

Como se puede observar, la composición de ambas funciones resultan la función identidad. Nótese que los jacobianos no están evaluado en el mismo punto, sin embargo como están evaluado en un entorno de 0 y posteriormente multiplicado por Δp el error debido a esto se encuentra en $O(\Delta p^2)$.

Composición del warp:

En muchos algoritmos, el paso después de la inversión es la composición con el warp particularizado con los parámetros p para, a partir de ese resultado, actualizar esos parámetros $W(x;p)\leftarrow W(x;p)\circ W(x;\Delta p)^{-1}$.

El primer paso de la composición es calcular $W(x;\Delta p)^{-1}$ que representa pequeños cambios en la malla base. Según lo visto en el apartado anterior, se podrán calcular los cambios en la malla base producidos por $W(x;\Delta p)^{-1} \simeq W(x;-\Delta p)$ de la siguiente manera:

$$\Delta s_0 = (\Delta x_1^0, \Delta y_1^0, \ldots, \Delta y_v^0, \Delta y_v^0) = -\sum_{i=1}^{n} \Delta p_i \cdot s_i \tag{57}$$

El paso posterior, es utilizar el resultado anterior como entrada de la funcion warp $W(x;p)$ que relaciona la malla s_0 con s. Para ello se actúa de la misma manera que se hizo con la implementación del warp. Véase la siguiente Ilustración.

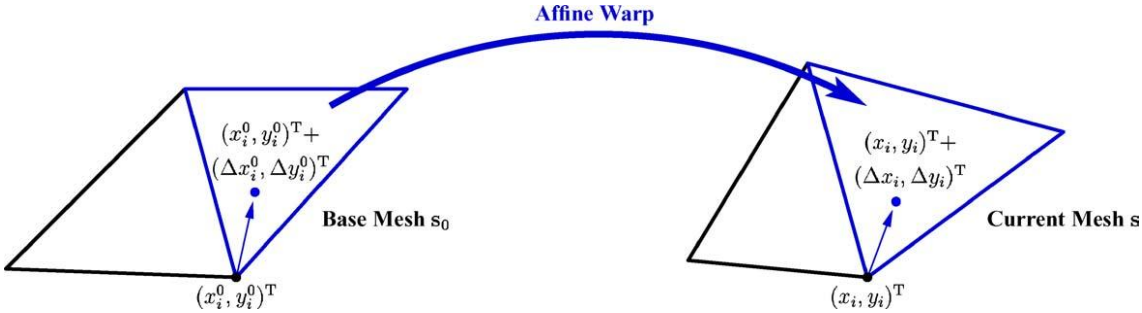


Ilustración 18: Composición warp

Sin embargo, si se calcula el warp utilizando únicamente el triángulo sobre el que cae $(x_i^0, y_i^0)+(\Delta x_i^0, \Delta y_i^0)$ es posible encontrar problemas cuando algún punto caiga fuera de s_0. En este punto se presentan varias opciones para la elección del triángulo, ya que dependiendo del triángulo utilizado el resultado obtenido variará.

En lugar de tomar un único triangulo se calcula el promedio del resultado obtenido para cada uno de los triángulos que comparte el vértice en cuestión. Esto hará que el resultado sea mucho más suave.

Una vez que se tiene la nueva posición de los puntos de s ($s' = s + \Delta s$) a partir del resultado de la composición anterior, suponiendo que s_i son ortogonales (condición que se cumple al haber utilizado PCA en el entrenamiento, y que de otra forma sería fácil de cumplir ortogonalizando la base con Grand-Smith), la actualización de los parámetros p se realiza mediante la resolución de la ecuación (8):

$$p'_i = s_i \cdot (s + \Delta s - so) \tag{58}$$

"Global Shape Normalising Transform"

En este punto se presenta un método para tener en cuenta el tamaño, posición o giro de la cara de manera independiente a los parámetros de forma y apariencia del AAM. Éste método admite dos soluciones:

1. Entrenando con imágenes de distintos tamaños y en distintas posiciones y realizar el entrenamiento mediante PCA.
2. Normalizar todas las imágenes del entrenamiento de tal forma que todas tengan el mismo tamaño, posición y orientación. Y posteriormente añadir una tranformación global que se encargue de modificar dichos aspectos.

La primera opción, mas sencilla ya que no conlleva añadir nada más al algoritmo, tiene como inconveniente el que la información de tamaño, posición, etc, se mezclaría con la relativa a los parámetros de forma. La mayor desventaja que esto presenta es a la hora de clasificar el gesto realizado en función de los parámetros. Ya que se obtendrían parámetros p diferentes ante la misma cara (con la misma forma) si esta tiene tamaños distintos y/o está rotada. Además, cabe la posibilidad de sobreentrenar el modelo haciendo que la dimensión de esta crezca haciéndolo poco útil.

Con la segunda opción se consigue independencia entre los parámetros de forma y los 4 nuevos parámetros que caracterizarán la transformación global. De esta forma, siguiendo con el ejemplo anterior, la misma cara con distinto tamaño únicamente hará variar en esos cuatro nuevos parámetros, permaneciendo los parámetros de forma iguales.

Para poder implementar esta ultima solución, tal y como ya se mencionó en el capítulo en el que se describian los AAM's, primeramente es necesario realizar una normalización en el entrenamiento, con el fin de eliminar toda información relativa a translación y escala, acompañado de un análisis de procrustes (ver pág 41) con el que se elimina toda aquella información relativa a la rotación de la cabeza.

Debido a que en el entrenamiento se ha eliminado toda información de tamaño, translación y giro, es necesario crear una nueva transformación paramétrica que sea capaz de generar estas dimensiones a partir del resultado entregado por el warp. Además es necesario que los parámetros que caracterizan a esta transformación puedan ser incluidos en la función de minimización.

Parametrizando la Transformación global

Si se denomina como N(x;q) a la transformación global para el AAM, se puede considerar un buen ejemplo de N(x;q) un conjunto de "Similarity Transforms":

$$N(x;q)=\begin{pmatrix}(1+a) & -b\\ b & (1+a)\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}+\begin{pmatrix}t_x\\ t_y\end{pmatrix} \tag{59}$$

Donde los parámetros (a, b , t_x, t_y) tienen la siguiente interpretación.

- Los parámetros a y b proporcionan la capacidad de escalar A y rotar θ la malla: $a = k\cos(\theta)-1$ y $b = k\sin(\theta)-1$.
- Los parámetros t_x y t_y permiten traslaciones en x e y respectivamente.

La ecuación (59) representa una parametrización tal que la transformación identidad (la que no realiza ningún cambio) se obtiene cuando el valor de todos los parámetros es 0.

La expresión (59) es sólo una manera de parametrizar la transformación global. Otra forma de parametrizar la transformación es definir un subconjunto especial de warps parecido al explicado hasta ahora usado para AAM's.

Si nuestra malla base es $s_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)$, tomando $s_1^* = s_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)$, $s_2^* = (-y_1^0, x_1^0, \ldots, -y_v^0, x_v^0)$, $s_3^* = (1,0,\ldots,1,0)$ y $s_4^* = (0,1,\ldots,0,1)$, entonces el conjunto de variaciones lineales de la malla AAM permitido es exáctamente el mismo que la transformación expresada en la expresión (59).

$$N(x;q) = s_0 + \sum_{i=1}^{4} q_i \cdot s_i^* \tag{60}$$

Donde los parámetros son $\boldsymbol{q} = (q_1, q_2, q_3, q_4)$ tienen relación directa con (a, b , t_x, t_y)

$$\begin{aligned} a &= q_1 \quad t_x = q_3 \\ b &= q_2 \quad t_y = q_4 \end{aligned} \tag{61}$$

La ventaja de utilizar la notación de (60) es que, al ser la misma notación que la utilizada para definir W(x;p) en (8), por lo que se podrá reutilizar todo el análisis del apartado “Función warp“ en este mismo capítulo.

Para que formen una base ortonormal, es importante que los vectores s_1*,s_2*,s_3* y s_4* sean ortogonales. Esta condición se cumple automáticamente si en el entrenamiento, se ha quitado la translación en todas las imágenes, ya que por su definición s_1* y s_2* siempre serán ortogonales. El único cambio que se produce al ser normalizados es que hay que ponderar las relaciones de los parámetros q con (a, b , t_x, t_y) expresadas en (61)

Dado N(x;q), se debe actualizar la definición de instanciación del modelo mostrada en (10) añadiendo esta transformación de la siguiente manera:

$$M(N(W(x;p);q)) = A(x) \quad \forall\, x \in s_0 \tag{62}$$

Donde M es la instanciación del modelo con su correspondiente tamaño y forma y A(x) está definida en s_0 . La instancia M(x) es creada mediante la transformación warp W(x; p) de A(x) y posteriormente la normalización mediante el warp N(x; q)

Nótese que la composición en la nueva definición no es igual a añadir los nuevos vectores s_1*,s_2*,s_3* y s_4* a los vectores originales de la base de forma s_1, … ,s_n para aumentarla, ya que de esta forma los cambios de tamaño y rotación no serían globales a todos los vectores de la base sino solamente a s_0.

Se asumirá que s_i y s_i* son ortogonales, esta condición se cumple si en el entrenamiento se ha normalizado y empleado Procrustes correctamente en cada imagen, ya que al realizar estas operaciones en realidad se está proyectando todos los vectores de entrenamiento a un espacio ortogonal a s_i*. No obstante, en la práctica no siempre se consigue que sean totalmente ortogonales, esto conllevaría un pequeño error en la operación de composición. Para evitarlo, se ortogonalizará conjuntamente s_i y s_i*.

Implementación

Para llevar a cabo el cálculo de I(N(W(x; p); q)) tal como se hacía con la función warp, se considerará una única función N o W (x; q,p) que se tratará de forma muy similar a como se hacía con W(x; p).

El primer paso es encontrar el destino de los vértices de la malla s, una vez encontrados el procedimiento es el mismo que el explicado en la Ilustración 17.

Para encontrar los puntos de *s* primero se calculará el destino de los vértices de s_0 al utilizar $W(s_{0,} p)= s' = s_0 + \sum_{i=1}^{n} p_i \cdot s_i$.

Seguidamente, se calculará la transformación global de la malla base con $N(s_{0,} q)= s'_0 = s_0 + \sum_{i=1}^{n} q_i \cdot s_i^*$.

Una vez obtenidos s' y s'_0 se puede hallar s de forma muy similar a como se muestra en la Ilustración 19, de la misma manera que se hacía con el warp.

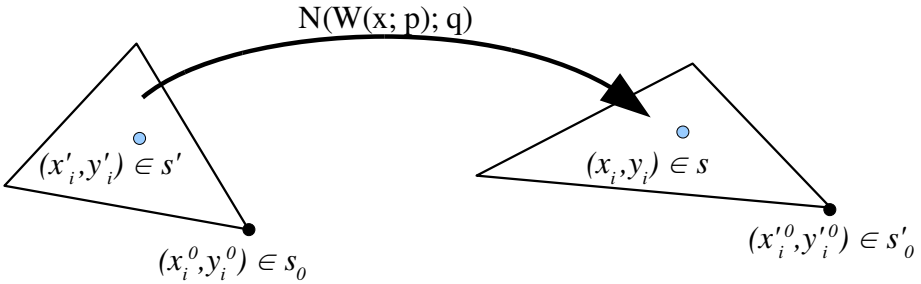


Ilustración 19: Implentación de N(W(x;p)q)

Derivación.

El jacobiano de la función N o W es $(\frac{\partial}{\partial q} N \circ W, \frac{\partial}{\partial p} N \circ W)$. Sin embargo, se deberá tener en cuenta para qué valor de parámetros *p* y *q* se está particularizando. A continuación, se verá que

según el algoritmo elegido se podrá o no simplificar el cálculo del Jacobiano lo que se traducirá en una reducción del tiempo de cómputo.

Algoritmo de Lukas-Kanade

Según lo estudiado en el correspondiente apartado (ver pág: 56), en este algoritmo, el Jacobiano de la función warp debe de ser recalculado en cada iteración y particularizado para el valor de los parámetros en ese momento.

Para el cálculo de el jacobiano en esta sección se utilizará de la notación matricial.

Según lo visto hasta ahora, la función N o W puede ser expresada de la siguiente manera:

$$N \circ W = \begin{pmatrix} (1+q_1) & -q_2 \\ q_2 & (1+q_1) \end{pmatrix} \cdot W(x;p) + \begin{pmatrix} q_3 \\ q_4 \end{pmatrix} \tag{63}$$

Esta expresión resulta fácil de derivar:

$$J(N \circ W)_{2x(n+4)} = \left(W(x;p), \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot W(x;p), \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} (1+q_1) & -q_2 \\ q_2 & (1+q_1) \end{pmatrix} \cdot \frac{\partial W(x;p)}{\partial q} \right) \tag{64}$$

El resultado es una matriz con 2 filas y tantas columnas como la suma del número de parámetros p y q. Para calcular $\frac{\partial W}{\partial p}$ se utilizará los resultados obtenidos en el apartado de derivación de la función warp.

En la expresión (64) no se ha tenido en cuenta el mencionado factor de escala que aparece al normalizar los vectores s_i*.

Algoritmos basados en el Composicional Inverso.

El composicional inverso y todos los algoritmos basados en él mejoran su rendimiento gracias a que no tienen que recalcular en cada iteración ni la matriz de paso ni el Hessiano. Esto se debe a que el Jabiano se particulariza para $p=q=0$.

Como $W(x;0)=N(x;0)=x$, Al calcular el jacobiano se obtiene:

$$\left(\frac{\partial}{\partial q} N \circ W = \frac{\partial}{\partial q} N\right) \tag{65}$$

Y:

$$(\frac{\partial}{\partial p}N \circ W, \frac{\partial}{\partial p}W) \tag{66}$$

Como consecuencia de que la representación de W es la misma que la de N en (60), el cáclulo del jacobiano $(\frac{\partial}{\partial q}N \circ W, \frac{\partial}{\partial p}N \circ W)=(\frac{\partial}{\partial q}N, \frac{\partial}{\partial p}W)$ se realiza de la misma manera que la explicada en el apartado de derívación de la función warp (ver pág: 69), con la salvedad que para el cálculo de $\frac{\partial}{\partial q}N$ se usará s_i* en lugar de s_i.

Función Inversa:

En la correspondiente sección de la función warp se demuestra que $W(x;\Delta p)^{-1} \simeq W(x;-\Delta p)$.

Si en el razonamiento empleado en dicha sección se remplaza W por N o W y los parámetros Δp por (Δp, Δq) se llega, fácilmente, a la siguiente conclusión:

$$N \circ W(x;\Delta p, \Delta q)^{-1} \simeq N \circ W(x;-\Delta p, -\Delta q) \tag{67}$$

Composición:

El propósito de la composición es doble: calcular la nueva posición de los vertices de la malla y recuperar los parámetros *p* y *q* que generan dicha malla.

El primer paso de los mencionados se lleva a cabo calculando el destino de s_0 bajo la función:

$$(N \circ W)(x;q,p) \circ (N \circ W)(x;\Delta q, \Delta p)^{-1} \approx (N \circ W)(x;q,p) \circ (N \circ W)(x;-\Delta q, -\Delta p) = s \tag{68}$$

La composición de $(N \circ W)(x;-\Delta q, -\Delta p)$ con $(N \circ W)(x;q,p)$ se realizará de forma muy similar a la composición de *W* con *N* explicada anteriormente:

En primer lugar, se calcula de forma separada $(N \circ W)(x;-\Delta q, -\Delta p)=s'$ y $(N \circ W)(x;q,p)=s'_0$ usando para ello el método explicado en el apartado de *Implementación,* posteriormente se realiza la operación de warp mostrada en la Ilustración 19.

Una vez encontrado el nuevo destino de los puntos de la malla, se debe encontrar el nuevo conjunto de parámetros *p* y *q* que consiguen que $(N \circ W)(s_0;q,p)=s^{\dagger}$.

Generalmente, resolver esta ecuación no es un problema lineal. Sin embargo para la representación matricial de la transformación N elegida, el problema puede ser solucionado de una manera bastante sencilla.

Sea:

$$(N \circ W)(s_0; q, p) = N(s_0 + \sum_{i=1}^{n} p_i s_i; q) \tag{69}$$

Si se expresa *N* de forma matricial tal y como en (59), la expresión anterior es igual a:

$$N(s_{0,} q) + \begin{pmatrix} (1+a) & -b \\ b & (1+a) \end{pmatrix} \cdot \sum_{i=1}^{n} p_i s_i \tag{70}$$

En esta expresión se comete un abuso de notación, ya que la matriz 2x2 multiplica a cada vector 2x1 formado por las coordenadas (x,y) de cada vértice. La expresión (70) se puede reescribir de la siguiente manera:

$$N(s_{0,} q) + \left[(1+a) \sum_{i=1}^{n} p_i s_i \right] + \left[b \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \sum_{i=1}^{n} p_i s_i \right] \tag{71}$$

En esta ecuación, el segundo término es ortogonal a s_i^* ya que todos los vectores s_i lo son. De la misma manera, el tercer término también es ortogonal debido a que cualquier vector en el subespacio generado por s_i^* si se intercambia x por y y se la cambia el signo a uno ellos, el resultado no pertenecerá al subespacio generado por s_i^*.

Sabiendo que $N(s_{0,} q) = s_0 + \sum_{i=1}^{n} q_i \cdot s_i^*$:

$$s_0 + \sum_{i=1}^{4} q_i s_i^* + \left[(1+a) \sum_{i=1}^{n} p_i s_i \right] + \left[b \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \sum_{i=1}^{n} p_i s_i \right] = s^{\dagger} \tag{72}$$

Los parámetros *q* que cumple dicha ecuación seran:

$$q = s_i^* \cdot (s^{\dagger} - s_0) \tag{73}$$

Una vez conocidos los parámetros *q*, se puede hallar *p* de la siguiente manera:

$$p_i = s_i \cdot (N(s^{\dagger}, q)^{-1} - s_0) \tag{74}$$

5.- Implementación

Introducción

En este capítulo se abordará la implementación práctica de los algoritmos estudiados hasta el momento en un lenguaje de programación estándar.

El lenguaje elegido para llevar a cabo la implementación es C.

C es un lenguaje ampliamente utilizado en todos los ámbitos y especialmente en entornos UNIX. Entre sus ventajas destacan su eficiencia, algo que es fundamental si quieren alcanzar condiciones de tiempo real; la gran cantidad de bibliotecas, documentación y programas existentes y la existencia de excelentes compiladores bien documentados, como GCC, que permiten optimizaciones que repercuten en el tiempo de cómputo final.

Entre sus principales inconvenientes está el que el manejo de memoria no se realiza automáticamente sino que debe de ser explícitamente programado. Esta circunstancia hace que se deba dedicar más tiempo a la programación y que halla más posibilidad de cometer errores.

Entre el amplio abanico de librerías existente, merece especial mención por su utilización en este trabajo las librerías OpenCV , una librería multiplataforma de visión artificial.

Para finalizar, se pondrá a prueba los programas realizados, se analizará su rendimiento y se tratará de sacar conclusiones y dar explicaciones a lo ocurrido en la medida de lo posible.

OpenCV

OpenCV es una librería de visión artificial desarrollada originalmente por Intel en 1999. Se distribuye gratuitamente bajo licencia BSD, esto quiere decir que permite su uso y modificación, incluso para fines comerciales, aunque el resultado no sea software libre. El uso de una licencia tan poco restrictiva convierte a OpenCV, prácticamente, en una librería de dominio público.

Además, OpenCV es multiplataforma (Linux, Windows y MAC) lo que garantiza una rápida portabilidad del código entre los distintos sistemas operativos.

La versión utilizada en este desarrollo ha sido la 1.1 que utiliza en su mayoría el lenguaje de programación C. Sin embargo, en el transcurso del desarrollo de este Proyecto Final de carrera han sido lanzadas las versiones 2.0 y 2.1 que aportan nuevas funcionalidades y el uso de C++.

La principal ventaja de OpenCV es que provee una infraestructura simple de usar como ayuda para construir programas complejos basados en visión artificial. Esta constituido por más de 500 funciones que recorren diversas áreas de la visión artificial. En complemento, OpenCV también desarrolla algunas funciones de aprendizaje automático e inteligencia artificial, ya que las tareas visión artificial frecuentemente van ligadas a estos conceptos. Además sus funciones han sido optimizadas de cara al uso en condiciones de tiempo real.

Visión general

En este Apartado se enumeran los principales programas creados en este proyecto para desarrollar un algoritmo AAM, su funcionalidad y la interacción entre ellos. En los apartados posteriores se tratarán mas a fondo cada uno de ellos.

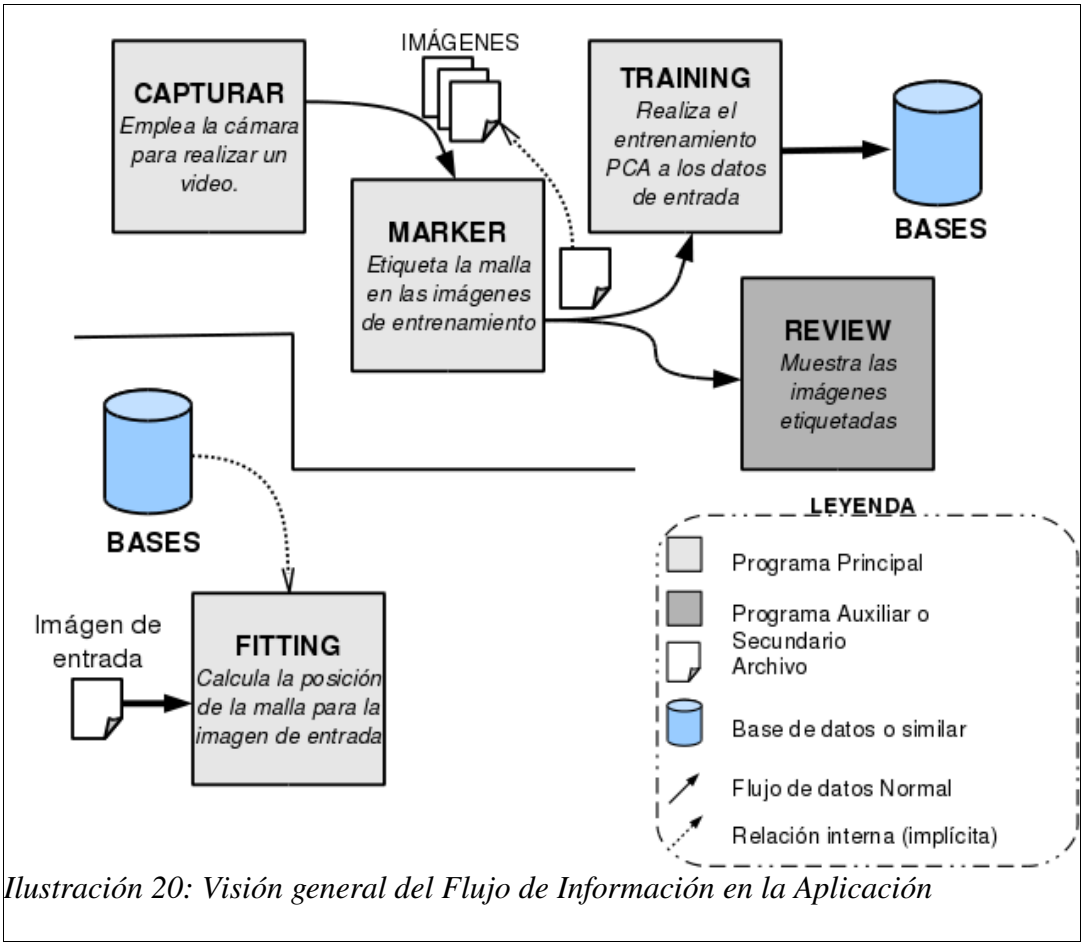


Ilustración 20: Visión general del Flujo de Información en la Aplicación

En la anterior ilustración se muestra un diagrama en el cual se trata de representar el flujo de información de nuestra aplicación.

El proceso comienza con la captura de datos del entrenamiento, en este paso el usuario tendrá que posar delante de la cámara realizando los gestos que se desean sean aprendidos.

Posteriormente se seleccionará aquellas imágenes más significativas de la secuencia tomada en el paso anterior, y se procederá a su marcado. En este paso se tendrán que marcar ***manualmente*** las imágenes seleccionadas.

Una vez marcadas un conjunto de imágenes lo suficientemente extenso se pasará ejecutar el entrenamiento ('training'). Este paso dará como resultado una serie de ficheros los cuales contienes información necesaria para ella realización del fitting tales como las bases de forma y apariencia.

Con todo ello se dispone de toda la información necesaria para poder realizar el 'fitting'. Como se vio en el correspondiente capítulo, existen varias posibilidades para realizar este algoritmo. El archivo de entrada a el programa de 'fitting' siempre es una imagen, si bien esta puede ser una archivo aislado o bien proceder de un flujo como una cámara.

Programa 'CAPTURAR'

Ilustración 21: Captura del programa 'CAPTURAR' funcionando

Este programa recibe por la linea de comandos el nombre de la persona de la que se quiere tomar el conjunto de imágenes. Se crea una carpeta con este nombre donde se guardarán secuencialmente las imágenes tomadas por la cámara nombradas con el nombre introducido seguido por el número de secuencia.

Obviamente las sentencias fundamentales en este programa son las que se encargan de operar con la cámara:

```
CvCapture* capture = cvCreateCameraCapture(0);
```

(capturar.c)

La anterior instrucción inicializa la estructura CvCapture de la cámara definida por el número que se pasa como parámetro. La estructura CvCapture contiene información relativa a la captura del vídeo, es decir un manejador.

Otra instrucción de importancia en el programa es aquella que toma del buffer un frame y lo devuelve descomprimido como una imagen:

```
img=cvQueryFrame(capture);
```

(capturar.c)

Programa 'MARKER'

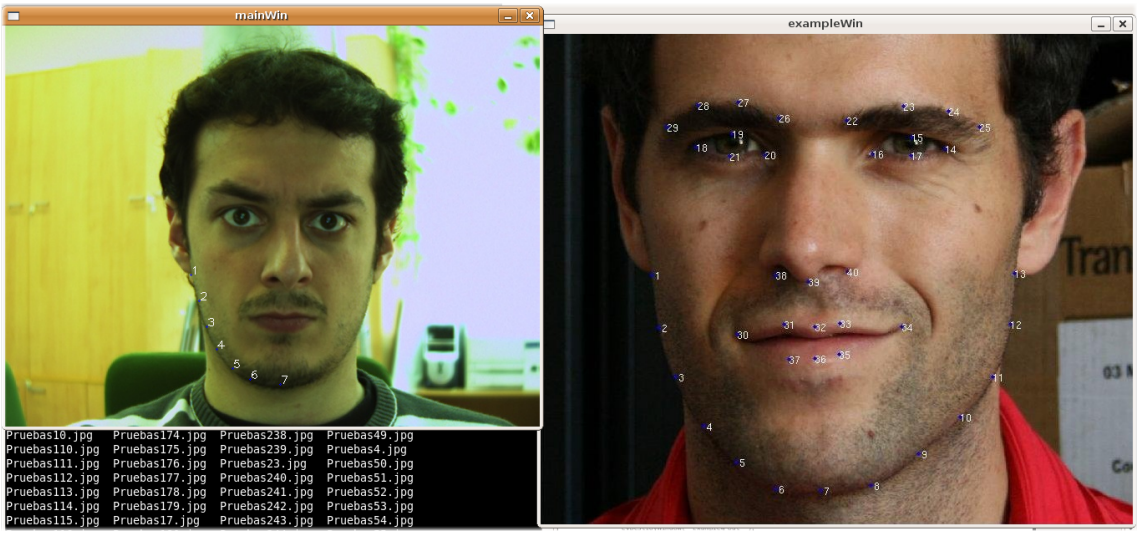


Ilustración 22: Captura del programa 'MARKER' funcionando

El programa 'Marker' se encarga de asociar una imagen con una malla de la que se introducen manualmente por el usuario sus vértices. A modo de orientación para el correcto marcado de la malla, se muestra una imagen que previamente ha sido marcada correctamente.

Los parámetros introducidos por linea de comando a esta función son en primer lugar la imagen a etiquetar y en segundo lugar el archivo de marcado al que se le quiere añadir, si este no existe se creará. Ambos parámetros son necesarios.

Por su parte el manejo es bastante intuitivo. Basta con ir hacer 'clic' en la posición que se desea, el marcado se realiza de forma secuencial siguiendo el orden mostrado en la imagen de ejemplo. Si se realiza 'clic' con el botón derecho del ratón el último punto señalado se borra. Una vez marcados todos los puntos se debe presionar una tecla para confirma el marcado.

La estructura creada para guardar la información obtenida es la siguiente.

```
struct PuntosCara
{
        char ImageName[MAX_NAME];
        int X[NUM_TOTAL_PUNTOS];
        int Y[NUM_TOTAL_PUNTOS];
};
```

(faces.h)

La imagen se graba por referencia, es decir, es necesario conservar la imagen utilizada en la misma ubicación en el sistema de ficheros al menos hasta que el entrenamiento sea ejecutado. El valor de X e Y no es más que la posición del vértice de la malla expresada en píxeles.

Programa 'TRAINING'

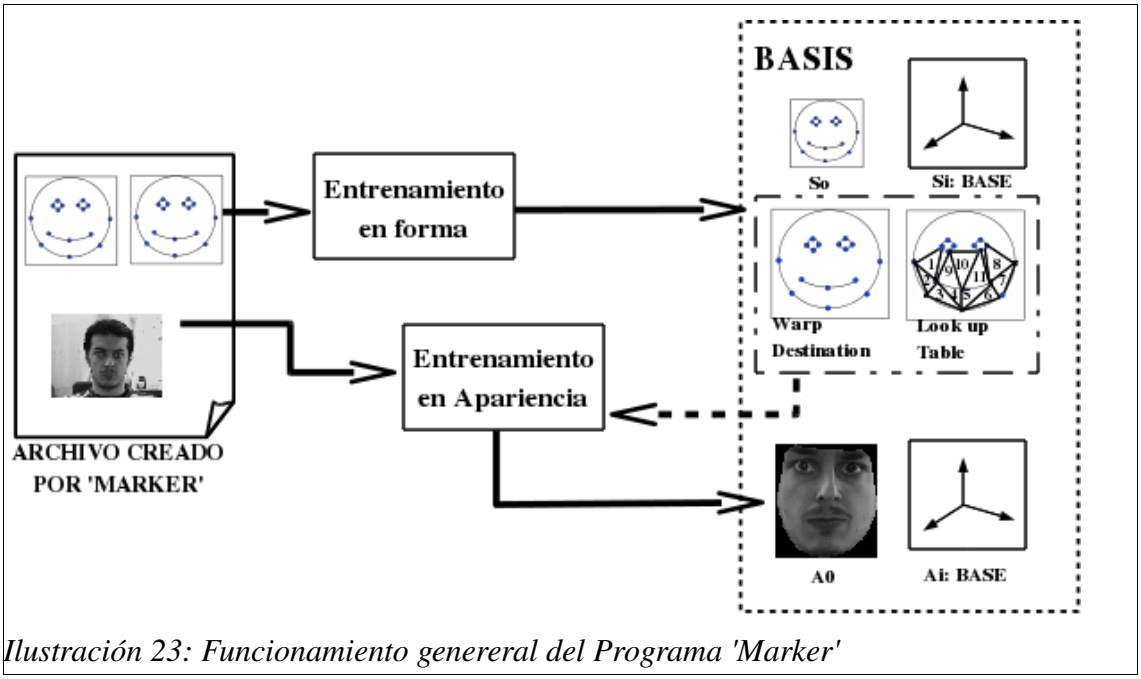


Ilustración 23: Funcionamiento genereral del Programa 'Marker'

Este programa trata de, tal y como se explicó en el capítulo 3, obtener unas bases de forma y apariencia necesarias para representar el modelo, y de esta forma poder realizar, según lo visto el fitting.

El punto de partida de la aplicación es el archivo generado por el programa explicado en la sección anterior. A partir de él, primeramente se realiza el entrenamiento de forma y se obtiene, mediante PCA, su correspondiente base y media.

Antes de comenzar el entrenamiento en apariencia, es necesario definir un destino para la función warp, donde se creará el espacio de apariencias. Normalmente ese destino suele coincidir con la media de la forma, sin embargo, en el caso de este proyecto, por razones que posteriormente se analizarán, esto no es posible. Además, para ahorrar tiempo de ejecución cada vez que se realiza el warp, se creará una tabla que relacione cada píxel de la malla de destino del warp con el triángulo al que pertenecen, de esta forma se podrá saber que coeficientes utilizar en la ecuación (51).

Una vez definido lo explicado en el párrafo anterior (En la Ilustración 23 “Warp Destintation” y “Look up Table”) se inicia el entrenamiento en Apariencia, basado también en PCA, obteniendo la base y la media de apariencia.

En los siguientes apartados se profuncizará más en cada una de las partes de entrenamiento

Entrenamiento en forma

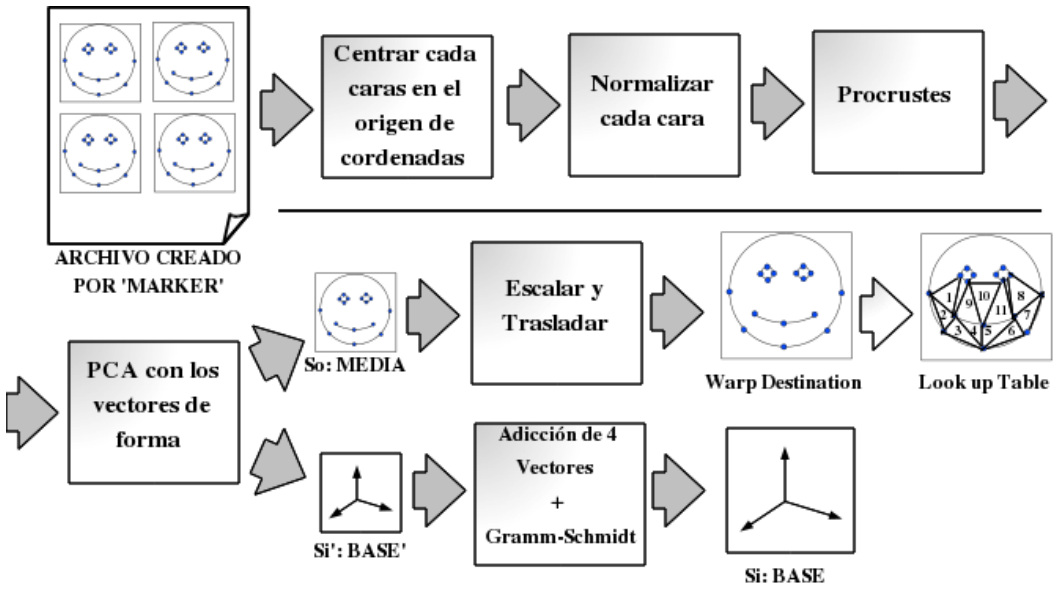


Ilustración 24: Implementación del entrenamiento en forma

En la primera parte del entrenamiento, se quita toda aquella información relativa al desplazamiento tamaño y rotación de la cara en la imagen, ya que esta información va a poder ser regenerada por la transformación global mencionada en el capítulo 4 (pág 73). Los tres primeros pasos en la Ilustración 24, encargados de eliminar la información menciada, se corresponden con el algoritmo de Procrustes generalizado, resumido en la página 44 de este documento.

Posteriormente se realiza un análisis PCA como el explicado en el correspondiente capítulo de este proyecto. Para ello el vector referido en la ecuación (1) será un vector de tantas dimensiones como el doble de vértices de la malla. Par realizar este paso, se ha utilizado la función de OpenCV destinada a PCA:

```
void cvCalcPCA (const CvArr* data, CvArr* avg, CvArr* eigenvalues, CvArr* eigen-
vectors,
int flags )
```

Esta función dada la matriz de datos *data* pasada como referencia, devuelve por referencia:

- La vector con la media de los datos pasados.
- Una matriz con los autovectores de la matriz de correlación.
- Un vector con los autovalores asociados a cada uno de los autovectores.

Tal y como se estudió en el capítulo de entrenamiento, se desea discriminar el número de autovectores que forman la base en función de la cantidad de 'energía' que contienen. Esto es posible

realizarlo atendiendo al valor al cuadrado del autovalor asociado a ese vector. El procedimiento es el descrito en el capítulo de entrenamiento.

Concluido el filtrado de los vectores que formarán la base se obtiene la base de forma y un vector malla media. Si no se utilizase una transformación global, el trabajo habría terminado aquí ya que se podría utilizar el vector media como destino del warp.

Sin embargo, al haber realizado la normalización explicada, se obtiene un vector media (s0) demasiado pequeño (recuérdese que $\sum X^2+Y^2=1$) y, ya que esta centrado en 0, tiene vértices con X, Y o de ambos menores que 0. OpenCV no permite utilizar el vector s0 como destino del warp ya que no es posible una representación subpixélica ni tampoco representar posiciones de píxeles negativas. Para solucionar este problema existen dos opciones:

- Escoger un destino aleatorio.
- Ampliar y trasladar el vector media lo suficiente para que pueda ser utilizado.

Es este proyecto se ha escogido por la segunda opción. En un principio los valores de traslación y amplitud son aleatorios, no obstante es necesario llega a una solución de compromiso ya que:

- Una malla de destino muy grande hará que el el procesado sea muy lento.
- Una destino muy pequeño, hará que perdamos mucha información de Apariencia perdiendo exactitud.

Finalmente, se ha considerado utilizar un valor de norma de 350 (definida como AP_NORM en warp.h) y una traslaciones horizontal y vertical de 100 píxeles (AP_TX y AP_TY)

Una vez obtenido la malla destino, es interesante crear una tabla de búsqueda que relacione los puntos en el interior de la malla con el triangulo que corresponden. Con esto se ahorra tiempo de ejecución cada vez que sea necesario realizar warp a una imagen.

En el desarrollo esta tabla no es más que una imagen cuyos píxeles toman el valor 0 si están fuera de la maya y el número del triangulo al que pertenecen si los píxeles se encuentran en el interior de la malla. Todo esto ha sido implementado en la función *'CreateLUT'*.

El último paso antes de acabar con esta fase del entrenamiento es la de añadir los 4 vectores de la base necesarios para la transformación global: $s_1^* = s_0 = (x_1^0, y_1^0, \ldots, x_v^0, y_v^0)$, $s_2^* = (-y_1^0, x_1^0, \ldots, -y_v^0, x_v^0)$, $s_3^* = (1,0,\ldots,1,0)$ y $s_4^* = (0,1,\ldots,0,1)$. Para asegurarnos que los vectores son ortogonales a los que ya se tenían, se realiza Gram-Schmidt con el conjunto de todos los vectores.

Todos los elementos creados en este paso, son guardados en respectivos ficheros, para que puedan ser recuperados por la función de fitting.

Entrenamiento de apariencia.

Una vez realizado el entrenamiento en forma, se dispone de todo lo necesario para poder realizar el warp de cualquier imagen.

La apariencia capturada en el entrenamiento, no es independiente a la forma de la malla. Por ejemplo, la apariencia que tiene un ojo puede depender de la posición de la cara.

Para tratar de independizar apariencia y forma, la apariencia es analizada en posición de la malla concreta, que en este proyecto coincide con lo que en el apartado anterior se ha llamado malla de destino.

Por lo tanto, el primer paso de el entrenamiento en apariencia trata de convertir todas las imágenes obtenidas a la mencionada forma común mediante la función warp

Ilustración 25: Ejemplos de Conversion de la apariencia Original a la forma común

Ya con todas las imágenes transformadas mediante warp, se representa cada imagen como un vector y se aplica PCA de la misma forma que en el apartado anterior. Los vectores obtenidos se filtran atendiendo a su ' Energía' tal y como se hizo con los vectores de apariencia, no obstante en este caso se ha comprobado que para un funcionamiento adecuado es conveniente utilizar un umbral mayor que el utilizado anteriormente (0.999).

Una mejora añadida es la de anexar a la base el vector media y vectores de offset para poder genera cambios en iluminación. Esta acción se realiza igual que en el caso de la forma, realizando Gram-Schmidt para asegurar ortogonalidad.

Programa 'fitting'

La implementación de el programa 'fitting' poco aporta a lo explicado en el capítulo en el que se analizo teóricamente cada una de las posibilidades para realizar esta tarea.

El motivo de incluir esta sección en el capítulo de implementación es poder explicar como se han superado ciertas limitaciones a la hora de llevar a cabo el algoritmo de AAM

Gradiente:

En todas las versiones del algoritmo, es necesario realizar el gradiente de alguna imagen (ya se de I(W(x;p)) o de $A_0(x)$). Para ello, tradicionalmente en procesado de imágenes, es necesario el uso de máscaras . Existen varias máscaras con este fin. En este proyecto se ha decidido utiliza la máscara de *Sobel 3x3* ya que es la más ampliamente utilizada, y además está implementada en la función *cvSobel* de OpenCV (cuando su último parámetro es un 3). Es importante tener en cuenta los siguientes dos aspectos cada vez que se utiliza la función *cvSobel*:

- Se realiza un filtrado gaussiano después de la máscara. Esto en principio puede afectar a la convergencia [Matthews & Baker, 2004].
- Es necesario normalizar la máscara, es decir hay que multiplicarla por un factor 1/8.

-1	0	1
-2	0	2
-1	0	1

Máscara de Sobel para derivada en x

-1	-2	-1
0	0	0
1	2	1

Máscara de Sobel para derivada en y

Ilustración 26: Máscaras de Sobel Utilizadas por OpenCV

Cambio de base. Función cvGEMM

A lo largo de todo el desarrollo realizado, un aspecto fundamental es el cálculo matricial. Una de las operaciones más repetidas es la multiplicación de dos matrices y su posterior suma con otra, es decir : $A \cdot B + C$. Ejemplos de esto son los cambios de base para, a partir de los parámetros de forma, hallar los vértices de la malla, o el cálculo del Hessiano mediante la ecuación (35).

Además, como se comprueba en los ejemplos dados, en ocasiones es necesario realizar la transposición de alguna de las matrices para hacer coincidir correctamente las dimensiones, o multiplicar alguna matriz por un factor

La función *cvGEMM* de OpenCV permite realizar de forma eficiente la siguiente operación:

$$A^{*} \cdot B^{*} + C^{*} \quad donde \; X^{*} \; puede \, ser \; X \, ó \, X^{T}$$

Asimismo el uso de esta función evita tener que reservar memoria para resultados intermedios (como la transposición o el producto) y esto conlleva un ahorro de memoria y de lo que más interesa en esta aplicación de tiempo real: el tiempo.

Inicialización: Viola & Jones.

Los algoritmos de minimización utilizados, son algoritmos locales que necesitan ser inicializados los suficientemente cerca de la solución para que estos converjan. Por ota parte, utilizando la transformación Global, se dispone de unos parámetros representativos de la posición y el tamaño de la cara que deben ser añadidos a la función de error que se pretende minimizar.

Nuestra propuesta es utilizar el algoritmo de Viola & Jones para establecer un punto de partida para los parámetros de la transformación Global. De esta forma, la mayor parte de la cara se encontrará en el interior de la malla (región donde se realiza la minimización).

OpenCV cuenta ya con una implementación de Viola & Jones y con sus propios archivos con imágenes de entrenamiento situados en la carpeta */usr/local/share/opencv/haarcascades.* La implementación de este proyecto se ha realizado de la siguiente manera:

- Se ha utilizado el clasificador *haarcascade_frontalface_alt.xml* ya que es el que mejor resultados ofrece para inicializar el programa cuando el usuario mira de frente.
- La función *detect_faces* basada principalmente en *cvHaarDetectObjects* devuelve una estructura cvRect que indica de la posición y el tamaño de la cara.
- Si Viola & Jones no encuentra ninguna cara. Se carga otra imagen y se trata de localizar otra vez una cara.
- Para convertir de la estructura devuelta por la función a los parámetros Q, se utiliza el siguiente código:

```
Q->data.fl[0]=sqrt(2.5*(float)
((pos.width*pos.width+pos.height*pos.height)));
Q->data.fl[1]=0.0;
Q->data.fl[2]=((float)pos.x+
(float)pos.width/2)*sqrt(NUM_TOTAL_PUNTOS);
Q->data.fl[3]=((float)pos.y+
(float)pos.height/2)*sqrt(NUM_TOTAL_PUNTOS);
```

El primer parámetro Q que se corresponde con el de la amplitud de la cara se multiplica por un factor para asegurar que la cara queda en el interior de la malla.

De la misma forma, los dos últimos vectores, correspondientes a la traslación horizontal y vertical, de multiplica por la raíz cuadrada del número de puntos para que al multiplicarlo por el correspondiente vector de la base se obtenga un vector que suma la traslación a la correspondiente coordenada de cada vértice.

Si se supone conocida la posición de la malla en un frame de video y que la persona no va a hacer movimientos bruscos, se puedeconcluir que no es necesario inicializar con Viola & Jones para cada frame del video sino que es suficiente con dejar la malla en la posición del anterior frame.

Esto hace ahorrar tiempo ya que **solo es necesario Viola & Jones al principio y cuando el algoritmo no converge.**

Reinicialización del Algoritmo.

Un aspecto interesante para una aplicación es primero conocer cuándo el algoritmo ha terminado de converger, y segundo si este lo ha hecho correctamente.

La literatura da la respuesta a la primera pregunta: se debe iterar hasta que el cambio de los parámetros calculados sea pequeño.

Sin embargo, la segunda pregunta es más difícil de responder. Es difícil establecer un umbral para el error (en apariencia o normal) ya que el error mínimo depende mucho de la posición de la cara.

La solución planteada pasa por establecer un número de iteraciones máximas permitidas. Sobrepasado ese límite se concluye que el algoritmo ha degenerado y que el programa debe reinicializarse con Viola & Jones.

Este método no es del todo robusto pues puede haber ocasiones donde los parámetros varíen poco y sin embargo no se halla alcanzado el mínimo deseado. Sin embargo, como se analizará en el siguiente cápitulo con una buena elección de los umbrales, se pueden minimizar los casos de falsos positivos.

Función warp.

La realización del warp atiende fielmente a lo explicado en el capítulo anterior. Sin embargo, para evitar un código muy repetitivo todas la funciones relativas al warp en cualquier implementación llama a la función *warp*. Esta función warp simplificada convierte la imagen dentro de una malla origen a otra imagen dentro de una malla destino utilizado el procedimiento explicado en la página 68. Así pues, la diferencia entre cada implementación es como se calcula la malla de origen y de destino.

Una de esas mallas (normalmente la de destino), suele coincidir con un destino común en el que se define la base de apariencia. Como ya se explicó en el apartado de *Entrenamiento en forma* (pág: 86), esta malla no puede coincidir con s_0 debido a la implementación en OpenCV y se debe elegir otra malla. No obstante, este cambio no es trivial pues afecta directamente al algoritmo *composicional inverso* en aspectos como la derivación.

La función de error que se desea minimizar sigue viniendo dada por la ecuación (23). Donde el warp *W* debe ser sustituido por la composición *N* y *W*, que son las mismas funciones que las definidas al final del capítulo 4, es decir $N \circ W$ es la función que transforma de la malla promedio s_0 a la malla destino *s.*

Además se definiá la nueva función *R,* encargada de llevar desde s_0 a la malla común (donde es definida la apariencia) denominada *WarpDestination* (s_D). En este caso como $s_D = K \cdot s_0 + t$, *R* se define así: $R(x) = K \cdot x + t$.

Como se sigue minimizando la misma función el desarrollo de la minimización es el mismo que el seguido en la pág: 60, con la excepción que OpenCV no es capaz de realizar el gradiente de A_0 ya que esta definido en s_0. Para poder calcular un gradiente en OpenCV es necesario que la imagen a derivar este en un espacio en el que la variable dependiente sea positiva y entera, como lo es el espacio $x' \in s_D$, donde se definirá A'_0 .

A partir de la siguiente igualdad:

$$A'_0(R(x)) = A_0 \tag{75}$$

Se obtiene el gradiente de A_0 relacionado con el de A'_0 .

$$\nabla \cdot A_0 = \nabla \cdot A'_0 \cdot \frac{\partial R}{\partial x} = K \cdot \nabla A'_0 \tag{76}$$

Como se puede observar, aparece una constante que multiplica directamente al gradiente y por lo tanto a la matriz SD.

El resto de la argumentación no se modifica y por tanto sus resultados siguen siendo válidos.

Para poder implementar la ecuación (33) en OpenCV se debe realizar el cambio $x = (x' - t)/K$, realizando todos los cálculos en x' definida en el espacio $s_{D.}$ Sin embargo, como $\frac{\partial W(x;p)}{\partial p} = \frac{\partial W(x';p)}{\partial p}$ esto no producirá ningún cambio.

El algoritmo de *Lucas-Kanade* no se ve afectado debido a a que no hay composición.

Derivada de la función warp

El Jacobiano de la función warp es una matriz 2x[Nº de Parámetros] que tiene un valor distinto en cada píxel del interior de la malla. Sin embargo, openCV no proporciona una estructura para trabajar con una "matriz de matrices".

Es por ello por lo que, para el cálculo del Jacobiano, se ha creado una matriz dinámica (asignada mediante malloc), de punteros a estructuras *CvMat,* de ancho y alto igual al ancho y alto máximo de la malla en píxeles respectivamente (mismo tamaño que la imagen *Look up Table*) . Esta matriz representa cada uno de los píxeles de la malla (Como la matriz tiene que ser cuadra-

da, los pixéles que no están en el interior de la malla contienen un puntero *NULL*). Cada píxel (posición de la malla) apunta a su correspondiente estructura CvMat con una matriz de tamaño 2x[Nº de Parámetros] que contiene el Jacobiano del warp en ese punto.

6.- Resultados experimentales

Estudio de la precisión de los algoritmos

Descripción de los experimentos.

Existen dos tipos diferenciados de imágenes con las que se realizan experimentos:

1. Imágenes que han sido utilizadas en el entrenamiento.
2. Imágenes no entrenadas.

En el primer caso, para el cálculo de errores se tomarán por buenos los puntos marcados inicialmente y utilizados para crear las bases utilizadas. En el caso de las imágenes no entrenadas se usará un marcado que se realizado manualmente, usando las mismas pautas que el marcado del caso anterior.

El conjunto de imágenes que se disponen para realizar el experimento son 50 imágenes, de las cuales 36 han sido utilizada para entrenar el AAM.

A partir de los puntos marcados se introducirán un error aditivo gaussiano cuya media y varianza cambiará dependiendo de cada experimento. Este tipo de experimento es diferente al que normalmente se realiza en la bibliografía utilizada, en ella se tomarán cuatro puntos (por ejemplo los vértices del warp) se les añade un error y se calcula la homografía entre los puntos modificados con el error.

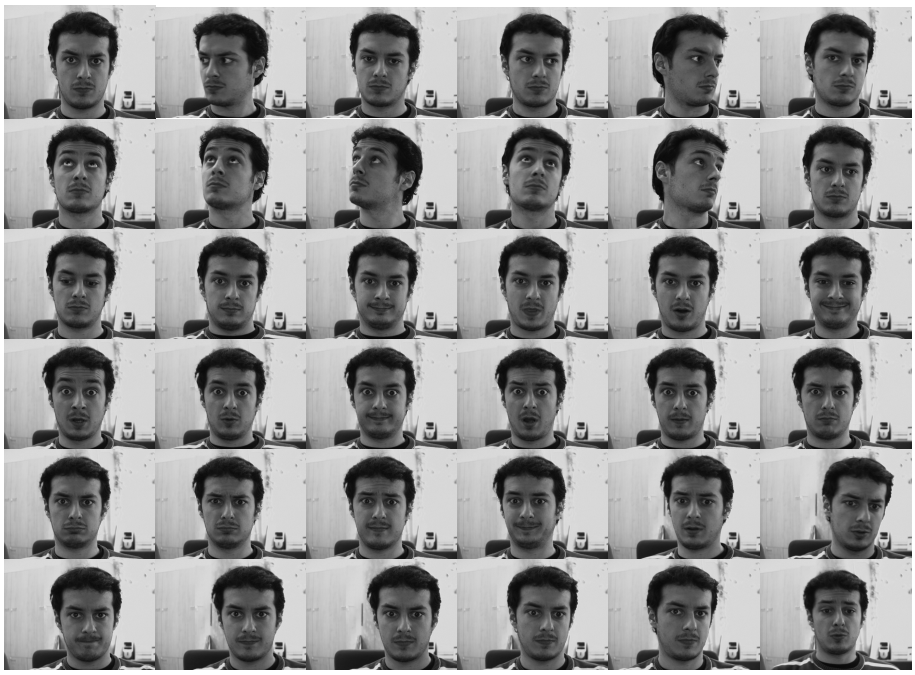


Ilustración 27: Conjunto de Imágenes de entrenamiento para los experimentos

Para los experimento, en general, se han entrenado el AAM con 36 imágenes en diversas posiciones. El entrenamiento ha sido realizado según lo explicado anteriormente, estableciendo los umbrales en 0,99 para la forma y 0,999 para la apariencia. Con esto consiguen un total de 6 vectores en la base de forma y 22 en la de Apariencia.

A la hora de realizar cada experimento se tomarán los datos aportados por el algoritmo de Lucas-Kanade como el máximo que se puede esperar experimentalmente. Ya que contiene toda la información (tanto de forma como de apariencia) dentro de la función que pretende minimizar.

Para cada imagen se dispondrá de un máximo de **15 iteraciones** para encontrar un mínimo. Este número máximo de iteraciones coincide con el utilizado en la implementación en tiempo real, tratando de obtener una respuesta, aunque sea errónea, en un tiempo acotado.

El criterio utilizado para considerar que el algoritmo ha convergido es que el incremento de la raíz cuadrada del error cuadrático de la malla es menor a 0,2 píxeles en alguna de las iteraciones del algoritmo.

Para los resultados en los que se muestra la evolución de algún valor en función del número de iteración esta ha sido calculada teniendo en cuenta las imágenes que han convergido según el criterio anteriormente explicado.

Primer Experimento. Error Implícito.

En este primer experimento, se parte del marcado original de los puntos de la cara y se pretende comprobar como evoluciona el error al ejecutar el algoritmo sin ningún error inicial.

Para ello, se inicializa los parámetros P y Q con la proyección de los puntos originales del marcado, sobre las bases de forma con un procedimiento parecido al explicado en el apartado de composición en el caso de utilizar la transformación Global (pág: 77, ecuaciones (72), (73) y (74)). Por su parte se realizará lo mismo con los parámetros de apariencia. Para inicializarlos primero se aplicará la función warp para convertir la apariencia desde los puntos marcados hasta el destino en el que se han definido las bases de apariencia, posteriormente se proyecta esta imagen sobre el espacio vectorial generado por la base de apariencia.

Con esto se pretende cuantificar el error debido a la pérdida de información inherente al entrenamiento con PCA y al propio algoritmo.

Primer Resultado: Imagen entrenada que converge correctamente.

En las siguientes gráficas se muestra el resultado obtenido en una imagen entrenada, en la que el algoritmo se comporta correctamente.

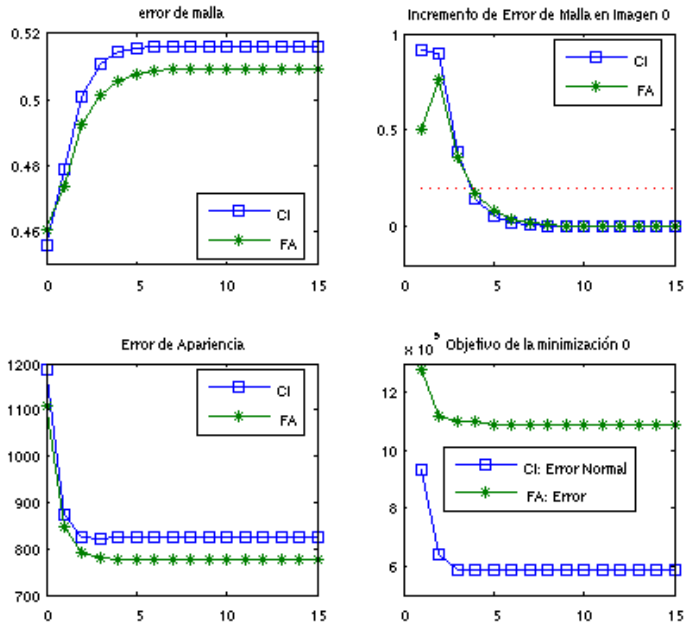


Ilustración 28: Gráfica de prueba 1 con imagen de entrenamiento que converge correctamente.

En las gráficas anteriores se comprueba lo siguiente:

1. El error de malla por píxel (definido como la raíz cuadrada del error cuadrático de la posición de la malla con respecto a la posición de la malla en el marcado original, dividido entre el número de puntos de la malla) no comienza en cero puesto que, al haber

realizado PCA en el entrenamiento, la base de forma no es capaz de generar completamente la malla entrenada.

2. Ocurre lo mismo que lo expuesto en el punto anterior con el error en apariencia.
3. El mínimo, tanto en el error de malla como en el error de apariencia, no se encuentra en la posición inicial. La explicación es que al haber realizado PCA y encontrarnos en un espacio con menor dimensión, los mínimos no tiene por que coincidir con lo entrenado ya que, como ya se ha podido comprobar, el sistema no tiene por que poder generar perfectamente las imágenes de entrenamiento.
4. El error de malla aumenta desde la posición inicial. En un principio esto podría resultar desconcertante, sin embargo no se debe olvidar que el error que se pretende minimizar es el expresado en la fórmulas (23) y (45), para el algoritmo 'Forward Additive' y el de Normalización del composicional inverso respectivamente, y como se muestra en la figura inferior derecha este siempre disminuye.
5. En este caso, la minimización realizada por 'Forward Additive' es mejor que la realizada por la Normalización del composicional inverso.

Segundo Resultado: Imágen entrenada que no converge correctamente.

La siguiente imagen con la que se probueba el algoritmo, es una imagen entrenada la cual se verá que ha convergido correctamente con el algoritmo 'Forward Additive' y sin embargo no ha hecho lo mismo con el otro algoritmo utilizado.

Se aprecia también como con el algoritmo de Normalización del composicional inverso se llega en la segunda iteración a un mínimo del cual se sale. Es por ello interesante saber detectar cuando se ha llegado a este mínimo para parar de iterar y de esta manera obtener mejores resultados.

Comparativamente, el error de malla y de apariencia de está imagen es mucho mayor (incluso al inicio). Esto parece indicar que el modelo genera peor imágenes parécidas a esta (de perfil).

En la Ilustración 30 se muestran los resultados de ambos algoritmo. La conclusión más interesante a la que se puede llegar es que, aún no habiendo convergido el algoritmo CI para el criterio utilizado (incremento de error de malla menor que 0.2) el resultado se aproxima bastante a la cara.

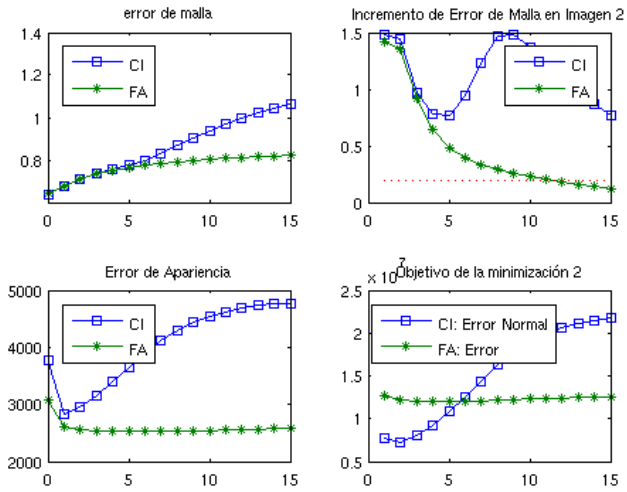


Ilustración 29: Gráfica de prueba 1 con imagen de entrenamiento que no converge correctamente.

Ilustración 30: Resultado de prueba 1 con imagen de entrenamiento que no converge correctamente.
(Malla con FA: superior izquieda. Malla con CI: superior derecha. Modelo generado por FA: inferior izquierda. Modelo generado por CI: inferior derecha)

Tercer Resultado: Imágen no entrenada que converge correctamente.

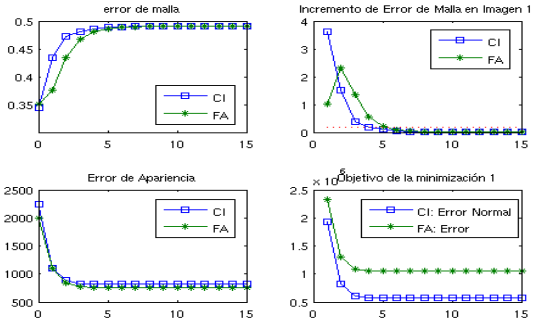


Ilustración 31: Gráfica de prueba 1 con imagen no entrenada que converge correctamente.

El resultado obtenido es muy similar al obtenido con la imagen entrenada.

Resulta interesante comprobar que el valor inicial de error de malla es incluso menor que el de cualquiera de los ejemplos anteriores que han sido entrenado

Otro aspecto importante es que ambos algoritmos tienen un rendimiento similar para los errores de malla y de apariencia.

Cuarto Resultado: Imágen entrenada que no converge correctamente.

La imágen utilizada en este caso es muy similar a la del segundo resultado (imagen de perfil).

Los resultados obtenidos también son similares: El algoritmo "Forward Additive" garantiza encontrar un mínimo. Sin embargo, el algoritmo de Normalización del compasicional inverso vuelve a "escaparse" del mínimo obteniendo peores resultados y no llegando a converger para la condición dada.

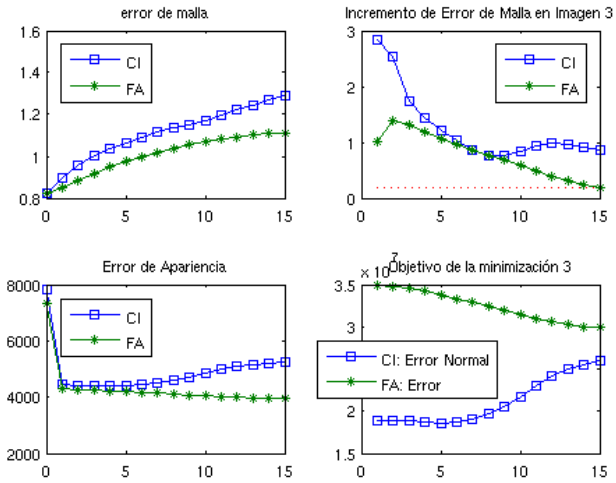
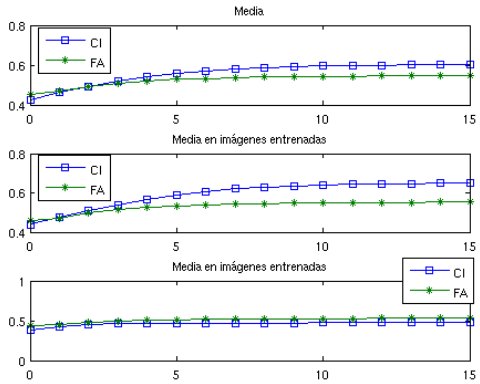


Ilustración 32: Gráfica de prueba 1 con imagen no entrenada que no converge correctamente.

Resultados globales:

Los siguientes resultados tratan de expresar una visión global de como funciona cada uno de los algoritmos para este experimento.

Para este propósito únicamente se tendrá en cuenta el error de malla y se realizará la media discriminando aquellas imágenes que se consideren que no han convergido según el criterio utilizado.



	Converge		No Converge	
	FA	**CI**	**FA**	**CI**
Entrendas	34	28	2	8
No Entrendas	13	10	1	4
TOTAL	47	38	3	12

Ilustración 33: Gráfica de resultados globales en prueba 1

Segundo Experimento: Media nula cambio de varianza.

Para este experimento, se comenzará a iterar desde una situación de partida de los puntos de la malla igual a la original mas un ruido aditivo gausiano con varianza cada vez mayor. Con esto, se pretende conocer como de bueno es cada algoritmo ante perturbaciones Iniciales.

Se ha elegido este tipo de prueba por que es similar a las realizadas en la documentación a cada algoritmo para compararlos entre sí. A diferencia que en [Baker et al,2002],[Baker et al, 2003a] y [Baker et al, 2003b] entre otros, en este proyecto se ha preferido añadir el ruido sobre la malla, en lugar de sobre cuatro puntos que determinan una homografía.

En la siguientes dos gráficas se relacionan el número de iteraciones necesarias para que cada algoritmo converja con la varianza inicial de los puntos de la malla.

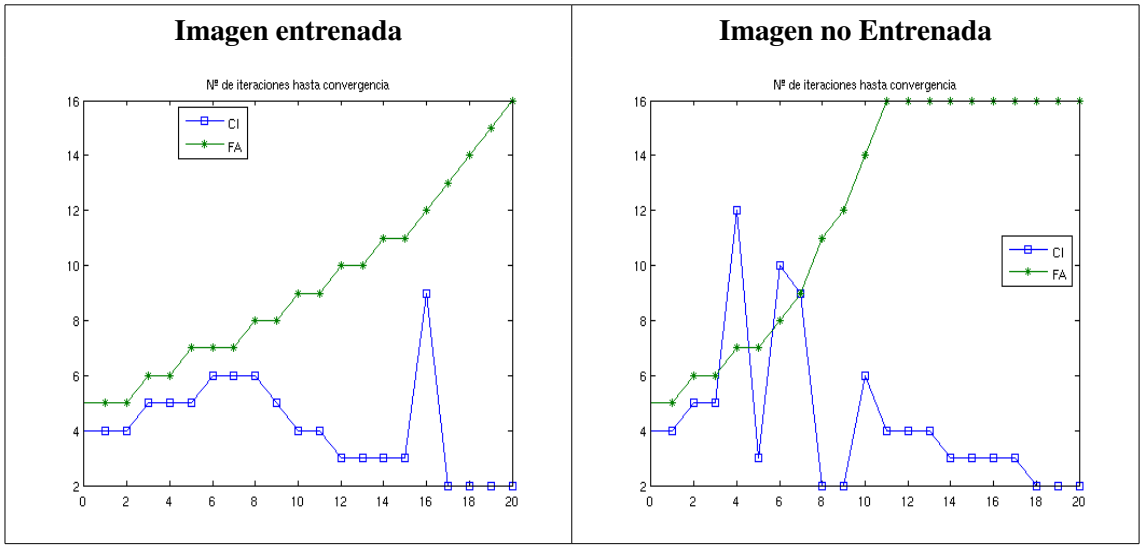


Ilustración 34: Numero de iteraciones en función del error inicial

En las anteriores gráficas, se ha representado con valor de 16 iteraciones aquellos casos en el que el algoritmo no convergió.

Al analizar las dos gráficas se ve que el algoritmo 'Forward Additive' se comporta de forma similar en ambos casos, si bien se aprecia que para imágenes no entrenadas es menos robusto y tarda más iteraciones en llegar a una solución.

Por otra parte, en el algoritmo composicional inverso vemos un comportamiento anómalo que, a priori, carece de sentido: Superado cierto nivel de ruido el número de iteraciones necesarias para que el algoritmo converja disminuye. Esto ocurre tanto para imágenes entrenadas como no entrenadas.

Para intentar explicar esto, se analizará el error en la malla modificando la varianza del error inicial. En las siguientes figuras se mostrará como evoluciona el error de malla en cada iteraciones para una varianza de error inicial de 0, 4, 8, 12, 16 y 20 píxeles al cuadrado.

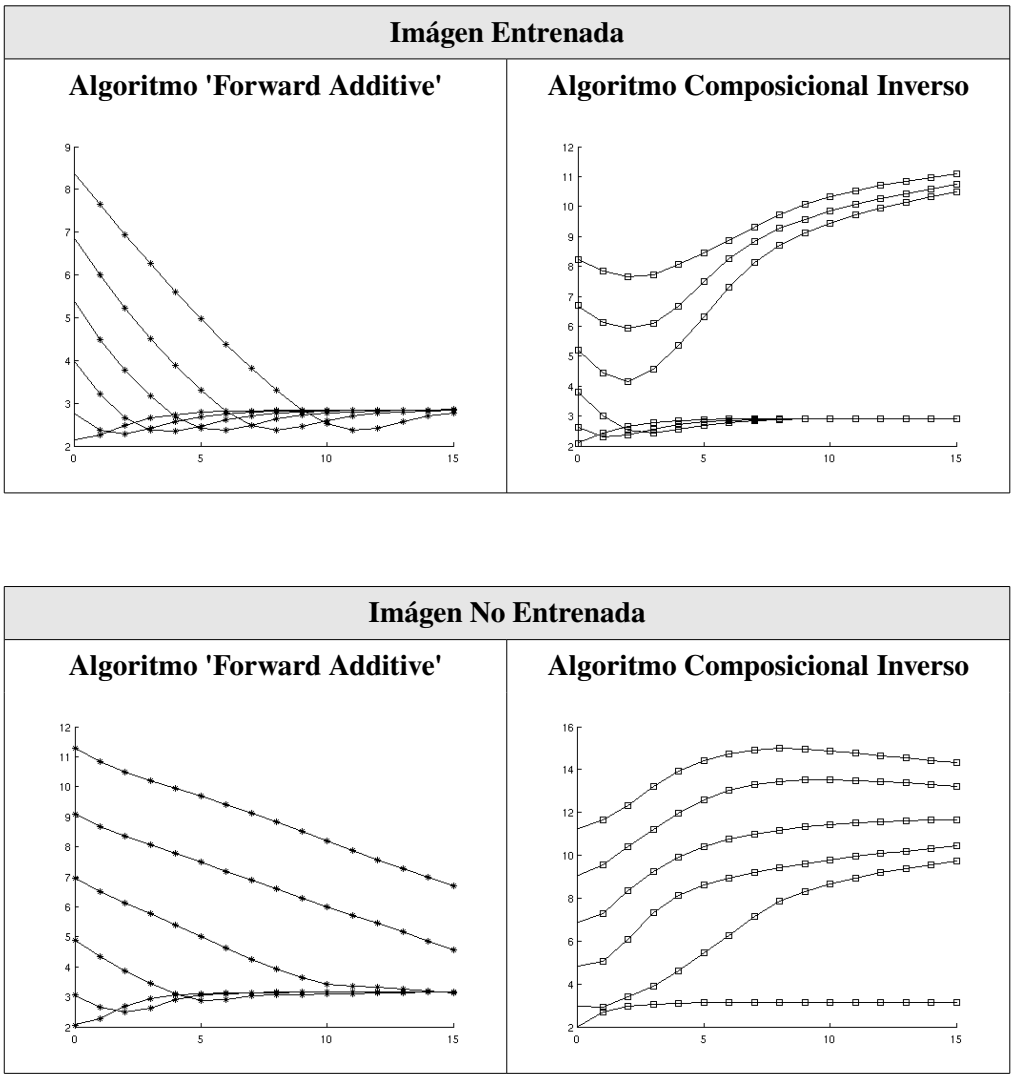


Ilustración 35: Variación del error de malla en función del numero de iteraciones para Imágenes entrenadas y no entrenadas, usando distinto tipos de algoritmo y de varianzas en el error inicial

A la luz de estos resultados, se llega a la conclusión que, al contrario del algoritmo 'Forward Additive', que siempre converge hacia el mismo mínimo, el algoritmo composicional inverso 'cae' en mínimos locales distintos al deseado.

De esto se deduce lo siguiente:

- El criterio de convergencia utilizado, no implica directamente que la solución encontrada por el algoritmo sea la correcta.
- El algoritmo composicional inverso funciona peor que el algoritmo de Lucas Kanade, en los casos que el estado inicial difiere mucho del objetivo.

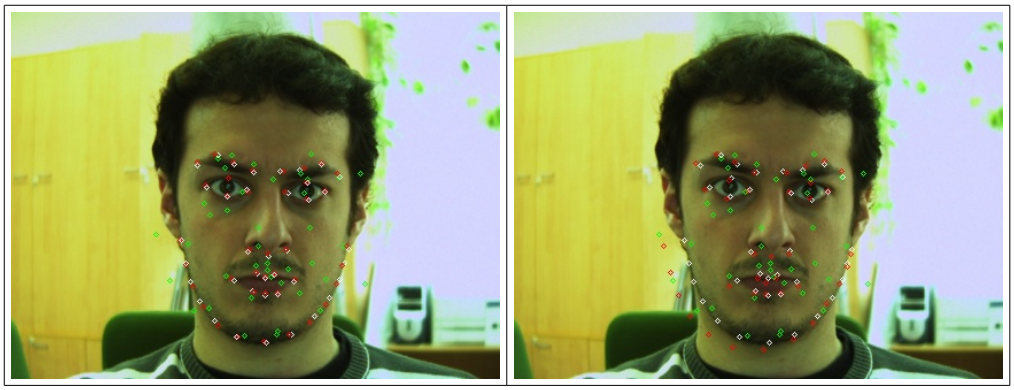


Ilustración 36: Resultado de la misma imagen con cada uno de los algoritmos (izquierda: FA, derecha: CI)

*En **Rojo**: Resultado,**Verde:** Posición de partida, **Blanco:** malla marcada manualmente.*

7.- Aplicación.

Introducción

En este capítulo, el fin es realizar una prueba de concepto de aplicación simple que muestre como se pueden aplicar cada uno de los elementos analizados a lo largo de este proyecto.

Se pretende que el objetivo del programa sea el de ayuda a la discapacidad mediante un interfaz humano-máquina por medio de visión artificial, que posibilite a un usuario interactuar con el ordenador mediante gestos faciales.

La aplicación que se describe a continuación pretende emular el comportamiento de un joystick o un ratón: Mediante el movimiento de la cara el usuario podrá ejecutar las ordenes "IZQUIRDA, DERECHA, ARRIBA, ABAJO" y además combinar todas estas ordenes con la acción de disparo o "CLICK" que se realizará mediante la apertura de la boca. Obviamente, se definirá una posición de la cara "NEUTRO" (mirada al frente) en la que no se realice ninguna acción.

Para ello es esencial que el programa sea lo más robusto posible, es decir, reducir el número de errores y/o, en caso de error, reconocer el error y reiniciar la aplicación de una forma segura. Ya que de otra manera, según la aplicación, se podría poner en peligro la integridad del individuo (Por ejemplo en el manejo de una silla eléctrica).

Otra cualidad que debe tener esta aplicación es la de trabajar en tiempo real, como ya se ha explicado con anterioridad, esto significa que, ante un gesto del usuario, la aplicación debería ofrecer una respuesta en un tiempo delimitado. Además, para esta aplicación, se debe tratar de que ese tiempo sea reducido.

Como se trata de una implementación real, se debe utilizar un lenguaje que permita llevar acabo los algoritmos explicados en capítulos anteriores. El lenguaje elegido ha sido C, debido a su versatilidad y gran número de librerías existentes así como referencias. Concretamente, para el desarrollo de las aplicaciones se ha utilizado la librerías libres de visión artificial OpenCV.

Adaptando AAM's a una aplicación.

Introducción:

Como ya se explicará en el capítulo dedicado a la implementación,para realizar cualquier aplicación a partir de un modelo de apariencia activa es necesario que existan, fundamentalmente dos tareas:

- Una dedicada al entrenamiento del modelo. Esta tarea, pensando en la aplicación buscada, debería ser ejecutada por otra persona distinta al destinatario final de la aplicación debido a que es necesario que los puntos sean marcados mediante los métodos de entrada convencionales. No obstante, esta labor debe ser complementada por otra clase de información: aquella que relaciona posiciones de la cara con un gesto determinada (sonrisa, mirada hacia arriba, etc..)
- Y una tarea llevada a cabo por un programa que realice el llamado 'fitting'. Este es el programa principal, destinado al usuario final, utiliza los resultados arrojados por los parámetros una vez que se ha llegado a la convergencia para conocer e interpretar el gesto del usuario

En este capítulo se tratará de describir una aplicación en la que se utilizará un modelo de apariencia activa para generar un modelo que permita discernir los gestos faciales de un usuario. Posteriormente, se utilizará esta información para proporcionar un interfaz de comunicación con el ordenador, a modo de 'ratón'.

No es objetivo de este proyecto realizar el controlador para que lo anteriormente pueda ser utilizado dentro de un sistema operativo. Más bien, la intención es presentar una prueba de concepto que demuestre que puede ser llevado a cabo.

El fin último de esta aplicación sería ser aplicada en un entorno de ayuda a la discapacidad. Tratando proponer una alternativa a los interfaces existentes que suelen estar compuesto de hardware costoso y no estandarizado.

No obstante a la luz, de los resultados arrojados por los experimentos del capítulo anterior, se ha decido no utilizar el algoritmo de Composicional Inverso ya que este no proporciona la robustez requerida por nuestra aplicación. Para esta prueba de concepto se utilizará el algoritmo 'Forward Additive' que, por sus características si proporcionan la robustez necesaria. Sin embargo, como ya se ha explicado, este algoritmo, debido a su complejidad no puede ser implementado (en el momento que se realizaron las pruebas) en condiciones de tiempo real. Esta circunstancia acentúa el tratamiento de prueba de concepto del desarrollo sobre la de aplicación final.

Entrenamiento:

La labor de crear una base PCA no requiere ningún cambio con la efectuada, en el capítulo de Implementación por los programas 'CAPTURAR', 'MARKER' y 'TRAINING' que interactúan, de la misma manera que lo hacían antes, respondiendo a lo mostrado en la siguiente figura.

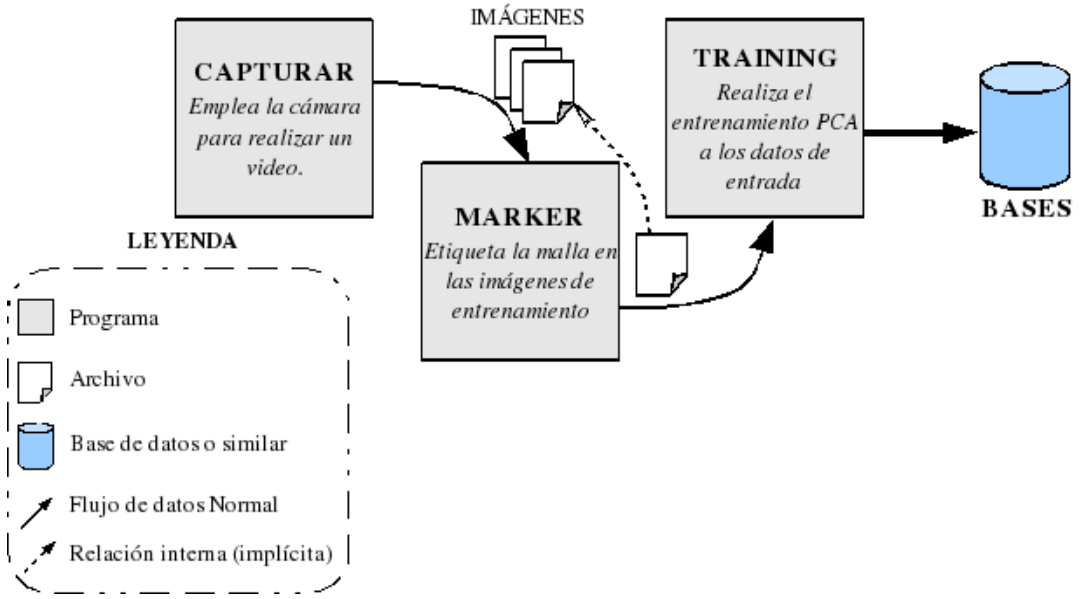


Ilustración 37: Esquema del Entrenamiento PCA de la aplicación final

Sin embargo, falta información que relacione cada conjunto de parámetros con una expresión.

Para encontrar esta información, primeramente se ha analizado que tipo de movimientos que se quieren encontrar. Es en esta etapa cuando se debe plantear como va a ser la interacción del usuario con el PC.

Finalmente, para esta prueba de concepto, se decidió que el usuario diera órdenes de posición mediante movimientos de la cara y que además tuviera la opción de realizar el comando 'click' a la vez que se realizan los comandos de posición (esto permitiría acciones como 'arrastrar'), por ello se asigna la apertura de la boca como gesto para hacer 'click' ya que es independiente a mover la cabeza.

En la parte del entrenamiento correspondiente a caracterizar los gestos, únicamente se relaciona el conjunto de parámetros, resultantes de proyectar la imagen de entrenamiento sobre las bases PCA, con la etiqueta que define ese gesto. Cada conjunto de parámetros se guarda en un archivo con el nombre del gesto que representan para que pueda ser recuperado por el programa principal.

En este paso, el algoritmo de fitting es lanzado con cada imagen de este segundo entrenamiento (ya que esta imagen no tiene porque estar en el entrenamiento PCA) para encontrar los parámetros de forma de la imagen de entrenamiento. Para ello, como es habitual, se inicializa cada

vez con el Algoritmo de Viola&Jones. Esto hace muy probable que para la misma imagen de entrada que la imagen de entrenamiento los parámetros obtenido no sean los mismo que los obtenidos en el entrenamiento ya que el algoritmo no ha sido inicializado en el mismo punto. Esta falta de precisión, no afecta al resultado final de la aplicación debido a la forma en la que realizase la clasificación de los gestos.

Método de Clasificación de gestos.

En esta prueba de concepto, el objetivo no es ser capaz de diferenciar un amplio abanico de gestos, sino que únicamente se pretende, de una forma sencilla, realizar una aplicación funcional que mueva un puntero a lo largo de la pantalla y sea capaz de seleccionar, arrastrar, etc...

EL método que aquí se plantea es usar una simple distancia euclídea entre el vector de forma de la imagen de entrada y cada unos de los vectores de forma de las imágenes de entrenamiento. Si bien esto no es trivial debido a que los elementos de una clase (por ejemplo "guiñar el ojo derecho") no tienen por qué ser los más próximos al representante de la clase (el de la imagen de entrenamiento) en el espacio vectorial elegido.

Ya que en PCA los vectores de la base son ordenados de forma decreciente según su covarianza, los primeros vectores, que son los que más energía acumulan y por tanto mayor covarianza tienen, deben ser los que modelen movimientos más bruscos, en cambio, movimientos más sutiles conllevarán cambios en las coordenadas relativas a los vectores con un menor autovalor asociado. Aún así, se debe ser consciente de que el entrenamiento PCA no garantiza la correspondencia de un subconjunto de vectores con un gesto determinado, o en otras palabras: todos los vectores pueden contribuir en cualquier movimiento.

Acorde con lo planteado en el apartado anterior y tras una inspección de los parámetros que se corresponde con cada gesto, se ha llegado a la conclusión que es posible discriminar movimiento de la cabeza mediante distancia euclídea si se premian los parámetros correspondientes a un vector de la base con un mayor autovalor asociado. Esto parece lógico ya que un movimiento de cabeza conlleva un movimiento considerable de muchos puntos de la cara y por tanto mucha energía.

Consecuentemente, se ha decidido ponderar el vector de parámetros de forma multiplicándolo (término a término) por un vector formado por los autovalores asociado al vector de la base correspondiente, antes de calcular la distancia euclídea.

Para detectar que el usuario ha abierto la boca se ha decidido emplear otro método debido a la dificultad que existe para encontrar una frontera que permita discriminar el gesto y no confundirlo con otros. Este método consiste en fijar un umbral en la distancia vertical entre los puntos superior e inferior de la boca (puntos 32 y 36 en la Ilustración 24).

Esta forma de diferenciar que el usuario ha abierto la boca es prácticamente independiente al método para identificar la dirección de la cara, debido a que la correlación de una cara en la misma dirección es grande ya tenga la boca abierta o cerrada. Por ello, los movimientos de la boca en la base PCA corresponde, en gran medida, a vectores asociados a autovectores pequeños de la matriz de correlación.

Obsérvese que en el planteamiento elegido obviaremos los parámetros de apariencia por ser más difíciles de interpretar, al verse afectados en mayor medida por cambios de iluminación. Sin embargo se debe tener en cuenta que se continua utilizando el modelo de apariencia en la fase de 'fitting' para lograr una mayor precisión. Algo similar ocurre con la parametrización de la Transformación global que aportan poca información sobre el gesto.

Debido a que en este planteamiento no es necesario calcular los parámetros de apariencia, si se solucionarán los problemas de precisión del algoritmo composicional inverso, se podría utilizar

el algoritmo 'Proyect Out' Explicado en la página 63, que es más eficiente que el al algoritmo de normalización [Baker et al,2002].

En adición a lo explicado en el capítulo correspondiente al “Project Out” del composicional inverso, si se desease modelar la ganancia en la base de apariencia tal como se explicó en el algoritmo de Normalización del composicional inverso, se podría hacer sustituyendo las ecuaciones (43) y (46) por las siguientes ecuaciones respectivamente:

$$\gamma = \frac{\sum_{x \in s_0} I(W(x;p)) \cdot A_0(x)}{\sum_{x \in s_0} (A_0(x))^2} \tag{77}$$

$$\Delta p = -\frac{1}{\gamma} \cdot H_{po}(x)^{-1} \cdot \sum_{x \in s_0} SD_{po}(x)^T \cdot E(x) \tag{78}$$

En el caso de este proyecto, como únicamente se trata de implementar una prueba de concepto, se ha obviado este aspecto.

Resultados Obtenidos.

En este apartado se tratará de evaluar los resultados de la aplicación propuesta. Para ello, se entrenará con dos personas distintas. Posteriormente se analizará la capacidad de nuestra aplicación para interpretar y actuar ante los gestos entrenados.

Primer experimento.

Con el mismo entrenamiento del AAM que el utilizado en el apartado de Entrenamiento, se han utilizado imágenes de la propia secuencia a analizar para definir las órdenes que se quieren interpretar.

Es importante hacer notar que el entrenamiento del AAM elegido, fue ideado para poder reproducir la mayor cantidad posible de gestos. Y, por lo tanto, no fue pensado de forma especifica para la aplicación analizada.

El resultado obtenido es bastante convincente, ya que al analizar la secuencia el software es capáz de interpretar correctamente el 98% de los frames analizados. El fallo cometido en esta secuencia consiste en interpretar un alzamiento de cejas como un movimiento hacia abajo.

La secuencia completa se puede encontrar en el fichero *video1.avi* del CD que acompaña a este proyecto

Segundo experimento

En esta ocasión, se ha ofrecido al programa una secuencia distinta (tanto al entrenamiento del AAM como al entrenamiento de gestos)

Con este experimento se quiere comprobar , como de reproducibles son los resultados obtenidos en el anterior experimento ante pequeños cambios en las imágenes de entrada.

El resultado obtenido empeora sutilmente con respecto al anterior (87% de éxito en el análisis). En esta ocasión se pierde precisión fundamentalmente al detectar la boca abierta junto con otro movimiento.

Una justificación a estos resultados puede ser el haber aprovechado la información sobre la dirección de la cara para aumentar o disminuir el umbral, a partir el cual se considera que la boca está abierta. Además, la informaciíon de la boca no es independiente a los parámetros de forma con los que se analiza la posición de la cara, pudiendo influir en estos y hacer que se tome una decisión incorrecta.

El resultado de este experimento se ha almacenado en el el video nombrado como *video2.avi* en el CD.

Tercer experimento.

En los casos anteriores, se ha utilizado un entrenamiento AAM genérico, es decir, en el momento que se realizó el entrenamiento AAM no se conocían que gestos iban a ser interpretados

por la aplicación. Simplemente se trató de tener un entrenamiento lo suficientemente generativo como para poder adaptarse al mayor número de gestos posibles.

Cabe plantearse la duda de qué ocurriría en el caso de tener un **entrenamiento AAM menor**, pero más adaptado a nuestras necesidad. Es decir, entrenando únicamente los gestos que se desean detectar.

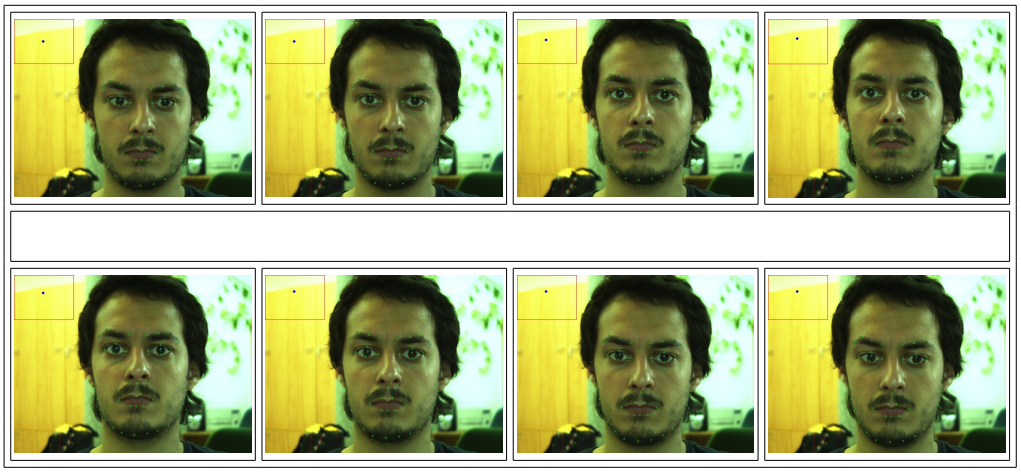


Ilustración 38: Secuencia de movimiento hacia arriba interpretada por la aplicación.

8.- Conclusiones.

A lo largo del proyecto se han mostrado las herramientas necesarias para implementar una interfaz hombre-máquina basado en el uso de visión por computador para el reconocimiento de gestos faciales. La principal herramienta de la que se ha hecho uso han sido los Modelos de Apariencia Activa que presentan una alternativa caracterizadas por los siguientes puntos fuertes:

- Precisión. Se ha comprobado que este método es capaz de reconocer de forma precisa la localización de puntos concretos de la cara.
- Permite la identificación de la persona. Debido a que la apariencia está incluida en el modelo.
- Rapidez. Utilizando el algoritmo Composicional Inverso, se es capaz de conseguir condiciones de tiempo real.

Entre los aspectos negativos del modelo destacan los siguientes:

- Es necesario realizar un entrenamiento específico previo. El entrenamiento no se puede generalizar.
- Poco robusto ante cambios de iluminación y sombras.

Si se atiende a los algoritmos de ajuste se llega a la conclusión de que si bien el algoritmo de Lucas-Kanade ofrece una alternativa con gran exactitud no cumplen con los requisitos de tiempo previstos. Por otra parte, el algoritmo Composicional Inverso proporciona un método más rápido aunque no tan exacto como el anterior pero lo suficiente, para la aplicación expuesta.

En cuanto a la aplicación, se ha demostrado que es posible es uso de los parámetros del modelo para interpretar el gesto realizado con la cara. Prueba de ello es la aplicación de muestra explicada a lo largo de este proyecto en la cual se consigue manejar un puntero.

III. Manual de Usuario

1.- Manual.

Introducción

El proyecto que se ha descrito a lo largo de este documento consta de diversos programas, ficheros de configuración y de datos. En el apartado de Implementación, en el capítulo de Memoria (pág 79) se describe el funcionamiento interno de los principales programas. En este capítulo se describirá aquellos aspectos importantes para la modificación, compilación y uso de los distintos ficheros.

A lo largo del proceso de desarrollo se ha tratado de cuidar la versatilidad y configurabilidad del resultado final. Haciendo que sea posible, por ejemplo, cambiar la malla que define la forma del modelo, o pasar de color a blanco y negro de forma muy sencilla.

Este capitulo está organizado en dos partes: en la primera se traró el plano del desarrollador, es decir, configuración y compilación del código fuente. En la segunda parte, se tratará el plano de usuario o, lo que es lo mismo, la manera de ejecutar y utilizar los programas una vez compilados.

Para organizar cada sección, se dividirá, como se ha hecho habitualmente a lo largo de este proyecto, en Entrenamiento y Ajuste o Fitting, todo lo relativo al warp, que esta relacionado está dentro de la parte de Ajuste.

Manual para el desarrollador

Listado de archivos de código fuente

Todos los archivos de código fuente, están situados en la carpeta *src/*. A continuación se presenta un listado alfabético de los archivo más importantes, y una breve descripción

ajuste.c : Entrenamiento para detectar gestos.

calcmalla.c : Pequeño programa para genera la estructura que define la malla

capturar.c : Realiza una secuencia de fotografiasy las guardas todas, secuencialmente en una carpeta.

fitting.c Algoritmo de Ajuste composicional inverso.

fitting_FA.c Algoritmo Lucas Kanade

fitting_PO.c Otra implemntación del Proyect Out.

fotos.c Toma una instantantea

marker.c Programa para marcar los puntos de la cara en las imágenes que serviran de entrenamient.

point.c Reconoce los gestos y mueve un punto. Usando Composicional inverso.

point_FA.c Igual que el anterior usando Lucas Kanade.

prueba1_1b.c Programa para realizar pruebas del composicional inverso.

prueba2_1b.c Igual que el anterio, con Lukas Kanade

review.c Ver imágenes etiquetada con marker

training.c Entrenamiento PCA

warp.c Funciones relativas al warp del composicional Inverso

warp_FA.c Funciones relativas al warp con el algoritmo de Lucas Kanade.

Cambio de Color a blanco y negro.

Se han realizado los ajustes necesarios para que tan solo con modificando la variable COLOR en el Makefile (0: en Blanco y negro y 1: en Color) se compilarán todos los programa con todo lo necesario para funcionar en el modo elegido.

No obstante, es altamente recomendable, realizar >> *make clean* desde la linea de comandos para borrar vestigios de la configuración anterior.

Entrenamiento

Estos son los posibles parámetros, relativos al entrenamiento que se pueden cambiar:

Malla:

Es posible definir una nueva malla distinta a la utilizada para ello se tendrá que cambiar o incluir las siguientes tres variables en el archivo calcmalla.c, compilarlo y ejecutarlo:

```
//MALLA 1:
const int NUM_TRIANGULOS=59;
const int NUM_TOTAL_PUNTOS=40;
const int tt[NUM_TRIANGULOS][3]={{14,25,13},{14,40,13},{14,25,24},
{3,30,4},{38,30,31},{38,30,1},{8,35,7},{37,30,4},{37,5,4},{37,30,31},
{34,35,10},{34,33,35},{34,40,13},{34,33,40},{2,30,1},{2,3,30},{9,35,10},
{9,8,35},{6,37,5},{36,35,7},{36,6,7},{36,6,37},{39,33,40},{39,38,31},
{39,33,31},{11,34,10},{16,39,40},{32,33,35},{32,36,35},{32,33,31},
{32,37,31},{32,36,37},{12,34,13},{12,11,34},{15,14,24},{15,16,22},
{17,14,40},{17,16,40},{17,15,14},{17,15,16},{20,39,38},{20,16,39},
{20,21,38},{20,16,22},{18,28,29},{18,29,1},{18,38,1},{18,21,38},{23,15,24},
{23,15,22},{26,20,22},{26,23,22},{19,20,21},{19,26,20},{19,18,28},
{19,18,21},{27,19,28},{27,19,26},{27,26,23}};
```

Donde:

- **NUM_TRIANGULOS**. Representa el número total de triángulo de la malla.
- **NUM_TOTAL_PUNTOS.** Igual al numero total de vértices que definen la malla.
- **tt** Relaciona cada triangulo con los tres vértices que lo componen.

Se han añadido sentencias de preprocesado para poder elegir directamente la malla, cambiando unicamente el valor de la variable MALLA del makefile;

Número de vectores PCA

Para cambiar el ratio de energía que deben acumular los vectores entrenados con PCA, únicamente se debe de modificar la constante RATIO_SH para controlar el ratio en los vectores de forma, y RATIO_AP, para los de apariencia.

Fitting

Cabe destacar que en todos aquellos programas basados en la ajustar el modelo a una imagen de entrada tales como fiting.c, fiting_FA.c, point.c, point_FA.c o los programas de pruebas son ajustables, los siguientes parámetros:

- N_MAX: numero de iteraciones máximas antes de considerar que ha habido un error.
- Condicion de ruptura: Dentro del bucle que itera con cada imágen existe una sentencia condicional *if* que describe la condición de ruptura del bucle (Cuando se considera que el algoritmo ha convergido) Normalmente etiquetada con el comentario //Break Condiction.

  Actuando sobre esta linea se cambiará el criterio para ser mas o menos restrictivo a la hora de considerar que una imagen a convergido correctamente.
- AP_NORM: Constante definida en warp.h y en warp_FA.h. Es la constante por la que se multiplica s_0 para calcular el destino común donde se calcula la apariencia (Explicado en el subaparatado Entrenamiento en forma del apartado de Implementación en el capítulo 2.

Compilado

En el directorio Raíz se ha creado un archivo Makefile, capaz de compila todo los programas descritos.

Manual de Usuario

En esta sección se explicará la manera de dar uso a los programas desarrollados para este proyecto.

En este apartado se seguirá el orden secuencial recomendado para la utilización de los programas.

Captura de Imágenes de entrenamiento

Para ello se pueden utilizar los siguientes programas:

Capturar

Crea una carpeta con una secuencia de imágenes capturadas con la cámara (Puede ser necesario el uso del comando sudo para obtener privilegios de administrador para acceder a la cámara)

El único argumento necesario es el nombre de la persona a la que se realiza la captura. Este coincidirá con el nombre del fichero creado y con el comienzo del nombre de todas las imágenes guardadas en él.

El uso de este programa es sencillo, y aparece impreso en pantalla: Se debe pulsar una tecla cuando se desee comenzar a capturar, y proceder de la misma manera para terminar la grabación. Si se desea salir del programa sin capturar, hay que presionar la tecla escape.

Fotos

Funciona de forma similar al anterior a excepción de no necesitar parámetros, y que al pulsar la tecla se toma una única instantánea, preguntando por linea de comandos si se desea guardar, y si es así el nombre con el que se desea guardar la imagen (Es necesario incluir la extensión .jpg en el nombre del archivo).

El procedimiento se repite hasta que se pulsa la tecla escape.

Etiquetado de Imágenes de entrenamiento

Se realiza con el programa **MARKER** ya descrito con anterioridad en este proyecto. Su funcionamiento es sencillo.

Por linea de comando se le debe pasar los siguientes argumentos:

1. La ruta de la foto que se desea marcar.
2. El nombre del fichero de entrenamiento al que se desea añadir la anterior imagen marca (si no existe se creará)

Seguidamente aparecerá la imagen y una imagen de ejemplo de como se debe marcar (ver Ilustración 22). Los puntos se marcarán consecutivamente en el orden marcado en la imagen de ejemplo haciendo clik con el botón izquierdo en el lugar de su posición. Para Borrar el último punto, se debe hacer click con el botón derecho.

Es fundamental no borrar o cambiar de nombre las imágenes que han sido etiquetadas

Visionado y revisión de un archivo de entrenamiento.

Se puede revisar las imágenes que han sido etiquetadas en un determinado fichero de entrenamiento. Para ello basta con ejecutar el programa **REVIEW** con el nombre del archivo a revisar como único parámetro.

Además, REVIEW tiene la siguientes funcionalidades cuando se revisa una imagen:

- Si se pulsa la tecla D se borra esa imagen del fichero.

- Si se pulsa la tecla E es imagen es marcada. Cuando se termina de recorrer todo el archivo se pide el nombre del archivo donde se guardarán únicamente las imágenes que han sido marcadas.

Entrenamiento PCA

Se realiza automáticamente al ejecutar el programa **TRINING** con la ruta del archivo de entrenamiento como único parámetro.

Fitting

Todos los programas que utilizan este algoritmo, ya sea con Composicional Inversoo con Lucas Kanade tienen tres modo de funcionamiento, que se diferencian en la fuente de las imágenes analizadas:

- Desde la cámara (necesarios privilegios) Este es el modo defectos cuando no se introduce ningún comando.
- Desde una Carpeta previamente creada con el programa 'Capturar' Para inicializar este modo es necesario introducir como parámetros " -p <Nombre de la carpeta>"
- Desde una única foto: Es necesario introducir como parámetros "-f <nombre de la foto>".

IV. Pliego de Condiciones

1.- Requisitos Hardware

- Cámara Progresiva Color ,Resolución SVGA,FIREWIRE - 53 img/seg (AVT-MARLIN-F-046C)
- Ordenador Mac Mini 1.83 Ghz (4).

2.- Requisitos Software.

- Linux Ubuntu v8.04 (hardy)
- Librerías OpenCV 1.0

V. Presupuesto

Presupuesto

En esta parte del documento se va a exponer de manera aproximada el coste del proyecto, desglosándolo en costes de software, costes de equipos y costes de recursos humanos.

Coste del software

La realización del proyecto se ha llevado a cabo, en su totalidad con software libre gratuito.

Sin embargo Las librerías utilizadas, también están implementadas en otros Sistemas Operativos por lo que la traducción se supone fácil.

Concepto	Precio Unitario	Unidades	Subtotal
Sistema Operativo: Ubuntu 8.04	*0,00 €*	*0*	*0,00 €*
Librerías de Visión Artificial: OpenCV 1.0	*0,00 €*	*0*	*0,00 €*
TOTAL			**0**

Coste del Hardware

Los equipos utilizados en el proyecto están formados por un ordenador mini-MAC y por una cámara Marlin F-046C, ambos ya disponibles en el Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte (GEINTRA). Sin embargo es posible el uso de cualquier ordenador PC o similar y cualquier cámara con características similares a las mencionadas en el pliego de condiciones,

Concepto	Precio Unitario	Unidades	Subtotal
Cámara Marlin F-046C	*1.090,00 €*	*1*	*1.090,00 €*
Ordenador Mac Mini 1.83 Ghz	*480,00 €*	*1*	*480,00 €*
TOTAL			**1,570,00 €**

Coste de Recursos humanos

El número de las horas dedicadas a la ingeniería equivalen a cuatro meses de trabajo a jornada completa.

Concepto	Precio hora	Horas	Subtotal
Ingeniería	*50,00 €*	*640*	*32.000,00 €*
Mecanografía	*35,00 €*	*160*	*5.600,00 €*
TOTAL			**37.600,00 €**

Coste total del Proyecto

Concepto	Subtotal
Software	*0,00 €*
Hardware	*1.570,00 €*
Recursos Humanos	*37.600,00 €*
TOTAL	**39.170,00 €**

VI. Bibliografía

Bibliografía

[Backer & Matthews, 2001]: Simon Backer and Iain Matthews, "Equivalence and Efficiency og Image Aligment Algorithms" *In Proceedings of the 2001 Conference on Computer Vision and Pattern Recognition, pp. 1090–1097.* 2001

[Baker et al, 2003a]: Simon Baker and Iain Mathews, "Lucas Kanade 20 Years On: A Unifying Framework: Part 2", Technical Report, The Robotics Institute, Carnegie Mellon University. 2003.

[Baker et al, 2003b]: Simon Baker and Iain Mathews, "Lucas Kanade 20 Years On: A Unifying Framework: Part 3", Technical Report, The Robotics Institute, Carnegie Mellon University. 2003.

[Baker et al,2002]: Simon Baker and Iain Mathews, "Lucas Kanade 20 Years On: A Unifying Framework: Part 1", Technical Report, The Robotics Institute, Carnegie Mellon University . 2002.

[Blanz & Vetter, 1999]: Volker Blanz and Thomas Vetter, "A morphable model for the synthesis of 3D faces" *Proc. SIGGRAPH, pp. 187–194*. 1999

[Cootes et al, 1998]: T.F.Cootes, G.J. Edwards and C.J.Taylor. , "Active Appearance Models" *Proceedings of the European Conference on Computer Vision, vol. 2, pp. 484–498. .* 1998

[Jonathon Shlens, 2009]: Jonathon Shlens, "A Tutorial on Principal Component Analysis" Center for Neural Science, New York University. 2009, http://www.snl.salk.edu/~shlens/pub/notes/pca.pdf

[Lanitis et al, 1997]: Andreas Lanitis, Chris J. Taylor, and Timothy F. Cootes, "Automatic Interpretation and Coding of Face Images Using Flexible Models" *IEEE Transactions on Pattern Analysis and Machine*Intelligence, 19(7):743-756. 1997

[Matthews & Baker, 2004]: Iain Matthews and Simon Baker, "Active Appearance Models Revisited" *International Journal of Computer Vision*. 2004

[Ross]: Amy Ross, "Procrustes Analysis",Technical Report, Department of Computer Science and Engineering, University of South Carolina. Http://www.cse.sc.edu/~songwang/CourseProj/proj2004/ross/ross.pdf

[Sclaroff & Isidoro, 1998]: Stan Sclaroff and John Isidoro, "Active Blobs" *ICCV'98 pp. 1146-1153*. 1998

[Sclaroff & Isidoro, 2003]: Stan Sclaroff and John Isidoro, "Active blobs: region-based, deformable appearance models" *Computer Vision and Image Understanding*, 89(2–3) . 2003

[Viola & Jones, 2003]: Paul Viola and Michel J. jones, "Robust Real-Time Face Detection" *Int. J. Comput. Vis.*, *57*(2), 137–154. 2003