Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Modelos Activos de Apariencia y Máquinas de soporte vectorial para Reconocimiento de Expresiones Faciales en Tiempo Real

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Julián Dondero

Director: Julio César A. Jacobo Berlles

Buenos Aires, 2013

Reconocimiento de Expresiones Faciales en tiempo real

Las imágenes faciales humanas pueden tener un gran grado de variación en textura y forma. Estas variaciones pueden ser causadas por diferencias de expresión, de pose e iluminación. Técnicas basadas en modelos pueden ser una buena aproximación para el análisis de expresiones faciales, en dónde se intenta adaptar un modelo, el cual representa una entidad de interés, con una imagen desconocida. Los modelos faciales son capaces de imitar variaciones de forma y de apariencia de un conjunto de entrenamiento representativo que describe las características de la cara con un reducido conjunto de parámetros. Esta tesis describe un trabajo realizado con determinado tipo de modelo facial en la implementación de un sistema de reconocimiento de expresiones en tiempo real. Este sistema se compone de tres partes principales, el reconocimiento facial, los modelos activos de apariencia (AAM), y el análisis y clasificación de expresiones faciales utilizando un clasificador SVM.

El alineado del modelo con una imagen de entrada, es un problema de optimización no lineal, dónde se reducen al mínimo los residuos de textura a medida que se actualizan los parámetros del modelo. La etapa de ajuste consiste en el aprendizaje de las correlaciones entre los residuos de textura y parámetros del modelo de formas con el fin de construir un algoritmo más rápido y eficiente para el alineamiento y ajuste del modelo en tiempo real.

Por último se clasifica la expresión en base a la información más relevante del modelo, utilizando diferentes descriptores de expresión y máquinas de soporte vectorial previamente entrenadas.

Palabras claves: Reconocimiento de Expresiones, Modelos Activos de Apariencia, Máquinas de Soporte Vectorial, clasificadores, Reconocimiento Facial, Tiempo Real, AdaBoost.

Facial Expression Recognition

Human facial images may have a large degree of variation in texture and shape. These variations may be caused by differences in expression, pose and illumination. Model-based techniques can be a good approximation for facial expressions analysis in which attempts to adapt the model, which represents an entity of interest, with an unknown image. The Faces models are able to mimic variations in shape and appearance of a representative training set that describes the features of the face with a small set of parameters. This thesis describes work done with certain types of facial model in the implementation of an real time expression recognition system. It consists of three main modules, facial recognition, active appearance models (AAM), and the analysis and classification of facial expression using SVM.

The model aligned with a target image is a nonlinear optimization based problem where minimizing waste texture as updating the model parameters. The forming step involves learning the correlations between the residuals of texture and model parameters in order to construct a fast and efficient algorithm for the alignment of the model in real time.

Finally expression is classified based on the most relevant information of the model, using Support Vector Machines, previously trained.

Keywords: Expression Recognition, Active Appearance Models, Support Vector Machines, classifiers, Facial Recognition, Real Time, AdaBoost.

Abreviaciones y Notaciones

0.1. Abreviaciones

- AAM - Modelos Activos de Apariencia
- PCA - Análisis de componentes principales
- SVM - Máquinas de Soporte Vectorial
- HCI - Interacción Hombre Maquina
- ROI - Área de Interés
- HLF - Haar-Like Feature
- PA - Análisis Procrustes
- GPA - Análisis Procrustes Generalizado
- MF - Modelo de Formas
- EA - Error de Alineación
- FPS - Frames Por Segundo
- II - Imagen Integral

0.2. Notación

- $\mathbf{M_{form}(p_k)}$: Modelo de formas generado a partir de los parámetros p_k.
- $\mathbf{Proj(F1, F2, I)}$: Proyectamos la textura contenida en el modelo de formas $F2$ en la imagen I en el modelo de formas $F1$.
- $\mathbf{Norm(g)}$: Normalizamos la textura g. En este trabajo este proceso es el de ecualizar los histogramas de cada componente color de la imagen y aplicar un filtro *Gaussiano* para hacer efecto smooth.

Índice general

Capítulo 1

Introducción

La interpretación de caras es una funcionalidad importante para la Interacción Hombre Máquina (HCI). Conocimientos de pose, orientación, y posición de la cara, como la habilidad de reconocer identidades o expresiones en estas, permite la construcción de pequeños sistemas interactivos, como: Sistemas de reconocimiento de Expresiones, estimación de estados de ánimo, aplicaciones de videoconferencia, etc.
Reconocimiento de expresiones es un ejemplo importante de técnicas de reconocimiento de caras usadas en ambientes inteligentes. Sin embargo, existen varias dificultades a superar. Imágenes de rostros humanos pueden mostrar un alto grado de variabilidad en la forma y la textura. Estas variaciones en la apariencia se deben a las diferencias entre los individuos, las deformaciones en la expresión facial, la postura y los cambios de iluminación.
La información extraída debe ser de dimensión acotada, ya que para procesar una imagen completa se requiere un enorme esfuerzo de cálculo.

El objetivo de este trabajo es construir un sistema que sea capaz de reconocer expresiones de una persona en un video o delante de una webcam en tiempo real. Las expresiones que se intentan reconocer son 5: Neutro, Alegría, Sorpresa, Tristeza y Enojo.

Se utilizaron modelos faciales para describir expresiones. Estos modelos faciales, son conocidos como Modelos Activos de Apariencia, y son construidos a partir de un conjunto de parámetros de entrada.

Este tipo de modelos es capaz de describir totalmente las características faciales mediante un reducido conjunto de parámetros, es decir extraer información relevante de la cara sin la interferencia del fondo, pose, etc.

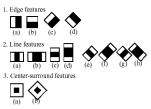
El sistema de reconocimiento de expresiones construido en este trabajo consta de 3 pasos fundamentales: reconocimiento de la cara, mecanismo para extraer información de expresión y un mecanismo para clasificar la información extraída en un conjunto de categorías o expresiones. Se utilizaron 5 expresiones a clasificar: neutra, alegría, sorpresa, tristeza y enojo.

Resumen Tesis

Estructura general de la aplicación.

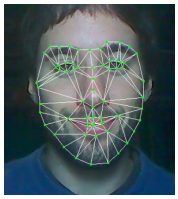
Se presenta aquí un diagrama de flujo, describiendo brevemente cada módulo y su funcionalidad dentro del proceso.

Reconocimiento de Puntos característicos con Adaboost.



Descripción de clasificadores en cascada entrenados con *Adaboost* y como pueden ser utilizados para el reconocimiento de puntos característicos. Como fueron entrenados y los resultados obtenidos.

Modelos Activos de Apariencia (AAM).

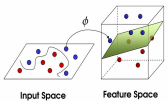


Se presentan en esta sección los Modelos Activos de Apariencia (AAM), su teoría como también el algoritmo de ajuste utilizado en la aplicación. Se describe el modelo utilizado para la cara, formas de entrenamiento de este y estrategias para lograr la actualización del modelo orientado a expresiones en tiempo real. Para este proceso, se utiliza una matriz de regresión previamente calculada.

Estimación de la pose inicial del AAM.

Se describe como obtener la pose inicial para el AAM, utilizando un clasificador en cascada entrenado con *Adaboost* y aplicando una transformación de rotación, traslación y escala al AAM.

Máquinas de Soporte Vectorial (SVM) para el reconocimiento de expresiones en tiempo real.



Se presentan en esta sección las Maquinas de Soporte Vectorial. Una pequeña descripción de su teoría como la forma de utilizarlas para el reconocimiento de expresiones faciales en conjunto con los Modelos Activos de Apariencia. Se reconocen en esta etapa 5 expresiones básicas: Neutro, Alegría, Sorpresa, Tristeza y Enojo. Se describe luego el proceso de entrenamiento de este clasificador, el kernel utilizado para el mismo y los descriptores obtenidos del AAM.

Referencias de implementación.

Se mencionarán algunos detalles de la implementación de las distintas partes del sistema:

- Librerías de AAM.
- Entrenador y creador de modelos.

- Sistema de reconocimiento de Expresiones.

Resultados y Conclusiones.

En esta sección se muestran los experimentos realizados y los resultados obtenidos. Se exponen luego las conclusiones del trabajo.

Trabajo futuro y posibles utilidades.

En esta sección se hace una breve descripción de hacia dónde podría extenderse este trabajo y posibles utilidades del mismo.

Capítulo 2

Estructura de la aplicación

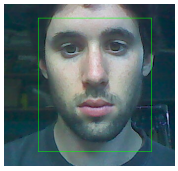
2.1. Módulos

El sistema de reconocimiento de expresiones consta de 3 módulos principales:

1. Módulo 1: Reconocimiento de cara y ojos - (Clasificadores en Cascada)
2. Módulo 2: Descriptor de expresión (Modelos Activos de Apariencia AAM)
3. Módulo 3: Clasificador de Expresiones con Máquinas de Soporte Vectorial (SVM)

2.2. Descripción

Módulo 1: Reconocimiento de cara y ojos:



El objetivo de este módulo será encontrar y determinar primero el área de interés (ROI) de la cara de un individuo frente a una webcam de una computadora. Luego dentro de este área de interés, se buscarán las áreas de interés respectivas a los ojos. Estas posiciones detectadas serán útiles luego para la configuración de la posición inicial del Modelo Activo de Apariencia (AAM).
Para este proceso se utilizará la técnica de **Clasificadores en cascada** y **descriptores Haar**, entrenados con **AdaBoost**.

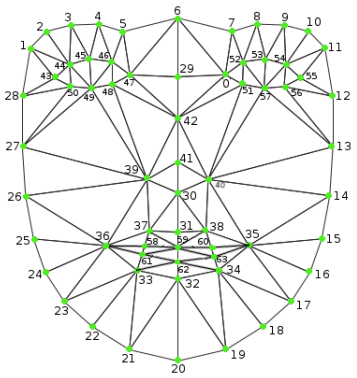
Módulo 2: Descriptor de expresión: Modelos Activos de Apariencia

Este será el módulo principal de la aplicación. En el se obtendrá el descriptor de la expresión, que luego será clasificado por el módulo siguiente (SVM). En una primera instancia se probó utilizar clasificadores en cascada para obtener los puntos característicos de la cara, luego fue remplazado por Modelos Activos de Apariencia para lograr un mejor resultado.

- **Adabost y puntos característicos:** Se explicará cómo utilizar los clasificadores Adaboost y cómo entrenarlos para obtener los puntos característicos. Se mencionarán algunos resultados de este método.

- **Modelos Activos de Apariencia:** Se creará y entrenará un modelo facial que resulte útil para representar una expresión. Se utilizarán puntos en lugares estratégicos, como ojos, cejas, boca, mandíbula, etc. Con este modelo creado, el siguiente paso es ajustarlo y llevarlo a la forma y expresión más próxima a la imagen obtenida desde la webcam (proceso de ajuste).
  Realizado esto, se utilizarán distintos descriptores obtenidos desde el AAM como descriptores de expresión. Este vector será el parámetro de salida del módulo, y el de entrada de módulo siguiente.

(a) Puntos característicos



(b) Modelo de Formas de cara - AAM

Módulo 3: Clasificador de Expresión con Máquinas de Soporte Vectorial (SVM)

Este módulo es el encargado de clasificar la expresión. Como parámetro de entrada toma el descriptor obtenido desde el AAM. Utilizando este descriptor, clasifica la expresión de la iteración actual en una de las 5 clases posibles: *Neutro, Alegría, Sorpresa, Tristeza y Enojo.*

2.3. Diagrama General de la Aplicación

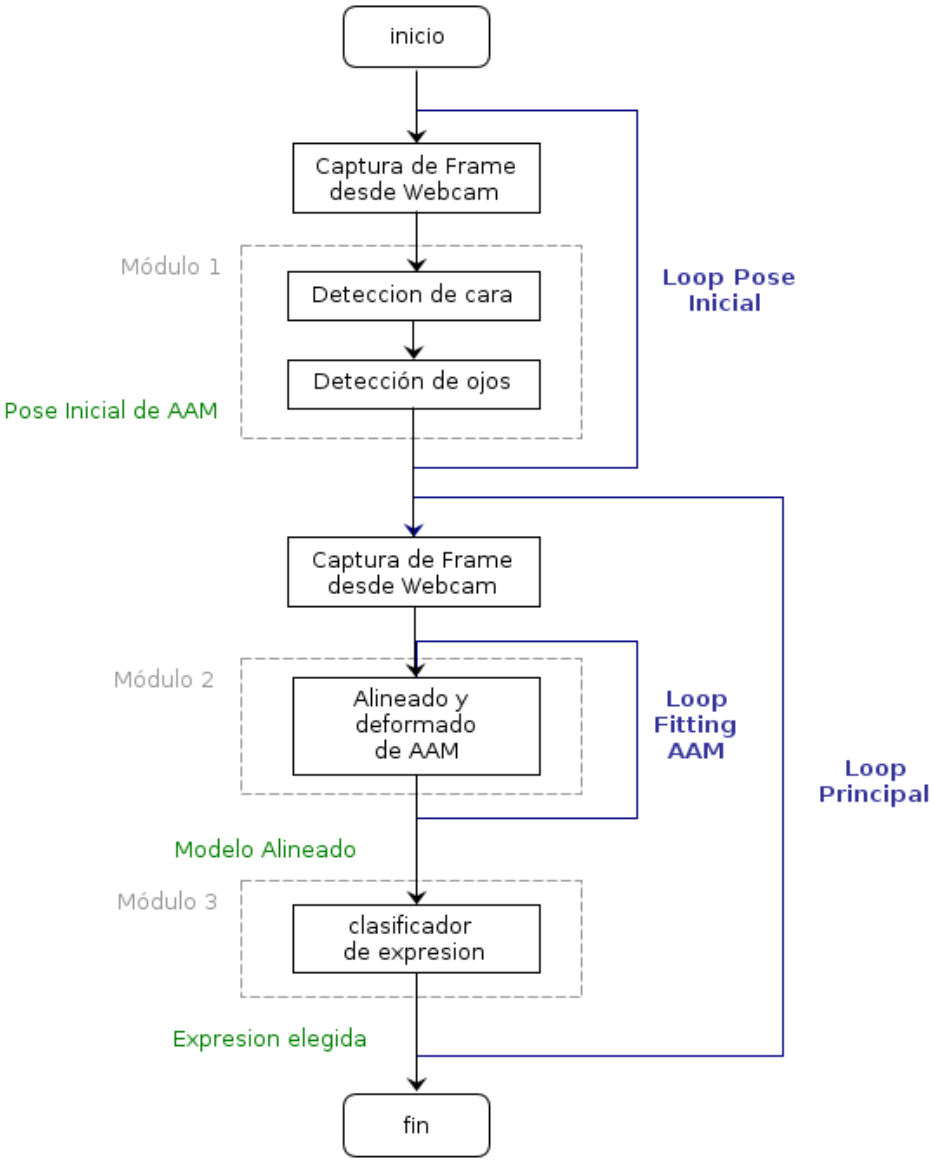


Figura 2.1: Diagrama de la aplicación

Capítulo 3

Reconocimientos de Puntos Característicos con Adaboost

Para clasificar expresiones, se necesita poder representar una expresión de alguna manera que describa sus características más importantes. Para ello se necesita algún tipo de descriptor que pueda ser capaz de discriminar entre las 5 expresiones distintas: Neutro, Alegría, Sorpresa, Tristeza y Enojo. En esta sección describimos cómo obtener los puntos característicos y cómo los resultados nos impiden utilizarlos para el reconocimiento de expresiones.

3.1. Haar-like Features y Adaboost

3.1.1. Haar-Like Features (HLF)

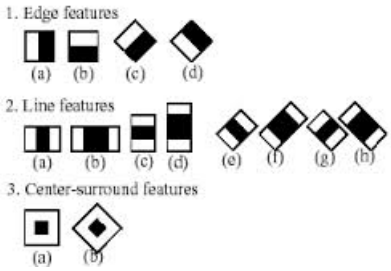
Para la clasificación de objetos, utilizamos *features* o características. Hay varias motivaciones para utilizar características en vez de los píxeles directamente, que se describen en [2], [1] y [12]. Supongamos un área dividida en dos rectángulos adyacentes (blanco y negro, ver en Figura 3.1 (a)), horizontal o verticalmente. Una *Haar-Like Feature* (simple) consiste en la resta, de la suma de los píxeles contenidos en la región marcada con un rectángulo blanco, con la suma de los píxeles contenidos en la región marcada por un rectángulo negro, en escala de grises [3].
Si consideramos RB (rectángulo blanco) y RN (rectángulo negro) entonces

$$HLF = \sum_{p_{rb} \in RB} p_{rb} - \sum_{p_{rn} \in RN} p_{rn} \tag{3.1}$$

También se pueden definir características con 3 rectángulos, o más, como también características de características.

Considerando cuadrados de aproximadamente 20 píxeles de lado, una imagen tendrá una gran cantidad de características. Para computar esta HLF puede utilizarse la siguiente aproximación.

(a) Haar-Like Features sobre imágenes de caras



(b) HLF Básicos

Figura 3.1: Haar-Like Features

Imagen integral.

Las características rectangulares, tales como las HLF, pueden ser computadas muy eficientemente utilizando una representación intermedia de la imagen. La imagen integral (II) en la posición x, y contiene la suma de los píxeles por encima y a la izquierda de la posición x, y inclusive.

$$II_{(x,y)} = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.2}$$

donde $i(x, y)$ es el valor del píxel en escala de grises de la imagen original en la posición x,y.

Utilizando las siguientes dos funciones:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{3.3}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{3.4}$$

donde $s(x, y)$ es la acumulación por columna, $s(x, -1) = 0$ y $II(-1, y) = 0$, la imagen integral puede ser computada iterando una sola vez sobre la imagen.

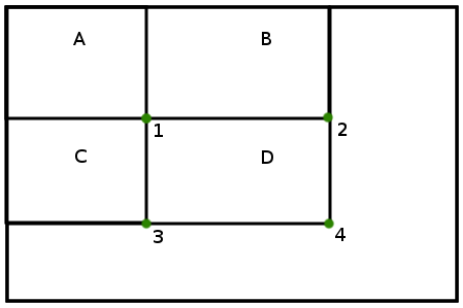


Figura 3.2: Ejemplo de calculo de la suma de los píxeles en el rectángulo D.

En la Figura 3.2 La suma de los píxeles en el rectángulo D, puede ser computada con 4 referencias de arreglo. El valor de la imagen integral en la posición 1, es la suma de los píxeles en el rectángulo A. El valor en la posición 2, es $A + B$. En la posición 3 $A + C$, y

en 4 es $A + B + C + D$. Entonces la suma de los píxeles dentro del rectángulo D, se puede calcular con 4 accesos al arreglo (imagen): $4 + 1 - (2 + 3)$.

HLFs como clasificadores débiles

Sea x, una región de una imagen, podemos obtener un valor de la HLF asociada. Podemos utilizar dicha HLF para construir un clasificador, donde llamamos clasificador a una función

$$h_c(x) = \begin{cases} 1 & \text{si } x \in c \\ -1 & \text{si } x \notin c \end{cases} \tag{3.5}$$

dónde es $h_c(x)$ vale 1 si x pertenece a la clase c o -1 si no.
Entonces si llamamos $HLF(x)$ al valor del HLF obtenido sobre la región de una imagen x, c a la clase que se desea discriminar, podemos construir un clasificador h de la siguiente manera:

$$h_c(x) = \begin{cases} 1 & \text{si } pHLF(x) \leq p\theta \\ -1 & sino \end{cases} \tag{3.6}$$

donde x es una subventana de la imagen, $HLF(x)$ es el resultado de calcular la HLF sobre la ventana x, p es el signo de la ecuación y θ es el parámetro de umbral, el que determina si la subventana x pertenece o no a la clase c.

Obtenemos de esta manera un clasificador, que determina si una región de una imagen pertenece o no a una clase. Sin embargo estos clasificadores se conocen como *Clasificadores Débiles*, ya que son muy imprecisos a la hora de detectar si una región de la imagen pertenece o no a una determinada clase (o es un determinado objeto). Para construir un clasificador que mejore la precisión de estos clasificadores y poder combinar varios de ellos utilizamos la técnica de AdaBoost.

3.1.2. Clasificador AdaBoost

AdaBoost (Adaptative Boosting) es una técnica de *aprendizaje automático* utilizada para construir un *Clasificador Fuerte* a partir de varios *Clasificadores Débiles* [4] y [15]. Dicho clasificador Fuerte se construye a partir de una combinación lineal de estos clasificadores débiles (HLFs, en nuestro caso).

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{3.7}$$

en dónde $H(x)$ es el clasificador fuerte, cada $h_t(x) \rightarrow \{-1, 1\}$ son los clasificadores débiles. Mientras cada uno de los clasificadores débiles por si solo no es capaz de detectar la pertenencia (con bajo porcentaje de error) de una porción de una imagen a un objeto o clase, si lo es el clasificador fuerte previamente entrenado (ver Algoritmo 1).

El clasificador fuerte entrenado será:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{3.8}$$

Algorithm 1 Algoritmo de entrenamiento de AdaBoost.

Sean: $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m); x_i \in X, y_i \in \{0, 1\}$ etiquetas, y h_i *clasificadores débiles*, t iteraciones y D_i conjunto de pesos en la iteración i

1: Inicializamos pesos en: $D_1(i) = 1/m$
2: **for** $t \in 1..T$ **do**
3: Buscar $h_t = \arg\ min_{h_j \in H}\ \epsilon_j = \sum_{i=1}^{m}\ D_t(i)[y_i \neq h_j(x_i)]$
4: **if** $e_t \geq 1/2$ **then**
5: parar
6: **end if**
7: $\alpha_t = \frac{1}{2}\ log((1 - e_t)/e_t)$
8: $D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{2\sqrt{e_t(1-e_t)}}$
9: **end for**
10: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$ {donde H, construido con los valores resultantes de los α_i es el clasificador fuerte entrenado.}

Para utilizarlo como un clasificador binario, podemos entonces utilizar un umbral como en el caso de los clasificadores débiles.

3.1.3. Clasificador en Cascada

Por último para que este método de reconocimiento de objetos en imágenes sea más veloz y eficiente, y poder así usarlo en tiempo real, se utiliza una estructura de cascada. Cada estado de la cascada será un clasificador entrenado por Adaboost, actualizando los umbrales para descartar falsos negativos. Se prioriza entonces no descartar muestras incorrectamente (falsos negativos) contra aceptar muestras negativas en ese estado, ya que estas deberán ser analizadas por los siguientes clasificadores, siendo así descartadas probablemente en un estado posterior [24]. Ubicando los clasificadores más restrictivos en los niveles más altos de la cascada se pueden descartar regiones de la imagen más rápidamente sin utilizar los siguientes clasificadores (3.3).

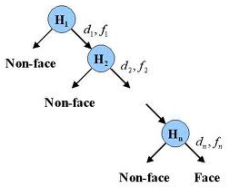


Figura 3.3: [24]

3.1.4. Procedimiento de búsqueda

La detección de objetos se hace entonces deslizando la ventana sobre la imagen de izquierda a derecha y de arriba hacia abajo, achicando de a poco la ventana y aplicando el clasificador en cascada. Si el clasificador da positivo, el objeto se encuentra en el área dada y se detiene la búsqueda.
También se utilizan dos límites para el tamaño del clasificador, superior e inferior (se acelera mucho el proceso de búsqueda si se conoce de antemano el tamaño aproximado que va a tener el objeto en la imagen). Por último, y para centrar mejor la ROI con el objeto, se utiliza un parámetro de vecinos que cumplen el clasificador. Esto requiere que el resultado de evaluar el clasificador desplazado en las 4 direcciones, también sea positivo, logrando que la ROI termine centrada.

Por último se busca restringir lo más posible la zona de búsqueda. Esto acelera mucho el proceso. Por ejemplo si buscamos ojos en una zona que sabemos que es de la cara, buscar solo en la mitad superior.

3.2. Reconocimiento de Puntos característicos utilizando *Clasificadores en Cascada.*

En esta etapa se intentan localizar 20 puntos característicos sobre la cara (Figura 3.4).

- 3 por ceja: extremos y centro. Total: 6
- 2 para nariz: Uno en cada fosa nasal. total: 2
- 4 por ojo: extremo izquierdo y derecho, párpados arriba y abajo en el centro. Total: 8
- 4 para la boca: extremo izquierdo y derecho. Centro labio superior e inferior.

Figura 3.4: Puntos característicos marcados automáticamente sobre la cara, con clasificadores en cascada entrenados con AdaBoost.

Para detectar los puntos, se entrenaron 13 clasificadores de cascada (12 referentes a los puntos característicos y 1 para la detección de la cara). Se utilizan 12 en vez de 20 gracias a la simetría facial, varios clasificadores pueden ser utilizados para ambos hemisferios, izquierdo y derecho. Estos 12 clasificadores son, los 4 de la boca, 1 para las fosas nasales, 3 para las dos cejas (extremos y centro), y los 4 respectivos a un ojo, que se utilizaran tanto para el ojo izquierdo como para el derecho.

Entrenamiento de los clasificadores

Para entrenar cada clasificador, se utilizaron muestras que constan de un cuadrado de 16 píxels por lado, centrados en el punto que se quiere reconocer. Se tomaron entonces 120 muestras marcadas por cada uno de los 12 clasificadores en cascada. Se aplicó una ecualización de histogramas para que la detección de estos sea más robusta frente a cambios de luz y lograr mejor contraste. Estos clasificadores son todos en blanco y negro (3.5).

Figura 3.5: Ejemplo de 3 muestras utilizadas para entrenar cada uno de los 3 clasificadores de cascada correspondientes.

Antes de utilizar los 12 clasificadores para buscar los puntos característicos se transforma la imagen en blanco y negro, y se ecualiza en forma general. Se detecta el ROI de la cara, utilizando el clasificador de la cara y luego se divide esta en distintas partes, para aplicar los clasificadores individuales: mitad superior e inferior, mitad izquierda y derecha. Queda así la imagen de la cara dividida en 4 partes iguales. Luego se aplican estos clasificadores en cada cuarto. En los cuartos de arriba a su vez se vuelven a dividir en dos mitades verticalmente, superior para cejas e inferior para ojos. El Mismo procedimiento se utiliza al dividir el tercio del medio horizontal de toda la cara, para buscar los puntos de la nariz.

Este proceso puede realizarse tan rápidamente que no hace falta un proceso de seguimiento (o tracking) en las siguientes iteraciones. En cada iteración se buscan los puntos nuevamente.

Clasificadores de cascada, Adaboost y puntos característicos en reconocimiento de expresiones

Al querer reconocer 5 expresiones, los puntos característicos tiene que ser encontrados en cada una de ellas sin excepción, sino el clasificador no tendrá información suficiente para determinar a que expresión pertenece la muestra. Al cambiar de expresión, el entorno del punto en la cara se modifica, en algunos casos completamente. Esto quiere decir que, o cada clasificador tiene que ser entrenado con muestras del mismo punto en todas las expresiones, o tienen que entrenarse 60 clasificadores (12 por expresión). Esta última es inviable en términos de eficiencia, con lo cual se intentó entrenar cada clasificador, con muestras del mismo punto en las 5 expresiones distintas. Los resultados que obtuvimos eran buenos para la cara sin expresión, pero en cuanto esta comenzaba a alterarse, los clasificadores dejaban de encontrar los puntos característicos o a confundirlos con otros aledáneos. Sobre todo en puntos como los bordes de la boca, que se alteran casi totalmente entre la expresión neutra, y por ejemplo, la de alegría (con sonrisa).

Por otro lado, la técnica de detección de objetos utilizando clasificadores a partir de descriptores o características Haar, funciona especialmente bien cuando los objetos que se intentan reconocer tienen una forma particular y no se parecen mucho al fondo o entorno en donde habitan. Como es el caso de una cara en un fondo cualquiera, o un ojo dentro de una cara, ambos son fácilmente discriminables respecto al entorno. Sin embargo, puntos como bordes de la boca, no tienen un cambio significativo con los puntos aledaños a ella. Tampoco puntos característicos con respecto a puntos aledaños a ellos (ej. borde de párpado, con centro de párpado). Esta poca precisión hace que los puntos se muevan, lo que genera confusión al querer clasificar una expresión. Un ejemplo gráfico de este tipo de errores puede verse en Figura 3.6.

Por estas razones es que se abandonó el método de detección de puntos característicos a partir de clasificadores entrenados con HLF, para la construcción del descriptor facial y sólo se utilizó para la estimación de la pose inicial del AAM, técnica explicada en los capítulos siguientes.

Figura 3.6: Ejemplo de un punto que no fue encontrado por el clasificador, y otro en una posición incorrecta.

Capítulo 4

Modelos Activos de Apariencia (AAM)

Como vimos anteriormente, necesitamos un modelo más robusto a la alteración de las expresiones. Modelos como los anteriores, se conocen como modelos abiertos en donde la posición de un punto no esta restringida por la posición del resto. Se propone ahora la utilización de Modelos Activos de Apariencia, en dónde la idea principal es deformar un modelo de formas, únicamente en direcciones provenientes de una etapa de entrenamiento, asegurando de esta manera que los modelos generados sean expresiones reales. Quitamos así la posibilidad de que desaparezcan puntos o que puntos se desplacen de una manera incoherente respecto del resto del modelo.

A diferencia del modelo anterior, en dónde los puntos se buscan dentro de la imagen, los AAM se generan y se ajustan a ella.

La responsabilidad de esta etapa de la aplicación será utilizar los AAM para extraer información de la expresión de una imagen o video. Esto se logra ajustando el AAM a la cara que esta en la imagen, y extrayendo información del modelo de formas ajustado, como por ejemplo distancias entre los puntos característicos. De esta manera podremos utilizar esa información para clasificarla luego y determinar a que expresión pertenece.

Los Modelos Activos de Apariencia son un método estadístico de ajuste entre un modelo (capaz de representar forma y apariencia), y una imagen dada [6], [5], [7], [11] y [8]. En los AAM las variaciones de forma y de textura son capturadas a partir de un conjunto representativo de entrenamiento. Se busca entonces realizar entrenamientos que abarquen todas las transformaciones posibles que el modelo tiene que ser capaz de reproducir, a la hora del ajuste con la imagen. En este trabajo, se desea detectar 5 expresiones: Neutro, Alegría, Sorpresa, Tristeza y Enojo. Los modelos son entrenados con muestras que abarcan las distintas expresiones, como también algún grado de rotación, traslación y escala (alteraciones de posición que puede tener una persona frente a la webcam).

4.1. Estado del arte

Modelos deformables capaces de ajustarse a imágenes de entrada tienen un gran interés en la visión por computadora. Trabajos iniciales sobre este tipo de modelos fueron los *Active Shape Models* [10]. Luego estos modelos fueron modificados al incluirse aspectos de apariencia creando los *Active Appearance Models* [11]. Se desarrollaron entonces métodos de ajuste para estos modelos permitiéndoles ajustarse a imágenes de entrada en tiempo real. El primer trabajo de ajuste fue propuesto en [8], luego otros desarrollos sobre algoritmos de ajuste pueden verse en [9]. Utilizaciones de modelos activos de apariencia con variaciones en texturas fueron propuestas por [6] y [7]. Por último modelos activos de apariencia utilizados en reconocimiento de expresiones y estimación de pose fueron propuestos en [5].

4.2. Modelos Activos de Apariencia

Los Modelos Activos de Apariencia entonces pueden ser construidos con 3 componentes principales:

- Modelo de Formas.
- Modelo de Apariencia o Textura.
- Transformación general de pose.

4.3. Modelo de Formas

Una *forma* puede ser representada como un conjunto de n puntos marcados, definidos en $\Re^k$ [5] [10]. Entonces una instancia de nuestro *modelo de formas*, puede ser representada por un vector de dimensión nk. En nuestro caso, con imágenes 2-D, $k = 2$.

Para una imagen 2-D podemos representar a n puntos marcados $\{(x_1, y_1), ..., (x_n, y_n)\}$, para una sola instancia del modelo de formas, por un vector $\boldsymbol{v}$ de longitud $2n$ dónde

$$v = (x_1, y_1, x_2, y_2, ..., x_{n-1}, y_{n-1}, x_n, y_n)^T \tag{4.1}$$

Notar que no hay información de las relaciones entre los puntos. Estas relaciones (líneas que unen los puntos, ver Figura 4.1) serán utilizadas y definidas luego en el modelo de texturas. En este trabajo, se utilizó el un modelo de formas para representar la cara.

Este modelo consta de 64 puntos marcados, interconectados como se ve en la figura, formando 98 triángulos utilizados luego en el modelo de apariencia. Como se utilizaron imágenes 2-D, $k = 2$, con lo cual, una sola instancia de nuestro *modelo de formas* es representada por un vector de longitud $2 * 64 = 128$.

4.3.1. Entrenamiento del modelo de formas

Supongamos ahora que tenemos $\{v_1, ..., v_s\}$ $\boldsymbol{s}$ instancias etiquetadas (cada punto marcado manualmente) de un conjunto de entrenamiento. Cada v_i será en nuestro caso, un vector

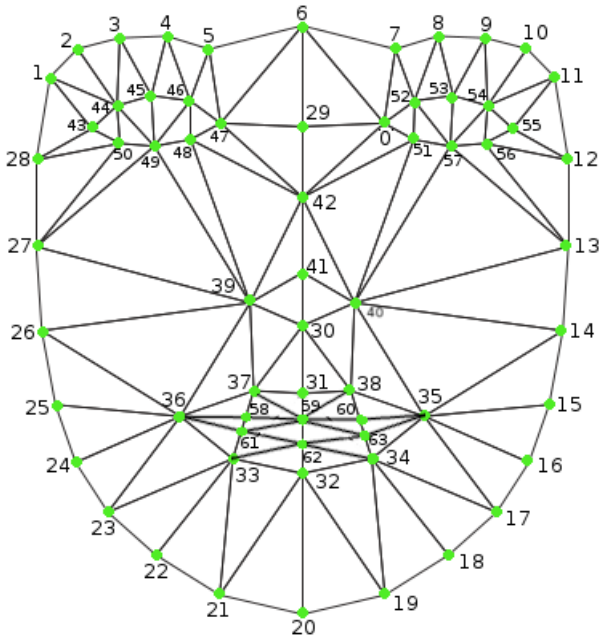


Figura 4.1: Modelo de Formas utilizado para la cara.

de 128 elementos. Para poder tener una representación estadística, es importante que todas las muestras tengan el mismo sistema de referencia. Para esto, se tiene que remover la posición, rotación y escala de cada muestra, llevando a las s muestras a un mismo sistema de referencias.

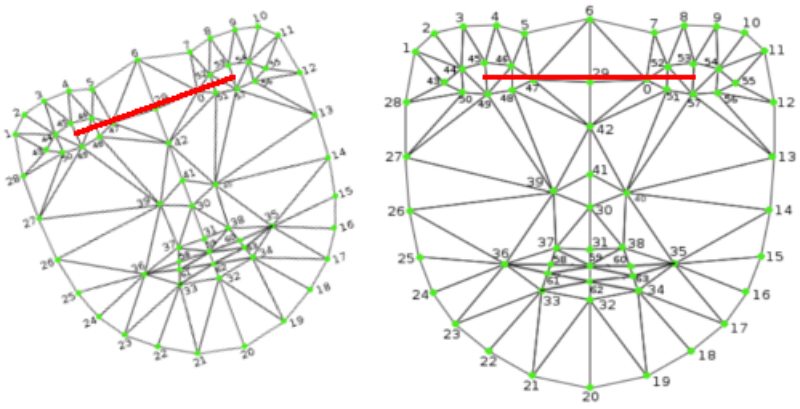


Figura 4.2: Puntos utilizados para alineación de modelos de forma.

En este trabajo, y por la naturaleza del modelo, elegimos alinear cada una de las s muestras entre sí en relación a a la posición de los ojos (ver Figura 4.2). Elegimos una instancia o muestra inicial, y luego calculamos la escala, rotación y traslación que llevan a las siguientes muestras a la posición de la inicial. Para ello, trazamos una linea entre los puntos centrales de cada ojo, izquierdo y derecho. Con ella, calculamos el ángulo de inclinación y la escala que se deberá aplicar a una de las dos formas para llevarla a la misma escala y rotación de

la segunda. Luego basta con superponer la posición de los ojos, desplazando a uno de los dos modelos en el eje x e y lo necesario. Esta decisión se tomo a partir de que en una cara, la posición de los ojos y distancia entre ellos se mantiene fija en el cambio de expresiones, siendo útil para el alineado de varias muestras.

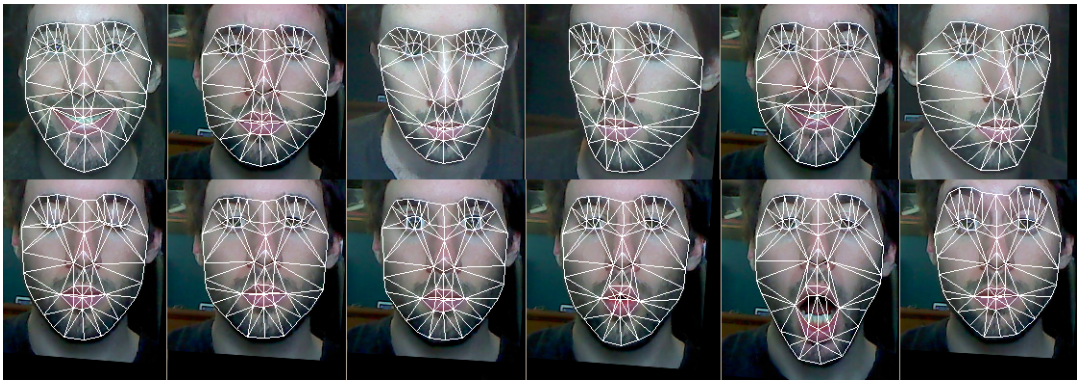


Figura 4.3: Imágenes alineadas en el proceso de entrenamiento

4.3.2. Análisis de Componentes Principales (PCA)

Cuando tenemos todas las muestras de los modelos alineadas y en un mismo sistema de referencia, se puede analizar la variación de los puntos en el conjunto de datos. Esto nos permite calcular su varianza utilizando *Análisis de Componentes Principales*. La utilidad principal que PCA nos trae, es poder representar el modelo (cada vector de $2n$ elementos, en este trabajo $2n = 2 * 64 = 128$) con un subconjunto mínimo de parámetros.

El *Análisis de Componentes Principales* es una técnica que nos permite reducir la dimensión de los datos. Este procedimiento busca las direcciones de mayor varianza y proyecta los datos en ellas.

Matemáticamente se define como una transformación lineal ortogonal que proyecta datos en un nuevo sistema de coordenadas definido por los ejes de coordenadas de las varianzas. La *reducción de dimensión* se obtiene dejando los datos que más contribuyen en la varianza, e ignorando el resto.

Consideremos un conjunto de entrenamiento de N vectores $\{v_i, ..., v_N\}$ dónde cada vector es de dimensión n (128 en este trabajo). Se requiere que el número de muestras sea mayor que el de la dimensión ($N > n$).

Algoritmo de PCA: ver Algoritmo 2.

Algorithm 2 Análisis de Componentes Principales.

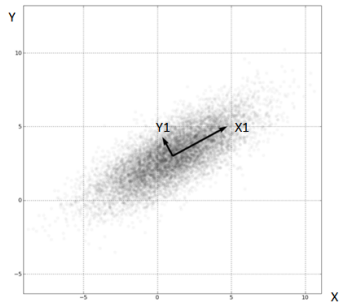
1: $\overline{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$ {Computamos el promedio de todas las muestras}
2: $C = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})(x_i - \overline{x})^T$ {Calculamos la matriz de covarianza}
3: Computamos los *autovectores* ϕ_i, y sus *autovalores* asociados λ_i, en orden tal que $\lambda_i \geq \lambda_{i+1}$. El autovector correspondiente al autovalor λ_1 corresponde a la dirección de mayor variación, el segundo al siguiente, etc. Algunos autovectores están asociados a autovalores con valores muy pequeños, estos pueden ser descartados, logrando así la reducción de datos.
4: Dejando en Φ los t autovectores asociados a los t autovalores más importantes, podemos reconstruir (o re-proyectar) cualquier valor del conjunto de entrenamiento como

$$x = \overline{x} + \Phi b \tag{4.2}$$

dónde $\Phi = (\phi_1|\phi_2|...|\phi_t)$ es una matriz de dimensión $n \times t$ (con n dimensión original de cada muestra) y b es un vector de dimensión t (o $1 \times t$), y $\overline{x}$ de dimensión $n \times 1$, quedando (en dimensiones): $(n \times 1) + (n \times t)(t \times 1) = (n \times 1)$ la muestra reproyectada en el espacio original. El vector b podría ser reconstruido de la siguiente manera

$$b = \Phi^{-1}(x - \overline{x}) = \Phi^T(x - \overline{x}) \tag{4.3}$$

Ejemplo de PCA



La imagen muestra las dos componentes principales, x_1, y_1 (PC) de un conjunto de puntos 2-D. Se puede ver como cada punto de la nube puede ser calculado como combinación lineal de ambos ejes x_1, y_1 (autovectores de mayor autovalor asociado)

$$x_i = \overline{x} + \sum_{j=1}^{t} \phi_j b \tag{4.4}$$

Observación: En OpenCV, las dimensiones de las matrices están invertidas, teniendo: $x_i = \overline{x} + \sum_{j=1}^{t} b^t \phi_j$ con dimensiones: $(1 \times n) = (1 \times n) + (t \times n)(n \times 1)$.

Número de PC

El número de componentes principales elegidos depende de las variaciones que quieren tenerse en cuenta en el modelo.
Aplicando *pca* en los datos de muestra alineados, podemos representar una instancia del modelo x

$$x = \overline{x} + \Phi b_s \tag{4.5}$$

donde x es la re proyección de la instancia, y $\overline{x}$ es la instancia (forma) promedio. Entonces podemos tener una representación de la instancia x, de dimensión $2n$, dada por el vector b de dimensión t, con $(t \leq 2n)$.

En nuestro caso, tenemos cada vector del modelo de formas $v = \{x_1, y_1, ..., x_{64}, y_{64}\}$ de longitud 128. $\overline{v}$ el modelo de formas promedio también de longitud 128, podría ser representado entonces por un vector b de por ejemplo tan solo 14 componentes

$$v = \overline{v} + \Phi b_{14} \tag{4.6}$$

con dimensiones: $(128 \times 1) = (128 \times 1) + (128 \times 14)(14 \times 1)$.

El vector b entonces es la representación de forma del *modelo de forma*. Alterando los valores de b, obtenemos distintos modelos, con valores que varían en las distintas direcciones obtenidas en las instancias de entrenamiento (Figura 4.4).

Agregando límite a los valores construidos por el modelo

La varianza de cada dirección ϕ_i obtenidas por el PCA en el conjunto de entrenamiento está dada por λ_i. Utilizando como cota $\pm 3\sqrt{\lambda_i}$ para cada valor de b_i correspondiente, aseguramos que el modelo de forma creado es similar al del conjunto de muestras de entrenamiento (no se generan modelos deformes) [5].

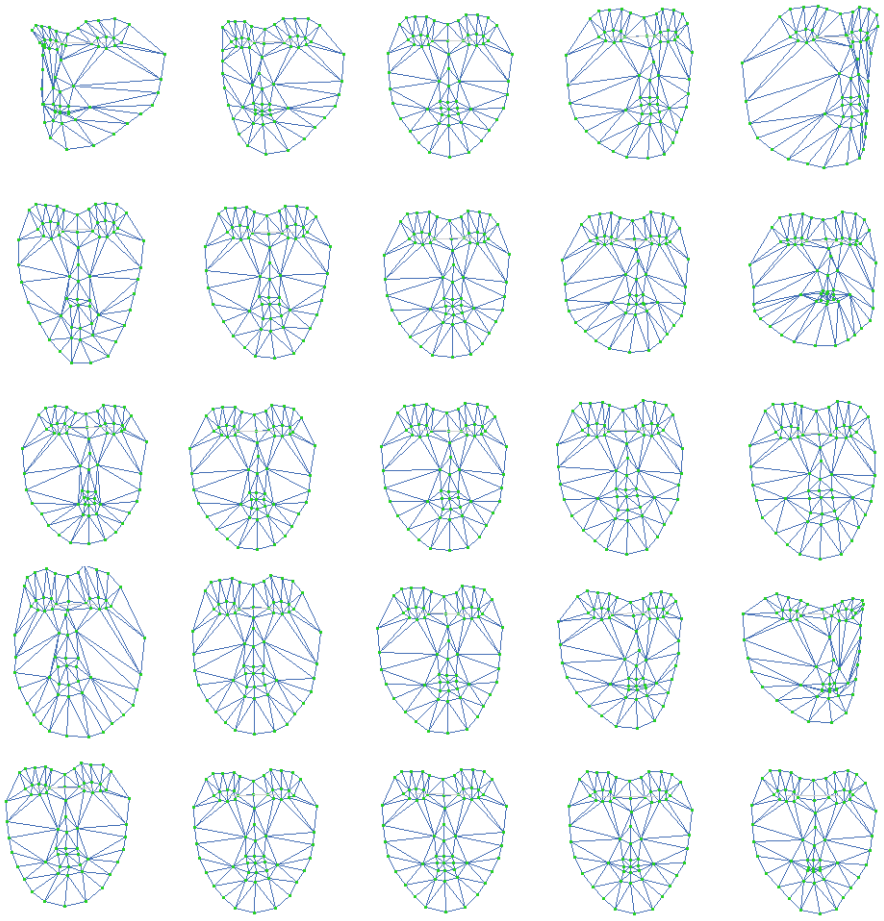


Figura 4.4: Variaciones de 5 PC (uno por línea), de izquierda a derecha variando $-3\sqrt{\lambda_i}$ - $+3\sqrt{\lambda_i}$. Se pueden ver los modelos de formas generados a partir de estas variaciones.

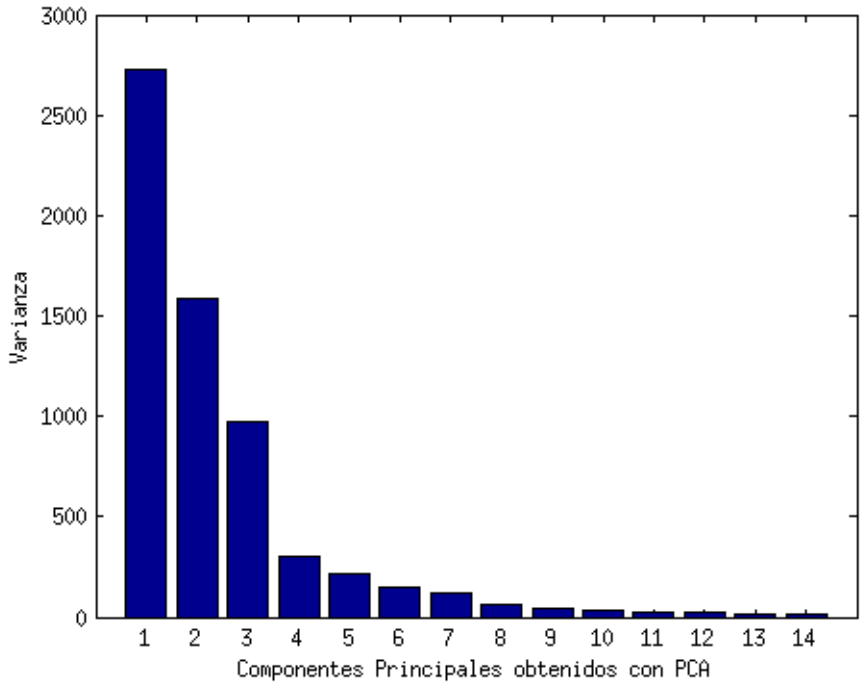


Figura 4.5: Decaimiento de la varianza, vs componentes principales (de mayor a menor)

De esta manera conseguimos no solo representar al modelo de formas con un subconjunto de parámetros, sino que también restringimos al modelo a deformarse en direcciones específicas que provienen del conjunto de muestras de entrenamiento. En este trabajo se utilizaron muestras con distintas expresiones para lograr que las direcciones principales en las que se deformaba el modelo de formas nos generen cada una de las distintas expresiones a reconocer.

4.4. Modelos de Textura

Para construir un modelo completo, no sólo tenemos que representar la forma, sino que también la apariencia. En esta sección se describe como construir un modelo de textura. Podemos considerar como textura, al contenido del modelo de formas, entre determinados conjuntos de puntos.
Del mismo modo que los modelos de formas, el modelo de textura requiere que todas las texturas estén alineadas dentro del mismo sistema de referencias.

Texturas

Las texturas son especialmente útiles a la hora del ajuste del modelo (proceso de fitting), en el cual las direcciones hacia donde será ajustado el modelo de formas, serán obtenidas a partir de los residuos de textura.

Las texturas, en los AAM, son vectores que almacenan el valor de los píxeles contenidos dentro de triángulos definidos en el modelo de formas. Estos triángulos tienen como vértices a 3 puntos del modelo de formas. Se elige la combinación de puntos de forma tal que los triángulos no se superpongan. También se intenta subdividir en mayor cantidad de triángulos las áreas en dónde la textura contiene mayor detalle. En nuestro caso esto sucede por ejemplo en el área de la boca, ojos, etc (Figura 4.1).

Para almacenar la textura se elige un tamaño de *lienzo*, para este trabajo de 120 píxeles de lado, en donde estará contenido el modelo de formas. Con lo cual nuestra textura será almacenada también como un vector de 120*120=14400 píxeles.

Como ya contamos con un modelo de formas, interesa que el modelo de texturas sea libre de forma e independiente de la pose en la que se encuentra el modelo. Para esto las texturas son mapeadas de una forma en donde todos los puntos del modelo de formas se encuentran en las mismas coordenadas. Esto se logra haciendo una proyección de la textura al modelo de formas promedio calculado anteriormente. En este trabajo, inicialmente, se utilizaron texturas en escala de grises, pero luego fueron modificadas por texturas a color, para tener mejor precisión. Para poder hacer este proceso fácilmente, se utilizaron secciones triangulares dentro del modelo para hacer las transformaciones que proyectan un triángulo en una pose dada, al mismo triángulo en referencia al modelo de formas promedio (Figura 4.6).

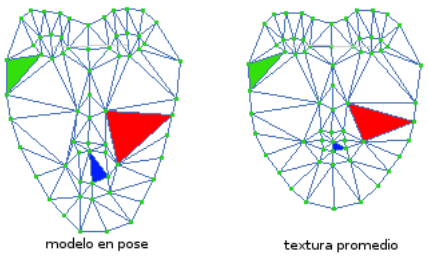


Figura 4.6: Ejemplo del mapeo de 3 triángulos del modelo de formas en una determinada pose, con los 3 triángulos respectivos en el modelo de formas promedio.

4.4.1. Mapeo de textura

En el mapeo de texturas, la configuración espacial de una imagen es transformada en la de otra. Formalmente definimos $I \in \Re^k \rightarrow I' \in \Re^k$ con $k = 2$ en nuestro caso, y en el caso de imágenes 2-D en general.

En Modelos Activos de Apariencia, cada punto del modelo de formas $\{v_1, ..., v_n\}$ es mapeado en $\{v'_1, ..., v'_n\}$. Entonces podemos escribir una función de mapeo

$$f(v_i) = v'_i, \forall i = 1, .., n. \tag{4.7}$$

en la práctica, si proyectamos una textura de un triángulo pequeño a uno de mayor tamaño, utilizando la función $f(v_i)$, tendríamos agujeros, ya que no todos los píxeles podrían ser proyectados desde la imagen fuente. Con lo cual, nos conviene utilizar la función inversa $f'(v'_i) = v_i, \forall i = 1, .., n.$ que mapea de $\{v'_1, ..., v'_n\} \rightarrow \{v_1, ..., v_n\}$.

4.4.2. Coordenadas Barométricas

Las Coordenadas Barométricas sirven para saber a qué triángulo pertenece un punto $p = [p_x, p_y]^T$.
En *coordenadas barométricas* un punto puede ser expresado en función de sus vértices

$$p = f(p) = v_1 + \beta(v_2 - v_1) + \gamma(v_3 - v_1) = \alpha v_1 + \beta v_2 + \gamma v_3 \tag{4.8}$$

dónde $v_1 = (v_{1x}, v_{1y}), v_2 = (v_{2x}, v_{2y})$ y $v_3 = (v_{3x}, v_{3y})$ son los tres vértices del triángulo y β, γ y α son números reales tales que $\beta + \gamma + \alpha = 1$. Los coeficientes β, γ y α son llamados coordenadas barométricas del punto p en relación a v_1, v_2 y v_3. Entonces podemos armar un sistema de 3 ecuaciones con 3 incógnitas

$$\begin{cases} \alpha v_{1x} + \beta v_{2x} + \gamma v_{3x} = p_x \\ \alpha y_{1y} + \beta y_{2y} + \gamma y_{3y} = p_y \\ \alpha + \beta + \gamma = 1 \end{cases} \tag{4.9}$$

con solución

$$\begin{aligned} \alpha &= 1 - (\beta + \gamma) \\ \beta &= \frac{p_y v_{3x} - v_{1x} p_y - v_{3x} v_{1y} - v_{3y} p_x + v_{1x} v_{3y} + p_x v_{1y}}{-v_{2x} v_{3y} + v_{2x} v_{1y} + v_{1x} v_{3y} + v_{3x} v_{2y} - v_{3x} v_{1y} - v_{1x} v_{2y}} \\ \gamma &= \frac{p_x v_{2y} - p_x v_{1y} - v_{1x} v_{2y} - v_{2x} p_y + v_{2x} v_{1y} + v_{1x} p_y}{-v_{2x} v_{3y} + v_{2x} v_{1y} + v_{1x} v_{3y} + v_{3x} v_{y2} - v_{3x} v_{1y} - v_{1x} v_{2y}} \end{aligned} \tag{4.10}$$

Para buscar si un punto p pertenece a un triángulo con vértices v_1, v_2 y v_3, sus coordenadas barométricas β, γ y α tienen que estar en el rango $0 \leq \alpha, \beta, \gamma \leq 1$.

En la práctica, utilizamos la transformación inversa para cubrir agujeros, esto se hace mapeando cada píxel de cada triángulo (del modelo de formas promedio) con una *transformación affine* [16] y su matriz respectiva, que lo proyecta en el mismo triángulo, en el modelo de formas de la instancia [**?**].
Entonces podemos construir a partir de los 3 puntos del triángulo fuente, y los 3 puntos del

triángulo destino una matriz de transformación affine que nos servirá para proyectar cada píxel del triángulo fuente al píxel correspondiente dentro del triángulo destino. De esta manera podemos construir inicialmente una matriz de transformación por cada triángulo de cada textura, y aplicarla (operación de multiplicación) a cada punto del triángulo correspondiente, haciendo el proceso más eficiente.

Finalmente podemos utilizar las coordenadas baromátricas para hacer las proyecciones de textura. Buscamos entonces proyectar la textura de un modelo de formas, en otro modelo de formas. Generalmente el modelo de formas destino, será el modelo de formas promedio. En dicho proceso necesitamos proyectar cada triángulo del modelo de formas fuente, al mismo triángulo en el modelo de formas destino. El algoritmo de proyección de texturas entre modelos de formas se puede ver en el Algoritmo 3.

Algorithm 3 :**Proj(**$MF_{destino}$**,**MF_{fuente}**,**I**)**:Algoritmo de proyección de textura.

1: **for** cada píxel $p = [p_x, p_y]^T$ dentro de la textura del modelo de formas $MF_{destino}$ **do**
2: Buscar el triángulo t al que pertenece el punto p (Coordenadas Baromátricas) en el $MF_{destino}$.
3: Usar la fórmula $f(p)$ (4.4.1) para determinar la posición relativa dentro del triángulo t' perteneciente a MF_{fuente}
4: Establecer $I'(p) = I(f(p))$.
5: **end for**
6: **return** $I'\{I'$ es la proyección de la textura de MF_{fuente} en el $MF_{destino}\}$

En está implementación inicialmente se utilizó opencv con las funciones getAffineTransform() y warpAffine(), luego, y por eficiencia, se utilizó openGL para las proyecciones de texturas, dejándole esta tarea a la placa de video de la PC.

Ejemplos de proyecciones de textura

Figura 4.7: Ejemplo de imagen original, y textura proyectada en el modelo de formas promedio, de izquierda a derecha respectivamente.

4.4.3. Normalización

Por último, y para que la textura sea más invariante a perturbaciones como por ejemplo la luz y detalles de calidad de la imagen, la textura es normalizada y esfumada.

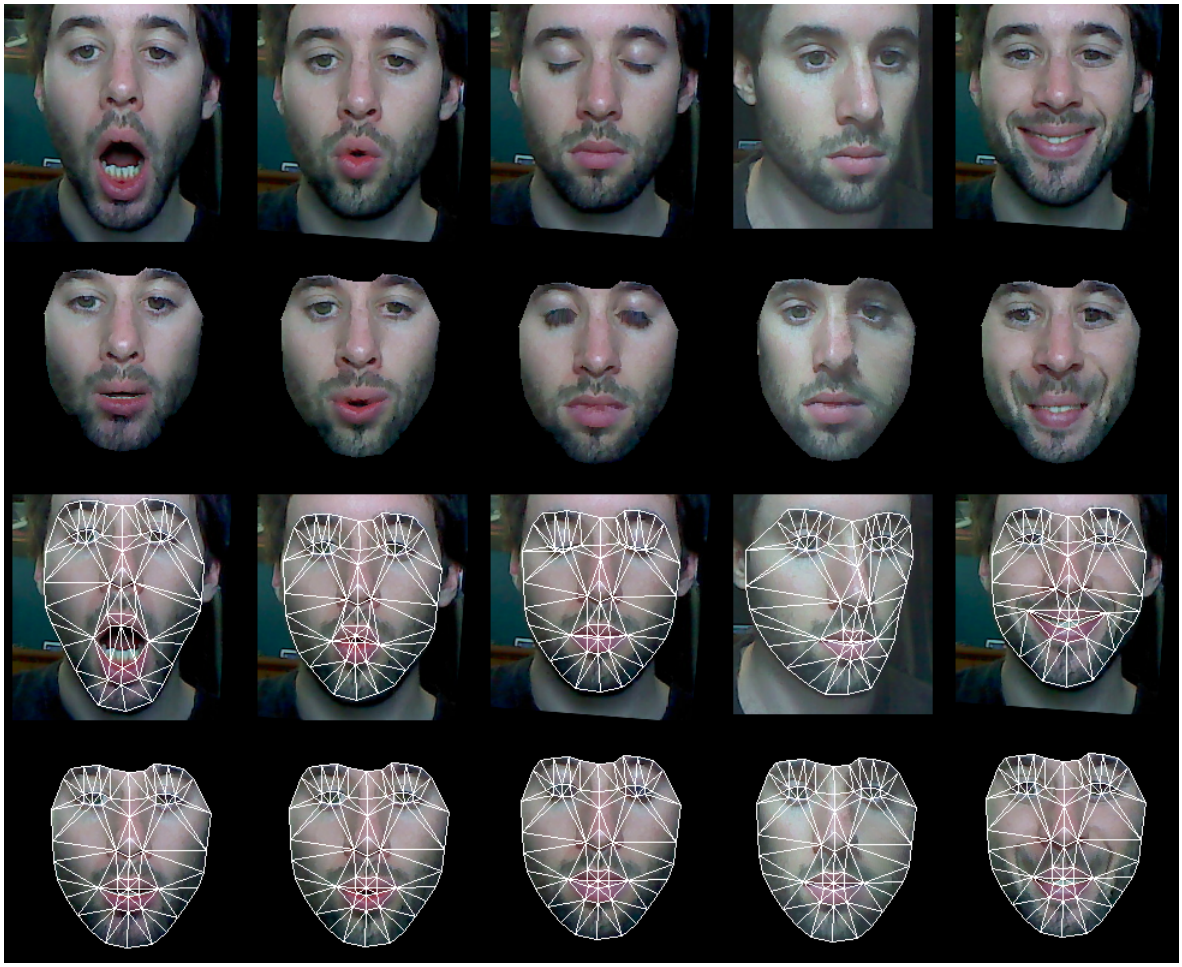


Figura 4.8: *Fila 1: Imágenes originales. Fila 2: textura proyectada en el modelo de formas promedio. Fila 3 y Fila 4, mismas imágenes con el modelo de formas dibujado encima.*

Para la normalización de color, se hace una ecualización del histograma en cada componente de color (RGB). Mientras que para tener menos errores por detalle, se aplica un filtro *Gaussiano* (Figura 4.9).

4.4.4. Modelo de Textura

El modelo de textura puede ser construido también con los mismos principios que el modelo de formas. Aplicando PCA podemos disminuir la dimensión de las variaciones de la textura. Sin embargo, y por cuestiones de eficiencia y alineado del modelo en tiempo real, utilizado específicamente para expresiones; utilizamos únicamente la textura promedio. En el proceso de entrenamiento, para cada instancia del modelo, se proyecta la textura al modelo de formas promedio. Luego todas estas muestras son sumadas y promediadas obteniendo así la textura promedio del entrenamiento. Con lo cual, nuestro modelo de texturas se reduce a un vector, del tamaño de la textura promedio. Cada elemento cuenta con 3 componentes (RGB) de cada píxel.

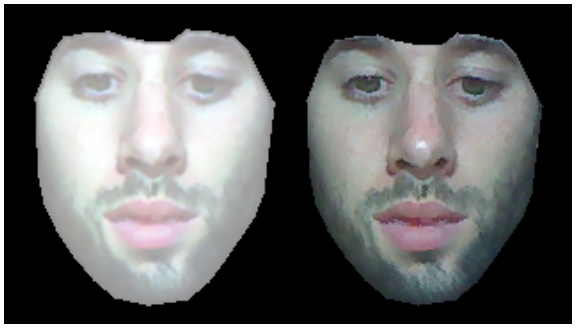


Figura 4.9: *textura normalizada vs. textura original*

En este trabajo se utilizaron texturas color de tamaño 120 píxeles por lado. Con lo cual nuestra textura promedio es un vector de $120 * 120 = 14400$ muestras de 24 bits cada una (8 bits por color).

4.4.5. Utilizando la placa de video para las proyecciones de textura

Como veremos luego en el algoritmo de ajuste del AAM, la proyección de textura es la operación más costosa y a su vez, que más se realiza en todo el proceso. Intentando realizar este proceso en tiempo real, se necesitó implementar dicha función utilizando la placa de video. De esta manera los procesadores de la placa son los responsables de las proyecciones de textura quitándole tiempo de procesamiento al procesador de la computadora. Para el método de proyección de texturas se utilizó la librería **openGL**.
Referencias del costo en milisegundos de la proyección de cada textura: $C/C++ \approx 200ms$, $OpenGL \approx 10ms$.

4.5. Transformación general de pose

Para tener una representación más limpia de la forma que adopta el modelo se intenta separar las transformaciones generales que tienen que ver con la pose, a las de la forma del modelo en sí más relacionadas a la forma. Para ello se separa el modelo de formas de la pose general del modelo, que incluye escala, rotación y traslación. Estas variables se representan con un vector

$$c = (\theta, s, t_x, t_y) \tag{4.11}$$

dónde θ es el ángulo de rotación, s es la escala y t_x y t_y parámetros de traslación. Las cuatro variables en función del modelo de formas promedio.

4.6. Modelo Activo de Apariencias Combinado

Por último, para construir nuestro modelo de apariencias entonces contamos con

- Modelo de Formas, representado por el vector de dimensión reducida b, el modelo de formas promedio $\overline{v}$ y la matriz de autovectores Φ:

$$v_i = \overline{v} + \sum_{j=1}^{t} \phi_j b \tag{4.12}$$

- Textura promedio, representado por un vector $\overline{t}$ de dimensión $120 * 120 = 14400$.
- Transformación general de Pose representada por

$$c = (\theta, s, t_x, t_y) \tag{4.13}$$

4.7. Ajuste del AAM

Supongamos ahora que tenemos una imagen de entrada $I(x)$ que tenemos que ajustar con el AAM. Esto significa buscar el mejor vector b (parámetros del modelo de formas) y c, parámetros globales de pose tal que el modelo generado a partir de estos, y la imagen (contenida entre los puntos del modelo de formas proyectado sobre ella) sean lo más similar posible. Podemos calcular el error entre la imagen y la textura en dos sistemas de referencia distintos, el sistema de referencias referente a la imagen I, o el referente al AAM. Por una cuestión de optimización decidimos utilizar el del AAM (Usando el modelo de formas promedio). Para definir el proceso de ajuste tenemos que definir más formalmente el proceso a optimizar.

Lo que se busca entonces minimizar en el proceso de ajuste es el **residuo de textura** entre el modelo de texturas y la imagen de entrada I. El modelo de texturas, se supone libre de formas ya que para calcular los residuos de textura, se hacen las proyecciones de estas en el modelo de formas promedio. Con lo cual, no importa en que forma se encuentre el modelo ni la imagen, siempre serán comparadas en misma forma, dejando solo el resultado ligado a la textura.
Si llamamos p al vector que incluye tanto los parámetros del modelo de formas b más los de pose c entonces buscamos

$$arg\ min_p |I_{image} - I_{model_p}|^2 \tag{4.14}$$

Entonces el proceso de ajuste de un AAM con una imagen es un problema de optimización en donde el residuo de textura es minimizado actualizando los parámetros de forma p.

4.7.1. Algoritmo de ajuste

Muchos métodos pueden ser usados para el proceso de ajuste [5], [9]. En este trabajo se necesita que el proceso sea eficiente para poder utilizarlo en tiempo real. Para ello necesitamos poder decidir como deformar el modelo de formas en cada iteración, de la forma más veloz posible. Se busca entonces poder precalcular la mayor cantidad de datos posibles fuera de la iteración central del ajuste. Para ello precalculamos una matriz que nos dice cómo el modelo tiene que ser actualizado o deformado en relación al residuo de textura de cada iteración.

Sean $p = (b^T|c^T)^T$ un vector que contiene los parámetros de forma b, más los 4 parámetros globales de pose.

δp perturbación de los parámetros de p del modelo, δg al residuo de textura (diferencia entre la textura promedio y la imagen contenida entre los puntos del modelo proyectada dentro del modelo de formas promedio).

Buscamos una matriz de regresión R tal que

$$\delta p = R\, \delta g \tag{4.15}$$

que mantiene la relación entre el residuo de texturas y la perturbación del modelo asociada a ese residuo [5].

4.7.2. Construcción y entrenamiento de la matriz de regresión R

Para construir la matriz de regresión R (ver Figura 4.15), generamos perturbaciones en el modelo y calculamos el residuo de textura asociado. Supongamos que tenemos s experiencias (s perturbaciones) construimos la matriz de textura y perturbación dónde Δp tiene las perturbaciones del modelo (perturbaciones al vector p), y Δg los residuos de textura correspondientes, ambas en columnas, y t_p y t_g representan la longitud del vector p, y de la textura g respectivamente.

$$\Delta p = (\delta_{p_1}|\delta_{p_2}|...|\delta_{p_s})_{t_p \times s} \tag{4.16}$$

$$\Delta g = (\delta_{g_1}|\delta_{g_2}|...|\delta_{g_n})_{t_g \times s} \tag{4.17}$$

Calcular un residuo de textura consiste en proyectar el modelo perturbado sobre la imagen, proyectar la imagen debajo de este sobre la forma promedio, y restarla con la textura promedio.

Perturbaciones del modelo para el entrenamiento

Para que cada muestra de textura este asociada a la alteración de un determinado valor del modelo de formas, cada perturbación la realizamos en un solo parámetro del vector b,

dejando los restantes en 0. El porcentaje en escala y traslación, son en referencia al modelo de formas promedio.

- **Perturbaciones modelo de formas (vector b):** Alteramos cada b_i: $\pm 0{,}25\sigma_i$,$\pm 0{,}5\sigma_i$
- **Perturbaciones de escala s:** $90\,\%, 110\,\%$
- **Perturbaciones de rotación** θ: $\pm 5°$, $\pm 10°$.
- **Perturbaciones de traslación**: t_x, t_y: $\pm 5\,\%$, $\pm 10\,\%$

Algorithm 4 Algoritmo de entrenamiento para Δp y Δg

Tomamos una instancia del modelo de la etapa de entrenamiento. Eso es una imagen etiquetada a mano I; y su respectivo modelo de formas con parámetro p (de pose y escala). Este se puede obtener proyectando el vector de puntos marcados con el PCA, obteniendo el vector b asociado. Sea g_{prom} la textura promedio calculada en el entrenamiento del modelo, y C_s el conjunto de perturbaciones

1: **for** $s \in C_s$ **do** {para cada experiencia s hacer}
2: calcular δp
3: $p' = p + \delta p$ {perturbamos el modelo}
4: $g' = Proj(MF_{prom}, M_{form}(p'), I)$ {Proyectar la textura de I contenida en $M_{form}(p')$ en el modelo de formas promedio MF_{prom}}
5: ecualizar g' y aplicarle filtro *gaussiano*
6: $\delta g = |g_{prom} - g'|$ {calculamos el residuo de textura}
7: almacenar la perturbación δp en la columna s de Δp
8: almacenar el residuo de textura δg en la columna s de Δg
9: **end for**

(ver Algoritmo 4).

Este procedimiento se repite para todo el conjunto de entrenamiento, obteniendo una matriz Δp y Δg para cada muestra del conjunto. Luego promediamos ambas matrices obteniendo la matriz Δp y Δg finales.

Construcción de la matriz de regresión R

Para construir la matriz de regresión R, utilizamos el método del *Jacobiano entrenado*, (ver [5] por detalles). Primero se construyen las matrices Δ_p y Δ_g utilizando el algoritmo 4, luego utilizar el algoritmo 5 para obtener la matriz R.

Algorithm 5 Algoritmo matriz de regresión R

1: Construir Δp y Δg utilizando el algoritmo anterior.
2: Calcular el Jacobiano: $J = \frac{\partial g}{\partial p} = \Delta g \Delta p^{\dagger}$
3: La matriz de regresión R está dada por $R = (J^T J)^{-1} J^T = J^{\dagger}$

Para calcular el *Jacobiano Pseudo Inverso* $J^{\dagger}$ se puede usar la Descomposición en Valores Singulares (SVD)[1].

Algoritmo de ajuste iterativo

Como se mencionó anteriormente, los parámetros del modelo son actualizados a partir del residuo de textura. Como tenemos una matriz que nos relaciona el residuo de textura con la perturbación que lo provoca, nos resta hacer la perturbación inversa para llevar al modelo en la dirección correcta. La actualización del modelo, está dada entonces por

$$P_k = P_{k-1} - \alpha(J^T J)^{-1} J^T \delta g \tag{4.18}$$

$$\mathbf{P_k} = \mathbf{P_{k-1}} - \alpha \mathbf{R} \delta \mathbf{g} \tag{4.19}$$

Dónde J es la matriz de Jacobiano entrenado en una etapa previa. Esto es lo que hace al algoritmo más eficiente, ya que esta matriz no tiene que ser calculada en cada iteración, sino que se tiene pre-computada desde la etapa de entrenamiento. Para ayudar a converger al algoritmo (y que no se deforme eternamente) se le agrega el factor de magnitud α. Este valor, amplifica o disminuye el nivel de ajuste dado por el residuo.

El Algoritmo de ajuste se puede ver en algoritmo (6).

El algoritmo de ajuste se realiza para cada frame obtenido desde la webcam. Mientras más alto sea el número de iteraciones del algoritmo, más bajo será el *frame rate* del algoritmo.

El proceso de ajuste de los AAM dependen mucho de la pose inicial del modelo. En la siguiente sección explicamos como obtenemos la pose inicial a partir de clasificadores en cascada entrenados con el algoritmo *AdaBoost*.

Un ejemplo de algunas iteraciones del algoritmo de ajuste sobre una imagen se pueden ser en la Figura 4.10, y algunos ejemplos de residuos de textura en la Figura 4.11.

[1] si $SVD(A) = USV^T$ entonces $A^{\dagger} = VS^{\dagger}U^T$

Algorithm 6 Algoritmo iterativo de ajuste

Sea MF_{prom} el modelo de formas promedio y T_{prom} la textura promedio, ambas obtenidas en el proceso de entrenamiento, I la imagen obtenida en la iteración actual y p los parámetros de forma contenidos en la iteración actual. Notar que este procedimiento se ejecuta para cada frame de la secuencia de entrada. Este frame es una imagen color.

1: i = 1
2: **while** $i < MaxIterations$ **do**
3: $g' = Proj(MF_{prom}, M_{form}(p), I)$\{Proyectar la textura de I contenida en $M_{form}(p)$ en el modelo de formas promedio MF_{prom}\}
4: $g' = Norm(g')$\{Normalizamos textura\}
5: $\delta g = |T_{prom} - g'|$\{Calculamos δg\}
6: $E_0 = |\delta g|^2 = \delta g \delta g^T$\{Calculamos error inicial \}
7: $\delta p = R\delta g$\{Calculamos δp, perturbación del modelo de formas. \}
8: **for** $\alpha \in \{1; 1{,}5; 0{,}5; 1{,}125\}$ **do**
9: $p' = p - \alpha\delta p$\{Actualizamos el modelo de formas con la amplificación α.\}
10: $g' = Proj(MF_{prom}, M_{form}(p'), I)$\{Proyectar la textura de I contenida en $M_{form}(p')$ en el modelo de formas promedio MF_{prom}\}
11: $g' = Norm(g')$\{Normalizamos textura\}
12: $\delta g = |T_{prom} - g'|$\{Calculamos δg\}
13: $E' = |\delta g|^2 = \delta g \delta g^T$\{Calculamos error\}
14: **if** $E' < E_0$ **then**
15: $p = p'$\{Aceptamos los parámetros p'\}
16: **break**
17: **end if**
18: **end for**
19: **if** $|E' - E_0| < umbralError$ **then**
20: **break**
21: **end if**
22: i++
23: **end while**

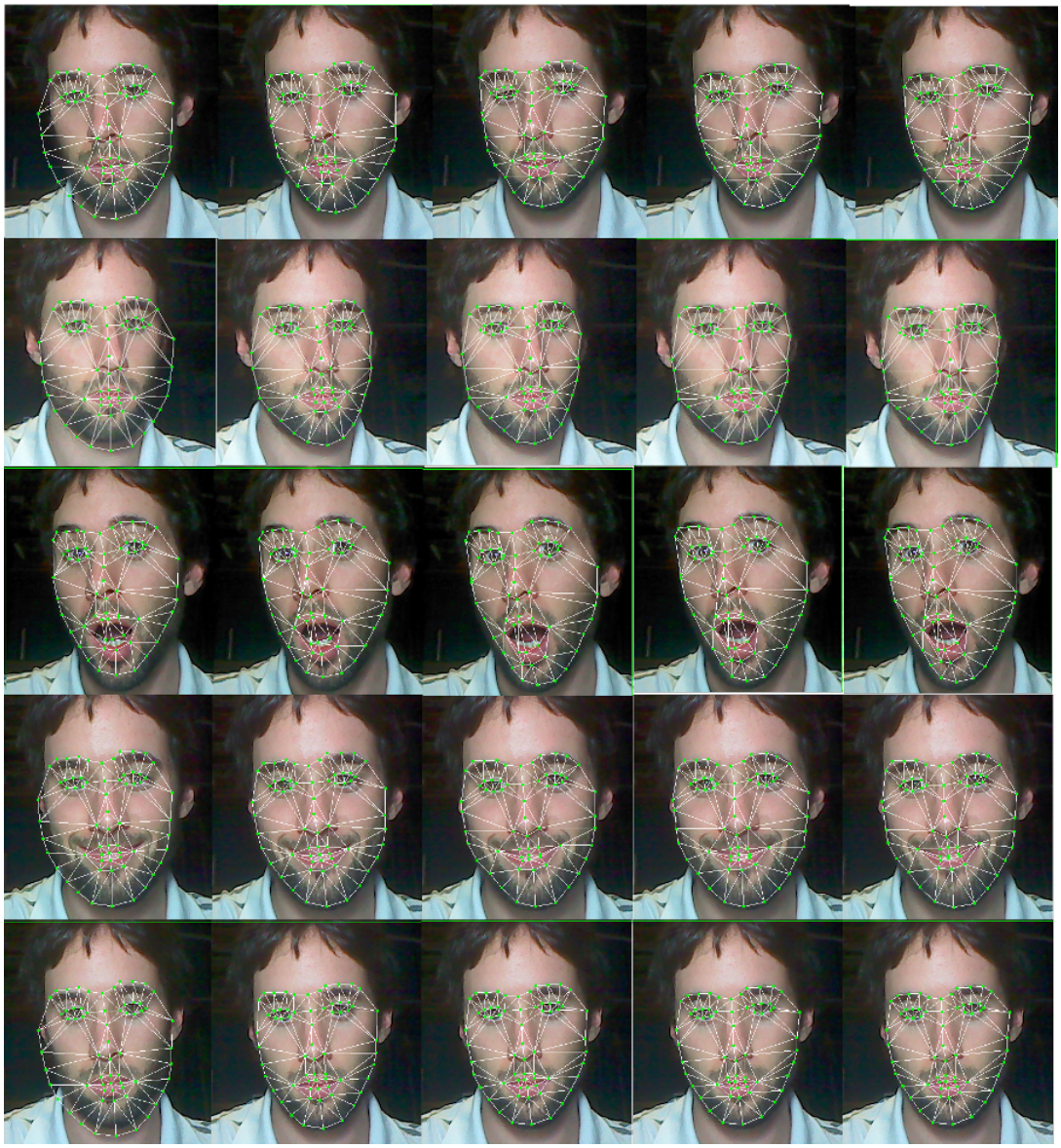


Figura 4.10: Cada fila muestra las iteraciones del algoritmo de ajuste, para distintas imágenes. La primera columna es la posición inicial del algoritmo.

Figura 4.11: Residuos entre modelos perturbados y textura promedio, en la etapa de entrenamiento de la matriz de regresión R.

4.8. Modelos divididos y texturas a partir de imágenes de bordes

En la etapa de desarrollo no solo se probó el modelo de la cara explicado anteriormente, también se probaron e implementaron algunas variaciones al modelo original:

- Texturas con imágenes de bordes.
- Modelos divididos en partes.

Texturas con imágenes de bordes

En esta variación, se intentó utilizar como textura en lugar de la imagen ecualizada directamente, una imagen filtrada con bordes resaltados. Este proceso se realiza aplicando dos filtros Gaussianos con diferente intensidad y tamaño del kernel en la función de convolución ([20], utilizado en detección de bordes) en dos copias de la textura, y luego restando ambas. También se probó ecualizar la textura resultante por canal de color, de la misma forma descripta anteriormente. El objetivo de esta variación era ver si se podían construir estos modelos con texturas más uniformes entre distintos rostros, y al mismo tiempo con información más referente a la expresión. Las imágenes de bordes tienen información más fuerte de los rasgos, y son más invariantes a los cambios de luz.

Ejemplo de texturas utilizadas en el proceso (Figura 4.12 4.13 y 4.14).

Figura 4.12: Texturas de diferencias Gaussianas sin ecualización de histograma por color

Figura 4.13: Texturas de diferencias Gaussianas con ecualización de histograma por color

Figura 4.14: Residuo de textura a partir de texturas con diferencia Gaussiana y ecualizacion de histograma por canal

resultados: La alineación del modelo a partir de estas texturas no resulta tan buena como la alineación utilizando texturas de la imagen entera sin filtración. Para movimientos

como determinadas rotaciones o escalas, la textura debajo de cada triángulo del modelo de formas es fundamental. Con las texturas filtradas de esta manera, obtenemos varios de los triángulos (en los que no hay prácticamente bordes, por ejemplo zona del cachete, o pómulos) totalmente oscuros (sin información). Esta falta de información hace que el modelo no sepa para donde perturbarse en la siguiente iteración, haciendo que el modelo no se ajuste correctamente.

Además de lo mencionado anteriormente, este proceso suma una carga adicional al proceso del cálculo del residuo, haciendo que por cada proyección de textura (operación realizada con mayor frecuencia en el proceso de ajuste) se tenga que copiar la textura dos veces, y aplicar por cada una un filtro Gaussiano, y luego hacer una resta matricial. Este proceso extra por cada proyección produce una carga notoria en el proceso.

Modelos divididos en partes

Como mencionamos anteriormente, las direcciones en las cuales se deforman los AAM se obtienen a partir de las muestras utilizadas en el proceso de entrenamiento. Se quiere probar en esta etapa, utilizar modelos más restringidos a determinadas zonas de la cara, intentando que las direcciones de cada modelo esten referidas a un único movimiento. Por ejemplo, con el modelo de la cara entera, direcciones del *pca* incluyen no solo movimiento de la boca, sino también de los ojos y cejas. Podemos pensar que cada movimiento caracteriza a un movimiento de pose, o expresión, pero global a la boca, ojos y cejas (ver Figura 4.4). Al dividir el modelo en submodelos, se intenta entonces restringir los movimientos a cada sub parte (boca, ojos), intentando tener mayor precisión.

Se probaron los siguientes sub modelos (ver Figura 4.15).

Resultados

Inicialmente se probaron los modelos de la boca (sin mandíbula) y el de los ojos y cejas. Si bien estos modelos se alineaban correctamente en la posición inicial, al producirse una rotación o un movimiento rápido, estos se desalineaban y perdían la posición fácilmente. A partir de esto se intentó agregar otra porción de textura hasta llegar a los bordes de la cara, en donde haya una diferencia notoria entre la textura y el fondo. Para el modelo de la boca y el de los ojos (3ro y 4to en Figura 4.15) esto funcionó correctamente. El modelo de la boca y la mandíbula funciona bien, ajustándose correctamente y teniendo direcciones del pca referidas a la boca únicamente. Sin embargo en el área de los ojos, el poco peso que tienen las cejas en la textura respecto al resto de la textura hace que un cambio de luz predomine sobre las cejas haciendo que no se ajuste tan bien en la parte superior (ver Figura 4.16). Por último, en cuanto a eficiencia, tener el modelo dividido en dos, hace que el paso de la textura a la placa de video y la vuelta de cada resultado sea más ineficiente. En el algoritmo de ajuste, vimos que se realizan una cantidad de iteraciones en el modelo ajustado, en dónde este no consigue un mejor resultado pasando por todas las amplificaciones (distitnos α). Este proceso se hace el doble de veces al tener el modelo dividido en dos partes. Por estas razones es que el mejor resultado sigue siendo obtenido con el modelo entero. El frame rate que se logra con este es cercano a 10fps, mientras que con el modelo divididos en dos cerca de 4fps.

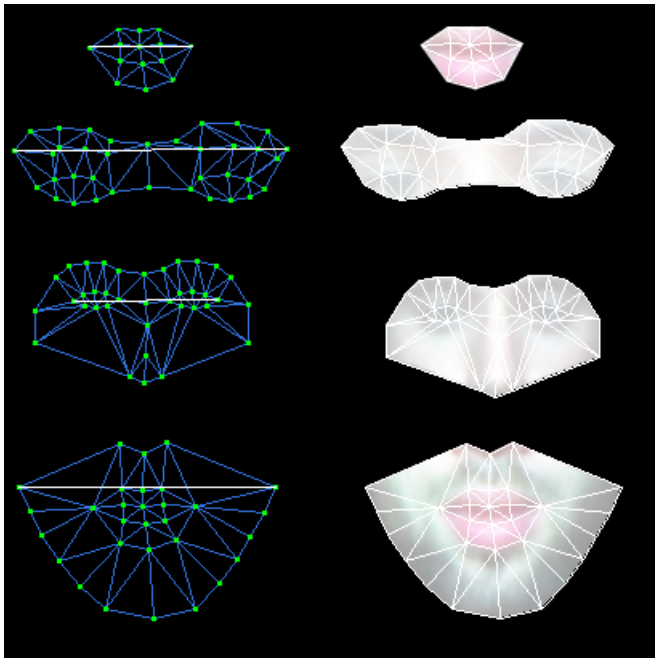


Figura 4.15: Otros modelos utilizados: boca, ojos, boca más mandíbula, ojos más nariz y cejas.

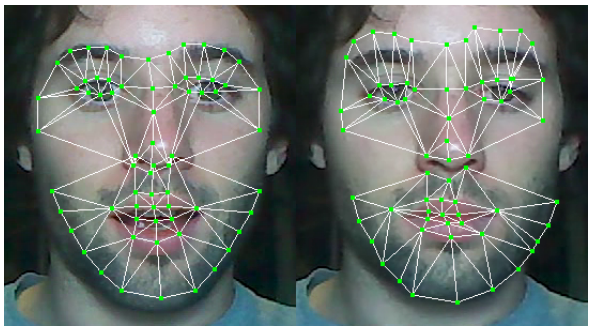


Figura 4.16: ejemplo de modelos en el proceso de ajuste. En el modelo de la derecha se puede ver el error de alineación del modelo de los ojos en la parte superior.

4.9. Estimación de la pose inicial del AAM

En esta sección se describirá como se encuentra el área de interés de la cara y dentro de ella el de cada ojo, para construir la pose inicial del AAM.
Para ello se utilizan clasificadores en cascada previamente entrenados con el algoritmo Adaboost, mencionados en la parte 2 de este trabajo.

Reconocimiento de Cara y ojos para la pose inicial del AAM, utilizando el clasificador en cascada, entrenados con AdaBoost.

Para la iniciación del Modelo Activo de Apariencia (AAM), se requiere de una buena pose inicial del modelo sobre la imagen. En este trabajo se utilizó como método de alineación,

superponer los ojos del modelo sobre los ojos encontrados en la imagen tomada desde la webcam (Figura 4.17).

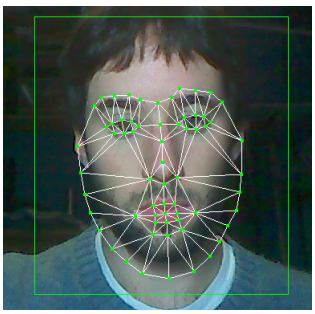


Figura 4.17: Pose inicial del AAM, alineada con los ojos, encontrados con el clasificador en cascada.

Para ello, se obtiene el área de interés de cada cara y luego en ella se buscan las áreas de interés de los ojos. Una vez obtenidos estos, se obtiene la inclinación de la cara, y la escala, trazando una linea que une el punto intermedio de cada ROI de cada ojo. Con eso basta para una buena alineación inicial del AAM.

Para obtener el área de interés ROI, de la cara como la de cada ojo, se entrenaron dos clasificadores:

- Clasificador de cara
- Clasificador de ojo, que luego es utilizado para el ojo izquierdo y el ojo derecho.

Ambos clasificadores fueron entrenados con una base de datos de caras y de ojos incluidos en OpenCV.

Proceso de búsqueda

Para el proceso de búsqueda de la cara, se itera desplazando el clasificador por toda la imagen mientras se va achicando la ventana y aplicando el clasificador en cascada en ella, para determinar si pertenece a el objeto buscado (cara, u ojos). También se utilizan dos cotas inferiores para la cantidad de vecinos que cumplen el clasificador, y para el tamaño mínimo del ROI. El primero requiere que al mover la ventana del clasificador hacia los 4 lados, este también de positivo. Eso logra centrar mejor el ROI con respecto al objeto buscado.

Para buscar los ojos se realiza el mismo procedimiento, solo que el área de búsqueda es el de la cara encontrado previamente. Cada ojo se busca en el cuarto izquierdo o derecho superior del ROI de la cara para disminuir la confusión entre ojos, y acelerar la búsqueda.

Capítulo 5

Clasificación de expresiones con Máquinas de Soporte Vectorial (SVM)

El reconocimiento de expresiones consta de 3 etapas principales, detección de caras, extracción de información de expresión y clasificación de esta en distintas expresiones. Como vimos anteriormente, para las dos primeras partes se utilizó: Haar-Like Features, Adaboost y Clasificadores en cascada (reconocimiento de caras y pose inicial de AAM), y Modelos Activos de Apariencia (AAM) para extraer información facial de expresión. Resta el problema de clasificar esta información obtenida en una de las 5 expresiones analizadas: *neutra, alegría, sorpresa, tristeza y enojo*.

Para este último paso se utilizaron *Máquinas de Soporte Vectorial - SVM* para clasificar la información obtenida desde el AAM y determinar a que expresión pertenece.

5.1. Maquinas de Soporte Vectorial - SVM

Las *Máquinas de Soporte Vectorial* (Support Vector Machines, SVM) son un conjunto de algoritmos de aprendizaje supervisado [18], [14], [17] y [13]. Estos métodos están relacionados con problemas de clasificación. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases manualmente y entrenar un clasificador SVM para construir un modelo que prediga automáticamente la clase de una nueva muestra.

Intuitivamente, una SVM es un modelo capaz de clasificar un conjunto de puntos pertenecientes a un espacio de muestras de distintas clases. Las SVM serán capaces de predecir la clase a la que pertenece una nueva muestra actuando como un clasificador.

Más formalmente, SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad (a veces más alta) que puede ser utilizado en problemas de clasificación. Una buena separación entre las clases permitirá una buena clasificación.

5.1.1. Idea Básica

Dado un conjunto de puntos pertenecientes a dos posibles clases, una SVM construye un modelo capaz de predecir la clase de una nueva muestra. Como en la mayoría de los métodos de clasificación supervisada, los datos de entrada (los puntos) son vistos como un vector p-dimensional. La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior.
En ese concepto de "separación óptima" es donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. Por eso también a veces se les conoce a las SVM como clasificadores de margen máximo. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado del mismo.

5.1.2. Formulación Básica (Vapnik [18])

Sean $S = \{(x_1, y_1), (x_2, y_2), ..., (x_l, y_l)\}$ un conjunto de entrenamiento con datos de entrada $x_i \in \Re^n$ y sus correspondientes clases binarias etiquedatas $y_i \in \{-1, 1\}$, de acuerdo con [18], el clasificador SVM estará dado por

$$D(x) = w^T \Psi(x) + b \tag{5.1}$$

donde $\Psi(x)$ es una función no lineal que mapea el espacio de entrada x, en el espacio de "características" posiblemente de mayor dimensión, w^T es un vector de dimensión m y b es un escalar.

Para separar los datos de forma lineal en el espacio de características, la función de decisión satisface lo siguiente

$$y_i(w^T \Psi(x_i) + b) \geq 1 \quad (\forall_{i=1,...,l}) \tag{5.2}$$

Con el fin de determinar la separación del hiperplano óptima, la cual tiene el margen máximo entre dos clases, podemos utilizar el siguiente problema de optimización

$$\min_{w,b} \frac{1}{2}||w||^2 = \frac{1}{2} w^T w \tag{5.3}$$

sujeto a

$$y_i(\Psi(x_i) + b) \geq 1 \quad (\forall_{i=1...l}) \tag{5.4}$$

Cuando los datos de entrada no pueden tener una separación lineal introducimos unas variables de holgura, el problema de optimización es el siguiente

$$\min_{w,b} \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{k=1}^{l} \xi_i \tag{5.5}$$

estando el problema sujeto a

$$y_i(\Psi(x_i) + b) \geq 1 - \xi_i \quad (\forall_{i=1...l}) \tag{5.6}$$

$$\xi_i \geq 0 \quad (\forall_{i=1...l}) \tag{5.7}$$

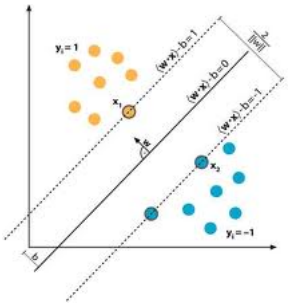


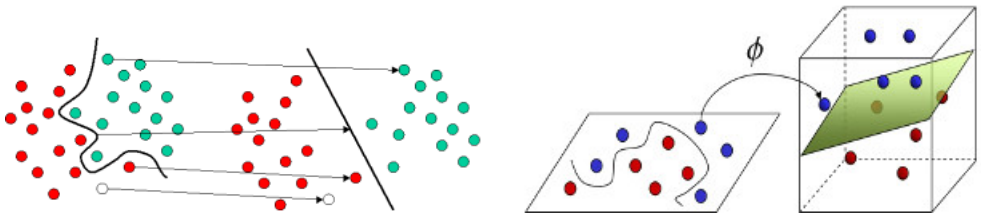
Figura 5.1: Ejemplo clasificador binario (2 clases).

en dónde γ representa una variable de equilibrio entre el margen máximo, y el error de clasificación.

Como no siempre se puede buscar una separación en el espacio de entrada, se intenta transformar el espacio en un espacio de características en donde esta separación pueda ser realizada. Es aquí en donde surge la función Kernel.

5.1.3. Kernels

El uso de está función es el de mapear el espacio de entrada en un espacio de características en donde no existe una clasificación lineal, en uno en dónde si.



(a) (b)

Figura 5.2: Dos ejemplos de espacio de entrada vs espacio de características

Un Kernel es una función de la forma $K(x_i, x_j) = x_i^T x_j$ (kernel lineal) que permite computar el *producto escalar* entre características, en el espacio de características sin transformar los datos a ese espacio, permitiendo que las operaciones se realicen en el espacio de entrada. La función kernel debe ser *simétrica y definida positiva.*

Kernels más utilizados (ver Figura 5.3).

Para este trabajo se utilizó una implementación de Máquinas de Soporte Vectorial de **OpenCv**. Esta implementación incluye los 3 kernels, y permite ajustar los parámetros del Kernel en la etapa de entrenamiento maximizando la tasa de acierto en la etapa de testing del clasificador.

Kernel	**función**
Lineal (el más eficiente)	$K(x_i, x_j) = x_i^T x_j$
Polinomial	$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d,\ \gamma \geq 0$
Radial Basis Function (RBF)	$K(x_i, x_j) = e^{-\gamma \|\|x_i - x_j\|\|^2},\ \gamma \leq 0$

Figura 5.3: Kernels estándar, γ, r *y* d don parámetros del kernel

Se utilizaron tanto el kernel *Lineal* como el kernel *RBF*, sin embargo se obtuvieron mejores resultados con RBF, descartándose así el kernel lineal (ver referencias [19]).

5.1.4. Máquinas de Soporte Vectorial Multi Clase

Las máquinas de soporte vectorial, inicialmente fueron propuestas para clasificaciones binarias (de dos clases), para hacer una clasificación multi clase se conocen 3 formas, *uno contra todos* (One-Against-All), *uno contra uno* (One-Against-One), y *DAGSVM*. Los tres métodos son cascadas de clasificadores binarios. Una breve descripción a continuación.

Uno contra todos (One-Against-All)

Se construyen k clasificadores SVM binarios $c_1, ..., c_k$, uno por clase. El clasificador c_i se entrena con las muestras de la clase i, como positivas, y todo el resto como negativas. Se crean así, k funciones de decisión

$$\begin{matrix} (w_1)^T \psi(x) + b_1 \\ \vdots \\ (w_k)^T \psi(x) + b_k \end{matrix}$$

la decisión final para una muestra x, está dada por la función de decisión con el valor más alto.

Uno contra uno (One-Against-One)

Este es otro esquema de votación como el anterior, pero esta vez se entrenan $\frac{k(k-1)}{2}$ clasificadores, dónde cada uno tiene muestras de dos clases, i, j. Suponemos una muestra x, entonces la clase ganadora será la que sume más votos. El esquema de votación es el siguiente. Se corren todos los clasificadores i, j para la muestra x. Se suma un voto a la clase resultante de cada clasificador. La decisión final será la clase que sume más votos.

Direct Acyclic Graph Support Vector Machines (DAGSVM)

Por último, DAGSVM consiste en el mismo esquema que *Uno contra uno*. En este método se arma un grafo a-cíclico dirigido (o árbol) con $\frac{k(k-1)}{2}$ nodos, uno por clasificador de a pares y k extremos (uno por clase). Cada nodo es un clasificador binario de la clase i, j. Empezando por el nodo raíz, la decisión binaria nos mueve por una salida u otra navegando por el árbol llegando por último a la hoja, dándonos la clase correspondiente.

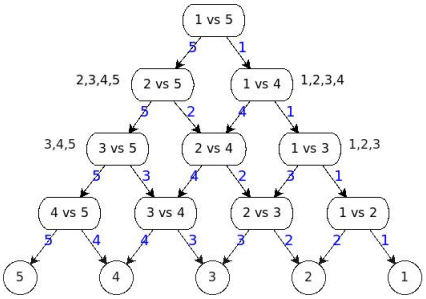


Figura 5.4: Ejemplo del árbol del método DAGSVM

5.1.5. SVM para reconocimiento de expresiones

Como vimos anteriormente las Máquinas de soporte vectorial son utilizadas para clasificar muestras de algún conjunto de datos, en distinta clases de equivalencia. En este trabajo fueron utilizadas para clasificar expresiones a partir de información obtenida del Modelo Activo de Apariencia (AAM). Hasta la etapa anterior, el AAM era ajustado a la cara en el proceso de ajuste. Resta entonces definir como utilizar la información obtenida del AAM, para clasificar la expresión de la imagen capturada.

Descriptores de expresión utilizados

Podemos definir entonces como descriptor, a un vector que contiene información que representa una expresión determinada. Se busca que este descriptor pueda representar lo mejor posible una expresión, teniendo información relevante que nos permita discriminar correctamente a cual pertenece. De esta manera podemos utilizar estos descriptores como muestras del SVM, y separar cada uno en diferentes clases de equivalencia (correspondientes a las 5 expresiones).

En este trabajo se utilizaron dos descriptores distintos, provenientes del AAM, más precisamente, del *modelo de formas*.

1. **Sub Conjunto del vector del *modelo de formas* del AAM,** b. Se tomaron solamente las componentes de b asociadas a las direcciones referentes a expresión, dejando afuera las direcciones asociadas con pose (rotación traslación, perspectiva de la cara, etc).

   El descriptor utilizado tiene dimensión 5 (5 componentes principales del vector b).

2. **Descriptor de expresión con información de distancia entre diferentes puntos del modelo de formas, normalizadas por un factor de normalización** (distancia entre ojos). Con el objetivo de tener información más pura de la expresión y dejar afuera información de pose o escala. Se considera en este caso, que la posición de la cara frente a la webcam será de frente.

   Las 6 componentes que se utilizan son

   - 1 - Distancia vertical entre cejas y labio inferior.

- 2 - Distancia vertical entre el centro de la linea de los ojos, con las cejas.
- 3 - Distancia horizontal entre extremos de la boca.
- 4 - Distancia vertical entre el medio de la boca (linea que une ambos extremos) y la nariz.
- 5 - Distancia vertical entre labio superior y labio inferior.
- 6 - Distancia vertical entre labio inferior y la mitad de la boca

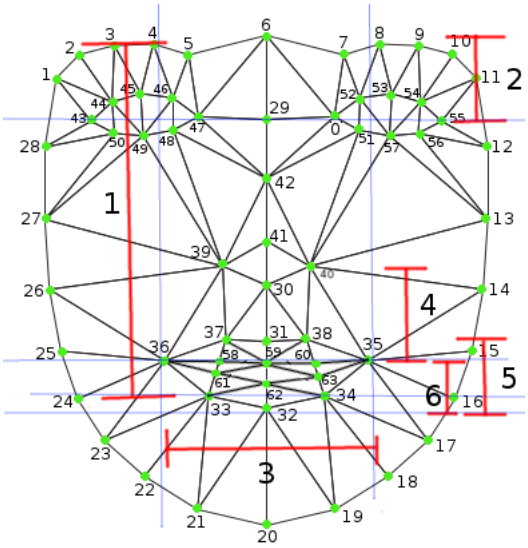


Figura 5.5: Descriptor de la cara (dimensión 6).

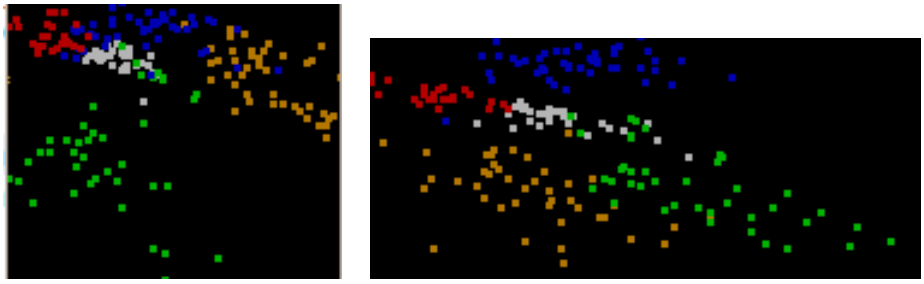
Para ambos casos se utilizó un clasificador SVM previamente entrenado con muestras de cada expresión, utilizando el descriptor correspondiente.

Los Kernels Utilizados fueron: Lineal, y RBF.

Ejemplos gráficos

Para ilustrar gráficamente algunos de los resultados del entrenamiento de SVM, tomamos dos coordenadas de las 6, que componen el **descriptor de expresión**, y las graficamos en el eje x-y. En color marcamos la clase a la que pertenece siendo

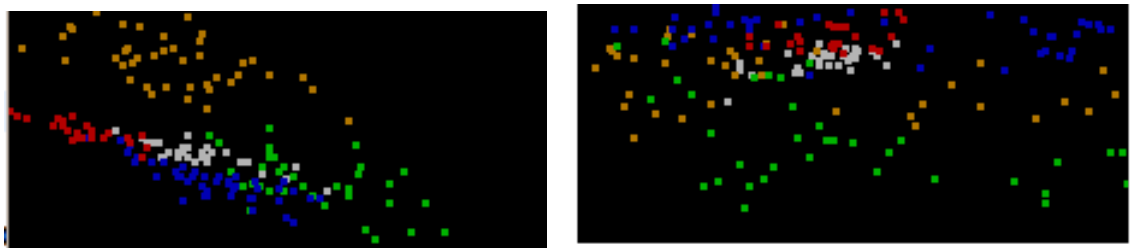
- Blanco: Neutro
- Amarillo: Alegría
- Verde: Sorpresa
- Azul: Tristeza
- Rojo: Enojo



(a) Dispersión de clases, vista en axis 2-4.

(b) Dispersión de clases, vista en axis 0-5.

Figura 5.6: Dispersión de clases



(a) Dispersión de clases, vista en axis 0-3.

(b) Dispersión de clases, vista en axis 1-4.

Figura 5.7: Dispersión de clases

ver Figura 5.6 y 5.7.

Entrenamiento de SVM

Para los entrenamientos de cada SVM se utilizaron aproximadamente 30 muestras por clase, para ambos descriptores.

Algunas de estas muestras correspondientes a instancias etiquetadas a mano mientras que la gran mayoría extraídas directamente del resultado de un modelo ajustado en una cara con la expresión indicada. Esto último se realizó para intentar utilizar en el entrenamiento muestras similares a las que van a ser obtenidas con el sistema.

Se graficaron algunos resultados en 3 dimensiones en dónde se puede ver la importancia de la cantidad de dimensiones para poder diferenciar determinadas expresiones como tristeza, enojo, y neutro.

Ver Figura 5.8 y 5.9.

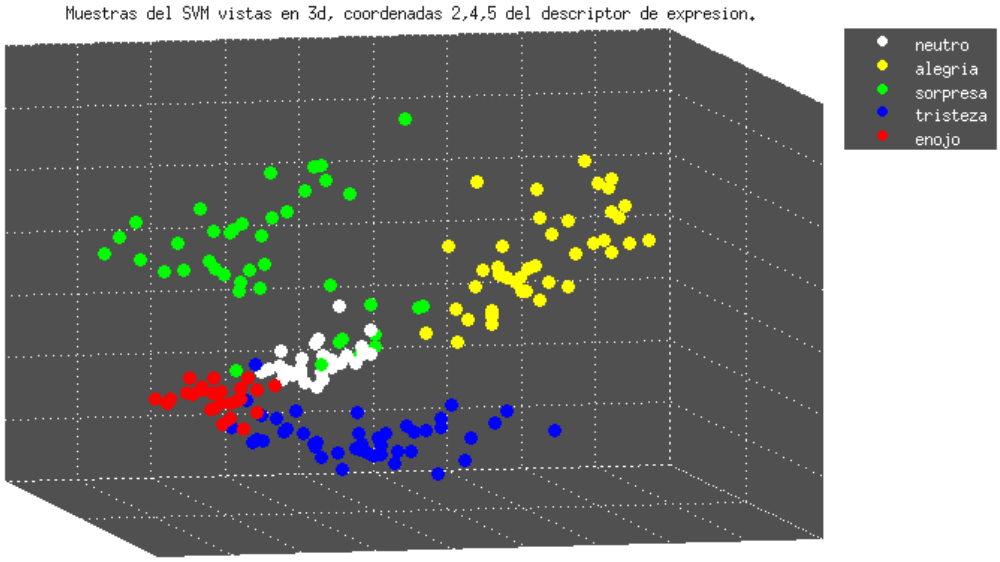


Figura 5.8: Dispersión de clases, vista en 3d para los componentes 2,3, y 5 del descriptor de expresión. Vista 1.

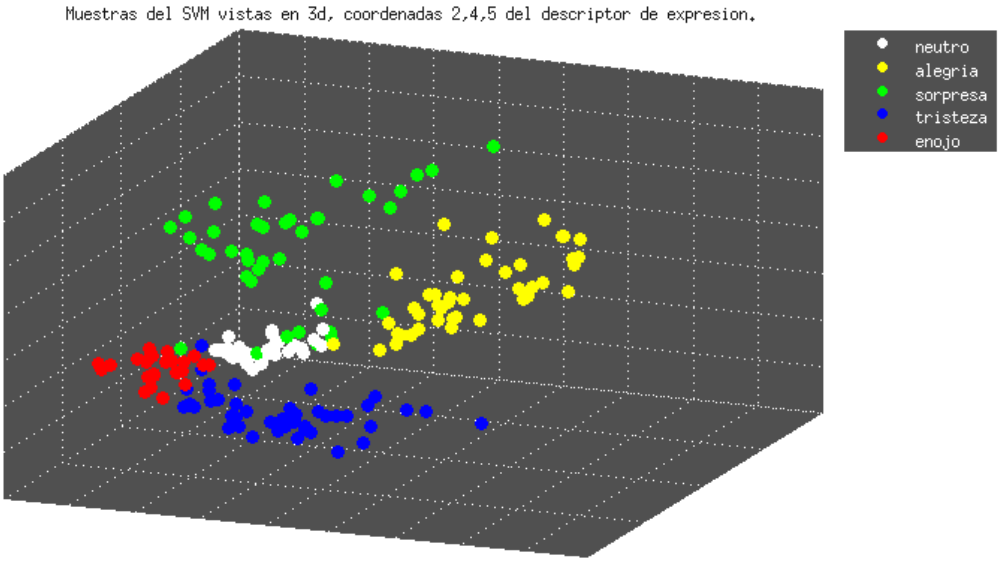


Figura 5.9: Dispersión de clases, vista en 3d para los componentes 2,3, y 5 del descriptor de expresión. Vista 2.

Capítulo 6

Referencias de implementación

En este trabajo se realizó un sistema de reconocimiento de expresiones. Para ello, se implementaron 3 partes principales

1. Libreria de Modelos Activos de Apariencia y SVM: aam-er
2. Programa entrenador de modelos y Svm
3. Sistema de reconocimiento de expresiones

Todas ellas implementadas en *C++*, utilizando *OpenCv 2.3* (Open Source Computer Visión) [21], *OpenGl* [22] y *GLut* (The OpenGL Utility Toolkit) [23].

1 - Libreria de Modelos Activos de Apariencia y SVM

Esta es la librería troncal de la aplicación. Contiene todos los modelos y métodos referentes a modelos activos de apariencia y a clasificación por SVM. Esta es la librería a ser utilizada en el desarrollo de una aplicación con modelos activos de apariencia.

Nombre: *aam-er*
Libreria Estática: *libaam-er.a*
Encabezado: *aam-er.h*
Dependencias: *opencv, opengl y glut.*

2 - Programa entrenador de modelos y Svm

Esta es la aplicación encargada de entrenar los Modelos Activos de Apariencia y clasificadores SVM, para ser utilizados en cualquier aplicación. Las funcionalidades principales de esta aplicación son

- Etiquetar muestras de modelos (de estructura definida en un archivo xml) sobre una imagen obtenida desde la webcam o desde sistema de archivos. Generar una muestra del modelo activo de apariencia, para el entrenamiento posterior.
- Entrenar un modelo activo de apariencia, a partir de muestras generadas. Este método calcula el PCA para el modelo de formas, la forma y textura promedio y la *matriz de regresión R utilizada en el proceso de ajuste.*

- Entrenar un clasificador SVM de AAM, a partir de muestras de modelos en determinada posición, etiquetadas con distintas clases.
- Testear el AAM entrenado. Se muestra una interfaz en donde se varían los componentes principales con tracksbar, y se ve la alteración del modelo a partir de esta perturbación, con la textura promedio proyectada en el. (Útil para probar distintas cantidades de componentes principales y ver que movimiento provoca cada una).
- Testear el clasificador SVM de AAM entrenado.

Sistema de reconocimiento de expresiones

Este es el sistema de reconocimiento de expresiones. Utiliza la librería aam-er y modelos entrenados con la aplicación de entrenamiento. En tiempo real, el modelo se ajusta a la imagen obtenida desde la webcam, y cada cierta cantidad de iteraciones, la pose del modelo es clasificada por el clasificador SVM, determinando a que expresión pertenece. Una muestra de la interfaz puede verse en la Figura 7.9.

Capítulo 7

Resultados y Conclusiones

7.1. Resultados

7.1.1. Error de Alineación

Para las mediciones realizadas, se utiliza como error de alineación (EA) a la distancia de cada punto del modelo obtenido, con el modelo etiquetado.

$$EA = \sum_{i=1}^{n} \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2} \tag{7.1}$$

dónde $p = (x_i, y_i)$ es un punto del modelo alineado en el proceso de ajuste, y $p' = (x'_i, y'_i)$ es el mismo punto, en el modelo etiquetado (dónde se tiene una posición correcta de p' en AAM) y n es la cantidad de puntos en el modelo de formas.

7.1.2. Resultados AAM

Perturbación en la Posición Inicial - Exp 1

Para medir el proceso de ajuste del modelo activo de apariencia, se utilizaron imágenes de las distintas expresiones y se ejecutó el proceso de ajuste variando la posición inicial en escala, y traslación.

Resultados obtenidos (Figura 7.1, 7.2 y 7.3).

Resultados Experimento 1

Podemos ver como el error aceptable (en el que el modelo se alinea correctamente) para las 5 expresiones se obtiene con variaciones de traslación entre $\pm 30px$ por cada eje. En cuanto a la escala, en relación a la posición inicial del modelo, los intervalos aceptables son 80 % - 120 % del tamaño real. Sin embargo, estas son deformaciones que no se producen en una secuencia a tiempo real. Al tener un *frame rate* de aproximadamente de 10 fps (Frame por Segundo), las variaciones entre un frame y otro son mínimas. Con lo cual, si se parte de una posición inicial dentro de estos márgenes, el modelo irá ajustándose correctamente a lo largo de la secuencia, perturbándose en poca medida en cada iteración.

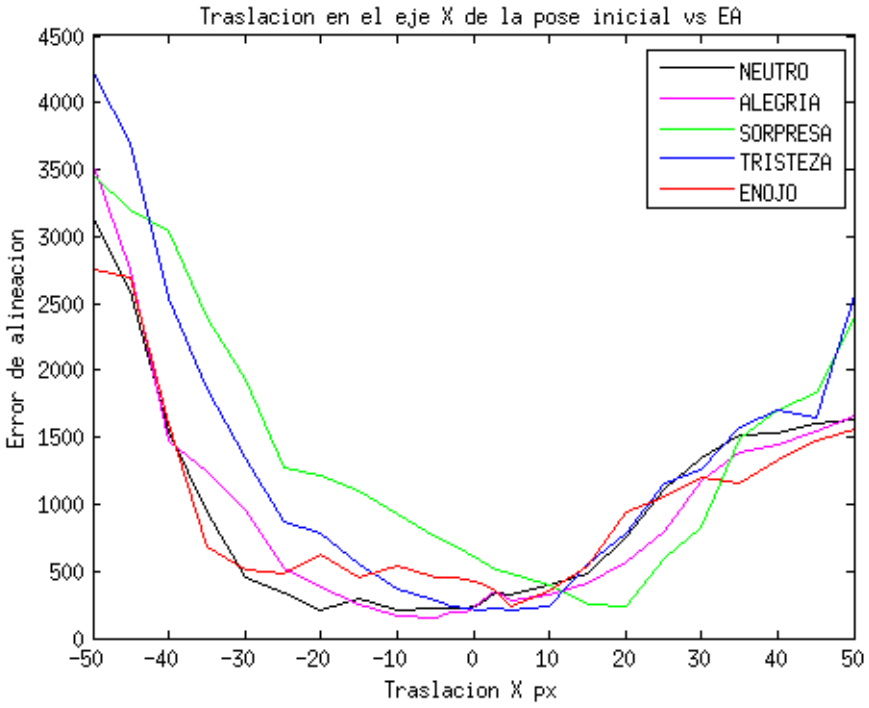


Figura 7.1: Perturbación del Eje X en la posición inicial del AAM vs EA

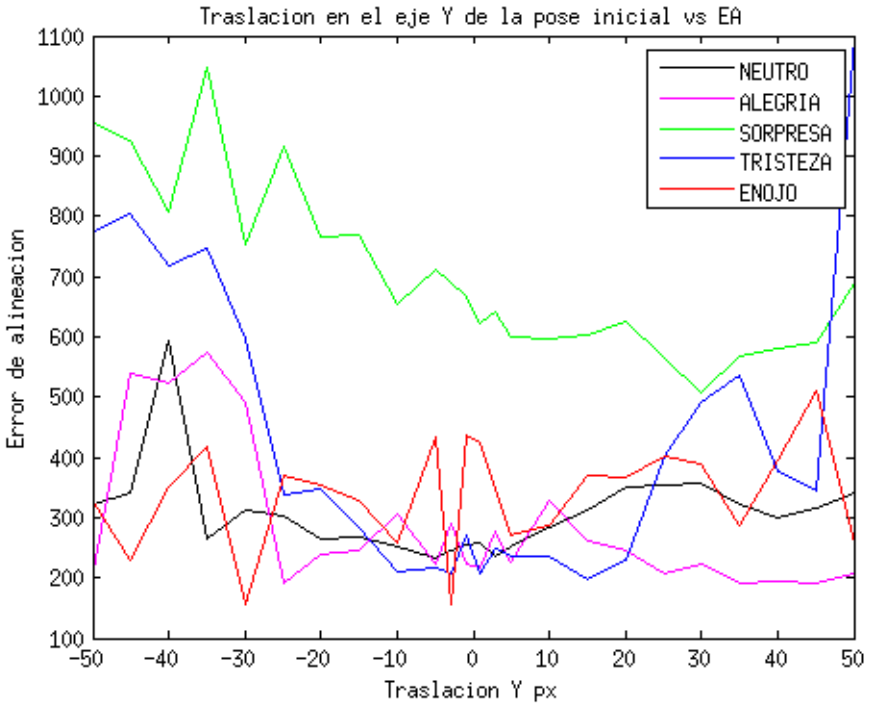


Figura 7.2: Perturbación del Eje Y en la posición inicial del AAM vs EA

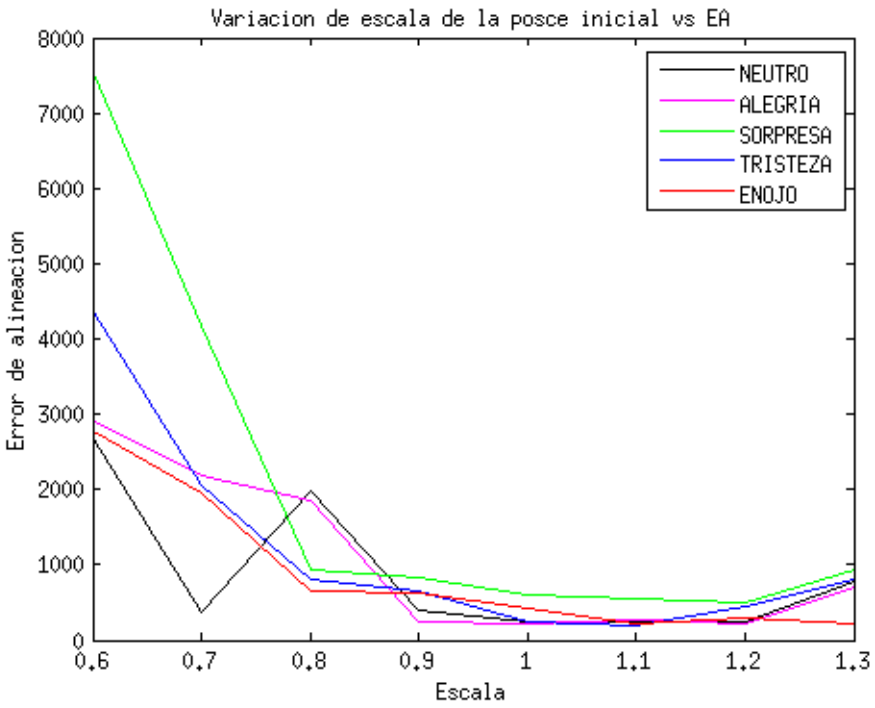


Figura 7.3: Perturbación de la escala en la posición inicial del AAM vs EA

También podemos observar, que las 5 expresiones se comportan de una manera similar para las distintas perturbaciones de la pose inicial. La más inestable a todas estas perturbaciones es la expresión de sorpresa.

Variación de Componentes Principales - Exp 2

En este experimento medimos la precisión de ajuste de los modelos de formas, con distinta cantidad de componentes principales. Estos resultados son utilizados a la hora de decidir cuántos componentes principales usar en el PCA del modelo de formas. Se corrió el proceso de ajuste para una imagen fija, y se varió la cantidad de componentes principales utilizadas en el modelo de formas.

Resultados obtenidos (Figura 7.4)

Resultados Experimento 2

Podemos ver como resulta el proceso de ajuste, para diferentes cantidades de componentes principales obtenidas por el PCA en el modelo de formas. Mientras más componentes principales agreguemos, más direcciones utilizará el AAM para el ajuste. Podemos ver como la convergencia a una posición correcta del AAM está directamente relacionada con la cantidad de componentes principales utilizadas. A partir de 14 componentes principales el modelo converge para las 5 expresiones iniciando el proceso en la pose inicial, y convergiendo directamente en la expresión. Si bien en las iteraciones de una secuencia,

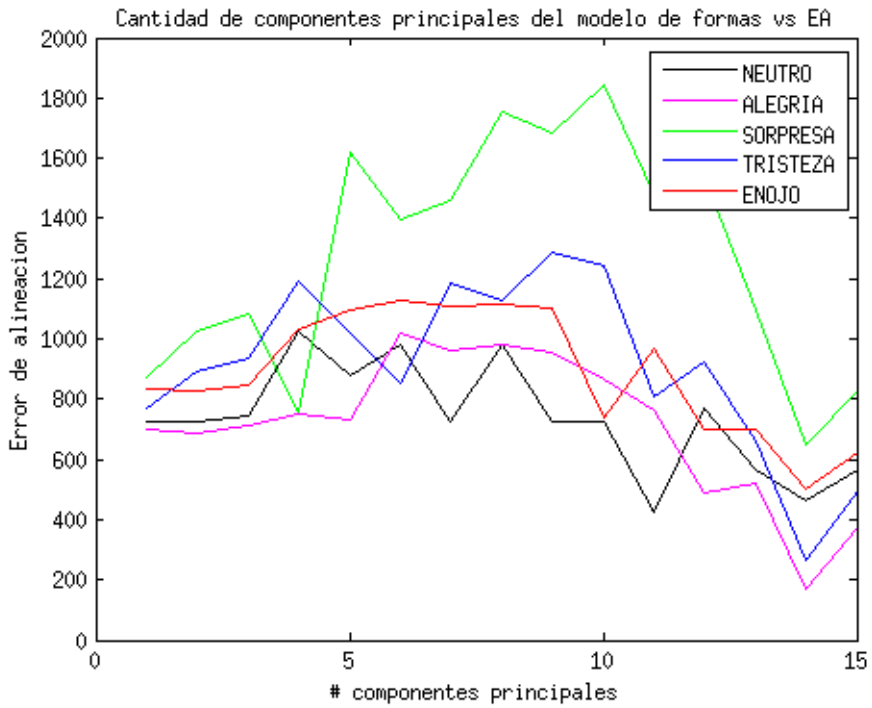


Figura 7.4: Variación de la cantidad de componentes principales del modelo de formas

las deformaciones son mucho más sutiles, con esta cantidad de componentes principales se puede converger en la mayoría de las expresiones para una imagen. Esto permitiría utilizar esto, por ejemplo, de manera offline para analizar imágenes que no son parte de una secuencia. También se busca manejar la menor cantidad de componentes o direcciones posibles, para mantener chicos los tamaños de las matrices y hacer que los costos de cada operación no sean tan costosos.

En la aplicación se determina utilizar 14 componentes principales.

7.1.3. Resultados SVM

En está sección se proveen resultados de experimentos referidos al clasificador SVM. Todas las muestras de descriptores del AAM, son muestras de modelos correctamente alineados.

En el **experimento 3** se testearon los clasificadores SVM, entrenados con aproximadamente 50 muestras de cada expresión. Luego del entrenamiento, se obtuvieron 200 muestras por cada expresión para el testing. Se presentan luego los resultados obtenidos para los dos descriptores (*sub conjunto de pcas* y *descriptor de expresión*).

Se presentan las Matrices de Confusión de cada expresión. Estas describen para una cantidad de muestras de una expresión determinada, las clases obtenidas por el clasificador. Por ejemplo, si para 100 muestras de expresión neutra, se obtienen 20 de resultado enojo, la confusión del neutro contra el enojo es del 20 %.

Ver resultados obtenidos en Figura 7.6 y 7.5 a continuación.

	Neutro	Alegria	Sorpresa	Tristeza	Enojo
Neutro	0.64	0.04	0.02	0.20	0.10
Alegria	0.45	0.46	0.00	0.00	0.09
Sorpresa	0.00	0.04	0.96	0.00	0.00
Tristeza	0.01	0.00	0.14	0.85	0.00
Enojo	0.10	0.20	0.01	0.20	0.49

Figura 7.5: Matriz de Confusión del descriptor **Sub PCA**

	Neutro	Alegria	Sorpresa	Tristeza	Enojo
Neutro	0.82	0.00	0.16	0.00	0.02
Alegria	0.09	0.90	0.00	0.00	0.00
Sorpresa	0.18	0.00	0.82	0.00	0.00
Tristeza	0.21	0.00	0.04	0.74	0.01
Enojo	0.28	0.00	0.00	0.00	0.73

Figura 7.6: Matriz de Confusión del **Descriptor de Expresión**

Resultados

Podemos ver que los resultados de las matrices de confusión utilizando el subconjunto de componentes principales (SUBPCA) no es tan bueno. Para determinadas expresiones como por ejemplo sorpresa, el resultado es incluso mejor que el del descriptor de expresión. Sin embargo tiene resultados privativos en cuanto a las expresiones alegría vs neutro y enojo vs tristeza. Esto se debe a que las direcciones principales obtenidas con pca, no representan expresiones limpias, ni direcciones limpias, con lo cual una expresión puede no estar formada siempre por la misma cantidad de cada componente principal en la re-proyección del modelo, esto hace que el entrenamiento del clasificador no de tan buenos resultados y las clases no queden tan bien distinguidas.

El resultado del descriptor de expresión es mucho mejor, y es el utilizado en este trabajo. Podemos ver que la mayor confusión obtenida, se produce entre la expresión de enojo, y la expresión neutra, siendo estas dos las más parecidas y las más distinguidas son la alegría y la sorpresa.

7.1.4. Mediciones de tiempo - Experimento 4

Finalmente se realizó una medición de *frame rate* para distinta cantidad de iteraciones por frame, del sistema entero. Estas mediciones se realizaron en una computadora con proce-

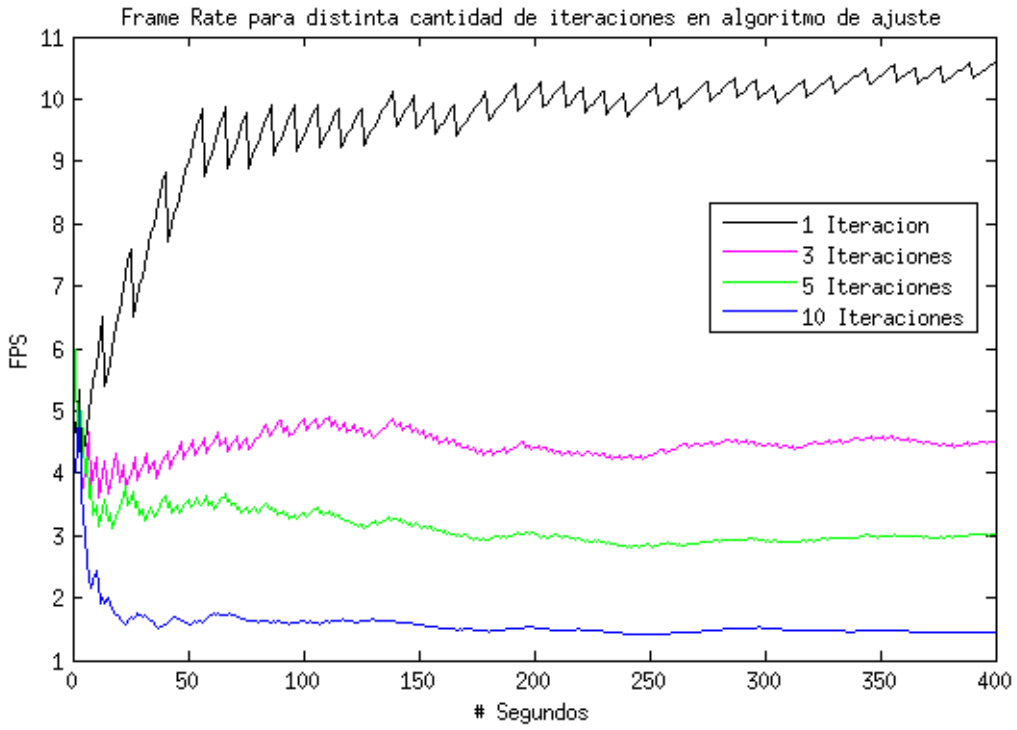


Figura 7.7: Medición de Frame Rate para distinta cantidad de iteraciones en el algoritmo de ajuste.

sador intel Core 2 Duo, de 2.2 Ghz, 4GB de memoria RAM y una placa de video intel inside.

Ver resultados en Figura 7.7.

En las figuras vemos mediciones del frame rate a lo largo del tiempo. Vemos que utilizar 1-3 iteraciones, rondamos los 5-9 fps, y utilizando 5-10, los 1-4 fps. Esto influye en buena medida en el funcionamiento de la aplicación, ya que si bien 4 fps pueden ser considerados tiempo real, en algunos contextos, el cambio posible entre un frame y el otro puede ser demasiado grande, haciendo que el modelo falle en el ajuste. Utilizando entre 1 y 3 iteraciones, el sistema se comporta correctamente, y es la configuración utilizada en la aplicación.

7.1.5. Extras

La Figura 7.8 muestra del análisis de una secuencia en dónde se pasa por las 5 expresiones. También se enseña una imágen de la interfaz de la aplicación Figura 7.9 y por último un ejemplo de las 5 expresiones utilizadas Figura 7.10.

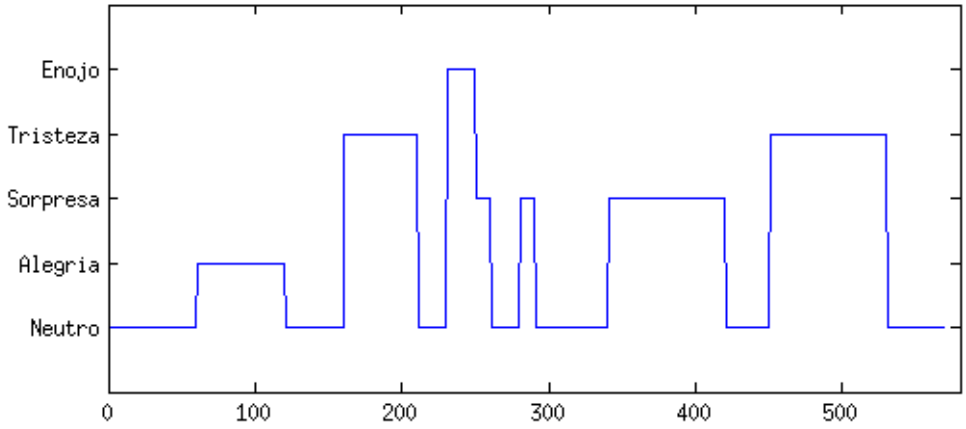


Figura 7.8: Análisis de secuencia en dónde se pasa por las 5 expresiones.

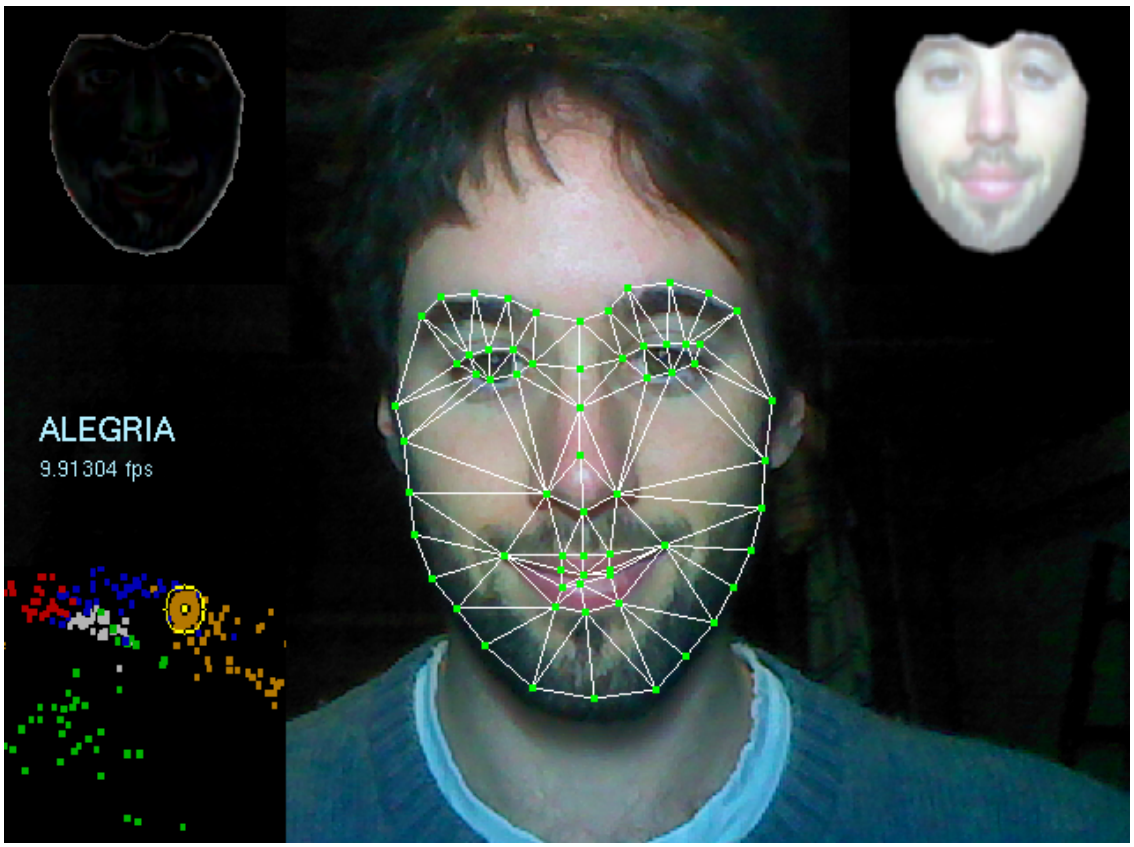


Figura 7.9: Interfaz de la aplicación. Arriba a la derecha, textura proyectada al modelo de formas promedio. Arriba a la izquierda, residuo de textura. Abajo a la izquierda, gráfico de las componente 2 y 4 del descriptor de expresión del clasificador SVM, y la expresión actual marcada en amarillo. A la izquierda al centro, expresión obtenida y frame rate aproximado.

Figura 7.10: Muestra de las 5 expresiones. De izquierda a derecha y arriba abajo: Neutro, Alegría, Enojo, Sorpresa, Tristeza.

7.2. Conclusiones

En este trabajo se presentó un sistema de reconocimiento de expresiones basado en Modelos Activos de Apariencia. Como vimos en los resultados anteriores, podemos utilizarlos para obtener información precisa de una imagen (como la cara humana), e ir ajustándolos en tiempo real, llegando a una velocidad de aproximadamente 9 fps en una computadora estándar.

Técnicas como clasificadores en cascada basados en HLF entrenados con AdaBoost funcionan correctamente en entornos en dónde el objeto que se quiere detectar no es muy parecido al entorno en dónde se encuentra (ej cara vs. fondo, ojos vs. cara). Sin embargo al querer utilizarlo para encontrar puntos determinados dentro de una cara, esté no da tan buenos resultados. Los descriptores (o características) que se consiguen en cada punto son muy similares entre sí haciendo que estos puntos se confundan con puntos aledáneos. Los modelos activos de apariencia son mucho más robustos en cuanto a forma, permitiendo variaciones solamente en las direcciones obtenidas del conjunto de entrenamiento. Esto hace que el modelo sea más robusto en cuanto a oclusión, y que no haya confusión de puntos, como en el caso de los clasificadores en cascada.
Modelos Activos de Apariencia tienen una desventaja en cuanto a eficiencia. Si bien podemos llegar a aproximadamente 9 fps, esto se logra implementando las funciones más costosas, como las proyecciones de textura, en placas de video. En este trabajo utilizamos OpenGL. Mientras más iteraciones se quiera realizar en el algoritmo de ajuste, más caro será el proceso. También los modelos activos de apariencia dependen notoriamente de la textura, y de las muestras utilizadas en el entrenamiento. En entornos en dónde la textura

del modelo que se quiera adaptar tenga menos variación este podrá ser utilizado sin inconvenientes. Sin embargo, en expresiones de caras humanas, la variación de textura puede ser mucha entre una cara y otra. Modelos Activos de Apariencia no son tan robustos en ese sentido. Deberán utilizarse técnicas que hagan que la textura sea menos influyente en el proceso iterativo. Algunas de ellas son aplicar un segundo pca, e incluir a las variaciones de textura en el proceso de ajuste. Sin embargo el tamaño de estas hacen que el proceso sea muy costoso para máquinas estándar, llegando difícilmente a tiempo real.

Existen técnicas mejores para el reconocimiento de expresiones, invariantes al cambio de textura, sin embargo se quisieron probar los modelos activos de apariencia ya que son una técnica novedosa que se está investigando actualmente.

Modelos Activos de Apariencia, requieren de una buena pose inicial para el proceso de ajuste, utilizar clasificadores en cascada entrenados con AdaBoost en este paso resultó conveniente, encontrando la cara y los ojos de una manera eficiente.

También se obtuvieron buenos resultados de clasificación utilizando Máquinas de Soporte Vectorial (SVM). En los resultados del clasificador, se logra obtener una buena separación de clases para las 5 expresiones. Utilizando únicamente los valores del pca, vimos que había expresiones que podían ser formadas por más de una dirección haciendo que el clasificador falle para determinadas muestras. Esto no sucede al armar un descriptor de la expresión a partir del modelo de formas del AAM, dando un resultado mucho más limpio en las matrices de confusión.

Finalmente el sistema combinado de estas 3 técnicas (HLF, clasificadores en cascada con AdaBoost, Modelos Activos de Apariencia, y Maquinas de Soporte Vectorial) dio buenos resultados, no solo de clasificación sino de eficiencia y precisión. Se conoció una técnica nueva (AAM) que puede ser utilizada para muchas otras cosas que tengan que ver con por ejemplo segmentación y se implementó también una librería, en C++, OpenCv y OpenGl de Modelos Activos de Apariencia, que puede ser utilizada con otros fines.

Capítulo 8

Trabajo futuro

8.1. Trabajo futuro y posibles utilidades

Como trabajo futuro se propone intentar hacer a los modelos activos de apariencia más robustos en cuanto a texturas. Intentar hacer más eficiente el proceso e implementar una actualización que permita utilizar estos modelos con muestras pre entrenadas, genéricas, sin la necesidad de entrenar el modelo específicamente con el objeto con el que va a ser utilizado. Quizás aplicar filtros a las texturas haciendo que estas sean menos vulnerables en diferencias de color, luz, etc.

Se podría pensar en implementar AAMs con CUDA (lenguaje de procesamiento en paralelo de la placa de video NVidia), y tratar de hacer una implementación que soporte varios modelos activos de apariencia simultáneos, por ejemplo para el análisis de respuesta de audiencia.

Otra posibilidad sería utilizar la precisión que se logra obtener con Modelos Activos de Apariencia, en formas más simples que una cara humana, por ejemplo la boca, e intentar realizar lectura automática de labios, ojos, etc.

Utilidades de segmentación. Detectar formas, o patrones de distintas imágenes, con determinadas restricciones de forma (provenientes de las muestras en el entrenamiento). Esto podría ser utilizado para segmentación automática en secuencias médicas, reconocimiento de partes del esqueleto humano en radiografías, etc.

Agregando al tiempo como una variable extra, podrían analizarse gestos, señas, etc y ser utilizadas para algún tipo de interfaz de uso de una computadora para personas con discapacidades motrices.

El reconocimiento de expresiones en sí podría ser utilizado en robótica, para interacción hombre maquina, etc.

Agradecimientos

Agradezco principalmente a Julio Jacobo, mi director, por la paciencia y la atención durante todo el desarrollo de este trabajo.

A mi familia, mi mamá Adriana y mi papá Tato. A mis hermanos Miguel, Pablo y Eva, a mi novia Ire, y a mis amigos que me ayudaron durante la carrera y el desarrollo de este trabajo.

¡Gracias!

Bibliografía

[1] Paul Viola, Michael Jones *Robust Real-time Object Detection*. Second international workshop on statistical and computational theories of vision-modeling, learning, computing,and sampling - Vancouver, Canada, July 13, 2001.

[2] Paul Viola, Michael Jones *Rapid Object Detection using a Boosted Cascade of Simple features, IEEE*. 2001.

[3] A. L. C. Barczak, M. J. Johnson & C. H. Messom *Real-time Computation of Haar-like features at generic angles for detection algorithms*. Institute of Information & Mathematical Sciences Massey University at Albany, Auckland, New Zealand, 2006.

[4] Jan Sochman, Jirí Matas *AdaBoost* Center for Machine Perception, Czech Technical University, Prague, 1998.

[5] Pedro Alexandre Dias Martins, *Active Appearance Models for Facial Expression Recognition and Monocular Head Pose Estimation*. University of Coimbra, Department of Electrical and Computer Engineering, 2008.

[6] Iain Matthews and Simon Baker, *Active Appearance Models Revisited*. Carnegie Mellon University, The Robotics Institute, 2004.

[7] Christian Martin, *A Real-time Facial Expression Recognition System based on Active Appearance Models using Gray Images and Edge Images*. MetraLabs GmbH, Germany, 2008.

[8] Simon Baker, Ralph Gross, and Iain Matthews, *Lucas-Kanade 20 Years On: A Unifying Framework*, 2003.

[9] Hugo Mercier, Julien Peyras, Patrice Dalle, *Toward an Efficient and Accurate AAM Fitting on Appearance Varying Faces*. Universite Paul Sabatier, Italia, 2006.

[10] Tim Cotes, *An Introduction to Active Shape Models*. Oxford University Press, 2000.

[11] G.J. Edwards, C.J. Taylor and T.F. Cootes, *Interpreting Face Images using Active Appearance Models*. Wolfson Image Analysis Unit, Department of Medical Biophysics, University of Manchester.

[12] Paul Viola, Michael Jones *Rapid Object Detection using a Boosted Cascade of Simple Features*. Mitsubishi Electric Research,Compaq Cambridge Research Lab Labs,Cambridge, 2001.

[13] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf *An Introduction to Kernel-Based Learning Algorithms*. 2001.

[14] Steve R. Gunn *Support Vector Machine for Classification and Regression*. 1998.

[15] Raúl Rojas *AdaBoost and the Super Bowl of Classifiers A Tutorial Introduction to Adaptive Boosting* Freie Universitat Berlin, 2009.

[16] Don Fussell *Affine Transformations* University of Texas at Austin, 2010.

[17] Bo Liu, Zhi-Feng Hao, Xiao-Wei Yang *Nesting support Vector Machine For Multi-Classification* Department of Applied Mathematics, South China University of Technology, China, 2005.

[18] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[19] *SVM Opencv* `http://docs.opencv.org/modules/ml/doc/support_vector_machines.html`

[20] *Gaussian Blur* `http://en.wikipedia.org/wiki/Gaussian_blur`

[21] *OpenCv - Open Source Computer Vision* `http://opencv.org/`

[22] *OpenGl* `http://www.opengl.org/`

[23] *GLUT - The OpenGL Utility Toolkit* `http://www.opengl.org/resources/libraries/glut/`

[24] *Overview Viola And Jons Detector* `http://bitsearch.blogspot.com.ar/2012/12/overview-of-viola-jones-detector-in.html`