UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

Sistema de reconocimiento de expresiones faciales aplicado a la interacción humano-computadora usando redes neuronales y flujo óptico

T E S I S

para obtener el título de:

MATEMATICO

Presenta:

Javier Arturo Porras Luraschi

Directora de Tesis:

Mat. María Concepción Ana Luisa Solís González-Cosío

2005

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ACT. MAURICIO AGUILAR GONZÁLEZ
Jefe de la División de Estudios Profesionales de la Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

"Sistema de reconocimiento de expresiones faciales aplicado a la interacción humano-computadora usando redes neuronales y flujo óptico"

realizado por Javier Arturo Porras Luraschi

con número de cuenta 40206555-1 , quien cubrió los créditos de la carrera de: Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director Propietario	Mat. María Concepción Ana Luisa Solís González-Cosío
Propietario	Dr. Boris Escalante Ramírez
Propietario	Dr. David Rosenblueth Laguette
Suplente	Dra. Sofía Natalia Galicia Haro
Suplente	Dra. Elisa Viso Gurovich

Consejo Departamental de Matemáticas

M. en C. Alejandro Bravo Mojica

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE
MATEMÁTICAS

Sistema de Reconocimiento de Expresiones Faciales aplicado a la interacción Humano-Computadora usando Redes Neuronales y Flujo Óptico

Por Javier Arturo Porras Luraschi

Resumen

Este trabajo analiza, describe e implementa un sistema capaz de reconocer distintas expresiones faciales basándose en el reconocimiento de características en las expresiones y del movimiento de la cara a través de sucesiones de imágenes. Se argumenta la factibilidad del sistema y propone usar métodos clásicos del flujo óptico y redes neuronales, investigando los movimientos faciales naturales del ser humano y determinando el campo de movimiento de los mismos. De esta forma, se complementará la investigación fisiológica con la tecnológica para asegurar un buen desempeño general.

Reconocimientos

Quiero agradecer primeramente a la Universidad Nacional Autónoma de México por acogerme en sus aulas e instalaciones durante mis estudios de licenciatura, a los profesores por su dedicación durante las clases para transmitirme claramente sus conocimientos y su apoyo en los momentos de dudas y dificultades.

A los administrativos por brindarme las herramientas para prosperar eficazmente en la realización de mis metas de corto plazo para lograr un excelente resultado a futuro, a mis sinodales: Boris Escalante Ramírez, David Rosenblueth Laguette, Elisa Viso Gurovich y Sofía Natalia Galicia Haro; muy en especial a Ana Luisa Solís González Cosío mi asesora, por dirigirme, motivarme, y orientarme en la realización de cada una de las etapas de esta tesis.

A mi familia por haberme apoyado en todo momento y por su motivación para que no me desanimara y alcanzara todas mis metas venciendo todos los retos que se me presentaron a lo largo de la carrera. A mi hermano por ser mi hermano estando a mi lado, a mi madre por preocuparse por mí, a mi padre por apoyarme incondicionalmente, a mis tíos y a mis abuelos por consentirme y apoyarme en todo momento.

A mis compañeros de licenciatura por su apoyo y aportaciones para facilitar la realización de este proyecto y en especial a los voluntarios que ayudaron en la recopilación de expresiones para el reconocimiento facial.

A mi novia, por estar junto a mí; a mis amigos Pedro Arzac, Mauricio Ledesma y Cesar Maguey quienes con su buen humor me animaron en todo momento.

Contenido

Lista de Tablas y Figuras

Una imagen vale más que mil palabras
Napoleón I, 1796-1821

Capítulo I - Introducción

Importancia del reconocimiento de expresiones faciales

El rostro es el espejo de la mente, y los ojos sin hablar confiesan los secretos del corazón.
San Jerónimo (374 -419)

Desde el inicio del hombre las herramientas tecnológicas han desempeñado un papel de gran importancia para la supervivencia de la especie. Hoy en día éstas son cada vez más complejas y por tanto se requiere facilitar la forma de utilizarlas. En el caso concreto de este trabajo se propone utilizar el reconocimiento facial de expresiones para mejorar la interacción del ser humano con las computadoras.

Las expresiones faciales en el ser humano son un mecanismo natural para mostrar satisfacción, disgusto, duda, etc. pues son medios de expresión que cualquier persona puede detectar sin importar raza, edad o sexo; sin embargo, esta tarea es compleja para los sistemas actuales pues determinar el estado interno de una persona mediante sus expresiones requiere ponderar muchísimas variables.

Un sistema capaz de obtener información de los gestos de las personas tendría un impacto gigantesco en aplicaciones como las que se presentan a continuación:

Medida de aprendizaje. Los sistemas de aprendizaje actuales no tienen modo de saber qué tan bien está asimilando los conocimientos el usuario, para satisfacer esta necesidad de evaluación se deben aplicar exámenes al final de cada lección; sin embargo, esto es un tanto ineficiente pues el usuario pudo no haber entendido desde el comienzo. Un sistema de reconocimiento facial puede ayudar a monitorear casi de manera instantánea el nivel de aprendizaje del usuario, lo cual es deseable pues se pueden aplicar medidas de reforzamiento o acelerar el aprendizaje oportunamente.

Compresión[1] de videos digitales. En el estándar MPEG-4 se describe un sistema de compresión de videos usando un modelo de rostro genérico. En el modelo del rostro genérico se proyecta la textura del rostro de la persona capturada, usando diferentes configuraciones del modelo que describen distintas expresiones faciales. Esto permite codificar sólo las expresiones faciales, la posición de la cabeza y la textura de la misma en lugar de 30 cuadros por segundo que contienen gran volumen información por imagen. [CHA01] describe un sistema de reconocimiento facial de expresiones con aplicaciones a la animación facial del MPEG-4.

Animación de personajes virtuales. Hoy en día un método común para capturar expresiones faciales se basa en usar marcadores físicos sobre la cara del actor, que a su

[1] Entendiendo compresión como la reducción del tamaño del video por cualquier tipo de algoritmo

vez se modelan sobre la retícula tridimensional que conforma al modelo virtual, estos equipos son costosos e inviables en el uso común; sin embargo usando una cámara Web podemos aplicar este reconocimiento para modelar las expresiones sobre avatares[2] de ambientes de interacción virtuales.

Detección de cansancio. Es claro que gran parte de los accidentes en la industria son causa del cansancio, al igual que los accidentes automovilísticos. De esta forma, detectar el nivel de cansancio expresado por el rostro, aunado a un mecanismo de alarma, sería muy adecuado tanto para la industria como para autos inteligentes, incorporando tan sólo una cámara y un dispositivo de procesamiento.

Interactividad en videojuegos. Los videojuegos se han convertido en un nicho de mercado demandante en el que el nivel de interactividad se convierte en un factor determinante; los juegos tienden a ser programados para brindarle un reto suficiente mas no excesivo al usuario, esto se logra generalmente haciendo uso de un incremento gradual en la dificultad. El problema está en que en muchas ocasiones el usuario juzgará el juego por su primera impresión tachándolo de difícil o sencillo tras los pocos segundos de haberlo iniciado, por lo que un juego equipado con un sistema de reconocimiento facial podría brindar la retroalimentación necesaria para ajustarse al perfil de jugador casi al instante en el que inicia su uso.

Medida de interés en buscadores y tiendas electrónicas. Una de las grandes pérdidas en el comercio electrónico ha sido la falta de agentes capaces de convencer a los compradores, y es que en muchos casos un buen vendedor es capaz de vender un producto que el usuario ni siquiera necesita. Tales vendedores hacen uso de las expresiones faciales, vestuario, modales, etc. de las personas con las que interactúan. Un sistema capaz de detectar el nivel de satisfacción de un producto en un cliente sería de mucha ayuda para incrementar las ventas de la empresa; de la misma forma, un buscador que ofrece búsquedas podría determinar cuales resultados están mejor relacionados con los deseos del usuario basándose nuevamente en su expresión facial.

Fundamento psicológico

Se afirma [EKM78] que existen cuatro clases de signos generales en las que un rostro muestra información:

1) Signos faciales **estáticos**. Son características propias al rostro como la forma natural del mismo, proporciones entre ojos y boca, color de ojos, etc.
2) Signos faciales **lentos**. Son cambios en la apariencia del rostro con el paso del tiempo como arrugas, color de piel, etc.
3) Signos **artificiales**. Son aditamentos ajenos al rostro como lentes, cosméticos, aretes, etc.
4) Signos faciales **rápidos**. Son cambios causados por actividad muscular que influye en la apariencia del rostro, tal es el caso de una sonrisa, guiño, mueca, etc.

[2] Un avatar es conocido como la representación gráfica del usuario bajo un ambiente virtual

El estudio de este trabajo se limita al de los "Signos faciales rápidos" pues son los que la neuropsicología afirma que están ligados al estado emocional de cada persona, y son controlados de forma involuntaria generalmente.

Se ha desarrollado un sistema para medir el movimiento facial denominado FACS *"Facial Action Coding System"* cuyo objetivo es clasificar los principales movimientos de la cara como subir una ceja, cerrar un ojo, etc. Éstos a su vez conforman los AU *"Action Unit"* que son combinaciones de FACS y determinan una expresión facial como sorpresa o disgusto.

La tabla [1.1] muestra las principales unidades de acción descritas en [EKM02]. El sistema de FACS extiende también las acciones a los movimientos de cabeza y ojos.

Sistema de Codificación de Acciones Faciales		
AU	**Nombre del FACS**	**Base Muscular**
1	Alzar ceja interna	Frontalis, Pars Medialis
2	Alzar ceja externa	Frontalis, Pars Lateralis
4	Bajar ceja	Hundimiento del Glabellae
5	Levantar párpado	Levator Palpebrae Superioris
6	Levantar cachetes	Orbicularis Oculi, Pars Orbitalis
7	Apretar párpados	Orbicularis Oculi, Pars Palebralis
8	Juntar labios	Orbicularis Oris
9	Contraer nariz	Levator Labii Superioris, Alaeque Nasi
10	Levantar labio superior	Levator Labii Superioris, Caput Infraorbitalis
11	Hundir área naso-labial	Zygomatic Minor
12	Levantar la esquina labial	Zygomatic Major
13	Contraer cachetes	Caninus
14	Hundir cachetes	Buccinnator
15	Hundir la esquina labial	Triangularis
16	Hundir el labio inferior	Depressor Labii
17	Levantar barbilla	Mentalis
18	Ondular labios	Incisivii Labii Superioris; Incisivii Labii Inferioris
20	Estirar labios	Risorius
22	Sacar labios	Orbicularis Oris
23	Encoger labios	Orbicularis Oris
24	Presionar labios	Orbicularis Oris
25	Separar labios	Hundimiento del Labii, o Relajación del Mentalis
26	Abrir mandíbula	Masetter; Temporal y Internal Pterygoid Relaxed
27	Abrir boca	Pterygoids; Digastric
28	Succionar labios	Orbicularis Oris
38	Dilatar cavidades nasales	Nasalis, Pars Alaris
39	Comprimir cavidades nasales	Nasalis, Pars Transversa and Depressor Septi Nasi
41	Semiabrir párpados	Relajación del Levator Palpebrae Superioris
42	Abrir ojos	Orbicularis Oculi
43	Cerrar ojos	Relajación del Levator Palpebrae Superioris
44	Semicerrar ojos	Orbicularis Oculi, Pars Palpebralis
45	Parpadear	Pars Palpebralis
46	Guiñar	Orbicularis Oculi

Sistema FACS. Acciones principales

Sistema FACS para Movimientos de Cabeza y de Ojos	
51	Rotar cabeza a la izquierda
52	Rotar cabeza a la derecha
53	Mover cabeza arriba
54	Mover cabeza abajo
55	Inclinar cabeza a la izquierda
56	Inclinar cabeza a la derecha
57	Cabeza hacia adelante
58	Cabeza hacia atrás
61	Ojos a la izquierda
62	Ojos a la derecha
63	Ojos hacia arriba
64	Ojos hacia abajo

Sistema FACS para Movimientos de cabeza y de ojos

Interfaz Humano – Computadora

Una interfaz Humano-Computadora constituye el hardware y software que tienen como finalidad realizar comunicación entre usuarios y computadoras. El ser humano cuenta con mecanismos de expresión naturales como los ademanes, gestos, tono de voz, etc. Sin embargo, la interacción Humano-Computadora o HCI[3] no explota estas características naturales a los hombres. Cualquier interfaz que utilice estas cualidades de manera eficiente y coherentemente tendrá una ventaja gigante con respecto a los dispositivos de interacción actuales (ratón, teclado, lápices ópticos, pantallas táctiles, etc.) pues el hombre tiende a tomar el camino más sencillo en la realización de sus tareas, así que la mayor eficiencia de comunicación se alcanza con dispositivos que interactúen con el usuario de forma natural.

El procesamiento del lenguaje natural, reconocimiento y generación de habla, reconocimiento de escritura y de expresiones faciales constituyen algunas de las áreas de desarrollo de interfaces de comunicación naturales. El presente trabajo centra sus esfuerzos en reconocer expresiones faciales en tiempo real durante un proceso de interacción directo con la computadora.

Planteamiento del problema

Con el fin de enfocar este trabajo a un problema en particular, se optó por plantear la aplicación del reconocedor facial para mejorar la interacción en ambientes virtuales, en específico para laboratorios virtuales.

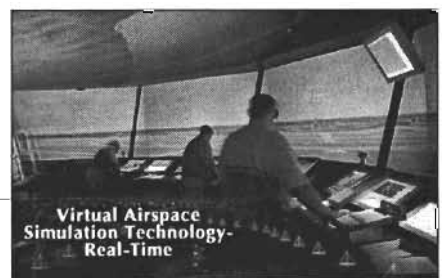
Los laboratorios hoy en día requieren de mayor especialización por lo que se vuelven más difíciles de construir y mantener. El error físico de una persona aprendiendo puede llegar a ser muy costoso; en ocasiones la capacidad de los laboratorios es insuficiente para la

[3] Human Computer Interaction

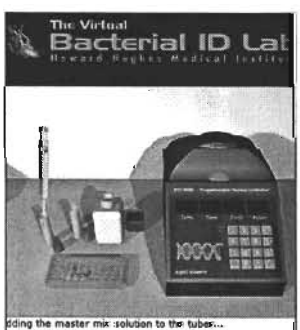
demanda que se requiere; por fortuna, las simulaciones por computadora son cada vez más precisas y las herramientas para interactuar dentro de éstas más realistas, así que podemos modelar laboratorios enteramente virtuales de manera eficiente y confiable. Estos ambientes de interacción con modelos digitales se llaman laboratorios virtuales cuando su finalidad es transmitir o generar conocimiento bajo condiciones controladas y simuladas en el usuario final.

Varias empresas de tecnología, sitios educativos y de investigación cuentan hoy en día con laboratorios virtuales tan complejos como aeropuertos o tan sencillos como modelos mecánicos unidimensionales; sin embargo tienen el mismo propósito (generar conocimientos) aunque ciertamente para distintas personas y necesidades. A continuación se presenta una pequeña muestra de los que se usan hoy en día.

NASA Simulation Laboratories. Cuenta con distintos laboratorios enfocados a la industria aeroespacial, como el VAST-RT[4] y el FFC[5] en el que el tráfico de un aeropuerto es simulado completamente en tiempo real con el fin de probar situaciones adversas, y aumentar la seguridad y eficiencia en los aeropuertos.



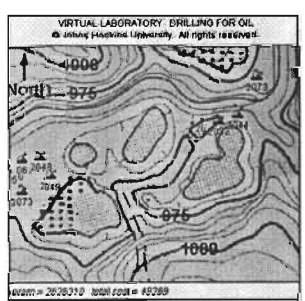
Nasa Simulation Laboratories



Howard Hughes

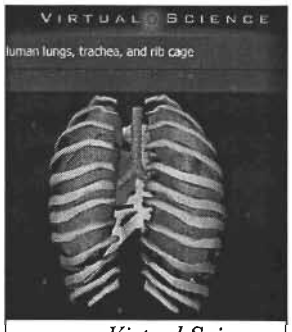
Howard Hughes Medical Institute. Presentan laboratorios de inmunología, cardiología y neuropsicología, con diversos objetivos como identificar bacterias o preparar moscas transgénicas.

Johns Hopkins University. Se presentan diversos laboratorios de ingeniería, algunos clásicos como lo es el de circuitos eléctricos, control de brazos robóticos, propagación de sonido y otros menos comunes como el de extracción de petróleo, que es inviable construirlo por las dimensiones geográficas en las que se busca un yacimiento petrolero.



John Hopkins

[4] Virtual Airspace Simulation Technology-Real-Time
[5] FutureFlight Central



Virtual Science

Virtual Science. Es una empresa dedicada a crear material multimedia con fines médicos; algunos de éstos simulan "laboratorios" que permiten examinar órganos en completo movimiento lo cual facilita el aprendizaje de los nuevos médicos e investigadores.

Objetivos

En el resto de este trabajo se propondrá, implementará y analizará un sistema de detección de expresiones faciales con el fin de mejorar el aprendizaje en ambientes virtuales haciendo uso de la retroalimentación visual de los movimientos del usuario. Los objetivos en específico que se consideraron son:

1. Diseñaremos un sistema capaz de detectar las principales expresiones faciales durante el proceso de capacitación en un laboratorio virtual. Las expresiones que se busca determinar son:
    1.1 Atención
    1.2 Distracción
    1.3 Sorpresa
    1.4 Duda
    1.5 Razonamiento

2. Detectaremos estas expresiones durante el proceso de aprendizaje con el fin de entender mejor el problema y proponer un modelo consistente para detectarlas de forma automática

3. Implementaremos el diseño del sistema de detección de expresiones faciales

4. Evaluaremos la efectividad del diseño e implementación propuestos

Resumen

Existe un crecimiento en el uso de dispositivos digitales para transmitir y generar conocimiento. Sin embargo, las interfaces de interacción entre humanos y computadoras no son del todo eficientes. Las expresiones faciales de cada persona están ligadas inconscientemente a la forma en la que asimilamos información por lo que podemos mantener una mejor comunicación si logramos incorporar expresiones faciales al modelo de interacción actual.

Se diseñará, implementará y analizará un modelo para reconocer las principales expresiones faciales resultantes de interactuar de forma continua con una computadora.

Los capítulos cubrirán los siguientes aspectos:

Capítulo I. Presenta una introducción al presente trabajo.
Capítulo II. Fundamento teórico de las redes neuronales y el flujo óptico.
Capítulo III. Definición de la arquitectura propuesta para el sistema.
Capítulo IV. Implementación y resultados.
Capítulo V. Conclusiones.

Capítulo II – Bases Teóricas

Redes Neuronales

Una mente muy activa no es mente después de todo.
Theodore Roethke (1908-1963)

Introducción

Existen procesos que el ser humano es capaz de realizar con facilidad y naturalidad. Para estos casos, en ocasiones es conveniente simular el modelo simplificado en el que trabaja nuestro cerebro usando el modelo del ANS[6]. Este modelo se fundamenta biológicamente en los componentes de una neurona y en la organización de estas para procesar información. En 1888 Ramón y Cajal mostraró que fluye información en una neurona al entrar a modo de impulsos a través de sus *dendritas*, después ponderarlos en el *soma* para determinar si la neurona emitirá a su vez un nuevo estímulo a través de su *axón*; de esta forma la neurona puede ser vista como un componente independiente que procesa información basada en impulsos eléctricos.

El modelo de Neurona

El modelo formal de una neurona se define como $f(xw') = y$ donde f es la función de activación, $x = [\,1, x_1, \ldots, x_p\,]$ es el vector de entradas, $w = [w_0, w_1, \ldots, w_p]$ es el vector de pesos y w_0 el valor de activación. La función de activación $f : \mathbb{R} \to (0,1)$ es creciente. En forma desarrollada tendremos que,

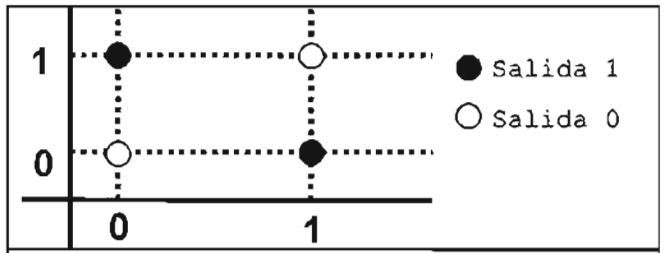
$$y = f\left(\sum_{i=0}^{p} x_i w_i\right), \text{ con } 0 < f(x) = \frac{1}{1+e^{-x}} < 1.$$

Restricciones al usar una neurona como clasificador

Un clasificador lineal (como es el caso de una neurona) es capaz de dividir el espacio de clasificación por una línea, plano o hiperplano; es decir, divide el espacio usando un polinomio de grado máximo uno. Por desgracia, no siempre es posible clasificar patrones usando únicamente clasificadores lineales; un problema que imposible clasificar linealmente es el del *"OR"* exclusivo, o XOR. Esto es una gran limitante para el modelo de neurona pues en sistemas complejos la clasificación generalmente no es lineal.

En la siguiente figura se muestra la tabla de verdad en forma gráfica del operador XOR, de la imagen se puede observar que es imposible definir una región que agrupe las salidas encendidas (1) y apagadas (0) usando tan solo una línea recta.

[6] Artificial Neural Systems – Sistema Neuronal Artificial



No linealidad en XOR

Algunos de los operadores booleanos pueden ser clasificados linealmente. Tal es el caso de la conjunción, disyunción o implicación; sin embargo la doble implicación o el XOR no son conjuntos clasificables por un operador lineal. La tabla y el archivo XML muestran estas características de forma concreta.

Caso	A	B	A and B	A or B	A xor B	B => A
0	0	0	0	0	0	1
1	0	1	0	1	1	0
2	1	0	0	1	1	1
3	1	1	1	1	0	1

Tabla de entrenamiento no lineal

Usamos el término patrones (archivo) de entrenamiento (en este caso la tabla de verdad) para referirnos al conjunto de patrones con el que alimentaremos a la neurona (o red neuronal) que a su vez los intentará clasificar.

El archivo de entrenamiento en este trabajo se implementó con formato XML. A continuación se muestra la estructura del archivo de entrenamiento que corresponden a la tabla de verdad anterior.

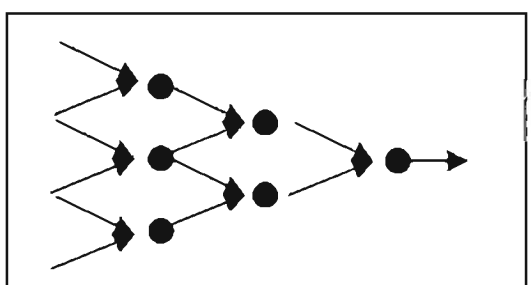
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="formato.xsl"?>
<entrenamiento>
  <definicion>
    <defcasos total="4"/>
    <defentradas total="2"/>
    <defsalidas total="4"/>
    <defepocas total="200"/>
  </definicion>
  <caso numero="0">
    <fuente imagen="" x="" y="" width="" height=""/>
    <entradas imagen="">
      <e v="0.0"/><e v="0.0"/>
    </entradas>
    <salidas>
      <s v="0.0"/><s v="0.0"/><s v="0.0"/><s v="1.0"/>
    </salidas>
    <clasificacion correcta="no" valor="0"/>
  </caso>
  <caso numero="1">
    <fuente imagen="" x="" y="" width="" height=""/>
```

```
    <entradas imagen="">
      <e v="0.0"/><e v="1.0"/>
    </entradas>
    <salidas>
      <s v="0.0"/><s v="1.0"/><s v="1.0"/><s v="0.0"/>
    </salidas>
    <clasificacion correcta="no" valor="0"/>
  </caso>
  <caso numero="2">
    <fuente imagen="" x="" y="" width="" height=""/>
    <entradas imagen="">
      <e v="1.0"/><e v="0.0"/>
    </entradas>
    <salidas>
      <s v="0.0"/><s v="1.0"/><s v="1.0"/><s v="1.0"/>
    </salidas>
    <clasificacion correcta="no" valor="0"/>
  </caso>
  <caso numero="3">
    <fuente imagen="" x="" y="" width="" height=""/>
    <entradas imagen="">
      <e v="1.0"/><e v="1.0"/>
    </entradas>
    <salidas>
      <s v="1.0"/><s v="1.0"/><s v="0.0"/><s v="1.0"/>
    </salidas>
    <clasificacion correcta="no" valor="0"/>
  </caso>
</entrenamiento>
```

Archivo de entrenamiento XML

El modelo de una red neuronal

Para solucionar la restricción de linealidad en la clasificación se utiliza un modelo con varias capas de redes neuronales, éste simulando el modelo natural en donde el axón a su vez transmite el impulso a hacia las dendritas de neuronas cercanas, a esta estructura la denominamos "red neuronal". A continuación se desarrolla la teoría que sustenta el proceso de entrenamiento de la red neuronal, el cual consiste en minimizar la función de error que genera la clasificación.



Estructura multicapa

Definamos L capas con k_i neuronas en la i-ésima capa, $k \in \mathbb{N}^L$. Buscamos minimizar los pesos $w_j^r(new)$ de la j-ésima neurona proveniente de la neurona r incluyendo el valor activación en w_j^0. Los pesos serán recalculados por un proceso de aproximación iterativo de la forma,

$$w_j^r(new) = w_j^r(old) + \Delta w_j^r \ , \ \ \Delta w_j^r = -\mu \frac{\partial J}{\partial w_j^r} \tag{2.1}$$

siendo $w_j^r(old)$ el valor de los pesos en la iteración anterior, J una función de error general ligada a la salida de la red que deberá ser minimizada, y ε la función de error específica para la capa; son definidas como

$$J = \sum_{i=1}^{N} \varepsilon(i) \ , \ \ \varepsilon(i) = \frac{1}{2} \sum_{m=1}^{K_L} \left(y_m(i) - \hat{y}_m(i) \right)^2 . \tag{2.2}$$

siendo N el número de parejas de entrenamiento $\left(y(i), x(i)\right)$ con $y(i) \in \mathbb{R}^{k_L}, x(i) \in \mathbb{R}^{K_0}$. Para minimizar J se usará el método convencional del "Algoritmo de Gradiente Descendiente", donde el gradiente dependerá de los pesos de la red, y estará definido por $\frac{\partial \varepsilon(i)}{\partial w_j^r}$.

Para calcularlo definiremos a $v_j^r(i)$ como la salida antes de aplicar la función de activación en la r-ésima capa de la j-ésima neurona,

$$v_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{j0}^r \equiv \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) . \tag{2.3}$$

Debemos suponer que $y_0^r(i) \equiv 1, \forall r, i$ para asegurar que el valor de activación no se altere.

```
float getV(vector<float> &pEst) {
   if(pEst.size() != w.size()-1) {
      cout << "[Error] float getV(vector<float> &)" << endl;
      cout << "  Los tamanos no coinciden" << endl;
   }
   float v = w[0]; // Asignar Umbral
   for(unsigned int i=1; i<w.size(); i++) {
      v += w[i]*pEst[i-1];
   }
   return v;
}
```

Implementación de $v_j^r(i)$

Aplicamos la regla de la cadena $\frac{\partial \varepsilon(i)}{\delta w_j^r} = \frac{\partial \varepsilon(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial w_j^r}$, y de (2.3) se tiene que

$$\frac{\partial}{\partial w_j^r} v_j^l(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} v_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} v_j^r(i) \end{bmatrix} = y^{r-1}(i), \quad y^{r-1}(i) = \begin{bmatrix} +1 \\ y_1^{r-1}(i) \\ \vdots \\ y_{k_{r-1}}^{r-1}(i) \end{bmatrix}, \tag{2.4}$$

pues para cada entrada de la matriz las derivadas parciales son precisamente el k-ésimo término de la sumatoria.

Definamos, $\frac{\partial \varepsilon(i)}{\partial v_j^l(i)} \equiv \partial_j^r(i)$, por lo que (2.1) se rescribe

$$\Delta w_j^r = -\mu \sum_{i=1}^{N} \partial_j^r(i) y^{r-1}(i). \tag{2.5}$$

Para encontrar el valor de $\partial_j^r(i)$ aplicaremos el algoritmo de propagación hacia atrás, iniciando desde $r = L-1, L-2, \cdots, 1$.

Consideremos $r = L$, entonces $\partial_j^L(i) = \frac{\partial \varepsilon(i)}{\partial v_j^L(i)}$, $\varepsilon(i) = \frac{1}{2} \sum_{m=1}^{K_L} \left(f(v_m^L(i)) - y_m(i) \right)^2$.

Tenemos,

$$\partial_j^L(i) = \frac{1}{2} \frac{\partial}{\partial v_j^L(i)} \sum_{m=1}^{K_L} \left(f\left(v_m^L(i) - y_m(i)\right)^2 \right) = \frac{1}{2} \sum_{m=1}^{K_L} \left(\frac{\partial}{\partial v_j^L(i)} f\left(v_m^L(i) - y_m(i)\right)^2 \right),$$

$$\partial_j^L(i) = \frac{1}{2} \sum_{m=1}^{K_L} \left(2 e_m(i) \frac{\partial}{\partial v_j^L(i)} f\left(v_m^L(i) - y_m(i)\right) \right) = e_j(i) \frac{\partial}{\partial v_j^L(i)} f\left(v_j^L(i)\right).$$

De esta forma, $\partial_j^L(i) = e_j(i) f'\left(v_j^L(i)\right)$, siendo f' la derivada de f

```
/// <metodo nombre="e" parametros="1">
///   <descripcion>
///     Error en la ultima capa (L-1) durante el entrenamiento
///   </descripcion>
float e(int j) {
     return vOut[j]-red[L-1][j].fSalida;
}
/// </metodo nombre="e" parametros="1">
```

Error en la j-ésima neurona de la ultima capa

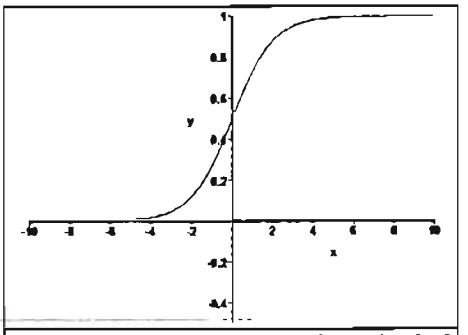
$f(x)=\dfrac{1}{1+e^{-x}}$, de ahí que $f'(x)=\dfrac{e^{-x}}{\left(1+e^{-x}\right)^2}=\left(\dfrac{1}{1+e^{-x}}\right)\left(\dfrac{e^{-x}}{1+e^{-x}}\right)$,

$$f'(x)=\left(\frac{1}{1+e^{-x}}\right)\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)=\left(\frac{1}{1+e^{-x}}\right)\left(1-\frac{1}{1+e^{-x}}\right)=f(x)\big(1-f(x)\big).$$

Por lo que f' será definida en términos de f para mejorar el uso de las salidas en la red.

```
static float f(float x) {
  return (float)(1.0/(1.0+exp(0.0-x)));
}
static float fp(float x) {
  return x*(1-x);
}
```

Definición de f y f'



Graficación de f

Considerando ahora $r<L$, $v_j^{r-1}(i)$ influye en todo $v_k^r(i), k=1,2,\cdots,k_r$ de la capa siguiente; aplicando la regla de la cadena se tiene,

$$\partial_j^{r-1}(i)=\frac{\partial\varepsilon(i)}{\partial v_j^{r-1}(i)}=\sum_{k=1}^{k_r}\frac{\partial\varepsilon(i)}{\partial v_k^r(i)}\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}=\sum_{k=1}^{k_r}\partial_k^r(i)\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}. \tag{2.6}$$

Ahora bien,

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}=\frac{\partial\left[\sum_{m=0}^{k_{r-1}}w_{km}^r y_m^{r-1}(i)\right]}{\partial v_j^{r-1}(i)}, \tag{2.7}$$

con

$$y_m^{r-1}(i)=f\left(v_m^{r-1}(i)\right),\ \text{así que}\ \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)}=w_{kj}^r f'\left(v_j^{r-1}(i)\right). \tag{2.8}$$

De (2.6) y (2.8),

$$\partial_j^{r-1}(i)=\left[\sum_{k=1}^{k_r}\partial_k^r(i)w_{kj}^r\right]f'\left(v_j^{r-1}(i)\right)=e_j^{r-1}(i)f'\left(v_j^{r-1}(i)\right). \tag{2.9}$$

```
/// <metodo nombre="e" parametros="2">

///   <descripcion>
///     Calcula el error por backpropagation en las capas de
///     la red previas a la capa de salida
///   </descripcion>
float e(int r, int j) {
      float sum = 0;
      for(int idx=0; idx<k[r+1]; idx++) {
            sum += red[r+1][idx].fDelta*red[r+1][idx].w[j+1];
      }
      return sum;
}
/// </metodo nombre="e" parametros="2">
```

Error en la j-ésima neurona de la r-ésima capa

```
/// <metodo nombre="delta">
float delta(int r, int j) {
      float ret;
      if(r == L-1)
            ret = e(j)*EREDN::fp(red[L-1][j].fSalida);
      else
            ret = e(r,j)*EREDN::fp(red[r][j].fSalida);
      red[r][j].fDelta = ret;
      return ret;
}
/// </metodo nombre="delta">
```

Delta en la j-ésima neurona de la r-ésima capa

donde,

$$e_j^{r-1}(i) = \sum_{k=1}^{k_r} \partial_k^r(i) w_{kj}^r . \qquad (2.10)$$

```
/// <bloque nombre="backprop">
///   <descripcion>
///     Aplica el algoritmo de propagacion hacia atras
///   </descripcion>
for(r=0; r<L; r++) {
 int tope = l+1;
 if(r>0)
  tope = k[r-1]+1;
 /// <for condicion="j<k[r]">
 for(int j=0; j<k[r]; j++) {
  float fDelta = red[r][j].fDelta;
  float fTFactor = cMiu*fDelta;
  red[r][j].w[0] += fTFactor;

  /// <for condicion="m<tope">
  for(int m=1; m<tope; m++) {
   if(r==0) {
    float fFactor = cMiu*fDelta*vIn[m-1];
    red[r][j].w[m] += fFactor;
   }
   else
   {
    float fFactor = cMiu*fDelta*red[r-1][m-1].fSalida;
    red[r][j].w[m] += fFactor;
   }
  }
  /// </for condicion="m<tope">
 }
 /// </for condicion="j<k[r]">
}
/// </bloque nombre="backprop">
```

Algoritmo de propagación hacia atrás

El algoritmo de entrenamiento propuesto difiere un poco en el implementado, pues el propuesto requiere que todos los casos de prueba se encuentren previamente seleccionados. Esta limitación se ve clara en

$$\Delta w_j^r = -\mu \sum_{i=1}^{N} \partial_j^r(i) y^{r-1}(i),$$

pues la sumatoria recolecta todos los casos de prueba. Sin embargo esto en ocasiones es inconveniente pues los archivos de entrenamiento deberán modificarse continuamente. Así que se adoptó el algoritmo de Robbins-Monro que permite agregar casos y entrenarlos arbitrariamente pues define la iteración como

$$w_j^r(i+1) = w_j^r(i) - \mu \partial_j^r(i) y^{r-1}(i).$$

Para el caso de entrenamiento de la tabla de verdad, se obtuvo la siguiente red usando una configuración de 2 capas con $k = [2, 6, 4]$.

	-6.18271	Muestras: 60000
	2.80046	Clasificaciones Correctas: 4
	-0.229343	Clasificaciones Incorrectas: 0
	5.99045	Uso Total: 48
	-3.45733	
-2.03166	2.7854	
4.4148	0.672571	
-0.0527267		
	-3.40059	
-2.48069	2.98881	
-3.04166	2.36615	
4.00351	2.50954	
	-1.98304	
-7.16854	3.02157	
4.45908	5.51885	
5.04381		
	-4.77914	
1.51611	-0.333696	
1.10255	0.906761	
-4.5407	-8.30347	
	0.709641	
-3.32594	-1.63574	
3.2022	9.76542	
1.61702		
	0.0456505	
-3.07836	3.9421	
6.85415	-5.13018	
6.33535	0.453445	
	5.36434	
	2.81663	
	-1.38256	

Pesos de la red de la tabla de verdad

		Sumatoria	Escalon			Sumatoria	Escalon
	-2.0317	-2.03166	0.115919		-6.1827	-8.57984	0.000188
0	4.4148			0.115919	2.80046		
0	-0.0527			0.077223	-0.2293		
				0.00077	5.99045		
	-2.4807	-2.48069	0.077223	0.819965	-3.4573		
0	-3.0417			0.034692	2.7854		
0	4.00351			0.044009	0.67257		
	-7.1685	-7.16854	0.00077		-3.4006	-4.1478	0.015593
0	4.45908			0.115919	2.98881		
0	5.04381			0.077223	2.36615		
				0.00077	2.50954		
	1.51611	1.51611	0.819965	0.819965	-1.983		
0	1.10255			0.034692	3.02157		
0	-4.5407			0.044009	5.51885		
	-3.3259	-3.32594	0.034692		-4.7791	-3.79929	0.021896
0	3.2022			0.115919	-0.3337		
0	1.61702			0.077223	0.90676		
				0.00077	-8.3035		
	-3.0784	-3.07836	0.044009	0.819965	0.70964		
0	6.85415			0.034692	-1.6357		
0	6.33535			0.044009	9.76542		
					0.04565	4.542235	0.969463
				0.115919	3.9421		
				0.077223	-5.1302		
				0.00077	0.45345		
				0.819965	5.36434		
				0.034692	2.81663		
				0.044009	-1.3826		

Pesos del 1er renglón en la tabla de verdad

		Sumatoria	Escalon			Sumatoria	Escalon
	-2.0317	-2.0843867	0.110624		-6.1827	-4.50744	0.010906
0	4.4148			0.110624	2.80046		
1	-0.0527			0.820953	-0.2293		
				0.106716	5.99045		
	-2.4807	1.52282	0.820953	0.046327	-3.4573		
0	-3.0417			0.153304	2.7854		
1	4.00351			0.962923	0.67257		
	-7.1685	-2.12473	0.106716		-3.4006	4.82593	0.992045
0	4.45908			0.110624	2.98881		
1	5.04381			0.820953	2.36615		
				0.106716	2.50954		
	1.51611	-3.02459	0.046327	0.046327	-1.983		
0	1.10255			0.153304	3.02157		
1	-4.5407			0.962923	5.51885		
	-3.3259	-1.70892	0.153304		-4.7791	4.227701	0.985624
0	3.2022			0.110624	-0.3337		
1	1.61702			0.820953	0.90676		
				0.106716	-8.3035		
	-3.0784	3.25699	0.962923	0.046327	0.70964		
0	6.85415			0.153304	-1.6357		
1	6.33535			0.962923	9.76542		
					0.04565	-4.33249	0.012964
				0.110624	3.9421		
				0.820953	-5.1302		
				0.106716	0.45345		
				0.046327	5.36434		
				0.153304	2.81663		
				0.962923	-1.3826		

Pesos del 2o renglón en la tabla de verdad

Introducción

Para cualquier objeto en movimiento en un espacio tridimensional visto por un observador podremos definir el concepto de *flujo óptico* como la proyección de la velocidad real del objeto (v_x, v_y, v_z) sobre el plano de visión del observador. Nuestros esfuerzos en esta sección radicarán en intentar deducir el flujo de imagen haciendo uso de dos imágenes subsecuentes capturadas del objeto en movimiento.

Existen diversas técnicas para calcularlo, en [YEO01] se considera un resumen de las principales que se presenta a continuación.

Técnicas diferenciales. Las técnicas diferenciales calculan derivadas espacio-temporales de las intensidades de la imagen o de las imágenes filtradas usando filtros pasa-bajas y pasa-altas. Suponen que la función de intensidad es diferenciable. Estas técnicas son representadas principalmente por:

- **Horn y Schunck.** Este método utiliza dos restricciones y una constante λ que define la influencia del término de suavidad. El método original utiliza derivadas de primer orden.
  Se busca minimizar $E = \int_S (\nabla I \cdot v + I_t) + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2) ds$
- **Lucas y Kanade.** El método de Lucas y Kanade implementa mínimos cuadrados sobre las restricciones de primer orden en una vecindad Ω. Minimiza $\sum_{x \in \Omega} W^2(x)\left(\nabla I(x,t) \cdot \bar{v} + I_t(x,t)\right)^2$ donde W es una ventana de pesos que le da más influencia a las restricción en el centro. Los pesos de W son (0.0625, 0.25, 0.375, 0.25, 0.0625).
- **Nagel.** Su método usa derivadas de segundo orden para medir el flujo óptico. La restricción esta basada en la suavidad, la cual no se impone sobre intensidades del gradiente abruptas, es decir, bordes; pues intentó manejar el problema de oclusión[7].
- **Uras, Girosi, Verri y Torre.** Utilizan derivadas de segundo orden y una solución local de la velocidad. Dividen la imagen en regiones de 8x8 pixeles y el mejor estimado que cumpla con $\|M^T \nabla I\| << \|\nabla I_t\|$ con $M = \begin{pmatrix} u_x & u_y \\ v_x & v_y \end{pmatrix}$.

[7] Problema en el que un objeto en movimiento pasa por atrás de un cuerpo rígido o semitransparente, por lo que el movimiento del objeto es difícil de predecir.

Basados en regiones. Esta familia de métodos busca encontrar la velocidad, como un el mejor desplazamiento entre regiones a lo largo del tiempo. Buscan la mejor alternativa definiendo medidas como la SSD[8] – Sum-of-squared difference.

- **Anandan**. Esta técnica utiliza SSD y pirámides para calcular desplazamientos grandes y mejorar la estructura de la imagen en los bordes. La técnica de pirámides utiliza distintas resoluciones de la imagen, para encontrar la mejor ocurrencia.
- **Singh.** Utiliza dos etapas, la primera calcula los valores de SSD, mientras que la segunda propaga la velocidad usando restricciones de vecindad.

Basados en energía. Los métodos basados en energía utilizan la energía que generan filtros de velocidad. También son llamados métodos basados en frecuencia porque el diseño de los filtros se realiza en un dominio de Fourier. Algunos métodos son equivalentes a los basados en gradientes o en correlación.

- **Heeger.** Heeger formulo una aproximación por mínimos cuadrados en un espacio de frecuencias. La energía local es extraida usando filtros de energía de Gabor que se configuran para responder a distintas orientaciones espaciales y distintas frecuencias.

Basados en fase. Estos métodos se basan en el comportamiento de la velocidad definida por su fase.

- **Waxman, Wu y Bergholm.** Waxman et. al. aplicaron filtros espacio-temporales sobre mapas de bordes binarios para monitorear el movimiento de los bordes en tiempo real. Los mapas de bordes se suavizan con un filtro Gaussiano. Los contornos son monitoreados usando métodos diferenciales.

- **Fleet y Jepson.** En su técnica definen las componentes de la velocidad en términos del movimiento normal instantáneo. Filtros de banda son usados para descomponer la señal de acuerdo a la escala, velocidad y orientación. Técnicas diferenciales son aplicadas a la fase.

Determinación del flujo óptico por el método de Horn y Schunck

En este subtema muestra la estructura general del método de Horn y Schunck para calcular el flujo óptico, este desarrollo esta basado en la explicación de [MAL94].

Definimos *flujo óptico* como un conjunto de vectores $\bar{v} = (u, v)$ de velocidad sobre cada píxel de la imagen que describirán la rapidez y dirección del movimiento.

Ahora bien, si definimos la intensidad para cada píxel en la imagen como función del tiempo $I(x, y, t)$, y analizamos un pequeño desplazamiento, podremos expandirla como

[8] SDC. Suma de la diferencia de cuadrados

$$I(x+dx, y+dy, t+dt) = I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt + \ldots,$$

Desearíamos que conforme este incremento es cada vez más reducido, se cumpliera que

$$I(x+dx, y+dy, t+dt) = I(x,y,t),$$

o lo que es lo mismo, que

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0. \quad (3.1)$$

Dividiendo (3.1) entre dt obtenemos

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v, \quad (3.2)$$

con $\frac{dx}{dt} = u$ y $\frac{dy}{dt} = v$. El término $\frac{\partial I}{\partial t}$ representa el cambio en la intensidad de la imagen, $\frac{\partial I}{\partial x}$ tanto $\frac{\partial I}{\partial y}$ representan el cambio espacial de la intensidad; así que estos tres términos pueden ser estimados, sin embargo por cada ecuación tenemos dos incógnitas por lo que necesitaremos una segunda restricción.

Escribiendo (3.2) como un gradiente tendremos que
$-\frac{\partial I}{\partial t} = \nabla I(u,v)$, si definimos $I_t = \frac{\partial I}{\partial t}$ tendremos que (3.2) se escribe como

$$\nabla I \, \bar{v} + I_t = 0. \quad (3.3)$$

La ecuación (3.3) se conoce como *ecuación de restricción del flujo óptico.*

Para proponer la segunda restricción analizaron que el movimiento en píxeles adyacentes debería ser similar, a excepción de en las esquinas. Podemos estimar cuanto varía el flujo óptico en la imagen si consideramos la variación de la velocidad en cada coordenada definiendo

$$S = \iint_S \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 ds. \quad (3.4)$$

Podemos considerar también la variación de la restricción de flujo óptico como

$$C = \iint_S \left(\nabla I \cdot \bar{v} + I_t\right)^2 ds, \quad (3.5)$$

Teorema: Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}$ y $g : U \subset \mathbb{R}^n \to \mathbb{R}$ funciones C^1 con valores reales dadas. Sean $x_0 \in U$ y $g(x_0) = c$, y sea S el conjunto de nivel de g con valor c. Suponer $\nabla g(x_0) \neq 0$.
Si $f \mid S$, que denota "f restringida a S", tiene un máximo o un mínimo local en S, en x_0 entonces existe un número real λ tal que

$$\nabla f(x_0) = \lambda \nabla g(x_0).$$

Método de los multiplicadores de Lagrange[9]

de esta forma, podemos minimizar (3.4) y (3.5) como $S + \lambda C$ usando la técnica de multiplicadores de Lagrange. Esto es, minimizaremos

$$\iint_S \left(\nabla I \cdot \bar{v} + I_t\right)^2 + \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 ds.$$

Al reacomodar y aplicar la técnica de los multiplicadores de Lagrange deberemos minimizar el conjunto de ecuaciones diferenciales siguiente:

$$\nabla\left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right) = \lambda\nabla\left(\left(\nabla I \cdot \bar{v} + I_t\right)^2\right),$$

desarollando,

$$2\left(\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)\bar{u} + \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)\bar{v}\right) = \lambda\left(\left(\nabla I \cdot \bar{v} + I_t\right)\nabla I\right),$$

es decir,

$$\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)\bar{u} + \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)\bar{v} = \lambda\left(\left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)\left(\frac{\partial I}{\partial x}\bar{u} + \frac{\partial I}{\partial y}\bar{v}\right)\right)$$

y finalmente expandimos como

$$\begin{aligned}
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= \lambda\left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)\frac{\partial I}{\partial x}, \\
\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} &= \lambda\left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)\frac{\partial I}{\partial y}.
\end{aligned} \qquad (3.6)$$

λ no se conoce, sin embargo aunque su valor generalmente se estima empíricamente; describe que tanta influencia recibe la restricción de suavidad. En [MAL94] se recomienda hacer λ grande si las mediciones de intensidad son correctas; por el

[9] Marsden, Tromba, **Cálculo Vectorial**, Prentice Hall, 4ª ed.

contrario, si se encuentra ruido, se procura que λ sea pequeña. Con λ definida, el conjunto de ecuaciones (3.6) podrá ser resuelto por métodos numéricos sobre cada píxel en la imagen.

Determinación del flujo óptico por correlación

Para encontrar el flujo óptico utilizando métodos de correlación deberemos realizar una búsqueda que minimice el error en el desplazamiento de una región en movimiento. Usando el supuesto de la conservación de la intensidad en la imagen, para cada píxel en nuestra imagen original buscaremos la mejor correlación alrededor de una vecindad cuadrada de radio fijo; finalmente supondremos que la componente horizontal y vertical de desplazamiento son la velocidad puntual del campo de movimiento para las imágenes evaluadas.

Definimos una ventana de búsqueda como una región rectangular alrededor de un píxel que delimita la región en donde se buscará la mejor ventana de correlación. Una ventana de correlación, es también una región rectangular pero esta define la semejanza con el patrón que se busca.

En [AJI91] se propone utilizar un filtro de intensidades y un análisis por pirámides en resolución, para cada píxel *P(x,y)* en la posición *(x,y)*. En la primera imagen, una ventana de correlación W_p se forma alrededor del píxel. Una ventana de búsqueda W_s se establece alrededor del píxel en la posición *(x,y)* de la segunda imagen. La medida $M(\partial x, \partial y)$ entre W_p y una ventana similar en cada píxel de W_s, desplazada de *(x,y)* en $(\partial x, \partial y)$ se calcula como la suma de las diferencias al cuadrado como,

$$M(\partial x, \partial y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \left(I_1(x+i, y+j) - I_2(x+\partial x+i, y+\partial y+j)\right)^2 ,$$

donde $I_1(x,y)$ e $I_2(x,y)$ se refieren a la intensidad del píxel en la posición (x,y) en la primera y segunda imágenes filtradas, respectivamente.

Normalización del histograma

Podemos definir una imagen de A píxeles de ancho y L de largo como una función $p:\mathbb{N}\times\mathbb{N}\rightarrow[0,255]$ que describe únicamente la intensidad del píxel puntualmente.

El histograma se define como el número de píxeles que hay en la imagen para un cierto tono. Más formalmente definimos a $\phi:[[0,255]\times[0,255]]\rightarrow\{0,1\}$ como

$$\phi(x,c)=\begin{cases}0 & x\neq c\\ 1 & x=c\end{cases}$$

Entonces el histograma $h:[0,255]\rightarrow\mathbb{N}$, será

$$h(x)=\sum_{i=1}^{L}\sum_{j=1}^{A}\phi(p(i,j),x)$$

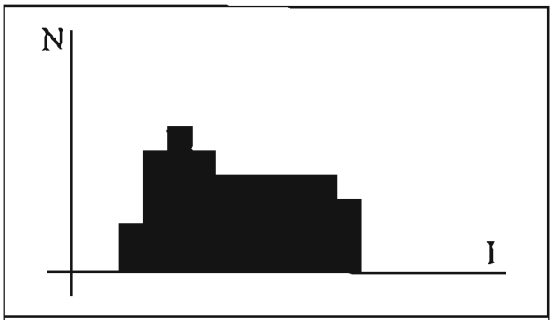
El siguiente fragmento de código define la función $h(x)$

```
/// <bloque>
///   <descripcion>Encontrar el Histograma de la Imagen</descripcion>
float d = 1.0f/fuente->width/fuente->height;
float hist[256];
for(int i=0; i<256; i++)
     hist[i]=0.0;
for(int j=0; j<fuente->height; j++)
     for(int i=0; i<fuente->width; i++)
          hist[(unsigned char)fuente->imageData[j*fuente->widthStep+i]] += d;
/// </bloque>
```

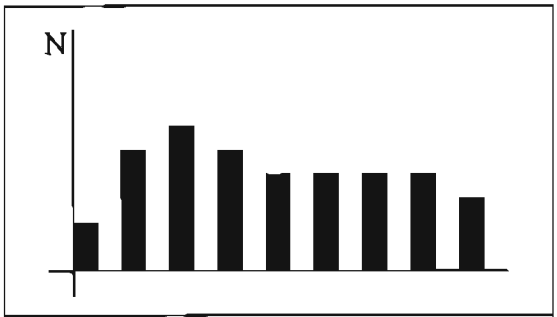
Implementación del cálculo del histograma

Existen dos operaciones principales que pueden realizarse para cambiar el histograma:

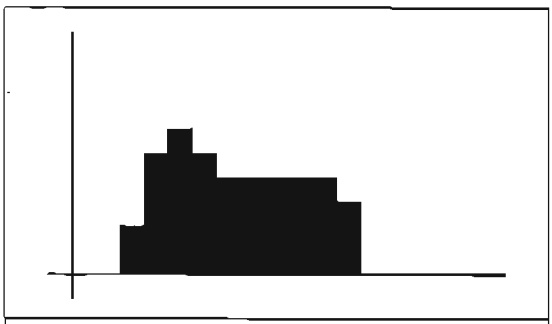
- La normalización modifica la imagen para intentar usar todas las intensidades.
- La ecualización modifica la imagen para intentar distribuir uniformemente las intensidades de los píxeles.



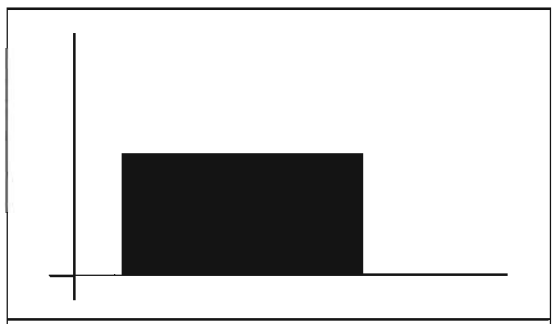
Histograma original



Histograma normalizado



Histograma original



Histograma ecualizado

Estamos interesados en normalizar el histograma durante el reconocimiento de patrones pues esto ayudará a reducir las diferencias entre patrones con distinta iluminación. De la misma forma la normalización puede ayudarnos a recuperar información perdida como la presentada en los ejemplos.

Si consideramos la función de distribución del histograma definida como,

$$H(c) = \frac{1}{AL}\int_0^c h(x)dx = \sum_{i=0}^{i=c} h(i),$$

```
/// <bloque>
///   <descripcion>Crear tabla de busqueda</descripcion>
float sum = 0.0f;
unsigned char lookup[256];
for(i=0; i<256; i++) {
     sum += hist[i];
     lookup[i] = (unsigned char)(sum*255.0+0.5);
}
/// </bloque>
```

Implementación de H(c)

H(c) nos mostrará la forma en la cual se usan las distintas intensidades en la imagen. Haciendo uso de *H* podemos modificar las intensidades en cada pixel si componemos

$$p'(i,j) = 255H\big(p(i,j)\big).$$

```
/// <bloque>
 ///   <descripcion>Encontrar el Histograma de la Imagen</descripcion>
 for(int j=0; j<fuente->height; j++) {
      for(int i=0; i<fuente->width; i++) {
            unsigned char idxLookUp = (unsigned char)
                  (fuente->imageData[j*fuente->widthStep+i]);
            char chrLookUp = (char)lookup[idxLookUp];
            dest->imageData[j*dest->widthStep+i] = (char)chrLookUp;
      }
 }
 /// </bloque>
```

Implementación del mapeo de intensidades

De esta forma, la nueva intensidad al ser compuesta con la distribución del histograma ampliará la distribución original a lo largo de toda la paleta de intensidades. Finalmente se muestran algunos ejemplos generados por la función de normalización del histograma implementada para este sistema, las imágenes normalizadas fueron generadas en su totalidad en este trabajo.

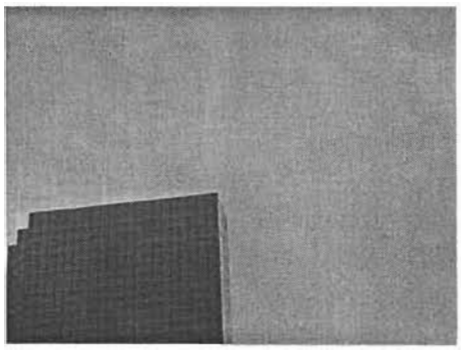


Imagen base. Ejemplo 1

Histograma normalizado. Ejemplo 1

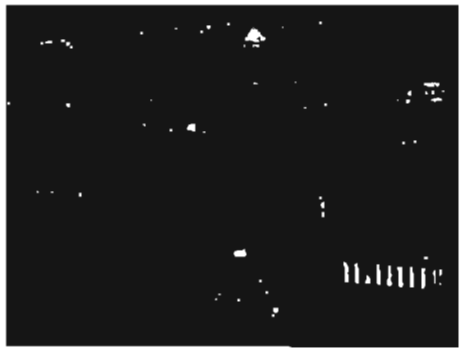


Imagen base. Ejemplo 2

Histograma normalizado. Ejemplo 2

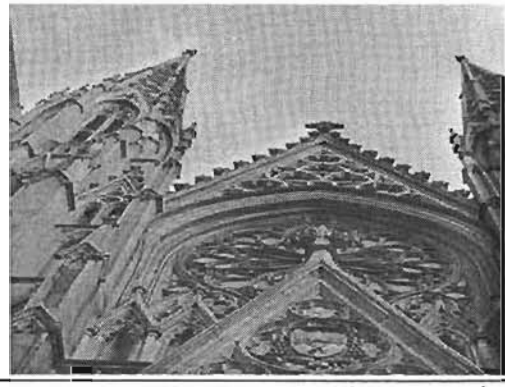


Imagen base. Ejemplo 3

Histograma normalizado. Ejemplo 3

Capítulo III – Arquitectura de Reconocimiento

Arquitecturas desarrolladas

El problema del reconocimiento de expresiones faciales se ha podido solucionar con buenos resultados realizando un seguimiento detallado sobre características importantes en la cara como cejas, boca, ojos, etc. Se afirma en [WIL90] que su método es capaz de dar buen seguimiento al modelo real de la cara sobre un modelo virtual realizando el seguimiento de los patrones usando flujo óptico. Sin embargo, su método requiere inicializar manualmente cada uno de los puntos a los que se les dará seguimiento, lo cual para nuestra aplicación no es deseable.

Varios investigadores han desarrollado trabajos relacionados con el reconocimiento de expresiones faciales. Suwa [SUW68] presenta un método que usa veinte puntos para monitorear el movimiento de la cara. Pentland [PEN97] desarrolló un modelo paramétrico basado en una geometría tridimensional de una cara para reconocer cinco expresiones prototipo. Mase [MAS91] seleccionó manualmente regiones faciales que corresponden a lo músculos usando flujo óptico. Yacoob [YAC96] desarrolló un sistema como el de Mase pero sobre regiones específicas como ojos, cejas y boca. En [YIN03] se continúa la descripción de algunos otros trabajos realizados, sin embargo concluye que estos métodos tienen como inconveniente que:

- Utilizan condiciones controladas de laboratorio con altas resoluciones en la imagen.
- La mayoría de los sistemas requieren de configuración manual
- Ninguno de los sistemas intenta lidiar con interacciones en un mundo real

La arquitectura inicial buscará combinar el método de [WIL90] con un reconocimiento automático de patrones usando redes neuronales, usando resoluciones comunes y sin esperar ambientes controlados. El seguimiento de flujo óptico se utilizará para capturar movimientos de la cara; esto es factible pues existen implementaciones en OpenCV capaces de detectar la posición de una cara en tiempo real.

Detectando la región de la cara y aplicando el flujo óptico sobre ésta, buscaremos determinar movimientos relevantes relacionados con diversas expresiones faciales.

Para realizar un análisis más detallado de expresiones en el rostro, aplicaremos detección de patrones sobre elementos importantes como ojos, boca, cejas, etc. que definan la expresión que la persona esta realizando.

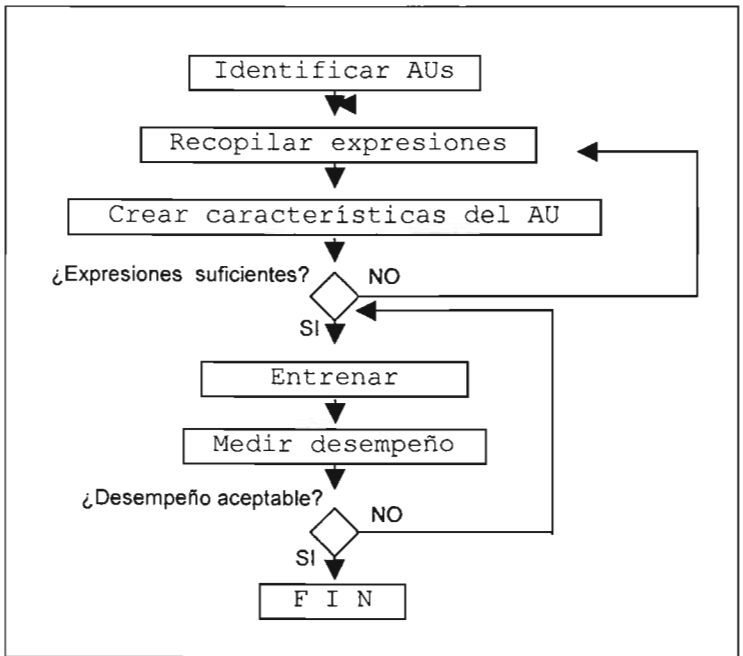
[YIN03] Utiliza una única red neuronal con una capa oculta, para determinar sobre una región de importancia como la boca, la expresión que la persona esta realizando. Nuestro trabajo implementará una red neuronal por expresión; esto nos permitirá evaluar más

fácilmente su desempeño, y en caso de ser necesario, realizar ajustes sólo sobre la expresión con desempeño pobre.

Propuesta

Considerando los objetivos planteados en el Capítulo I de este documento, se propone crear el sistema mediante un modelo versátil que permita expandir y corregir expresiones faciales. Esto se logrará usando un esquema de optimización del sistema con distintas fases:

a) Identificar AUs viables para el reconocimiento facial
b) Recopilar expresiones que muestren los AUs
c) Crear características del AU
d) Entrenar la red neuronal y definir las regiones de movimiento
e) Mostrar el sistema en funcionamiento usando los archivos entrenados



Proceso de optimización del sistema

Módulos

Para satisfacer el modelo de optimización y escalabilidad mencionados en la propuesta, se definen cinco módulos principales que se ajustan a las necesidades de cada fase.

a) Examen facial.
b) Captura de caras.
c) Clasificación.

d) Entrenamiento.
e) Reconocimiento.

Examen facial

Descripción

El examen facial tiene como finalidad capturar diferentes expresiones faciales durante el aprendizaje, haciendo uso de un cuestionario con preguntas extremas para crear en el usuario las expresiones faciales que buscamos. Durante la encuesta el usuario desconoce que está siendo grabado y que además sus expresiones están siendo clasificadas continuamente.

Este módulo opera haciendo uso de la definición del examen facial en un archivo XML donde se define la pregunta y el tipo de expresión que ésta causa, por ejemplo; una serie de preguntas repetitivas causa distracciones, preguntas irracionales como la 4 y 5 muestran expresiones de duda y confusión.

```
// Conjunto de expresiones disponibles en la evaluacion
```

nula	desinteres	interes	duda	sorpresa
0	1	2	3	4

```
// Conjunto de Preguntas
```

Num	Expresion	Pregunta
0	[0.nula]	Iniciar prueba? SI NO
1	[2.interes]	Cuanto es 7*7+1?
2	[2.interes]	Como buscarias un libro en especifico en una biblioteca que no tiene catalogo ni personas a quien preguntarle?
3	[2.interes]	Cuantas veces al dia las manecillas de reloj se enciman?
4	[3.duda]	Se ha demostrado que la relacion entre un homocubo y un espacio tetradimensional linealmente convexo e hiperseparable mantiene una proporcion equidistante, que entero positivo define la relacion de proporcianalidad en espacios con estas propiedades?
5	[3.duda]	Si definimos una serie puntualmente hiperconvergente de Lebesgue en un espacio semiplano potencialmente acotado en un subconjunto de Weirstrass, cuantas particiones hiperbolicamente convexas se pueden adjuntar de forma homeomorfa?
6	[4.sorpresa]	Cuanto es 1+1?

Archivo de entrenamiento en el Examen Facial

Justificación

Están bien estudiados los movimientos musculares que generan cada expresión facial. Sin embargo, la clasificación de las distintas expresiones tomadas por la cámara de video permitirá decidir que tipo de rasgos son fáciles de detectar con una cámara de videoconferencia común. Asimismo, algunas imágenes serán usadas para extraer rasgos significativos.

Captura de caras

Descripción

La finalidad de la "Captura de caras" es crear expresiones para el entrenamiento haciendo uso de actores que modelan las expresiones faciales más significativas que se obtuvieron del conjunto generado por el examen facial. El programa debe almacenar el rostro en un formato con pérdidas mínimas de información para clasificar los rasgos posteriormente.

Justificación

Crear expresiones adicionales a las generadas por el módulo de Reconocimiento Facial es necesario ya que para el entrenamiento se necesitan algunos miles de muestras del patrón por lo que es costoso tomar todas las muestras de expresiones naturales en distintos voluntarios durante un proceso de interacción.

Para determinar una aproximación del tiempo y muestras totales requeridas realizaremos la siguiente suposición: el examen puede durar algunos minutos, digamos 10, y las expresiones como sorpresa pueden durar tan sólo unos pocos segundos, digamos 3, una parte reducida estarán libres de ruido y viendo hacia la cámara, suponiendo que obtenemos el 5% de las muestras totales. Si suponemos que capturamos 20 cuadros por segundo, tendremos en 3 segundos 3 imágenes útiles para la clasificación; de esta forma si usamos 2000 muestras en el entrenamiento requeriremos realizar 666 exámenes faciales y clasificar las expresiones de estos exámenes podría tomar meses pues tan sólo se requerirían 111 horas en los exámenes suponiendo que somos perfectamente eficientes, despreciando el tiempo que tardemos en conseguir voluntarios, y sin cometer error alguno.

De esta forma, vemos que es muy importante usar este módulo para reducir el tiempo que requiere recopilar las muestras. Ahora bien, estas expresiones adicionales serán generadas con base en las muestras que se obtengan del examen facial por lo que se deben mantener en buena medida las características encontradas en el examen.

Clasificación

Descripción

La clasificación tiene como función determinar las características principales de la expresión, su finalidad es reducir el número de muestras que se requieren para entrenar la red neuronal, ya que si no se clasificasen deberíamos entrenar la red con la expresión facial completa, lo que provocaría un número de muestras mayor.

Las características principales que se clasificarán son los ojos y boca, definiendo también durante la clasificación la región donde es más probable encontrarlas.

Otra de las funciones de este módulo es la de preprocesar la imagen, realzando los detalles y ajustando la resolución de la imagen. Para este proceso se usará la normalización del histograma local.

Justificación

Al determinar las características de una expresión reducimos el problema de reconocimiento considerablemente y mejoramos la eficiencia del sistema, pues al preprocesar la imagen calculando el histograma local se realzan las características y se homogenizan los distintos niveles de luz en los que se puede realizar el reconocimiento.

Entrenamiento

Descripción

El módulo de entrenamiento toma como entrada el archivo de salida del módulo de clasificación, y entrenará a la red neuronal. En caso de que ésta ya esté entrenada se realiza su actualización con respecto a los nuevos casos de entrenamiento. Deberá desplegar estadísticas y resultados del proceso de entrenamiento para medir la efectividad en el aprendizaje.

Justificación

Una red neuronal es capaz de reconocer patrones distribuidos de forma no lineal; mientras que el proceso propio de reconocer patrones en la expresión facial representa un proceso complejo que difícilmente podrá ser implementado como algoritmo. El uso de redes neuronales permite realizar el proceso de reconocimiento de manera más natural, pues las expresiones se aprenden empíricamente sin usar mecanismos precisos; así que es una buena área de aplicación.

En el Capítulo II se modificó el algoritmo convencional de propagación hacia atrás por la versión de Robbins-Monro con el fin de realizar un entrenamiento continuo y permitirle al sistema agregar nuevos casos de entrenamiento con facilidad, lo cual es deseable por los niveles de escalabilidad que deseamos.

Reconocimiento

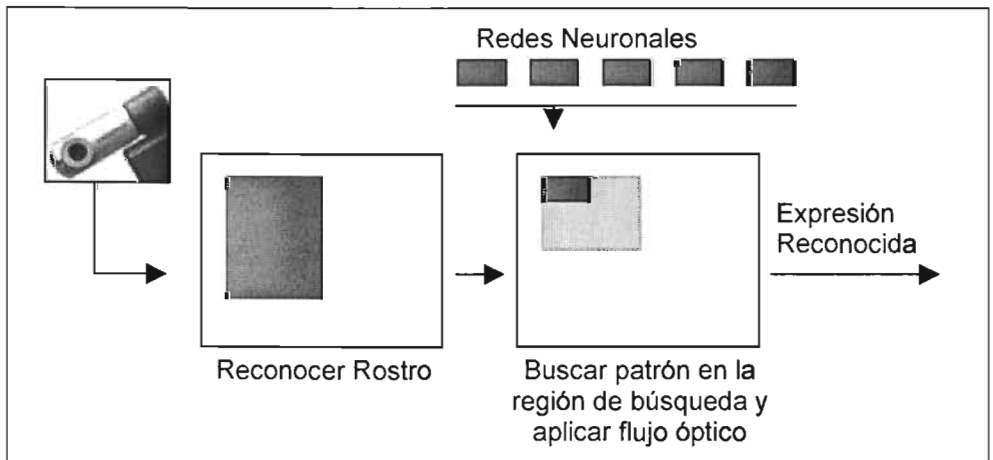
Descripción

La función del módulo de reconocimiento es utilizar la red neuronal entrenada y/o el flujo óptico en la imagen en tiempo real para determinar si el usuario se encuentra en alguno de los siguientes estados:

a) Atención.
b) Sorpresa.
c) Duda.
d) Distracción.
e) Razonamiento.

El proceso deberá utilizar tanto redes neuronales como flujo óptico según convenga. Sin embargo el algoritmo de reconocimiento será esencialmente:

1. Capturar imagen del dispositivo.
2. Reconocer la posición del rostro.
3. Iterar sobre el total de redes neuronales.
   3.1. Dentro del rostro desplazar la red neuronal en la región de búsqueda para encontrar un patrón usando la red neuronal.
4. Aplicar el flujo óptico para determinar movimientos importantes.
5. Mostrar la expresión reconocida.



Algoritmo de reconocimiento

Capítulo IV – Implementación y Resultados

Plataforma

El desarrollo se realizó sobre Windows XP y Visual Studio .NET usando la biblioteca OpenCV orientada al procesamiento de visión por computadora. Así mismo, para el manejo de XML se utilizó la biblioteca de código abierto Expat para C++. El dispositivo de captura fue una cámara convencional de videoconferencia Creative Labs con resolución de 320x240, bajo un procesador Celeron 1400MHz con 480Mb en RAM.

Características del Sistema	
General	Especifico
Sistema operativo	Windows XP
Biblioteca de visión	OpenCV
Biblioteca XML	Expat
Fuente	C++
Ambiente de trabajo	.NET
Procesador	Celeron 1400MHz
RAM	480Mb
Cámara	320x240 Creative Labs

Características del Sistema

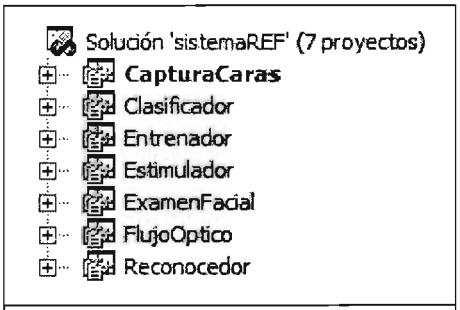
Aplicaciones

Se desarrollaron las aplicaciones principales que se proponen en el diseño de la arquitectura:

a) ExamenFacial.
b) CapturaCaras.
c) Clasificador.
d) Reconocedor.

y además algunas aplicaciones adicionales:

e) Módulo de generación de estímulos.
f) Módulo de flujo óptico.



Estructura Aplicaciones

Tanto el *módulo de generación de estímulos* como el *Módulo de flujo óptico* son aplicaciones que se usaron para facilitar el desarrollo; el *módulo de generación de estímulos* toma como entrada una imagen, convierte la imagen en un estímulo y finalmente obtiene el valor del estímulo que generó la red; por otra parte, el *módulo de flujo óptico* prueba la clase que lo implementa, esto es deseable pues depurar la salida del

flujo óptico en el propio *reconocedor* sería más complejo ya que debe utilizar distintos hilos[10] y además tiene la tarea de reconocer patrones sobre la imagen.

Existen aplicaciones para depuración (sus nombres concatenan una 'd') y las aplicaciones de distribución. Cada aplicación toma distintos parámetros de entrada pero estos generalmente serán a su vez archivos XML en donde se especifica el funcionamiento de la aplicación.

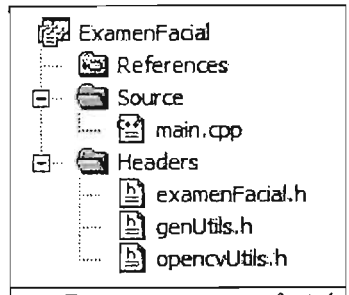
capcaras.exe	56 KB	Application	3/18/2005 10:44 AM
capcarasd.exe	104 KB	Application	2/27/2005 1:19 PM
clasificador.exe	244 KB	Application	2/14/2005 10:34 AM
clasificadord.exe	412 KB	Application	2/27/2005 1:19 PM
entrenador.exe	276 KB	Application	2/14/2005 12:06 PM
entrenadord.exe	616 KB	Application	2/27/2005 1:19 PM
estimulador.exe	228 KB	Application	3/13/2005 4:37 PM
estimuladord.exe	408 KB	Application	2/27/2005 1:19 PM
examenfacial.exe	244 KB	Application	3/13/2005 4:38 PM
examenfacialď.exe	336 KB	Application	3/4/2005 1:25 PM
flujooptico.exe	124 KB	Application	3/13/2005 11:21 PM
flujoopticod.exe	208 KB	Application	3/13/2005 11:11 PM
reconocedor.exe	272 KB	Application	3/14/2005 12:02 AM
reconocedord.exe	392 KB	Application	3/13/2005 11:18 PM

Listado de aplicaciones

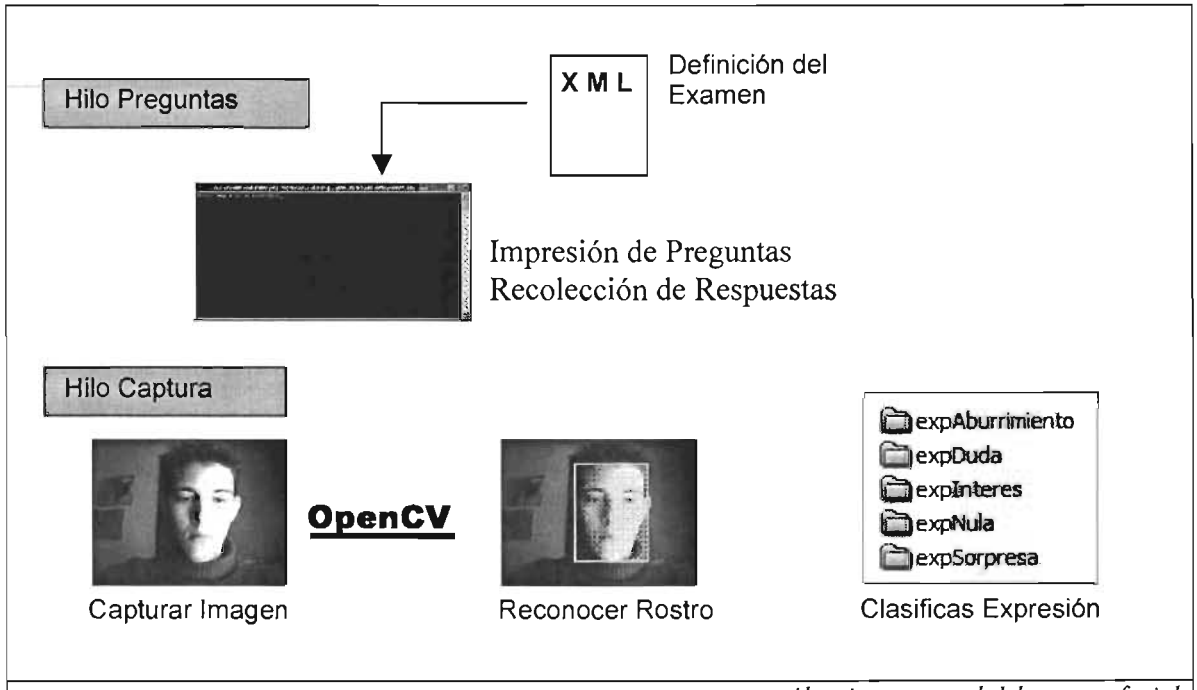
[10] Los hilos es un fragmento de código que corre de manera concurrente en una aplicación

Implementación

El archivo ExamenFacial.xml contiene los estados de ánimo que generan las distintas preguntas definidas en él. El programa las despliega en una ventana de consola cada una de las preguntas del examen, mientras otro hilo se encarga de reconocer en las imágenes el rostro (haciendo uso de las funciones de reconocimiento de OpenCV) y enviar las imágenes al directorio que le corresponde.



Estructura examen facial



Algoritmo general del examen facial

El hilo de preguntas se encarga de presentar las preguntas y capturar las respuestas. La aplicación busca recopilar las distintas expresiones y bajo el supuesto de que las preguntas generan la expresión facial deseada es irrelevante guardar la respuesta del usuario a la pregunta, así que simplemente las respuestas del usuario se desechan.

```
/// <funcion nombre="interfacePreguntas" tipo="hilo">
///   <descripcion>
///     Realiza las preguntas contenidas en la clase
///     del examen facial, la respuesta del usuario
///     es desechada
///   </descripcion>
void interfacePreguntas(void * data) {
      char respuesta[1000];
      while(efac->imprimePregunta()) {
            cout << endl << "Respuesta: ";
            cin.getline(respuesta,1000);
            cout << endl << endl << endl << endl;
      }

      bActivo = false;
      _endthread();
}
/// </funcion nombre="interfacePreguntas">
```

Implementación del hilo de preguntas

El examen propuesto se compone de preguntas clasificadas en: nula, desinterés, interés, duda y sorpresa. Las preguntas nulas omiten las expresiones realizadas. Esto es conveniente pues en la primera pregunta permitiremos que el usuario se adecue al ambiente, y en algunas otras como las previas a las expresiones de desinterés simplemente las expresiones no son significativas por lo que también se desechan.

Para las preguntas de interés (preguntas 1,2,3) se usaron preguntas sencillas de entender y algunos acertijos para captar la atención. En contraste, para las expresiones de duda se propusieron preguntas totalmente irracionales pero que parecen complejas y difíciles de contestar (preguntas 4,5). Después de un arduo intento por contestar preguntas complejas se presenta una pregunta en extremo sencilla (pregunta 6) para causar sorpresa; finalmente se hace una serie de preguntas repetitivas que son desechadas hasta llegar a las últimas 5 (preguntas 23-27) que continúan con el modelo de preguntas repetitivas para causar desinterés por el examen en el voluntario.

// Conjunto de expresiones disponibles en la evaluación

nula	desinterés	interés	duda	sorpresa
0	1	2	3	4

// Conjunto de Preguntas

Num	Expresión	Pregunta
0	[0.nula]	Iniciar prueba?
1	[2.interes]	Cuanto es 7*7+1?
2	[2.interes]	Como buscarias un libro en especifico en una biblioteca que no tiene catalogo ni personas a quien preguntarle?

3	[2.interes]	Cuantas veces al dia las manecillas de reloj se enciman?
4	[3.duda]	Se ha demostrado que la relacion entre un homocubo y un espacio tetradimensional linealmente convexo e hiperseparable mantiene una proporcion equidistante, que entero positivo define la relacion de proporcianalidad en espacios con estas propiedades?
5	[3.duda]	Si definimos una serie puntualmente hiperconvergente de Lebesgue en un espacio semiplano potencialmente acotado en un subconjunto de Weirstrass, cuantas particiones hiperbolicamente convexas se pueden adjuntar de forma homeomorfa?
6	[4.sorpresa]	Cuanto es 1+1?
7	[0.nula]	Cuanto es 2+1?
8	[0.nula]	Cuanto es 1+2?
9	[0.nula]	Cuanto es 2+2?
10	[0.nula]	Cuanto es 2+2?
11	[0.nula]	Cuanto es 3+3?
12	[0.nula]	Cuanto es 3+23?
13	[0.nula]	Cuanto es 3+10?
14	[0.nula]	Cuanto es 113+10?
15	[0.nula]	Cuanto es 3+6?
16	[0.nula]	Cuanto es 1+2?
17	[0.nula]	Cuanto es 3+3?
18	[0.nula]	Cuanto es 2+1?
19	[0.nula]	Cuanto es 2+2?
20	[0.nula]	Cuanto es 3+2?
21	[0.nula]	Cuanto es 2+1?
22	[0.nula]	Cuanto es 2+2?
23	[1.desinteres]	Cuanto es 3+2?
24	[1.desinteres]	Cuanto es 3+2?
25	[1.desinteres]	Cuanto es 2+1?
26	[1.desinteres]	Cuanto es 2+2?
27	[1.desinteres]	Cuanto es 3+2?

Contenido del archivo del examen facial

Resultados

Se realizaron 28 preguntas en total durante el examen facial a 25 personas, algunas expresiones fueron sencillas de causar en el voluntario, como interés, duda, y algunas más complejas como sorpresa; una vez clasificadas en directorios se tomó una muestra de las más representativas. Estos resultados se muestran en la tabla de imágenes del examen facial.

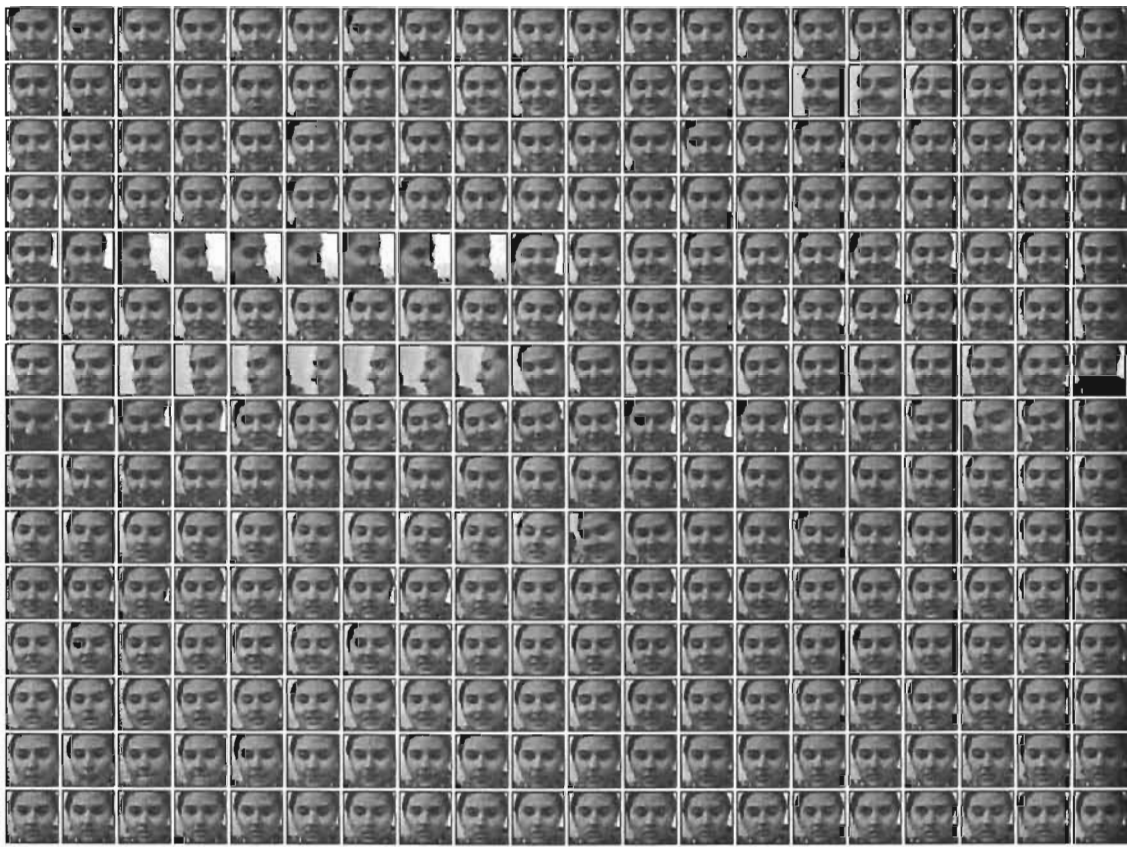
Preguntas por Expresión	
Expresión	Preguntas
Distracción	5
Duda	2
Interés / Pensando	3
Sorpresa	1
Total	11

Desglose de preguntas por expresión

Muestras por Expresión	
Expresión	Muestras
Distracción	607
Duda	4387
Interés / Pensando	4883
Sorpresa	320
Total	10197

Desglose de muestras por expresión

En la sucesión de imágenes de la tabla se aprecia claramente el movimiento de la cabeza horizontalmente al existir algún tipo de distracción.



Algunas imágenes capturadas durante el examen facial

Las imágenes muestra se encuentran recopiladas en el directorio "Muestras" y representan las expresiones más significativas durante el proceso.

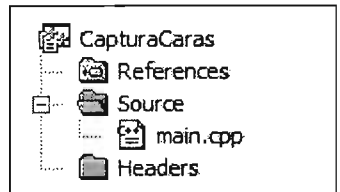
Distraccion	File Folder	2/14/2005 10:13 AM
Duda	File Folder	2/14/2005 10:15 AM
Interes	File Folder	3/17/2005 1:13 AM
Pensando	File Folder	2/14/2005 10:23 AM
Sorpresa	File Folder	2/14/2005 10:26 AM

Directorio de muestras

Imágenes seleccionadas del examen facial

Implementación

El programa permite capturar rostros haciendo uso de OpenCV y los almacena en el directorio especificado en los parámetros, el nombre de las imágenes se genera haciendo uso de un contador incremental como sufijo concatenado al nombre de la imagen.



Estructura captura caras

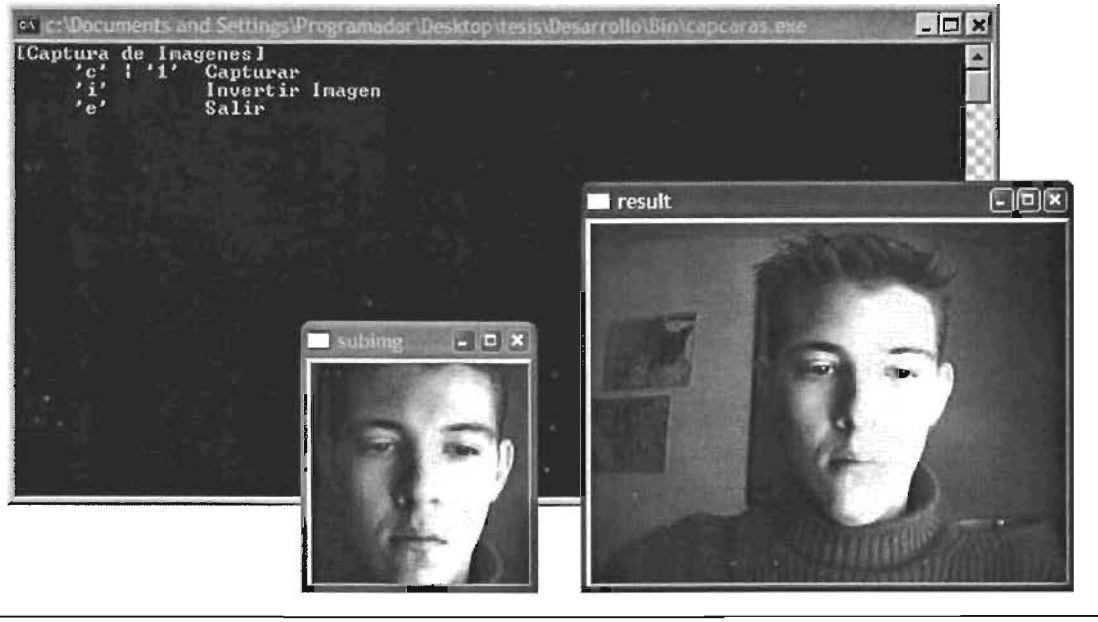
```
/// <funcion nombre="printParams">
///   <descripcion>
///     Imprime los parametros de este programa
///   </descripcion>
void printParams() {
      printf("[Parametros de Entrada]\n");
      printf("  -destino Ruta en la que se almacenaran las imagenes\n");
      printf("  -prefijo Prefijo de las imagenes\n");
      printf("  -cascada Cascada XML generada en openCV\n");
}
/// </funcion nombre="printParams">
```

Parámetros para la captura de caras

```
/// <funcion nombre="printUso">
///   <descripcion>Imprime el uso del programa</descripcion>
void printUso() {
      printf("[Captura de Imagenes]\n");
      printf("     'c' | '1'  Capturar\n");
      printf("     'i'        Invertir Imagen\n");
      printf("     'e'        Salir\n");
}
/// </funcion nombre="printUso">
```

Uso de la captura de caras

Parte del listado de las imágenes capturadas



Ejecución durante la captura de caras

Resultados

Este proceso fue satisfactorio pues, entre otras aportaciones, permitió tomar imágenes de la misma expresión con distintas iluminaciones tal como se muestra en la figura siguiente de imágenes, lo cual habría sido difícil de obtener usando únicamente el examen facial.

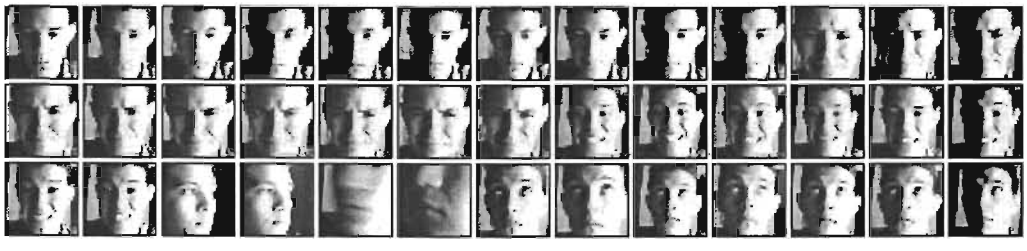
Algunas imágenes adicionales generadas durante la captura de caras

La siguiente tabla resume el total de imágenes generadas por expresión. Se consideró contar sólo las imágenes usadas en el proceso de entrenamiento pues el total de imágenes capturadas fue de 1719 y en la tabla sólo aparecen 1233.

Muestras capturadas por Expresión			
Expresión	Muestra	Expresión	Muestra
Distracción	306	Duda	208
Interés	223	Pensando	122
Sorpresa	335	Verificación	39
Total			1233

Desglose de las muestras clasificadas por expresión

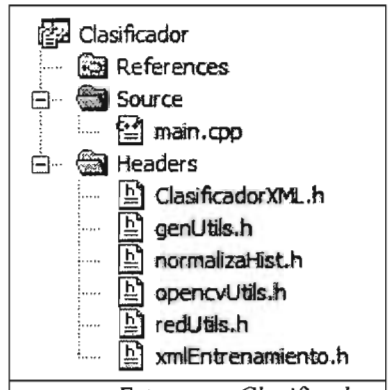
Las imágenes de verificación son usadas por el entrenador para medir la eficiencia con la que fue entrenada la red por lo que no se usan durante el entrenamiento. Esto se explica a detalle en la sección del entrenador.



Algunas imágenes de verificación

Implementación

La finalidad del clasificador es clasificar patrones eficientemente haciendo uso de las imágenes generadas por módulos anteriores, así que se optó por presentar imagen por imagen y haciendo uso del cursor, manualmente se selecciona sólo el área del patrón sobre la imagen que se desea clasificar.



Estructura Clasificador

El único parámetro de entrada es un archivo XML que describe las especificaciones de la aplicación.

```
/// <funcion nombre="printParams">
///   <descripcion>Imprime el uso del programa</descripcion>
void printParams() {
        cout << "[Parametros de Entrada]" << endl;
        cout << "  -archivoxml Especificaciones de la clasificacion en XML" << endl;
}
/// </funcion nombre="printParams">
```

Parámetros del clasificador

Las variables que se especifican son:

Errores. Representa el error en píxeles tanto en escala como en traslación que existe al seleccionar un patrón; al tener valores positivos se agregan al entrenamiento patrones con traslaciones y escalas distintas pero sin que estos patrones adicionales rebasen el error de escala o traslación que se indica.

Neurona. Identifica el ancho y largo de la neurona en píxeles; cada píxel será convertido en una entrada para la neurona durante el proceso de clasificación.

Archivos. Se especifica el directorio donde se encuentran almacenadas las imágenes que serán usadas para realizarla clasificación. También se especifica el archivo y ruta de salida; la ruta de salida es necesaria pues se crearán imágenes para cada patrón seleccionado.

Despliegue. Es el tamaño de la ventana en la que se visualizará el patrón.

Incrementos. Representan los píxeles que se agrandará la ventana a lo ancho y largo al seleccionar el patrón. Es importante que guarden proporción con el tamaño de la neurona pues de lo contrario el patrón se distorsionará.

```
<clasificador>
  <errores escala="0" traslacion="1"/>
  <neurona width="20" height="10"/>
  <archivos rutaentrada="..\..\Data\imgCaras\"
            imagenactual=""
            rutasalida="..\..\Data\localexp\Atencion\OjoDer\"
            entrenamiento="entrenamiento.xml"/>
  <despliegue width="200" height="100"/>
  <incrementos width="2" height="1"/>
</clasificador>
```

Parámetros del clasificador del archivo XML

Antes de explicar el funcionamiento del clasificador debemos tener claro que existen distintos tipos de muestras. En nuestro caso, como la red se entrenará para discernir si un patrón está dentro o no de un conjunto, podemos dividir las muestras en positivas (pertenecientes al conjunto) o negativas (no pertenecientes) y más específicamente en:

a) Muestra de entrenamiento. Es una muestra que usa la red neuronal directamente para generalizar el patrón.
   a. Positivas. Definen patrones de lo que se quiere clasificar.
   b. Negativas. Definen conjuntos que no son patrones, y por tanto la red no debe aceptar.
b) Muestra de verificación. Son muestras seleccionadas manualmente que funcionan como indicadores de la eficiencia con la que la red neuronal aprendió la información con la que se le entrenó. Estos ejemplos nunca son vistos por la red neuronal durante su entrenamiento
   a. Positivas. Definen patrones que deberían haber sido clasificados exitosamente.
   b. Negativos. Definen patrones que no deben haber sido clasificados como elementos del conjunto.

Los comandos para clasificar un patrón haciendo uso del teclado se especifican en las tablas de "Comandos", el carácter se delimita por [].

Tipos de muestras y sus comandos		
	Positivas	Negativas
Entrenamiento		[p]
Verificación	[v]	[v] [p]

Comandos para definir las muestras

Comandos del Clasificador	
Comando	Descripción
[e]	Cierra el programa y guarda los cambios en el archivo XML
[n]	Carga la siguiente imagen en el directorio
[m]	Salta 20 imágenes hacia delante en el directorio de imágenes
[v]	Alterna entre muestras de entrenamiento y verificación
[1]	Reduce la ventana de captura en (<incrementos.width> , <incrementos.height>)
[2]	Aumenta la ventana de captura en (<incrementos.width> , <incrementos.height>)
[h]	Normaliza el histograma global. Desactivado por omisión.
[j]	Normaliza el hisograma localmente. Activado por omisión.
[p]	Alterna entre muestras positivas y negativas
[i]	Reinicia la posición en el directorio de imágenes
<click>	Agrega la imagen al archivo de entrenamiento

Comandos del clasificador

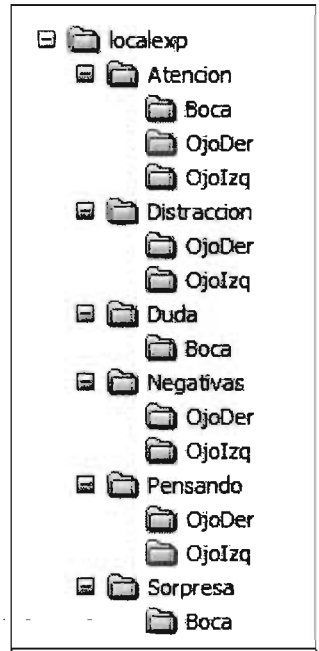
```
/// <funcion nombre="imprimeUso">
///   <descripcion>Imprime el uso del programa</descripcion>
void imprimeUso() {
        cout << endl << " [Clasifica Patrones] " << endl;
        cout << "   'e' Salir" << endl;
        cout << "   'n' Siguiente Imagen." << endl;
        cout << "   'm' Ir 20 Imagenes adelante." << endl;
        cout << "   'v' Agregara la imagen para verificacion." << endl;
        cout << "   '1' Reduce Capura en (incrementowidth,incrementoheight)" << endl;
        cout << "   '2' Aumenta Captura en (incrementowidth,incrementoheight)" << endl;
        cout << "   'h' Normaliza Histograma Global [Activar/Desactivar]" << endl;
        cout << "   'j' Normaliza Histograma Local [Activar/Desactivar]" << endl;
        cout << "   'p' Seleccionar Entradas [Positivas/Negativas]" << endl;
        cout << "   'i' Reiniciar en Capturas" << endl;
        cout << "   <click izquierdo> Agrega Entrada XML" << endl;

        cout << endl;
}
/// </funcion nombre="imprimeUso">
```

Uso del clasificador

Ejecución del clasificador

Resultados

Se clasificaron regiones de interés como ojos y bocas para los distintos tipos de expresiones; los resultados se muestran en las tablas de resultados.



Estructura de directorios

Información General de la Clasificación				
Expresión	Región	Entradas	Salidas	Casos
Atención	Boca	200	1	2630
	Ojo Derecho	200	1	3808
	Ojo Izquierdo	200	1	3393
Distracción	Ojo Derecho	200	1	5180
	Ojo Izquierdo	200	1	7018
Duda	Boca	200	1	3646
Negativas	Ojo Derecho	200	1	1462
	Ojo Izquierdo	200	1	1489
Pensando	Ojo Derecho	200	1	2939
	Ojo Izquierdo	200	1	2939
Sorpresa	Boca	200	1	3901
Total				38405

Información general de la clasificación

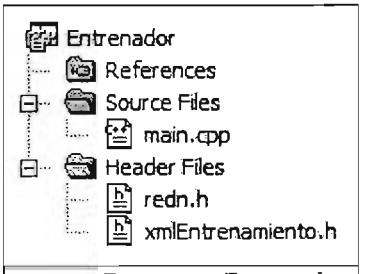
Con el fin de reducir el área de búsqueda durante el reconocimiento, el clasificador define el área posible donde el patrón se puede encontrar. Así mismo genera información adicional como la posición más probable en donde se puede encontrar al patrón.

Regiones Faciales Clasificadas										
Expresión	Región	Mínima			Máxima			Media		
		x	y	escala	x	y	escala	x	y	escala
Atención	Boca	0.277	0.659	0.243	0.692	0.992	0.348	0.341	0.756	0.291
	Ojo Derecho	0.533	0.277	0.151	0.887	0.483	0.236	0.710	0.380	0.193
	Ojo Izquierdo	0.113	0.282	0.139	0.449	0.492	0.227	0.279	0.388	0.174
Distracción	Ojo Derecho	0.518	0.226	0.128	0.952	0.515	0.285	0.730	0.351	0.211
	Ojo Izquierdo	0.030	0.208	0.140	0.480	0.513	0.262	0.143	0.325	0.224
Duda	Boca	0.237	0.648	0.269	0.737	0.971	0.458	0.465	0.710	0.280
Pensando	Ojo Derecho	0.513	0.255	0.153	0.891	0.513	0.236	0.740	0.336	0.178
	Ojo Izquierdo	0.513	0.255	0.153	0.891	0.513	0.236	0.740	0.336	0.178
Sorpresa	Boca	0.203	0.646	0.268	0.772	1.000	0.462	0.342	0.730	0.324

Regiones faciales clasificadas

Implementación

Los algoritmos de entrenamiento de redes neuronales se implementaron como se especificó en el Capítulo II; existen dos principales archivos de entrada usados en esta fase, primeramente el archivo de entrenamiento XML creado por el clasificador y un archivo de definición de la red neuronal, que a su vez será el archivo en el que se establecerán los pesos de la red.



Estructura Entrenador

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="..\..\..\xmlData\entrenamiento.xsl"?>
<entrenamiento>
  <definicion>
    <defcasos total="3808"/>
    <defcapturadas total="3088"/>
    <defverificacion total="44"/>
    <defentradas total="200"/>
    <defsalidas total="1"/>
    <defepocas total="50"/>
  </definicion>
  <estadisticas>
    <statregion minx="0.533784" maxx="0.887218" miny="0.277311"
      maxy="0.483607" minescala="0.151515" maxescala="0.236364"
      promx="0.71051" promy="0.380491" promescala="0.193927"/>
  </estadisticas>
  <caso numero="0" verif="no" generada="si">
    <fuente imagen="distraccion001.bmp" x="87" y="46" width="20" height="10"/>
    <entradas imagen="i0_c0_distraccion001.bmp">
      <e v="1"/>
    </entradas>
    <salidas>
      <s v="0"/>
    </salidas>
    <clasificacion correcta="si" valor="0.0 "/>
  </caso>
</entrenamiento>
```

Definición muestra del archivo de entrenamiento

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="redneuronal.xsl"?>
<red>
  <creacion>
    <descripcion></descripcion>
    <fecha></fecha>
    <autor></autor>
  </creacion>
  <definicion capas="3">
    <entradas total="200"/>
    <defparametros usarvalesperado="si" cmiu="0.25"/>
    <defcapa numcapa="0" neuronas="12" neuwidth="20" neuheight="10"/>
    <defcapa numcapa="1" neuronas="6" neuwidth="8" neuheight="1"/>
    <defcapa numcapa="2" neuronas="1" neuwidth="4" neuheight="1"/>
  </definicion>
</red>
```

Definición XML de una red neuronal

Podemos monitorear el estado en el que se encuentra la red al entrenarla usando el archivo de estado de la carpeta de "estados", este archivo nos muestra bloques de información relevantes como:

Época Actual. Una época tiene como objetivo dividir el proceso de entrenamiento de patrones en etapas; en nuestro desarrollo, una época se define como el entrenamiento de 10,000 patrones. La época actual representa la época que está siendo entrenada.

Número de ciclos en las entradas. Este número indica cuántas veces se le ha presentado un cierto patrón a la red neuronal en promedio.

Distancia mínima. Esta distancia es el mínimo error que se alcanzó durante el entrenamiento de la red.

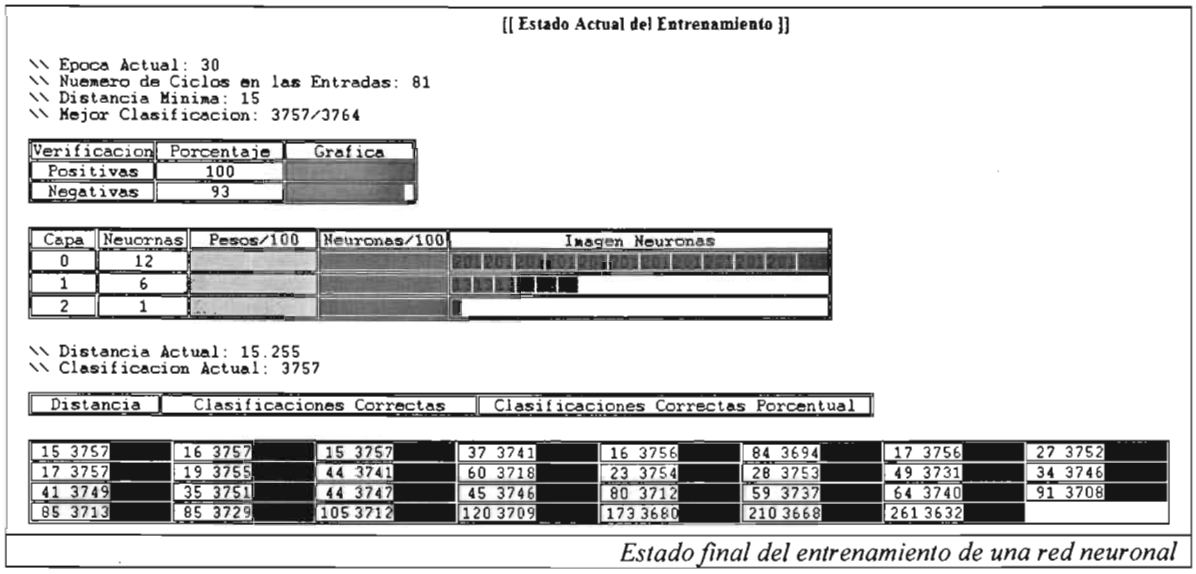
Mejor clasificación. La *mejor clasificación* es el mejor número de muestras que se clasificaron correctamente.

Tabla de verificación. Este indicador es bueno para saber si la red está bien entrenada, pues representa el porcentaje con que se clasificaron correctamente patrones que no fueron usados durante el entrenamiento.

Tabla de activación. Este indicador muestra cuántas neuronas o pesos se activaron durante el entrenamiento de la red neuronal; siempre deberá ser el total de neuronas y pesos, pues de lo contrario significaría que algunas neuronas o pesos están siendo desaprovechadas.

Tabla de épocas. Para cada época se detalla el valor de la distancia, el número de clasificaciones correctas y una gráfica con la clasificación porcentual. Las épocas

se presentan en orden inverso, es decir, la primera época en la tabla es la época que esta siendo procesada.



[[Estado Actual del Entrenamiento]]

\\ Epoca Actual: 30
\\ Nuemero de Ciclos en las Entradas: 81
\\ Distancia Minima: 15
\\ Mejor Clasificacion: 3757/3764

Verificacion	Porcentaje	Grafica
Positivas	100	
Negativas	93	

Capa	Neuornas	Pesos/100	Neuronas/100	Imagen Neuronas
0	12			
1	6			
2	1			

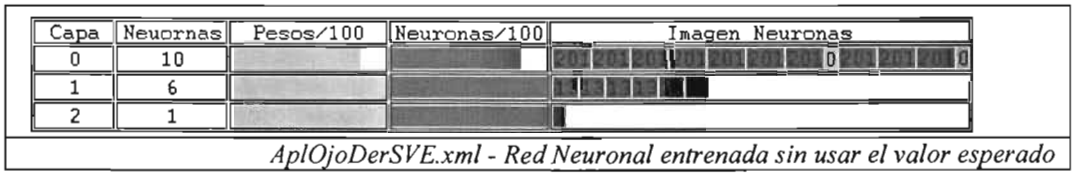
\\ Distancia Actual: 15.255
\\ Clasificacion Actual: 3757

Distancia	Clasificaciones Correctas	Clasificaciones Correctas Porcentual

15 3757	16 3757	15 3757	37 3741	16 3756	84 3694	17 3756	27 3752
17 3757	19 3755	44 3741	60 3718	23 3754	28 3753	49 3731	34 3746
41 3749	35 3751	44 3747	45 3746	80 3712	59 3737	64 3740	91 3708
85 3713	85 3729	105 3712	120 3709	173 3680	210 3668	261 3632	

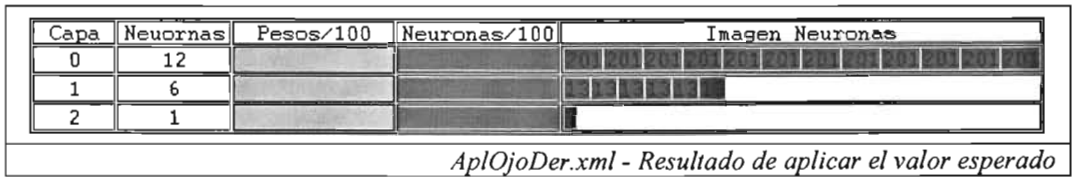
Estado final del entrenamiento de una red neuronal

Para iniciar el entrenamiento verificamos si ya se ha inicializado la red neuronal; de no ser así, se generan pesos aleatorios y se les asignan a las neuronas. Sin embargo, dejar el valor de la activación completamente aleatorio puede inutilizar neuronas por la gran diferencia entre la suma de sus pesos y la activación, así que usaremos un valor aleatorio cercano al valor esperado[11] de la activación de la neurona. En términos concretos, la red neuronal AplOjoDerSVE.xml fue entrenada sin usar el valor esperado, por lo que existe una pérdida de 2 neuronas que no fueron usadas durante el proceso de entrenamiento, mientras que AplOjoDer.xml fue entrenado usando el valor esperado y por esto todas las neuronas se utilizaron.



Capa	Neuornas	Pesos/100	Neuronas/100	Imagen Neuronas
0	10			
1	6			
2	1			

AplOjoDerSVE.xml - Red Neuronal entrenada sin usar el valor esperado



Capa	Neuornas	Pesos/100	Neuronas/100	Imagen Neuronas
0	12			
1	6			
2	1			

AplOjoDer.xml - Resultado de aplicar el valor esperado

Podemos aproximar al valor esperado de la neurona por,

$$\varepsilon(v_m) = \varepsilon\left(\sum_{i=0}^{K_{L-1}} w_{mi}^{L-1} y_i^{L-1}\right) = \sum_{i=0}^{K_{L-1}} w_{mi}^{L-1}\varepsilon\left(y_i^{L-1}\right) = \frac{1}{2}\sum_{i=0}^{K_{L-1}} w_{mi}^{L-1}$$

[11] Valor que se debe esperar obtener de una función ligada a un evento probabilístico

si suponemos que $y_i^{L-1} \in [0,1]$ está uniformemente distribuida.

```
/// <metodo nombre="random">
///   <descripcion>
///     Inicializa los pesos de la red con valores en [0,1)
///     y la activacion se aproxima al valor esperado de
///     la salida de la red para maximizar su utilizacion
///   </descripcion>
void random() {
 for(int i=0; i<L; i++)
  for(int j=0; j<k[i]; j++) {
   float fAct = 0.0;
   for(unsigned int m=1; m<red[i][j].w.size(); m++) {
    red[i][j].w[m] = (float)(rand()%100)/100;
    fAct += red[i][j].w[m];
   }
   if(xmlUsarValorEsperadoEnActivacion == true)
    red[i][j].w[0] =  -fAct/2+(float)(rand()%100)/1000;
   else
    red[i][j].w[0] =  (float)(rand()%100)/100;
  }
}
/// </metodo nombre="random">
```

Implementación de la inicialización de la red

El proceso de entrenamiento de la red neuronal se realiza presentándole a la red 10,000 patrones consecutivos aleatorios, y por defecto se entrenan 50 épocas. Sin embargo, al monitorear el estado del archivo de entrenamiento detendremos el proceso si consideramos que la red ya ha sido entrenada exitosamente. El proceso se detuvo cuando el entrenamiento de las imágenes de verificación era mayor al 90%, sin embargo no siempre se alcanzó esta meta.

```
/// <metodo nombre="entrenaxml">
///   <descripcion>
///     Entrena la red usando el archivo de entrenamiento XML
///     generado por el programa de clasificacion, el entrenamiento
///     se realiza por epocas y las muestras se toman
///     aleatoriamente del archivo de entrenamiento
///   </descripcion>
void entrenaXML(char *xmlRed, char * xmlEntrada, char * xmlDebug) {
  unsigned long nCaso=0, totCasos=entradas.size(), nCiclosEntradas=0;
  for(int nEpoca=0; nEpoca<xmlent.totEpocas; nEpoca++) {
    // Entranar Epoca
    for(long iter=0; iter<10000; iter++, nCaso++, contCien++) {
     int rndNum = rand()%totCasos;
     while(xmlent.entrenamiento[rndNum].bVerificacion == true)
      rndNum = rand()%totCasos;
     entrena(xmlent.entradas[rndNum],xmlent.salidas[rndNum]);
    }
  }
}
/// </metodo nombre="entrenaxml">
```

Código resumido del ciclo de entrenamiento

Resultados

Todos los patrones se entrenaron hasta obtener clasificaciones efectivas mayores al 97%, y para los patrones de verificación se obtuvo un rendimiento promedio (muestras positivas y negativas) de al menos el 75%.

Resultados del Entrenamiento						
Expresión	Región	Épocas	Ciclos	Verificación Porcentual		Clasificación Porcentual
				Positivas	Negativas	
Atención	Boca	7	30	90	100	99.6
	Ojo derecho	30	81	100	93	99.8
	Ojo izquierdo	7	23	90	92	98.9
Distracción	Ojo derecho	16	32	100	100	99.6
	Ojo izquierdo	2	4	75	89	97.1
Duda	Boca	1	5	48	99	92.2
Pensando	Ojo derecho	2	10	100	100	98.5
	Ojo izquierdo	2	10	100	100	98.5
Sorpresa	Boca	5	15	100	100	99.4
Media		8.00	23.33	89.22	97.00	98.17

Resultados del entrenamiento

El archivo de entrenamiento después de la clasificación muestra en color blanco los casos que se entrenaron exitosamente y de color negro aquellos que no se lograron entrenar. Estos representan el valor de clasificación porcentual de la tabla superior.

0.999631	0.999611	0.999402	0.999184
0.999273	0.999125	0.999228	0.999235
6.45524e-005	0.00304241	0.000109077	0.000143822
0.998995	0.999149	0.997995	0.999335
0.961642	0.98919	0.998307	0.994999
0.998163	0.999311	0.999175	0.999051
0.998186	0.998966	0.999195	0.999396
0.995423	0.998598	0.998904	0.937171
0.998405	6.53915e-005	0.000194031	0.998366
0.993414	0.999211	0.999578	0.947405
0.99876	0.000226318	6.55089e-005	6.74023e-005
0.00301945	0.999528	0.999304	0.996657
0.999468	0.999165	0.99917	0.999011
0.999421	0.999519	0.9993	0.999227
0.999067	0.999003	0.999041	0.998288
0.00317417	0.000141028	0.999341	0.999446
0.999617	0.999634	0.999515	0.999337
0.998927	0.000126135	6.69295e-005	0.998301
0.982969	0.999386	0.999364	0.999132
0.999588	0.999204	0.999192	0.999085
0.999513	0.99946	0.999363	0.999631
0.999604	7.28638e-005	0.000269339	0.981422
0.726255	0.998215	0.996136	0.966715
0.998349	0.986524	0.000863302	6.44581e-005

Región entrenada con casos positivos

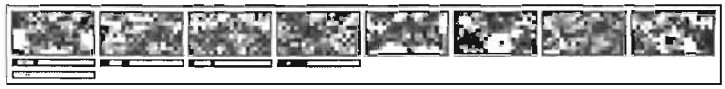
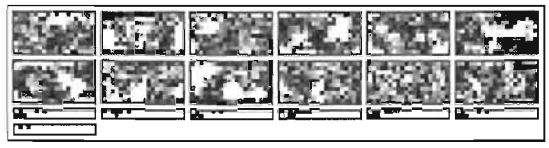
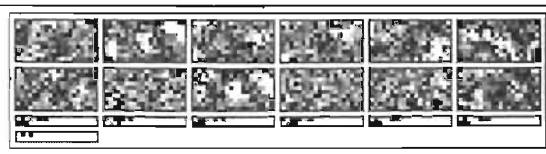
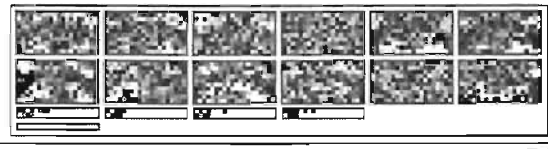
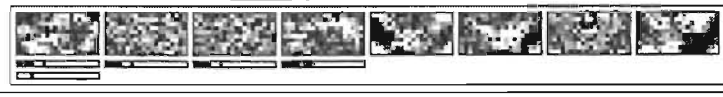
9.1801e-005	9.8425e-005	7.04756e-005	0.000142966
9.97501e-005	7.6986e-005	7.39721e-005	7.01502e-005
7.65185e-005	7.17026e-005	0.000161704	0.00020052
0.000101578	0.000112483	0.000174174	9.80286e-005
6.49953e-005	7.19562e-005	0.00014063	7.49064e-005
7.11748e-005	0.000118793	6.30487e-005	6.7203e-005
6.63231e-005	6.70078e-005	6.7034e-005	0.000148107
0.00363242	0.0231548	0.000158527	6.70104e-005
7.23347e-005	6.73665e-005	0.00187596	0.000468566
8.10551e-005	7.52542e-005	0.000126878	0.00013321
0.000349181	7.1419e-005	0.0127141	0.00233755
8.76566e-005	8.75651e-005	9.09262e-005	0.000486161
0.000133001	0.00057535	0.00168577	0.000181781
6.70629e-005	0.000921348	0.000431423	0.000192142
0.000259323	0.000610425	0.00058239	0.00227917
8.85404e-005	7.67898e-005	8.84491e-005	0.000126796
6.28246e-005	0.00202732	6.49735e-005	0.000782962
8.08155e-005	0.000303738	0.000218801	7.64271e-005
9.11283e-005	7.08121e-005	0.000125632	6.76412e-005
6.33116e-005	6.70104e-005	6.74487e-005	6.71184e-005
7.08253e-005	6.28401e-005	6.23592e-005	6.51058e-005
6.82405e-005	0.00011415	8.51346e-005	0.000106105
7.59903e-005	6.96035e-005	0.000413132	9.81898e-005
0.00081088	6.79817e-005	8.13032e-005	9.14982e-005

Región entrenada con casos negativos

0.621493	0.539175	0.322369	0.547	
0.282769	0.301594	0.412369	0.550887	0.471717
0.294445	0.602557	0.563079	0.534005	0.407804
0.680013	0.822773	0.810185	0.746917	0.741916
0.889918	0.760497	0.830528	0.782436	0.778925
0.787836	0.739338	0.626281	0.519886	0.632379
0.662882	0.579893	0.608616	0.625687	0.47793
0.333338	0.426691	0.159908	0.303382	0.25322
0.173541	0.0507328	0.119518	0.0700274	0.0463292
0.0727441	0.0630372	0.015139	0.0223872	0.5870
0.0482957	0.0300899	0.0151149	0.0386053	0.0284759
0.0207905	0.0132864	0.0120602	0.01066	0.0131308
0.0100692	0.00951929	0.0691082	0.0405329	0.0167434
0.0139527	0.0160228	0.0139969	0.0118984	0.0101287
0.0231506	0.010649	0.00979971	0.030556	0.0187019
0.0113734	0.0128733	0.0255268	0.126853	0.0185152
0.021675	0.0225461	0.0311197	0.0128155	0.0152857
0.0269448	0.00945095	0.0836737	0.0123401	0.126454
0.0101448	0.0210169	0.0134663	0.0159254	0.00945108
0.0102377	0.01037	0.0161874	0.024363	0.018499
0.0125063	0.0162129	0.0137434	0.0116209	0.0134636
0.0222446	0.0219183	0.0272989	0.0144964	0.00997031
0.0109133	0.0214605	0.00989333	0.0100565	0.0267134
0.0126137	0.0138131	0.0435744	0.00980261	0.0441114
0.124371	0.00980809	0.0323076	0.00948172	0.00963252
0.0097368	0.0102888	0.0479903	0.0142972	0.0113038
0.0107554	0.0195413	0.01452	0.0720663	0.0184327
0.0116121	0.166773	0.022125	0.0180349	0.0150049
0.0144793	0.0172622	0.0132823	0.0100064	0.0190182
0.142539	0.0158999	0.0207187	0.0128219	0.159712
0.0132843	0.0205541	0.0337658	0.0110608	0.0170402
0.0191058				

Algunos patrones de verificación

La siguiente tabla muestra los pesos ponderados en cada capa de la red neuronal; algunas capas fueron reacomodadas para mejorar la presentación.

Principales Redes Neuronales Entrenadas		
Atención	Boca	
	Ojo derecho	
	Ojo izquierdo	
Duda	Boca	
Sorpresa	Boca	

Principales redes neuronales entrenadas

[[Estado Actual de la Red Neuronal]]

```
\\ Descripcion: Detecta bocas expresando duda
\\ Autor: Javier Porras
\\ Fecha: Marzo 14 del 2005
```

```
-47.0753
0.610338
 0.72203
0.378444
-0.020616
0.675586
0.261794
0.852208
0.711912
0.834068
0.806352
0.128528
0.579483
 1.16667
0.725821
0.933986
 1.07596
0.986475
0.421716
0.265453
```

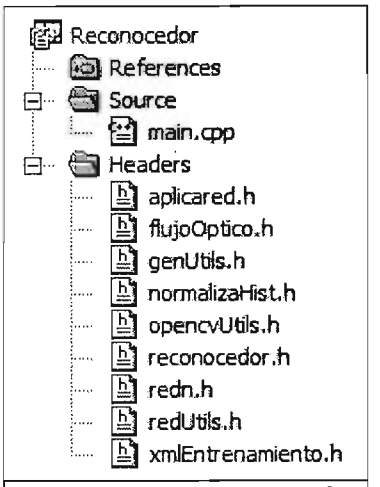
```
Muestras: 20000
Clasificaciones Correctas: 3221
Clasificaciones Incorrectas: 425
Uso Total: 14274
```

Muestra de red neuronal entrenada

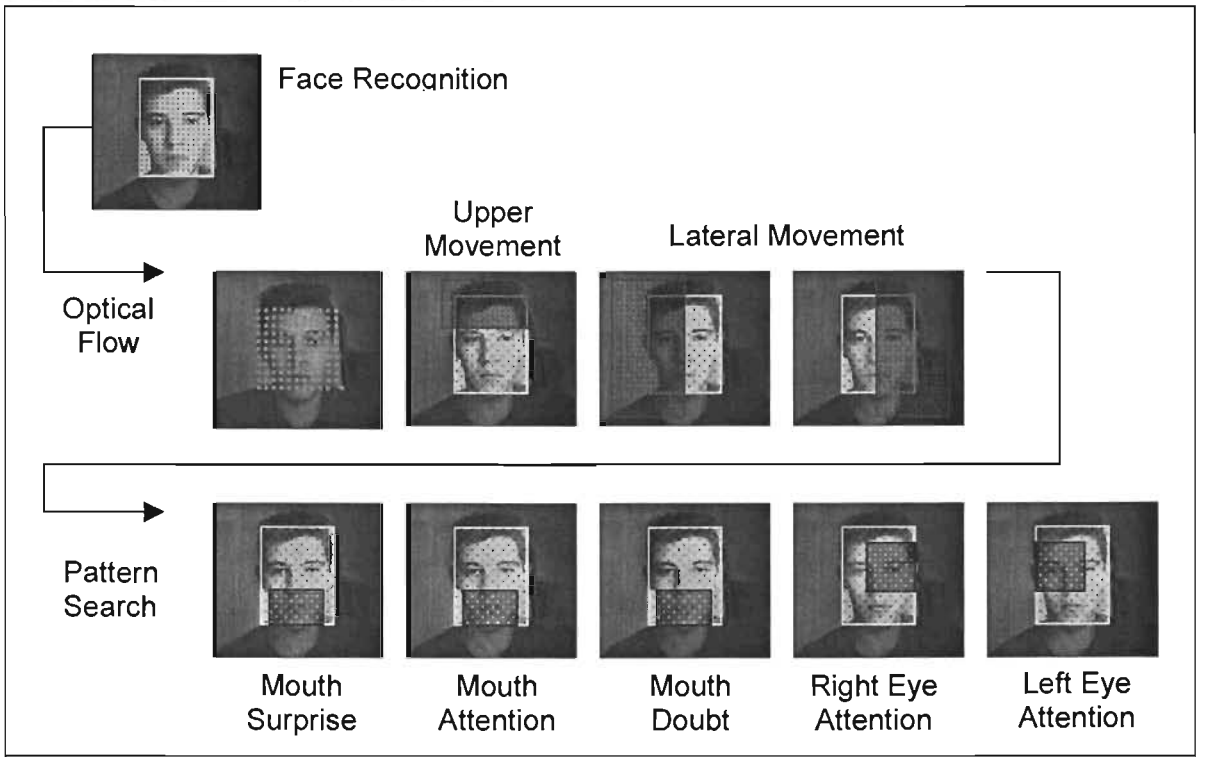
Implementación

El reconocedor utiliza principalmente los módulos de reconocimiento de patrones por redes neuronales y flujo óptico, combinando la forma en la que se detectan expresiones usando un archivo XML de entrada.

El sistema intentará reconocer características de la expresión facial; al encontrar alguna característica se suspenderá el proceso de búsqueda y se asimilará la expresión facial de la característica que se encontró. El sistema primero reconoce el rostro y sobre él aplica continuamente el flujo óptico para encontrar movimientos de cabeza; si no existieran, buscará cada patrón en la región donde es probable encontrarles.



Estructura Reconocedor



Esquema general del algoritmo de búsqueda de expresiones

```
<aplicacion>
  <definicion>
    <deffuente tipo="camara" archivo="..\..\Data\xmlData\frontalface.xml"
               normalizahistogramaglobal="no"
               normalizahistogramalocal="si"/>
    <defdebug archivo="edorec.xml" guardarsubimgs="no"
              dirsalida="..\..\Data\imgReconocimiento\"/>
    <defmovimientos segundos="1">

    </defmovimientos>
    <defpatrones total="5">
      // Definición de Patrones
    </defpatrones>
  </definicion>
</aplicacion>
```

Archivo de entrada para el reconocedor

El esquema de movimiento de la cara para las expresiones de distracción y pensamiento se obtuvo analizando los resultados del módulo de *examen facial*, los rangos de movimiento para cada expresión se establecieron de manera empírica, y se muestran a continuación.

Regiones de Movimiento				
Expresión	Desplazamiento		Desplazamiento	
	Mínimo X	Máximo X	Mínimo Y	Máximo Y
Distracción	10	40	-3	3
Distracción	-40	-10	-3	3
Pensando	-5	5	9	30
Pensando	-5	5	-30	-9

Regiones de movimiento en el reconocedor

```
<defmovimiento expresion="distraccion"
               minMovX="10" maxMovX="40" minMovY="-3" maxMovY="3"/>
<defmovimiento expresion="distraccion"
               minMovX="-40" maxMovX="-10" minMovY="-3" maxMovY="3"/>
<defmovimiento expresion="pensando"
               minMovX="-5" maxMovX="5" minMovY="9" maxMovY="30"/>
<defmovimiento expresion="pensando"
               minMovX="-5" maxMovX="5" minMovY="-30" maxMovY="-9"/>
```

Definición XML de las regiones de movimiento en el reconocedor

```
<defpatron numero="1" archivo="..\..\Data\xmlData\redes\SorBoca.xml"
           patronwidth="20" patronheight="10" patronstep="2"
           escalastep="0.03" minestimulovalido="0.90" expresion="sorpresa">
  // Definición de la Region de Estadisticas
</defpatron>
<defpatron numero="4" archivo="..\..\Data\xmlData\redes\AtnBoca.xml"
           patronwidth="20" patronheight="10" patronstep="2"
           escalastep="0.03" minestimulovalido="0.90" expresion="atencion">
</defpatron>
<defpatron numero="0" archivo="..\..\Data\xmlData\redes\DudaBoca.xml"
           patronwidth="20" patronheight="10" patronstep="2"
           escalastep="0.03" minestimulovalido="0.90" expresion="Duda">
```

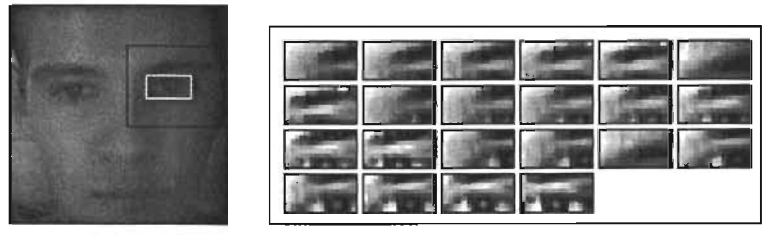
```
</defpatron>
<defpatron numero="2" archivo="..\..\Data\xmlData\redes\AtnOjoDer.xml"
           patronwidth="20" patronheight="10" patronstep="2"
           escalastep="0.03" minestimulovalido="0.90" expresion="atencion">
</defpatron>
<defpatron numero="3" archivo="..\..\Data\xmlData\redes\AtnOjoIzq.xml"
           patronwidth="20" patronheight="10" patronstep="2"
           escalastep="0.03" minestimulovalido="0.90" expresion="atencion">
</defpatron>
```

Definición de la búsqueda de patrones

La región de estadísticas es generada por el programa de clasificación y representa el área donde puede ser encontrado cada patrón.

```
<statregion minx="0.277778" maxx="0.692308" miny="0.659722"
            maxy="0.992064" minescala="0.24359" maxescala="0.348624"
            promx="0.341211" promy="0.756475" promescala="0.291937"/>
```

Tabla de estadísticas en el archivo del reconocedor



Búsqueda en la región donde es probable encontrar al patrón

La búsqueda del patrón sobre una región se realiza de forma exhaustiva iniciando en la parte superior izquierda y terminando en la inferior derecha; por cada movimiento de la subregión de búsqueda se alimentan los píxeles que contiene a la red neuronal y si la red acepta el patrón, éste se dibuja sobre la imagen y se informa al evento de la clase que maneja el reconocimiento.

```
/// <bloque>
///   <descripcion>Busca un patron usando la calse AplRed</descripcion>
do {
 if(ared->aplicaPatron() == true) {
  CvRect rPat = ared->getRectPatron();
  dibujaRectangulo(subicolor,rPat,CV_RGB(255,255,0));
  detectaExpresion((char *) ared->getExpresion().c_str());
  break;
 }
} while(ared->siguientePatron());
/// </bloque>
```

Ciclo de búsqueda de patrones

La búsqueda se realiza internamente primero por escala y después por traslación,

```
/// <while condicion="escalaPatron()">
while(escalaPatron() == true) {
 patimg = cvCreateImageHeader(cvSize(r->width,r->height), 8, 1 );
 inicializaTraslado();
 int iVentana = 0;
 /// <while condicion="transladaPatron()">
 while(trasladaPatron() == true) {
```

Estructura interna de la búsqueda de patrones

La retícula del flujo óptico se reinicia cada segundo para aprovechar que siempre se está detectando la nueva posición del rostro.

```
/// <metodo nombre="procesaReinicio">
///   <descripcion>
///     Reinicia la reticula del flujo optico cada segundo
///   </descripcion>
void procesaReinicio() {
 int segundos = 0;
 while(bThreadReinicio == true) {
   _sleep(1000);
   segundos++;
   flOptico->getMovimiento();
   if(segundos == ared->getMovSegundos()) {
     segundos = 0;
     flOptico->reiniciaReticula();
   }
 }
}
/// <metodo nombre="procesaReinicio">
```

Estructura del reinicio del flujo óptico

Flujo óptico

OpenCV implementa 4 métodos para calcular el flujo óptico:

Funciones flujo óptico en OpenCV		
Identificador	Función	Método
HS	CalcOpticalFlowHS	Horn y Schunck
LK	CalcOpticalFlowLK	Lucas y Kanade
BM	CalcOpticalFlowBM	Basado en regiones
PyrLK	CalcOpticalFlowPyrLK	Lucas y Kanade usando pirámides

Funciones flujo óptico en OpenCV

Cada método utiliza distintos parámetros que nos servirán para determinar el mejor método para nuestra aplicación.

Función del flujo óptico por método de Horn y Schunck	
`void cvCalcOpticalFlowHS( const CvArr* prev, const CvArr* curr, int use_previous, CvArr* velx, CvArr* vely, double lambda, CvTermCriteria criteria );`	
Parámetro	**Descripción**
prev	Primera imagen, 8-bit, canal único
curr	Segunda imagen, 8-bit, canal único
use_previous	Usar el campo de velocidades previo
velx	Componente horizontal del flujo óptico
vely	Componente vertical del flujo óptico
lambda	Multiplicador de Lagrange
criteria	Criterio para terminar con el cálculo de las velocidades
Función del flujo óptico por método de Horn y Schunck	

Función del flujo óptico por método de Lucas y Kanade	
`void cvCalcOpticalFlowLK( const CvArr* prev, const CvArr* curr, CvSize win_size, CvArr* velx, CvArr* vely );`	
Parámetro	**Descripción**
prev	Primera imagen, 8-bit, canal único
curr	Segunda imagen, 8-bit, canal único
win_size	Tamaño de la ventana de ponderación usada para agrupar pixeles
velx	Componente horizontal del flujo óptico
vely	Componente vertical del flujo óptico
Función del flujo óptico por método de Lucas y Kanade	

Función del flujo óptico por método basado en regiones	
`void cvCalcOpticalFlowBM( const CvArr* prev, const CvArr* curr, CvSize block_size, CvSize shift_size, CvSize max_range, int use_previous, CvArr* velx, CvArr* vely );`	
Parámetro	**Descripción**
prev	Primera imagen, 8-bit, canal único
curr	Segunda imagen, 8-bit, canal único
block_size	Tamaño del bloque básico que es comparado
shift_size	Incrementos en las coordenadas del bloque
max_range	Tamaño de la vecindad de los píxeles analizados alrededor del bloque
use_previous	Usar el campo de velocidades previo
velx	Componente horizontal del flujo óptico
vely	Componente vertical del flujo óptico
Función del flujo óptico por método basado en regiones	

Función del flujo óptico por método de Lucas y Kanade usando pirámides

```
void cvCalcOpticalFlowPyrLK( const CvArr* prev,
                             const CvArr* curr, CvArr* prev_pyr,
                             CvArr* curr_pyr,
                             const CvPoint2D32f* prev_features,
                             CvPoint2D32f* curr_features,
                             int count, CvSize win_size,
                             int level, char* status,
                             float* track_error,
                             CvTermCriteria criteria,
                             int flags );
```

Parámetro	Descripción
prev	Primera imagen, 8-bit, canal único
curr	Segunda imagen, 8-bit, canal único
prev_pyr	Buffer de la pirámide del primer frame
curr_pyr	Buffer de la pirámide del segundo frame
prev_features	Arreglo de puntos que se les calculará la velocidad
curr_features	Arreglo de puntos que contiene las posiciones calculadas de los puntos
features	Número de puntos en la segunda imagen
count	Número de puntos originales
win_size	Tamaño de la ventana para cada nivel de la pirámide
level	Máximo nivel en la pirámide
status	Arreglo asociado a los puntos que asigna 1 si fue encontrada su velocidad
error	Arreglo de diferencias entre las regiones de los puntos originales, puede ser nulo
criteria	Define cuando se detendrá la búsqueda para cada punto en cada nivel de la pirámide
flags	Banderas adicionales

Función del flujo óptico por método de Lucas y Kanade usando pirámides

Para nuestra aplicación, buscaremos que el método que implemente el flujo óptico tenga las siguientes características:

- Deberá funcionar en tiempo real.
- Debe omitir cualquier tipo de calibración, pues el sistema de reconocimiento facial debe ser autónomo y carecer de configuración alguna.
- Los movimientos de cabeza pueden ser rápidos, por lo que debemos detectar cambios abruptos en el movimiento de los objetos.
- De ser posible, deberá calcular el flujo óptico solo sobre la región de la cara, esta región la conocemos por el reconocimiento que realizó la red neuronal.

La siguiente tabla relaciona las principales características de cada uno de los métodos que implementa OpenCV, las funciones que nos interesan están marcadas en color claro, al igual que las características que cumple cada algoritmo.

Tabla comparativa de las funciones de OpenCV				
Funciones	HS	LK	BM	PyrLK
Es fácil su implementación	1/2	1/2	Si	No
Calcula eficientemente el flujo óptico denso	Si	Si	Si	No
Utiliza el flujo óptico anterior	Si	No	Si	No
Funciona en tiempo real	Si	Si	Si	Si
Necesita calibración	Si	No	No	No
Detecta movimientos abruptos	No	No	No	Si
Puede definirse una región de puntos	No	No	No	Si

Tabla comparativa de las funciones de OpenCV

Como vemos, el algoritmo de *PyrLK* - Lucas y Kanade usando pirámides, representa la mejor opción para nuestra aplicación. Su implementación se describe en la siguiente tabla:

```
cvCalcOpticalFlowPyrLK( prev_grey, grey, prev_pyramid, pyramid,
 points[0], points[1], count, cvSize(win_size,win_size), 3, status, 0,
 cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), flags );
```

Utilización de OpenCV para calcular el flujo óptico

La clase del reconocedor

Finalmente toda la implementación del reconocimiento se engloba en la clase Reconocedor. Se pretende que las aplicaciones que usen el reconocedor de expresiones faciales deriven esta clase e implementen los métodos:

detectaExpresion. Llamado cada vez que una expresión es reconocida.

procesoGeneral. Hilo paralelo a la detección, debe llamar necesariamente a terminar() al finalizar.

```
protected:
  virtual void detectaExpresion(char * expresion) {
    cout << "Expresion Facial: " << expresion << endl;
  }
  virtual void procesoGeneral() {
    getch(); terminar();
  }
```

Métodos usados al derivar la clase padre

La implementación es muy clara pues se encapsulan satisfactoriamente los procesos de reconocimiento. Para implementar la clase debemos:

1. llamar al constructor padre con la ruta al archivo de configuración XML,
2. implementar los métodos virtuales, y

3. escribir nuestra aplicación en el hilo del proceso general sin olvidar llamar a *terminar()*, pues éste se encarga de destruir el hilo y liberar los recursos del reconocedor.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

#include "reconocedor.h"

class miReconocedor : public Reconocedor {
 void detectaExpresion(char *exp) {
  cout << "Expresion: " << exp << endl;
 }

 void procesoGeneral() {
  cout << "[Inicio]" << endl;
  getch();
  cout << "[Fin]" << endl;
  terminar();
 }
public:
 miReconocedor(char * xmlFile) : Reconocedor(xmlFile) {}
};

int main(int argc, char** argv )
{
 if(argc != 3) {
  printf("[Parametros de Entrada]\n");
  printf("  -aplicacionred Archivo de configuracion XML\n");
  return -1;
 }
 miReconocedor miRec(argv[2]);
 miRec.setMostrarVentanas(true);
 miRec.iniciar();
 return 0;
}
```

Implementación del reconocedor

Resultados

El reconocedor discrimina satisfactoriamente las AUs que se propusieron. Sin embargo existen variaciones bajo distintas iluminaciones y situaciones de ruido.

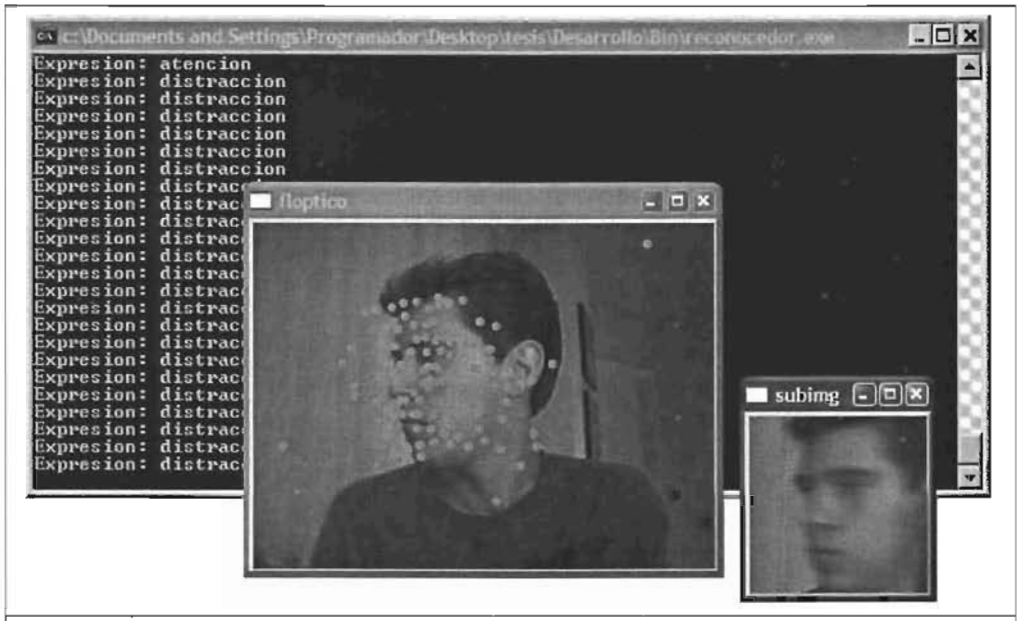
Los patrones para distracción y pensamiento en ojos, aunque fueron entrenados, generaron gran inestabilidad por su parecido con los patrones de atención y por la baja resolución en las muestras y el video. Por esto se optó por usar el flujo óptico con movimientos de cabeza que son descritos también en las especificaciones de las AUs.

Reconocimiento por AUs			
Expresión	Región	AU	Descripción
Atención	Boca	8	Juntar Labios
	Ojo Derecho	42	Abrir Ojos
	Ojo Izquierdo	42	Abrir Ojos
Distracción	Cabeza	51	Cabeza a la Izquierda
		52	Cabeza a la Derecha
Duda	Boca	20	Estirar Labios
		15	Hundir Esquina Labial
Sorpresa	Boca	27	Abrir Boca
Pensando	Cabeza	53	Cabeza Arriba

Reconocimiento por AUs

Expresión de atención reconocida

Expresión de sorpresa reconocida



Expresión de distracción reconocida

Expresión de duda reconocida

Expresión de pensamiento reconocida

Capitulo V – Conclusiones

Conclusiones

"Una máquina puede hacer el trabajo de 50 hombres corrientes. Pero no existe ninguna máquina que pueda hacer el trabajo de un hombre extraordinario."

Elbert Green Hubbard

El reconocimiento de expresiones faciales representa un reto latente con grandes implicaciones para los sistemas e interfaces que estamos acostumbrados a usar hoy en día. Existen bastantes desarrollos (principalmente en el área de laboratorios virtuales) que requieren urgentemente mejorar las interfaces de interacción con las computadoras, lo que ofrece un ambiente favorable para desarrollar interfaces orientadas al reconocimiento de expresiones.

La estrategia que se usó para este proyecto tuvo buenos resultados. Sin embargo es fácil confundir al sistema si el usuario detecta que hay una cámara oculta. Se probó que las redes neuronales pueden ser lo suficientemente efectivas como para buscar los patrones que constituyen a una expresión facial en tiempo real; no obstante se requiere de mayor resolución en las cámaras de video convencionales por lo que el uso de interfaces de reconocimiento facial para el uso masivo deberá esperar hasta que los dispositivos de captura de video mejoren. Ahora bien, existen dispositivos de captura actuales con mayores resoluciones que sin duda pueden ser usados para el reconocimiento facial en ambientes controlados, como es el caso de laboratorios virtuales. Bastaría entrenar la red con patrones de mayor resolución manteniendo la arquitectura e implementación propuestas en este desarrollo.

El uso del flujo óptico es bueno y preciso para analizar movimientos rígidos como lo fue el movimiento de la cara. Sin embargo, existen problemas para aplicarlo directamente sobre el rostro pues primero debemos normalizarlo y mantener un seguimiento minucioso de los movimientos, y en tiempos extendidos de monitoreo se pueden desacomodar o incluso perder los patrones que originalmente fueron detectados.

La arquitectura que se planteó brindó la escalabilidad deseada hasta finalizar el proyecto e incluso continuarlo. La mayor parte de las expresiones se entrenaron gradualmente hasta alcanzar los resultados deseados; sin embargo, la expresión de duda fue difícil de detectar pues existen varios patrones que presenta la gente como lo es el fruncimiento de las cejas, cambio de vista, y distintos tipos de muecas.

El reconocimiento de expresiones faciales es un problema difícil para todo sistema automatizado que puede ser solucionado satisfactoriamente conjugando métodos de procesamiento de video como lo es el flujo óptico y redes neuronales.

Trabajo futuro

Algunas mejoras y desarrollos futuros que se contemplan son,

Patrones espaciotemporales. Existen AUs que sólo pueden detectarse si comparamos distintas imágenes en el tiempo; los patrones podrían definirse tanto en una región en el espacio para encontrar características como cejas, ojos, etc; y sobre un lapso del tiempo para completar el patrón con el movimiento que se realizó; de esta forma tendríamos patrones como boca-cerrada a boca-abierta, ojo-abierto a ojo-cerrado, etc.

Expresión dominante. Crear métodos en la clase del reconocedor para generar información global de la expresión facial pues la implementación actual sólo avisa en el instante en que se detecta la expresión; sin embargo, para otras aplicaciones puede ser útil saber cuál fue la expresión dominante en un intervalo de tiempo más que de forma instantánea.

Agregar expresiones. Agregar otras expresiones faciales como disgusto, frustración, ansiedad, risa, etc.

Examen y análisis facial. Se podrían crear otros exámenes faciales probando su efectividad. Además sería conveniente agregar al examen funcionalidad para capturar el flujo óptico de la misma forma en la que se capturan los rostros.

Interfaz. Mejorar la interfaz gráfica de todas las aplicaciones, pues esto permitiría capturar, clasificar un mayor número de expresiones faciales y presentar más amigablemente los resultados.

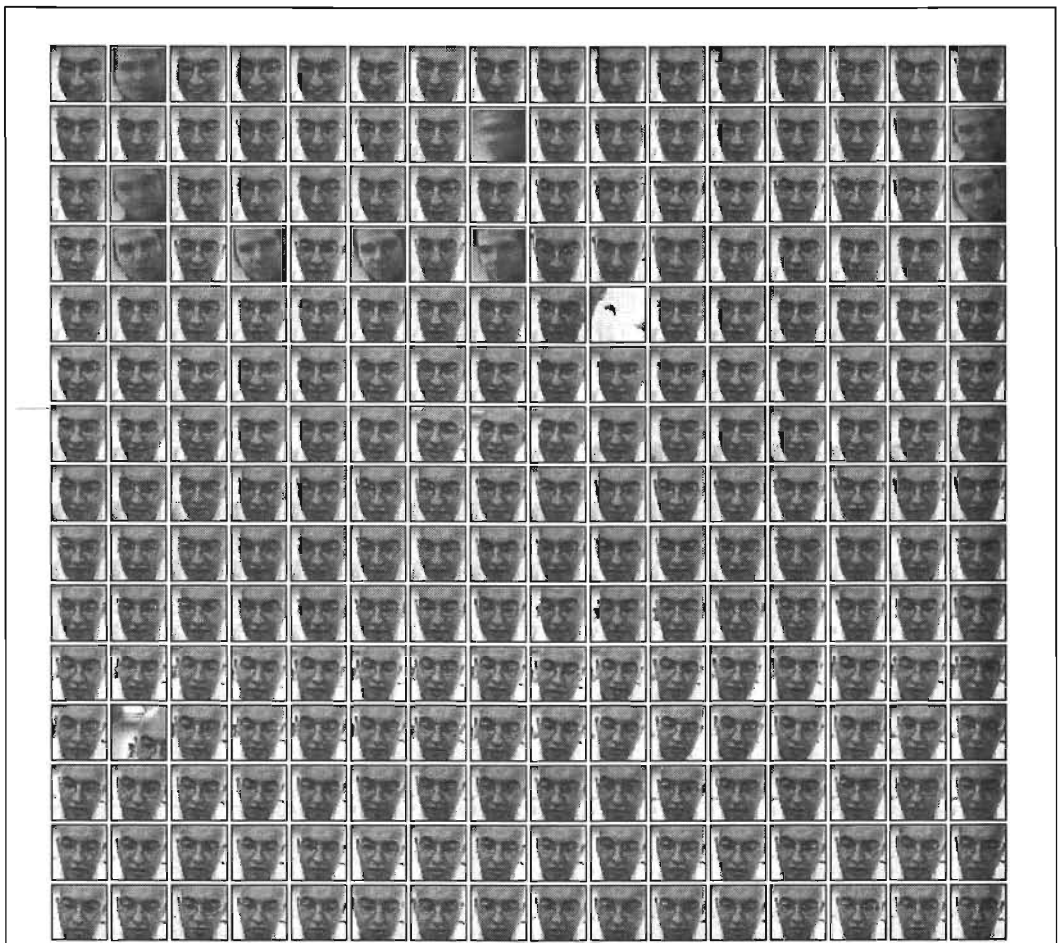
Mejorar el algoritmo de búsqueda. El patrón inicia a buscar sobre la esquina superior izquierda en la región donde puede encontrarse; sin embargo, sería más conveniente iniciar la búsqueda desde la posición donde es más probable encontrarlo, que generalmente es cercana al centro de la ventana de búsqueda.

Mayor resolución. Entrenar la red neuronal con muestras de mayor resolución.

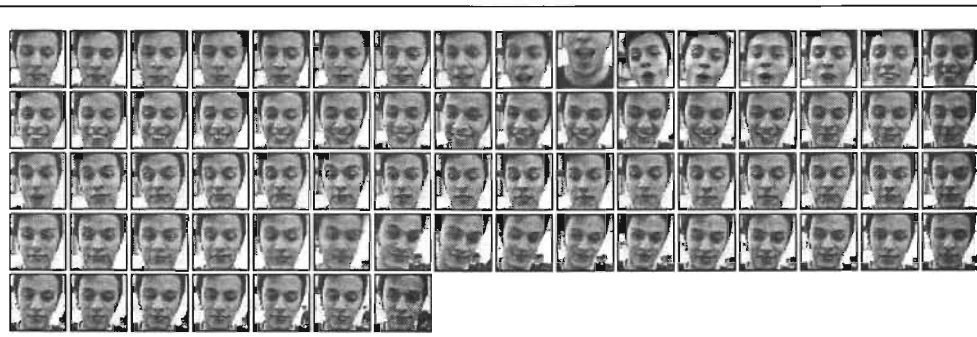
Nuevas muestras. Generar del examen facial mayores muestras.

ANEXO I – Secuencias de Imágenes

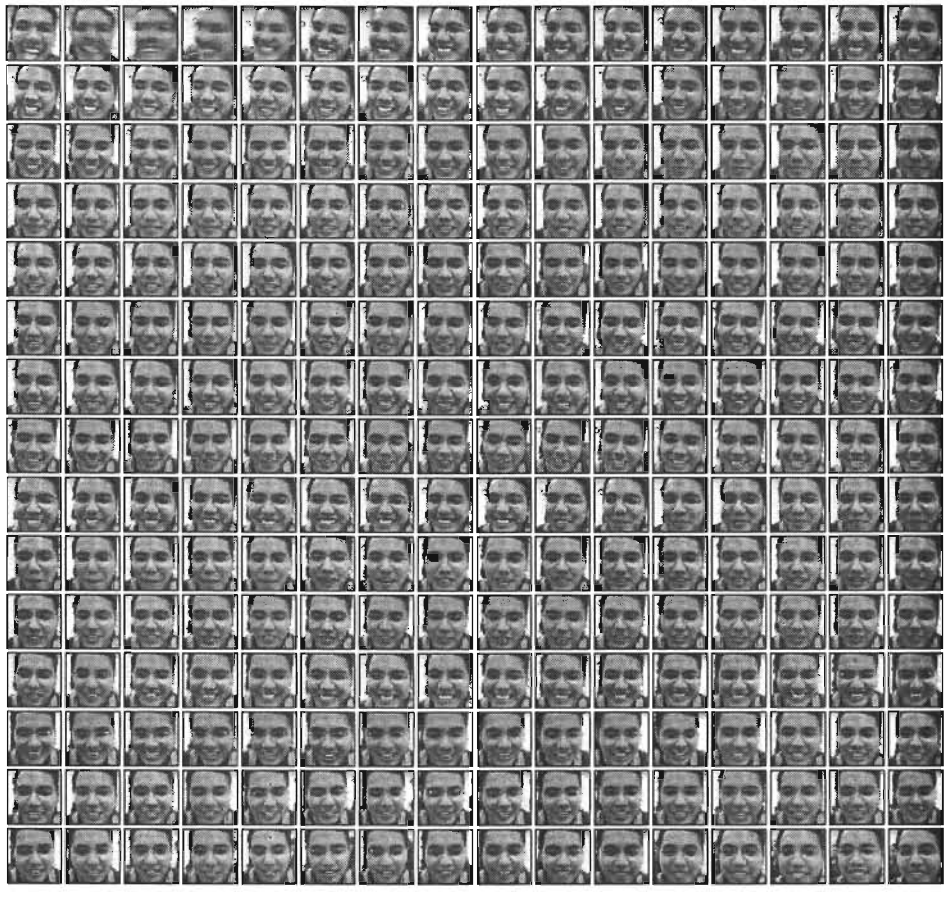
Examen Facial



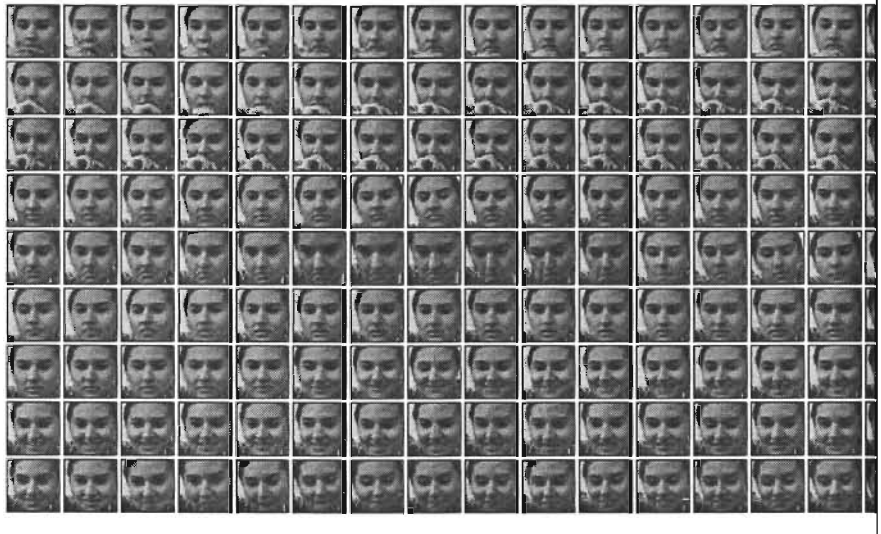
Examen Facial en categoría de Interés



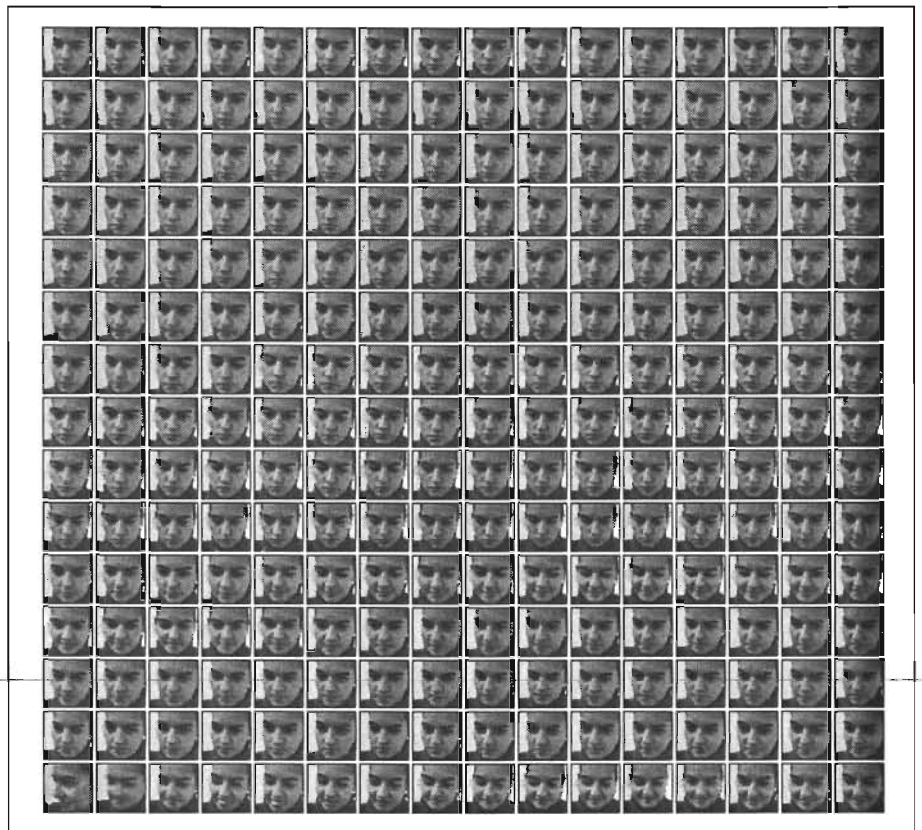
Examen Facial en categoría de Sorpresa



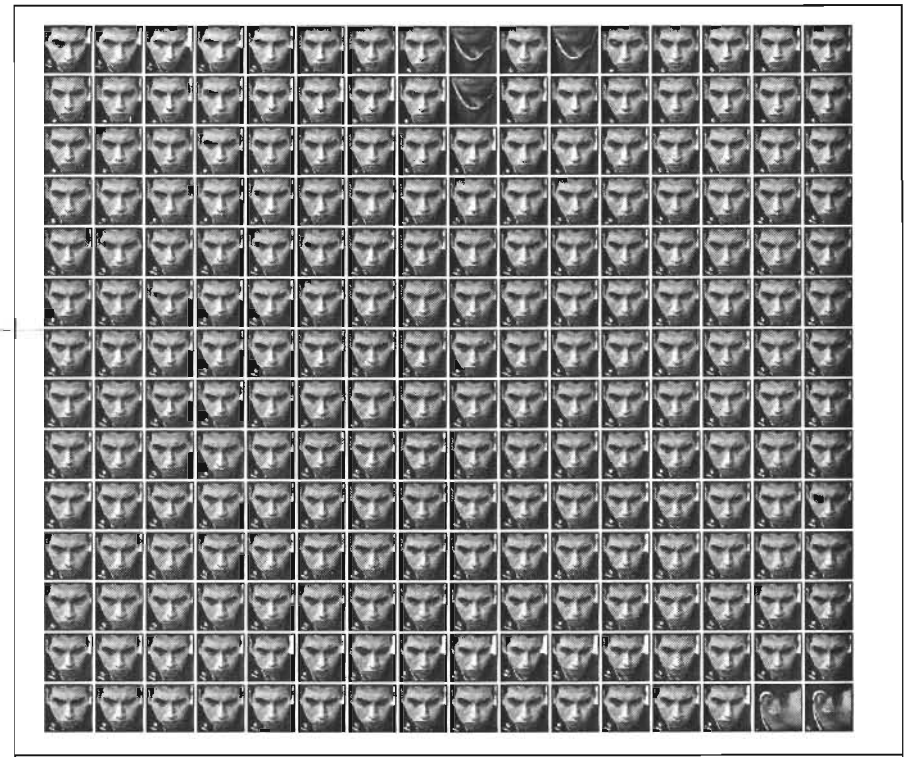
Examen Facial en categoría de Interés



Examen Facial en categoría de Duda



Examen Facial en categoría de Interés



Examen Facial en categoría de Duda

Referencias

[AJI91] Ajit Singh, **Optic Flow Computation: A Unified Perspective**, IEEE Computer Society Press, 1991

[AZI95] Irfan Aziz Essa, **Analysis, Interpretation and Synthesis of Facial Expressions,** MIT Media Laboratory, 1995

[BEA95] Beauchemin S.S y Barron J. L. **The Computation of Optical Flow**, University of Wester Ontario, 1995

[BER92] Bernd Jähne, **Digital Image Processing. Concepts, Algorithms, ans Scientific Applications**, Springer, 1992

[CAR90] Carme Torras, **Computer Vision: Theory and Industrial Applications**, Springer-Verlag, 1990

[CHA01] N. P. Chandrasiri, Takeshi Naemura, and Hiroshi Harashima, **Real Time Facial Expression Recognition System with Applications to Facial Animation in MPEG-4**. IEICE Trans. Inf. & Syst., Vol. E84-D, No. 8, Aug. 2001

[EKM78] Paul Ekman. **Facial signs: Facts, fantasies and possibilities**. In T. Sebeok, editor, Sight, Sound and Sense. Indiana University Press, 1978.

[EKM02] Paul Ekman, Wallace V. Friesen, and Joseph C. Hager. **The new (2002) Facial Action Coding System (FACS).**
http://face-and-emotion.com/dataface/facs/manual/TOC.html

[FON01] Luciano da Fontoura, Roberto Marcondes, **Shape Analysis and Classification**, CRC Press, 2001

[MAL94] Cardiff University - David Marshall, **Vision Systems,**
http://www.cs.cf.ac.uk/Dave/Vision_lecture/node45.html

[MAR02] Bonifacio Martín del Brío, Alfredo Sanz Molina, **Redes Neuronales y Sistemas Difusos,** Alfaomega, 2002

[MAS91] Kenji Mase. Recognition of facial expressions from optical-flow. IEICE Transactions, 1991

[PAN02] Igor S. Pandzic, **MPEG-4 Facial Animation: The Standard, Implementation and Applications,** John Wiley & Sons, 2002

[PEN97] I.A. Essa y A.P. Pentland. **Coding, Analysis, interpretation, and recognition of facial expressions**. IEEE Trans. On Pattern Analysis and Machine Intelligence, 1997.

[SUW78] M. Suwa, N. Sugie, K. Fugimora, **A preliminary note on pattern recognition of human emotional expression**. International Joint Conference on Pattern Recognition, 1978.

[THE03] Sergios Theodoris, Konstantinos Koutroumbas, **Pattern Recognition**, Academic Press, 2003

[WIL90] L. Williams, **Performance-driven facial animation**, ACM SIGGRAPH Conference Proceedings, 1990

[YAA96] Y. Yaacob y L.S. Davis. **Recognizing human facial expressions from long image sequences using optical flow.** IEEE Trans. on Pattern Analysis and Machine Intelligence, 1996.

[YEO01] Yeonho Kim, **Optical Flow Analisys. Peformance of Optical Flow Techniques**, Purdue Univeristy, 2001.
http://rvl2.ecn.purdue.edu/~yeonho/research/OpticalFlowAnalysis/

[YIN03] Ying-li Tian, Lisa Brown, et. al. **Real World Real-Time Automatic Recognition of Facial Expressions**, Exploratory Computer Vision Group, IBM Watson Research Center, 2003.

Referencias adicionales

Expat XML library
http://expat.sourceforge.net/

Howard Hughes Medical Institute
http://www.hhmi.org/biointeractive/vlabs/index.html

Johns Hopkins University
http://www.jhu.edu/~virtlab/virtlab.html

OpenCV
http://sourceforge.net/projects/opencvlibrary/

OpenCV en Intel
http://www.intel.com/research/mrl/research/opencv/

University of Kansas - Elena Popel
http://www.ittc.ku.edu/~jgauch/teaching/742.s05/Elena.ppt

NASA Simulation Laboratories
http://www.simlabs.arc.nasa.gov/vast/vast.html

The Computation of Optical Flow
http://www-etsi2.ugr.es/depar/ccia/mia/complementario/t7/p433-beauchemin.pdf

Virtual Science.
http://www.virtualscience.com

Glosario