UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA

LOCALIZACIÓN, SEGUIMIENTO Y RECONOCIMIENTO DE ROSTROS EMPLEANDO MÉTODOS ESTADÍSTICOS PCA Y HMME

TESIS

PARA OPTAR EL TITULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

JOSÉ ANTONIO HUAMÁN LAYME

PROMOCIÓN

2002-II

LIMA – PERÚ

2007

SUMARIO

A pesar de la existencia de sistemas biométricos que permiten el reconocimiento de rostros desde comienzos del siglo XX, estos aún no han alcanzado un porcentaje óptimo necesario. Aunque se trata de una tecnología no madura, en los últimos años han aumentado la inversión y las expectativas depositadas en ella. Básicamente se trata del desarrollo de algoritmos que permiten una adecuada identidad de la persona a través del rostro de la misma. Al ser las imágenes de rostros representaciones en 2D y algunos casos 3D, implican un procesamiento alto en operaciones.

A medida que la tecnología va incrementándose el problema de velocidad pasa a ser de segundo plano, el objetivo principal es obtener un porcentaje alto de aciertos y esto se logra a través de algoritmos más robustos. Estos algoritmos son probabilísticos por lo que se requiere un umbral que separe los aciertos y desaciertos.

En el desarrollo de la tesis se han empleado algoritmos para la detección y reconocimiento de rostros. Para la detección de rostros se empleó algoritmos en cascada que sumados permitieron un mejor desempeño. Para el reconocimiento de rostros se evaluaron varios algoritmos cuyas salidas probabilísticas determinan si es un rostro reconocido dentro de una base de datos. Finalmente se implementó el sistema reconocedor de rostros de acuerdo al mejor algoritmo evaluado, la implementación se realizó utilizando el lenguaje c++ y el hardware consistió de una cámara de video, una tarjeta digitalizadora de video, una PC y un sistema controlado por motores paso a paso para el seguimiento de personas en un ambiente determinado.

INDICE

INTRODUCCIÓN

El tema de tesis tiene como objetivo el desarrollo e implementación de un localizador, seguidor y reconocedor de rostros a través de una imagen de video en tiempo real empleando procesamiento de imágenes y métodos estadísticos como el Análisis de Componentes Principales (PCA) y el Modelo Oculto de Markov Embebido. Actualmente las instituciones educativas en el ámbito mundial, implementan muchos proyectos basados en tratamiento de imágenes conocido como visión artificial o visión por computador. Por tanto se pretende dar iniciativa al desarrollo de visión por computadora que incluye una diversidad de temas de desarrollo en el área de procesamiento de señales.

Desde que hubo los atentados de las torres gemelas en New York en el año 2002 [1] los sistemas de seguridad experimentaron un creciente interés y con ello una mayor investigación, especialmente los basados en características de tipo biométrico. Este éxito se debe en gran medida a que, cuando una persona pretende acceder a un sistema, la decisión se toma en base a características específicas de esa persona, y no en base a lo que conoce (como cuando se emplea claves alfanuméricas) o a lo que posee (como en el caso de tarjetas magnéticas). . Un sistema reconocedor de rostros es una tecnología biométrica emergente debido a que estos actúan pasivamente, es decir que personas no se dan cuenta que están siendo examinadas por cámaras de video.

Existen diferentes algoritmos para el reconocimiento de rostros como son las técnicas de Eigenspace, igualamiento de plantilla, redes neuronales, LFA, HMMs, etc. La diferencia que existe entre estos algoritmos es la eficiencia que se da en el reconocimiento. Actualmente el método de LFA (Local Features Análisis) es el más utilizado, pero su algoritmo es resguardado por sus creadores, cuenta con un porcentaje de reconocimiento aceptable aproximadamente de 98% a 99% [2] (FAR y FRR menores a 1%) de acuerdo a las condiciones de ambiente (Iluminación, sensor de la cámara, digitalizador, Potencia del computador, etc.) y a la base de datos empleada. La eficiencia se saca de la toma de muestras, ya que estos métodos son estadísticos.

Para el desarrollo de este tema el capitulo uno empezará con una breve teoría básica sobre procesamiento digital de imágenes, luego se describirá la adquisición y la

base de datos de imágenes de rostros utilizada en desarrollo de la tesis. Seguidamente el pre-procesamiento de estas imágenes de rostros.

En el cuarto capitulo se describe como se realiza la localización y seguimiento de un rostro en una imagen de video, luego en los capítulos siguientes se da la teoría y el desarrollo de los métodos a emplearse en el reconocimiento rostros.

Finalmente se implementa el sistema seguidor de movimiento de personas que esta tendrá la función de localizar los rostros y reconocerlos, esto se realizada mediante una computadora personal, una cámara de video y un controlador de posición.

CAPITULO I
TEORIA BASICA DE PROCESAMIENTO DIGITAL DE IMÁGENES

1.1 Elementos Básicos para el Procesamiento de Imágenes

Los elementos básicos para los sistemas de procesamiento digital de imágenes se muestran en la figura 1.1. Estos sistemas realizan las tareas de Adquisición, almacenamiento, tratamiento, comunicación y representación de imágenes.

Figura1.1. *Elementos básicos de un sistema de procesamiento de imágenes*

La formación de la imagen es un proceso en el cual la escena (3 dimensiones) es proyectada como una imagen (2 dimensiones), un ejemplo claro son nuestros ojos tal como se ilustra en la figura 1.2. Por lo tanto los dispositivos de adquisición de imágenes tienen la función de imitar y mejorar el proceso llevado a cabo por nuestros ojos.

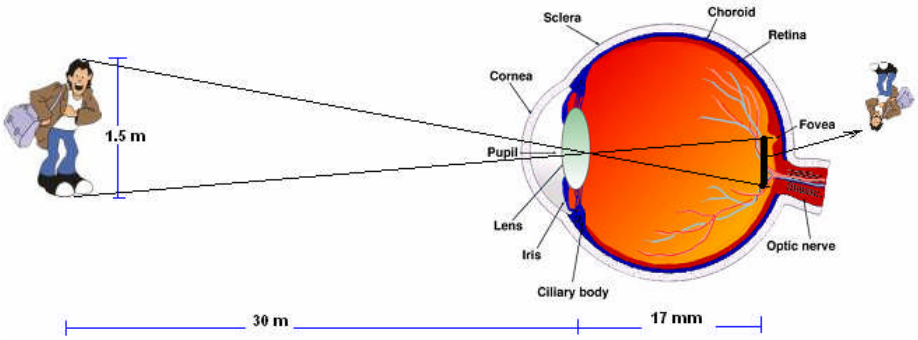


Fig. 1.2. *Adquisición de una imagen a través del ojo humano*

Para la adquisición de imágenes se necesita de dos elementos [3]. El primer elemento es un sensor sensible a una determinada banda del espectro electromagnético (en nuestro caso el espectro visible), de tal manera que produzca una señal eléctrica de salida proporcional al nivel de energía detectado. El segundo elemento es el digitalizador, que convierte la señal de salida en forma digital. La Fig. 1.3 muestra este proceso. Los dispositivos mas frecuentes para esta función son los scanner's, webcam's, cámaras fotográficas, etc.

Los tipos de sensores más usados están basados en dos tipos de tecnologías, CCD (Charged Couple Device) o CMOS (Complementary Metal Oxide Semiconductor).

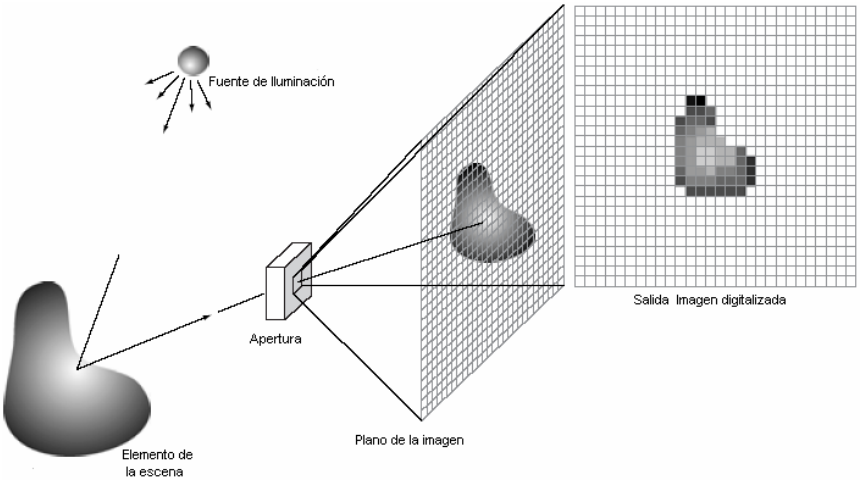


Fig. 1.3. *Adquisición de imágenes de un sistema ideal*

Los sensores CCD tienen mayor sensibilidad a la luz, y requieren un consumo eléctrico entre 2 y 5 Watts, en tanto que los de tipo CMOS son menos sensibles a la luz y requieren un consumo eléctrico entre 30 y 50mW. Actualmente, el nivel que ofrecen las cámaras con CCD es mejor que el de las equipadas con CMOS. Sin embargo, la calidad

de estas últimas aumenta progresivamente y tiene las mejores perspectivas para el futuro.

Una Imagen digital es una matriz o array bidimensional de números, donde cada celda de la matriz es un píxel, tal como se muestra en Fig. 1.4.

Un píxel

90	67	68	75	78	98	185	180	153	139	132	106	70	80	81	69	69	67	35	34
92	87	73	78	82	132	180	152	134	120	102	106	95	75	72	63	75	42	19	29
63	102	89	76	98	163	166	164	175	159	120	103	132	96	68	42	49	46	17	22
45	83	109	80	130	158	166	174	158	134	105	71	82	121	80	51	12	50	31	17
39	69	92	115	154	122	144	173	155	105	98	86	82	106	83	76	17	29	41	19
34	80	73	132	144	110	142	181	173	122	100	88	141	142	111	87	33	18	46	36
37	93	88	136	171	164	137	171	190	149	110	137	168	161	132	96	56	23	48	49
66	117	106	147	188	202	198	187	187	159	124	151	167	158	138	105	80	55	59	54
127	136	107	144	188	197	188	184	192	172	124	151	138	108	116	114	84	46	67	54
143	134	99	143	188	172	129	127	179	167	106	118	111	54	70	95	90	46	69	52
141	137	96	146	167	123	91	90	151	156	121	93	78	82	97	91	87	45	66	39
139	137	80	131	162	145	131	129	154	161	158	149	134	122	115	99	84	35	52	30
137	133	56	104	165	167	174	181	175	169	165	162	158	142	124	103	67	19	31	23
135	132	65	86	173	186	200	198	181	171	162	153	145	135	121	104	53	14	15	33
132	132	88	50	149	182	189	191	186	178	166	157	148	131	106	78	28	10	15	44

Fig. 1.4. *Representación de una imagen con valores de luminosidad*

La manera de visualizar una imagen, es dándole un nivel de color de acuerdo al valor de la misma, por ejemplo entre valores de 0 (negro) y 255 (blanco). Ver Fig. 1.5.

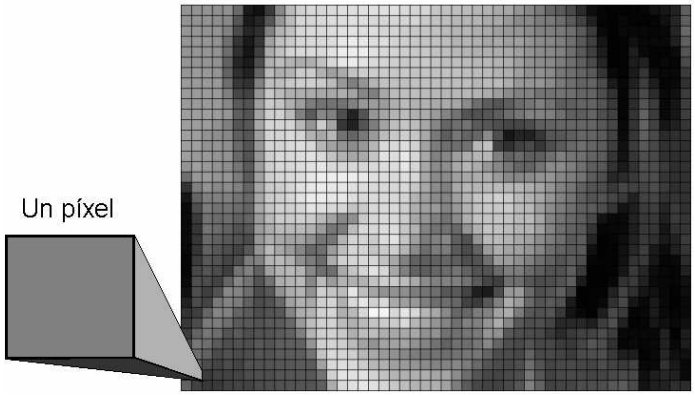


Fig. 1.5. *Representación de una imagen en escala de grises*

El número de columnas de la matriz representa el ancho de la imagen, el número de filas de la matriz representa la altura de la imagen, El tamaño de la imagen (resolución) se representa como Ancho x Alto, El tamaño típico de una imagen es de 320x240, 640x480, 800x600 o 1024x768.

El tipo de datos de cada celda puede ser binaria, escalas de grises o de color. En una imagen binaria el píxel esta compuesto de un bit por tanto solo puede tomar dos valores (0 = negro y 1=blanco). En una imagen en escala de grises el píxel toma 8 bits equivalente a 1 byte, por tanto puede tomar valores entre 0 y 255. En una imagen a color

cada píxel consta de 3 valores (Rojo, Verde, Azul) y 1 byte por cada valor, por tanto existen 16.7 millones de valores (colores).

Fig. 1.6. *Tipos de datos en imágenes. Binario, escala de grises y color*

Cada imagen también se puede representar con más o menos bits, llamado también profundidad de bits y profundidad de color.

Fig. 1.7. *Tipos de imágenes de acuerdo al número de bits*

Una imagen de 8 bit's y 320 x 240 de resolución necesita 76800 bytes de memoria. Esta imagen se puede ser usada a corto plazo, para ser empleado durante el procesamiento o puede ser usado en línea, para una reutilización relativamente rápida, como también se puede almacenar en archivo bajo un formato adecuado.

Existen muchos formatos de almacenamiento en archivo, entre ellos destacan BMP, GIF, TIF, JPG. Cada formato tiene sus ventajas y desventajas de acuerdo al uso que se le emplee. El formato BMP es el usado en procesamiento de imágenes ya que guarda la imagen cruda sin comprimir, la desventaja de este formato es el tamaño en bytes que ocupa en memoria.

1.2 Transformación de imágenes

El procesamiento de datos en una imagen se puede enfocar de dos maneras diferentes, la alteración píxel a píxel de los datos en una escala global (individuales) y operaciones basadas en múltiples puntos (vecindad).

En las transformaciones globales, cada píxel de salida depende solo de un pixel de entrada, osea que no se tiene en cuenta la relación de vecindad entre pixeles. En

cambio en la transformación local, el valor de un pixel depende de la vecindad local de ese pixel.

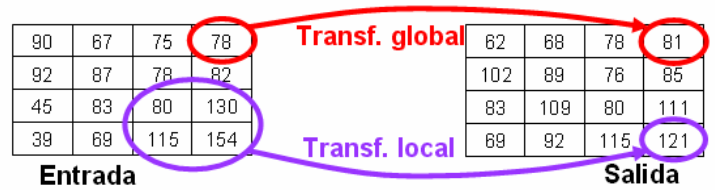


Fig. 1. 8. *Tipos de transformacion de una imagen*

En las transformaciones globales existe un operador individual que implica la generación de una nueva imagen modificando el valor del píxel en una simple localización basandose en una regla global aplicada a cada localizacion de la imagen original, por tanto el operador individual es una transformación uno a uno.

$$q(x,y) = f(p(x,y)) \qquad (1.1)$$

El operador f se aplica a cada pixel p de la imagen, el proceso matemático es relativamente simple. La imagen resultante es de la misma dimensión que la original. Existen diferentes operadores[4] que cumplen una determinada función, asi el operador Identidad, que crea una imagen de salida identica al original. Operador inverso o negativo, utilizado en imagen médicas. Operador umbral, que crea una imagen de salida binaria a partir de una imagen de grises. Operador intervalo de umbral binario, similar al anterior pero el nivel de gris esta en un intervalo definido. Operador reducción de nivel de gris. Existen otros operadores que trabajan con dos imágenes asi se tiene la operación adición, operación sustracción entre otros.

Fig. 1.9. a*) Imagen original b) imagen Invertida (negativo del anterior)*

Las operaciones de vecindad en las transformaciones locales utilizan el mismo procedimento excepto que el nuevo valor de píxel en la imagen de salida, depende de una combinacion de valores de lo píxeles en la vecindad del pixel en la imagen original que esta siendo transformada. Algunas de estas transformaciones son operaciones de convolución.

Respecto a las transformaciones lógicas, estas son empleadas en las imágenes binarias, con una imagen de estas caracteristicas es posible realizar todo tipo de operaciones lógicas, asi por ejemplo “negación”, “or”, “and”, “xor” entre otros.

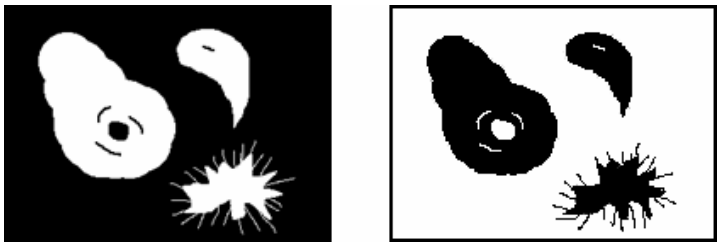


Fig. 1.10. *a) Imagen binaria b) Imagen binaria invertida*

Transformaciones geométricas, que son utilizadas para investigar más especificamente en un area dentro de la imagen, llamada tambien region de interes (RDI). Para realizar esta transformación se necesitan hacer operaciones que modifiquen las coordenadas espaciales de la imagen, las cuales se denominan operaciones geométricas. En estas operaciones estan por ejemplo la interpolación, desplazamiento, cambio de escala, zoom y giros.

En las transformadas en el dominio de frecuencia tenemos la DFT2 que es la transformada discreta de Fourier de dos dimensiones, asi mismo la DCT que es la transformada discreta de Fourier.

Otras transformadas se puede hacer referencia [6] a las siguientes: transformada de Hadamard, transformada haar, transformada Slant, transformada KL entre otros.

1.3 Representación de bordes y regiones

Los puntos de borde, o simplemente bordes son píxeles alrededor de los cuales la imagen presenta una brusca variación en los niveles de gris, este término de bordes se refiere a cadenas conectadas.

Para aplicar los siguientes métodos, se consideran que las regiones a estudiar son suficientemente homogéneas para que la transición entre dos de ellas se pueda determinar sobre la base de discontinuidades de nivel de gris. En la Fig. 1.11 el módulo de la derivada primera se utiliza para detectar la presencia de un borde en una imagen, y

la segunda derivada tiene un paso por cero en el punto medio en una transición de nivel de gris.

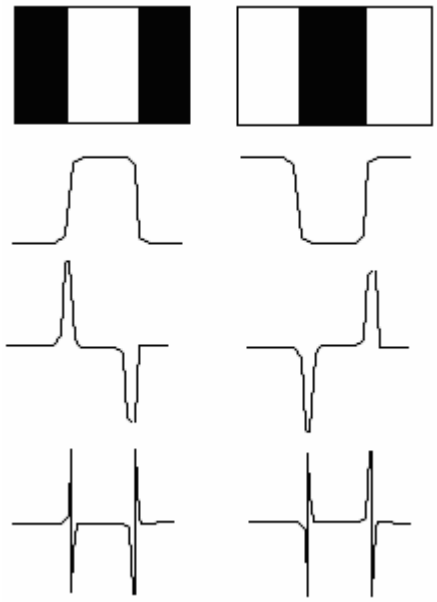


Fig. 1.11. *Detección de bordes por operadores de derivación*

La 1era derivada se obtiene utilizando el módulo del gradiente en un punto, la 2da derivada se obtiene de forma similar utilizando el laplaciano.

El operador gradiente se define en la ecuación 1.2,

$$\nabla \mathrm{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{1.2}$$

Se sabe del análisis vectorial que el gradiente de un vector indica la dirección de la máxima variación de f en (x,y). El módulo del vector llamado gradiente con la notación: ∇f , se muestra en la ecuación

$$\nabla f = mag(\nabla \mathrm{f}) = \left[G_x^2 + G_y^2\right]^{1/2} \tag{1.3}$$

Esta cantidad es igual a la máxima variación de f(x,y) por unidad de distancia en la dirección de ∇ f. En la práctica se aproxima a la ecuación

$$\nabla f \approx |G_x| + |G_y| \tag{1.4}$$

La dirección del vector gradiente

$$\alpha(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \qquad (1.5)$$

Donde el ángulo se mide con respecto al eje X. Los operadores de sobel tienen la ventaja de presentar tanto una diferenciación como un efecto de suavizado, como las derivadas realzan el ruido, este efecto de suavizado es una característica particularmente atractiva de los operadores de sobel.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

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

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

Fig. 1.12 *a) Región 3x3 b) Máscara G_x c) Máscara G_y*

En la Fig. 1.12 se observa que:

$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \qquad (1.6)$$

$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \qquad (1.7)$$

Donde, las z son los niveles de gris de los píxeles solapados por las mascaras en cualquier posición de una imagen. Luego se utiliza la ecuación (1.6) y (1.7), que proporcionan un valor de gradiente.

Una vez completado el procedimiento en todas las posibles ubicaciones, el resultado es una imagen gradiente del mismo tamaño de la original

El Laplaciano es una función bidimensional f(x,y) de segundo orden definido por:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad (1.8)$$

Esta ecuación se puede implementar de forma digital de varias maneras. La forma más frecuente en la práctica es:

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8) \qquad (1.9)$$

El requisito básico para la definición del laplaciano digital es que el coeficiente asociado con el píxel central sea positivo y los coeficientes asociados con los píxeles exteriores sean negativos, como el laplaciano es una derivada, la suma de los coeficientes debe ser cero.

El laplaciano responde a las transiciones de intensidad, rara vez se utiliza en la práctica para la detección de bordes, por varias razones. Como es una derivada de segundo orden, normalmente es inaceptablemente sensible al ruido. Además produce bordes dobles, y es incapaz de detectar direcciones de borde, por estas razones el laplaciano desempeña un papel secundario de detección para establecer si un píxel esta en la parte clara u oscura de un borde.

Un empleo más general del laplaciano consiste en encontrar la ubicación de bordes utilizando las propiedades de paso por cero.

Respecto a la representación de regiones, estas se basan en el hecho de constituyen agrupaciones de píxeles conectados entre sí, pero además de la conexión, dichos píxeles presentan propiedades o características comunes.

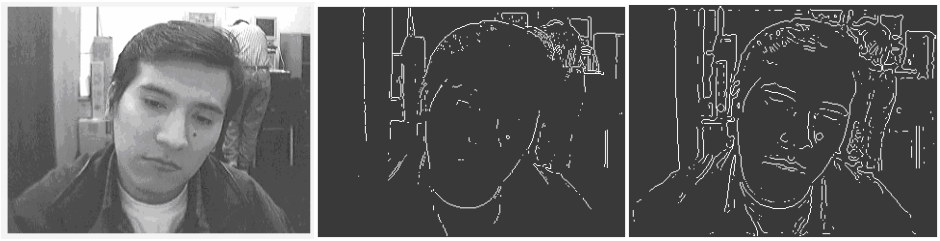


Fig. 1.3 *a) Imagen original b) Operador Sóbel c) Operador del laplaciano*

El histograma de una imagen digital con niveles de gris en el rango [0,L-1] es una función discreta $p(r_k) = n_k/n$, donde r_k es el k-esimo nivel de gris, n_k es el número de píxeles de la imagen con este nivel de gris, n es el número total de píxeles de la imagen con ese nivel de gris y k=0,1,2,...,L-1

La representación grafica de la función antes mencionada para todos los valores de k que proporciona una descripción global de la apariencia de la imagen, la Fig. 14 se muestra la imagen y su respectivo histograma.

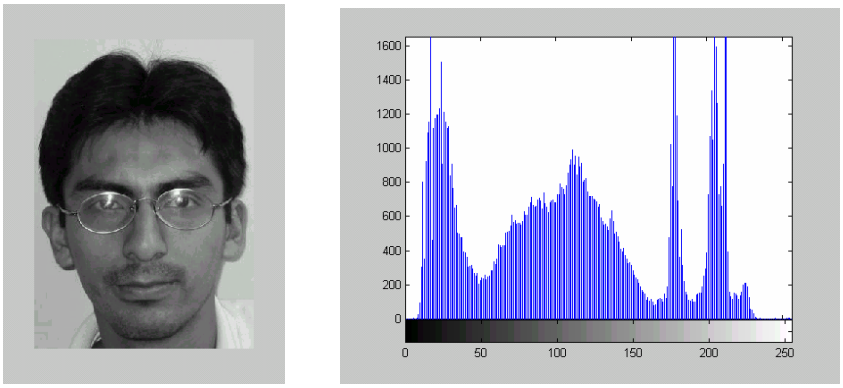


Fig. 1.14 *a) Imagen b) Histograma de la imagen anterior*

1.4 Mejora, filtrado y restauración de imágenes

La etapa de preprocesado de la imagen tiene la función básica de mejorar la imagen de forma que aumenten las posibilidades de éxito en los procesos posteriores. Este proceso utiliza técnicas para mejorar el contraste, eliminar el ruido y aislar regiones de las partes irrelevantes de una imagen.

Un modelo de color es utilizado para facilitar la especificación de colores en algún formato Standard. En esencia, un modelo de color es una especificación de un modelo de coordenadas 3-D y un subespacio dentro de ese sistema donde cada color se representa por un punto único.

La mayoría de los modelos de color que se usan hoy se orientan hacia hardware como monitores o impresoras o aplicaciones de manipulación de color. El modelo orientado a hardware más común es el RGB (rojo-verde-azul), el modelo CMY (cian-magenta-amarillo) para impresoras en color y el YIQ que es el Standard para la difusión de TV, en este caso la componente Y, esta corresponde a luminancia y las componentes I y Q son dos componentes cromáticas. Para la manipulación del color se usan normalmente los modelos HSI (matiz, saturación e intensidad) y HSV (matiz, saturación y valor). Los modelos de color más usados en el procesamiento de imágenes son RGB, YIQ y HSI.

En el modelo RGB, cada color aparece con sus componentes primarias espectrales rojo, verde y azul. El modelo está basado en un sistema de coordenadas cartesiano. El subespacio de interés es el cubo que se muestra en la Fig. 1.15.

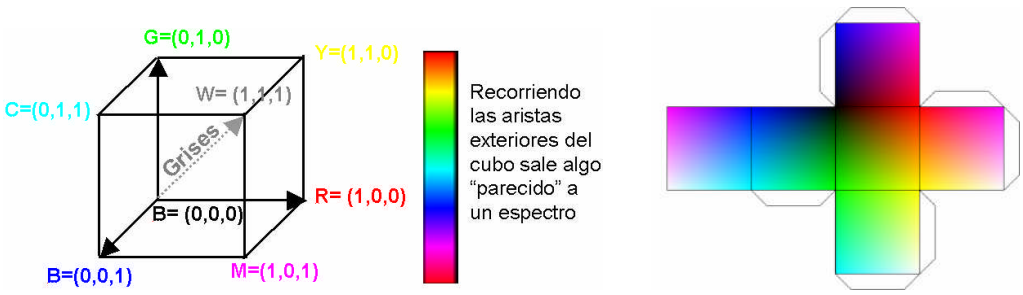


Fig. 1.15. *Cubo de valores RGB. Los puntos sobre la diagonal principal corresponden a valores de gris, desde el negro, en el origen, hasta el blanco en el punto (1,1,1).*

Fig. 1.16 *a) Imagen Original RGB; b) Plano de Rojo; c) Plano de Verde; d) Plano de Azul;*

Ahora no siempre es éste el mejor modelo de color para el procesamiento de imágenes. Consideremos, por ejemplo que queremos realzar una imagen en color de una cara humana, parte de la cual está oculta por sombra.

Fig. 1.17 *a) Imagen Original RGB; b) Plano rojo; c) Plano verde; d) Plano azul*

La ecualización del histograma es una técnica ideal para este tipo de problema. Debido a la presencia de las tres imágenes y a que la ecualización del histograma solo considera los valores de la intensidad, la técnica más trivial consiste en someter cada plano de la imagen de forma independiente a la ecualización del histograma. La parte de la imagen oculta por la sombra será mejorada en los tres casos. Sin embargo, las intensidades de los tres planos de la imagen se alterarán de forma diferente, con el resultado de un cambio de las intensidades relativas entre ellos. El resultado neto será que las propiedades cromáticas importantes de la imagen, como los tonos de carne, no parecerán naturales cuando se muestren en un monitor RGB. Algunos de los modelos de color son más adecuados para este tipo de problemas.

Fig. 1.18. *Después de la ecualización: a) plano rojo; b) plano verde; c) plano azul; d) imagen final*

El modelo HSV (Tono-Saturación-Valor) se debe su utilidad a dos hechos básicos, primero el componente de valor esta desacoplada de la información cromática contenida en la imagen. Segundo las componentes de tono y saturación están íntimamente relacionadas con las formas en que los seres humanos percibimos el color.

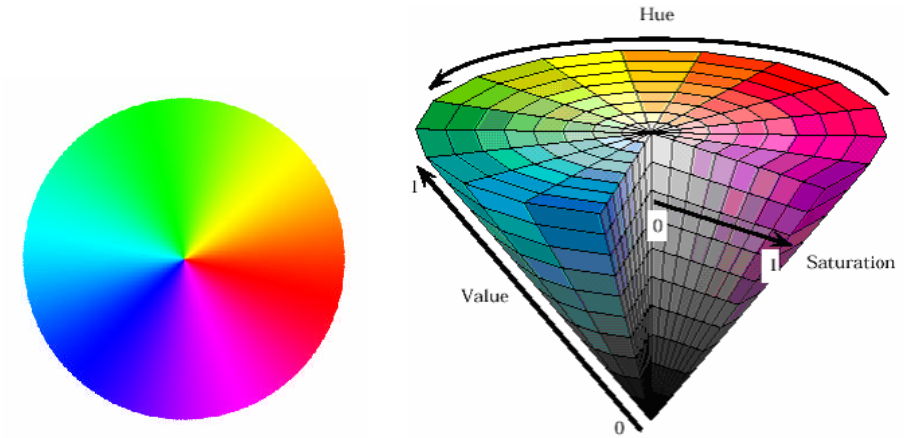


Fig. 1.19. *a) Rueda Cromática b) Sólido del modelo de color HSV*

Del problema anterior del rostro se pasa a emplear este modelo de color:

Fig. 1.20. *a) Imagen original; b) Plano del Tono; c) Plano de Saturación d) Plano de Valor*

La imagen se convirtió en HSI y su componente de intensidad se sometió a la ecualización del histograma. La imagen se convirtió posteriormente otra vez a RGB, obteniendo como resultado lo siguiente:

Fig. 1.21 *a) Imagen Original; b) Imagen ecualizada en el modelo de color HSV*.

1.5 Segmentación de imágenes

La Segmentación [6] de imágenes es un primer paso en el análisis de imágenes, la segmentación subdivide una imagen en sus partes constituyentes u objeto. El nivel al que se lleva a cabo esta subdivisión depende del problema a resolver. Esto es, la segmentación deberá detenerse cuando los objetos de interés de una aplicación hayan sido aislados. En nuestro caso la adquisición automática de rostros en una imagen el interés tiende, entre otras cosas a la identificación de rostros.

Los algoritmos de segmentación de imágenes monocromáticas generalmente se basan en una de las dos propiedades básicas de los valores del nivel de gris: discontinuidad y similaridad. En la primera categoría, el método consiste en dividir una imagen basándose en los cambios bruscos del nivel de gris. La detección de puntos aislados, la detección de líneas y bordes de una imagen son las áreas de interés de esta categoría. En la segunda categoría los métodos utilizados están basados en la umbralización, crecimiento de región, y división y fusión de regiones. Estos algoritmos son aplicados tanto en imágenes estáticas como dinámicas (video).

1.6 Morfología

La morfología matemática esta basada en la geometría y la forma de una imagen. Estas operaciones morfológicas simplifican las imágenes y preservan las formas fundamentales de los objetos.

La morfología tiene la tarea de suavizar los bordes de una región, esto es útil si se necesita mejorar un borde que fue obtenido defectuosamente mediante técnicas de segmentación normales. La morfología también separa determinadas regiones que el proceso anterior de segmentación las mantiene unidas o por el contrario también nos permite unir regiones que han sido separadas durante la segmentación.

Estas operaciones morfológicas se aplican generalmente en imágenes binarias. Dentro de las transformaciones morfológicas se tiene la dilatación, erosión, apertura y cierre que son base para otras transformaciones morfológicas.

1.7 Extracción y análisis de características

Para poder realizar el reconocimiento de patrones, las imágenes deben ser representadas mediante un vector característico del mismo. Por tanto un patrón es una descripción estructural o cuantitativa de un objeto o de alguna otra entidad de interés en una imagen. En general, un patrón [7] esta formado por uno o mas descriptores o características. Una clase de patrones es una familia de patrones que comparten algunas propiedades comunes.

Por ejemplo en una imagen el vector característico puede ser toda la imagen o una serie de medidas efectuadas sobre la imagen en un proceso de extracción de características (por ejemplo, en reconocimiento de rostros la distancia entre los ojos, grosor de las cejas, etc.) de forma que se obtenga un conjunto de datos.

Los vectores P-dimensiónales de características se podría representar en un espacio vectorial R^P. Sin embargo, para dimensiones mayores a 3 (P>3) no resulta factible una representación gráfica.

La extracción de características llamado también parametrización proporciona innumerables ventajas, entre ellas la reducción del número de datos necesario a procesar, tamaño del modelo de una imagen, entre otros. Y la transformación a un nuevo espacio de características en el cual es más fácil discriminar entre rostros.

La separabilidad entre clases no depende únicamente de su distribución en el espacio de señal, sino que viene condicionada por el tipo de clasificador a utilizar. Por ejemplo, el conjunto óptimo de características para un clasificador lineal puede no ser óptimo para otro clasificador, como por ejemplo una red neuronal.

1.8 Clasificación de imágenes y reconocimiento de patrones

El reconocimiento de patrones tiene como objetivo la clasificación de objetos, personas, señales, representaciones en un cierto número de categorías o clases. Las aplicaciones son amplias, pero por lo general están asociadas a la visión y audición por parte de una maquina. Por ejemplo, el reconocimiento de caracteres que a partir de un texto o manuscrito se genera un archivo que posee los caracteres de ese texto y es aplicado a reconocimiento de placas de automóviles o lectura de textos mediante un conversor de voz.

Las clases deben ser separables entre sí, esta separabilidad no depende únicamente de su distribución en el espacio si no que viene condicionada con el tipo de clasificador a utilizar.

Generalmente existían dos enfoques en el reconocimiento de patrones, estos eran estadísticos (teoría de decisión) y el sintáctico (estructural). Recientemente el desarrollo de las redes neuronales utilizan patrones de naturaleza cuantitativa, mientras que el enfoque sintáctico utiliza las relaciones geométricas asociada a la forma del objeto de la imagen.

En este tema nos concentraremos en los métodos estadísticos. Existen muchos métodos estadísticos para diversos usos de aplicación por lo que nos basaremos en los utilizados en imágenes especialmente en el reconocimiento de rostros.

Estos métodos son los siguiente, análisis de componentes principales (PCA), que tiene como objetivo buscar una reduccion de dimensionalidad manteniendo la representación de los datos. Análisis Discriminante Lineal (LDA) que tiene como objetivo el anterior además de buscar la separabilidad entre clases. Modelo Oculto de Marvok, este método genera un modelo de estados basados en la correspondencia entre secuencia de observación y secuencia de estados.

A estos dos métodos se le dara mayor interes ya que según la literatura poseen mayor eficiencia frente a otros.

CAPITULO II
ADQUISICIÓN Y BASE DE DATOS DE IMÁGENES DE ROSTROS

2.1 Adquisición de imágenes

Esta es una etapa primordial para el sistema, es aquí donde la imagen es adquirida por medio de cámaras y de esto depende bastante el rendimiento eficaz del sistema que se desea obtener.

Para lograr tener una imagen clara es necesario elegir una cámara con su digitalizador en buenas condiciones. Además de tener una buena iluminación acorde con el ambiente a ser usado

2.1.1 Descripción en cuanto a cámaras de videos existentes

A partir del año1980 empezaron a aparecer sensores CCD en el mercado. Inicialmente se utilizaban en camcorders, luego se empleo en cámaras de vigilancia. A medida que se expandía el mercado y debido al avance de altas resoluciones de los sensores CCD nacieron las cámaras digitales y estas ampliaron su aplicación a PDAs, juegos, Cámaras Digitales, Cámaras en vehículos. En el año 2000 nacen los módulos de cámaras incorporados en los equipos celulares, empleando sensores CCD/CMOS.

A medida que se requiera más resolución con el número de píxeles se debe emplear el sensor adecuado, la figura (2.1) muestra las aplicaciones de acuerdo al tamaño del sensor óptico y al número de píxeles.

Para el propósito que deseamos, existen una gran variedad de cámaras de video entre analógicos y digitales. Las cámaras de video analógicos necesitan de una tarjeta digitalizadora instalada en una computadora para que esta pueda procesar los datos del video analógico. Las cámaras de video digital generalmente tienen incorporados una plaqueta digitalizadora y la transmisión a la PC se realiza mediante el bus IEEE 1394 o mediante el USB.

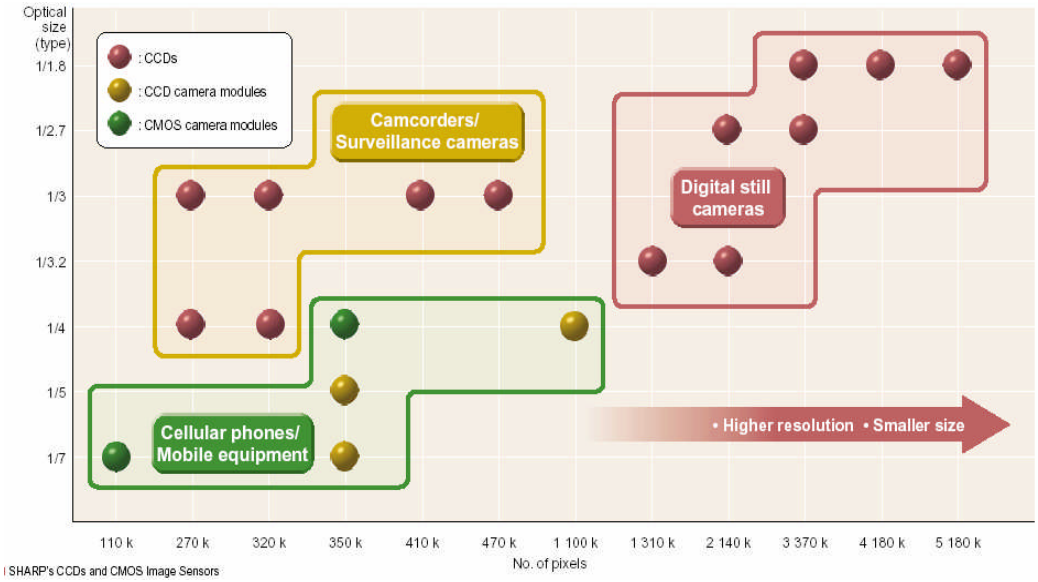


Figure 2.1 *Aplicaciones de acuerdo al número de píxeles y tamaño óptico.* Información de sharp-world [8]

Los parámetros o características que definen una cámara de video analógico son el tipo de sensor, resolución, número de píxeles, iluminación mínima, nivel de ruido S/N, distancia focal (lente), ángulo de visión, entre otros.

El tipo de sensor para cámaras de vigilancia es generalmente el CCD y el tamaño optico es de 1/3" y ¼", según se muestra en la Fig. (22).

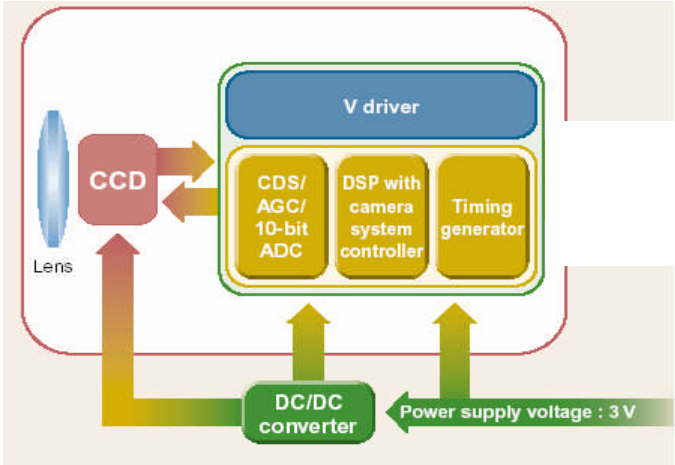


Fig. 2.2 Ejemplo de un sistema con sensor CCD *[8]*

Entre los componentes de las cámaras de video analógicas tipo CCD están la lente, el censor CCD, un conversor DC/DC y un bloque que controla los parámetros de la señal de video como es la ganancia o el tipo de salida (NTSC/PAL) tal como es muestra en la Fig. 2.2.

En las cámaras de video con sensor CMOS se encuentra la lente, y a diferencia de la camara CCD el sensor CMOS va en un bloque junto con la ganancia y el tipo de salida. Este bloque es manejado por medio de un DSP controlar, tal como se muestra en la Fig. 2.3

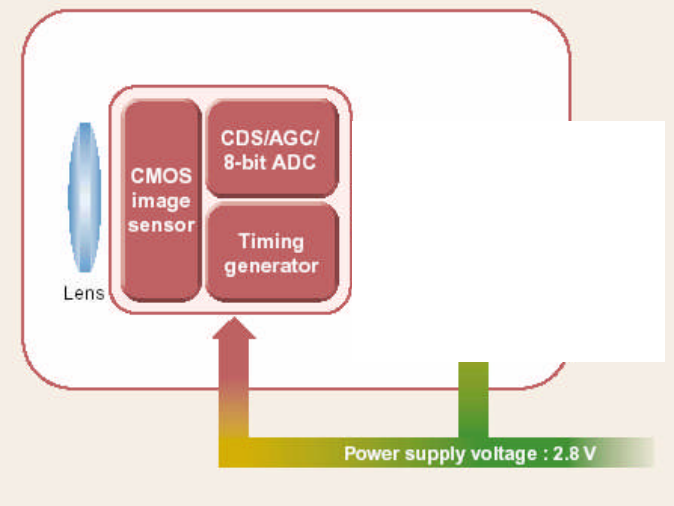


Fig. 2.3 Ejemplo de un sistema con sensor CMOS *[8]*

Respecto al tamaño del sensor de imagen en las cámaras CCD, se tiene de 1/3" y ¼". Para las cámaras en Blanco y Negro B/W generalmente se usan los sensores de 1/3" con resoluciones de 270K píxeles y 320K píxeles. Para las cámaras a color se pueden usar de 1/3" y ¼" y las resoluciones don de 270K, 320k píxeles para el sensor de ¼" y para sensores de 1/3" las resoluciones de 410k y 470k píxeles.

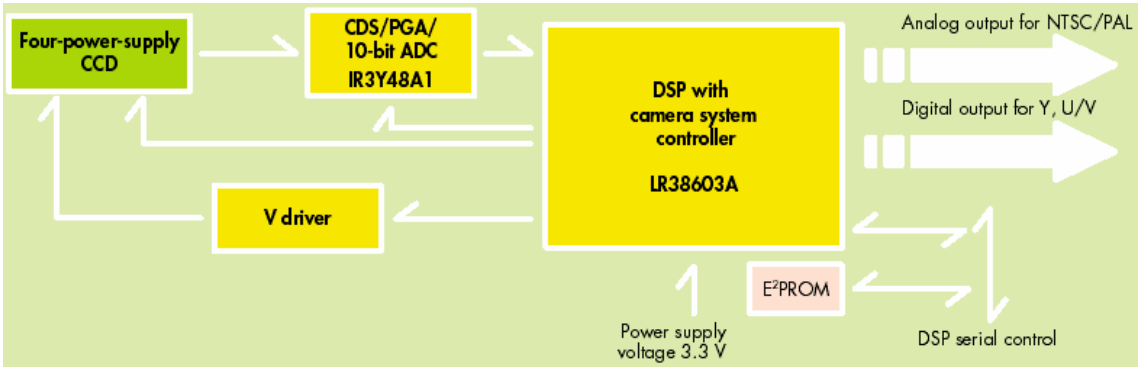


Fig. 2.4 *Dispositivos en el bloque del circuito de control [8]*

El Chip LR38603 es un procesador de señales para sensores CCD de 270k,320k,410k y 470k píxeles. Incorpora un DAC de 9 bits, un circuito generador de señales sincronizadas, un generador de tiempos driver CCD.

Circuito procesador para el control AWB/EE, salida digital Y/U/V, salida analogica NTSC/PAL y soporta monitoreo de salida de IR3Y48A1.

Este bloque además posee un procesador de señal, que es el IR3Y48A1 y es un chip con tecnología CMOS en IC que tiene la función de procesar la señal para sensores de área CCD, incluye un circuito doble muestreo de correlación (CDS: Circuit double sampling), un amplificador de ganancia programable, generador de voltaje referencial, circuito de detección de nivel negro, un conversor ADC 18MHZ a 10 bits. Generador de tiempos para pulsos internamente y una inferface serial para el control de ganancia PGA.

■ **1/3-type CCDs**

No. of pixels	Standard		Electronic shutter (s)	Resolution		Pixel size H x V (μm²)	Sensitivity (mV) TYP.	Smear ratio (dB) TYP.
				Horizontal TV lines	Image pixels (H x V)			
270 000	Color	NTSC	1/60 to 1/10 000	330	512 x 492	9.6 x 7.5	1 300	–120
320 000		PAL	1/50 to 1/10 000		512 x 582	9.6 x 6.3		
410 000		NTSC	1/60 to 1/10 000	480	768 x 494	6.4 x 7.5	800	–105
470 000		PAL	1/50 to 1/10 000		752 x 582	6.5 x 6.3	750	

■ **1/4-type CCDs**

No. of pixels	Standard		Electronic shutter (s)	Resolution		Pixel size H x V (μm²)	Sensitivity (mV) TYP.	Smear ratio (dB) TYP.
				Horizontal TV lines	Image pixels (H x V)			
270 000	Color	NTSC	1/60 to 1/10 000	330	512 x 492	7.2 x 5.6	800	–105
320 000		PAL	1/50 to 1/10 000		512 x 582	7.2 x 4.7	720	

Fig. 2.5 Sensor de tamaño óptico de 1/3" y ¼". *[8]*

La resolución se mide en líneas de TV, de acuerdo a la norma que se utilice (NTSC, PAL). La Fig. 26 muestra estos parámetros.

La iluminación mínima para que el sensor trabaje adecuadamente esta en el orden de 0.5Lux/F 2.0, donde 0.5Lux indica la iluminación o luz que llega a los objetos, y F2.0 es la apertura del diafragma de la cámaras. Algunas cámaras traen incorporados led's Infrarrojos (IR) que funcionan a 0 Lux. Las lentes estándar en la mayoría de cámaras están en el orden de f3.6mm/F2.0, donde 3.6mm es la distancia focal entre el centro óptico y la superficie del sensor. La figura 22 muestra las lentes de 3.6mm y 8mm y estas tienen un ángulo de visión de 70º y 30º aprox. respectivamente.

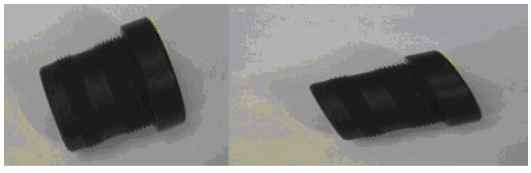


Fig. 26. *Lentes de 3.6mm y de 8mm*

Las cámaras analógicas necesitan de una alimentación externa, generalmente de 12 Voltios DC. +/- 10%, 150mA. Y la salida de video esta en el orden de 1Voltio p-p a 75 Ω .

Entre las cámaras analógicas existen de varios modelos que dependen de la instalación y utilidad que le puedan dar.

En el caso de vigilancia, las cámaras analógicas tienen diferentes modelos como pueden ser el DOMO, que es utilizado mayormente en interiores y los de tubo sellado que son utilizados en exteriores. También existen cámaras inalámbricas con múltiples funcionalidades. En su mayoría utilizan sensores CCD. Se puede ver estos modelos en la figura 2.7.

Figura 2.7. *Algunos modelos de cámaras analógicas de vigilancia existentes*

En el caso de cámaras digitales estas se pueden diferenciar en la manera de cómo se conectan a la PC. Algunas utilizan el puerto USB para transmitir sus datos como es el caso de la WebCam y otras cámaras utilizan el bus IEEE 1394 *[8]*. La diferencia entre estos buses es la velocidad de transmisión. En el caso de USB 1.1 la velocidad máxima de transmisión es de 12Mbps, en cambio con la IEEE 1394 es de 200Mbps.

Las cámaras Web Cam ya incluyen la tarjeta digitalizadora de video mediante un circuito integrado, en el caso del webcam utilizado (WebCam creative Pro), esta posee un CI OV511 que tiene un bajo costo y alta integración. Este CI necesita de una memoria EDO DRAM de 256Kx16 y una cámara digital como la OV7620 para resolución VGA, sensor CMOS y lente para conformar una WebCam-USB.

El diagrama interno de una WebCam se muestra en la figura 29. La interface de la cámara genera imágenes en formatos de 16 bits YUV4:2:2/RGB raw image. También de 8 bits Y4:0:0/RGB raw data inputs. El rendimiento es de 10 a 15 cuadros por segundo para VGA(Video Graphics Adaptor) y 30 cuadros por segundo para CIF(Common Interchange Format).

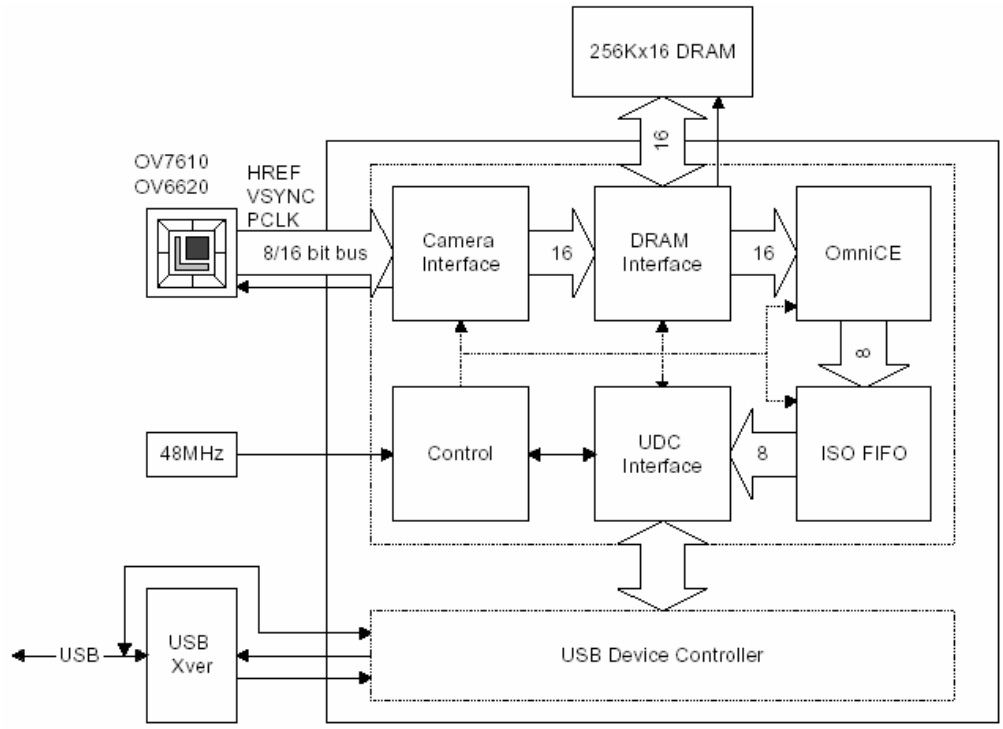


Figura 2.8 *Diagrama interno de una WebCam-USB*

[Omnivision Technologies Inc, www.otv.com]

Algunos de estos modelos se pueden ver en la figura 2.9

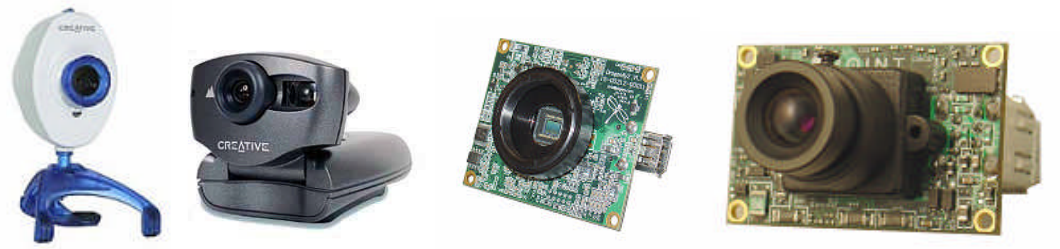


Figura 2.9. *Modelo de cámaras digitales existentes.*

2.1.2 Descripción en cuanto a tarjetas digitalizadoras de video

Como se mencionó anteriormente, las cámaras de video analógicos necesitan de una tarjeta digitalizadora que pueda digitalizar las imágenes que provengan de la cámara.

Estas tarjetas pueden ser internas conectadas a la PC mediante el slot PCI, o pueden ser externas a través del puerto USB.

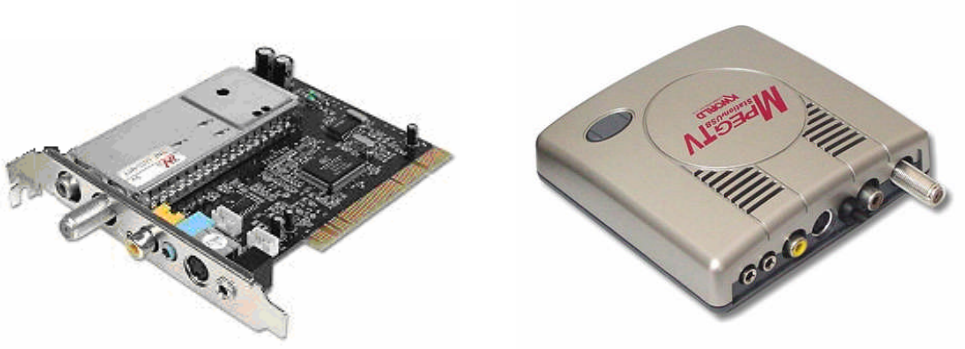


Fig. 2.10 *Tarjetas sintonizadoras de TV y receptoras de video*

Estas reciben video mediante la conexión de entrada de video en RCA o Súper-Video. En la Fig. 2.10 se muestra dos tarjetas sintonizadoras de video MpegTV de Kworld Computer. La primera es una tarjeta sintonizadora de TV a través del slot PCI del computador tal como se muestra en la Fig. 2.11 las características son las siguientes:

a. Conector FM (para recepción de radio FM)

b. Conector TV para antena o cable

c. Salida de audio (3.5mm stereo jack)

d. Entrada de video compuesto (RCA)

e. Entrada SVHS (mini DIN 4 pin)

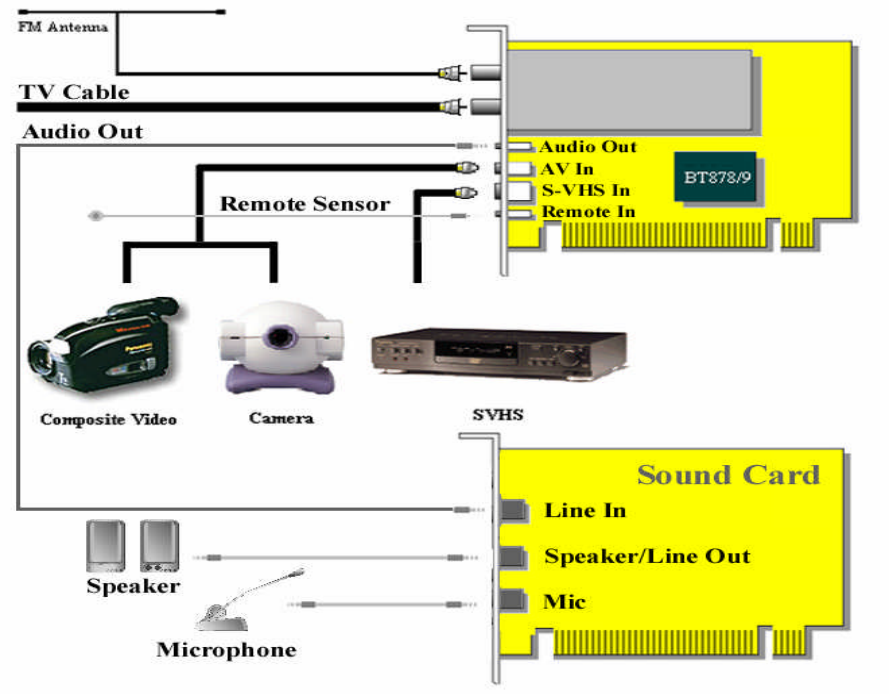


Fig. 2.11 Interface de una sintonizadora de TV a través de PCI

La segunda imagen es una tarjeta sintonizadora a través de USB, las características son similares a la primera imagen de la figura 2.12.

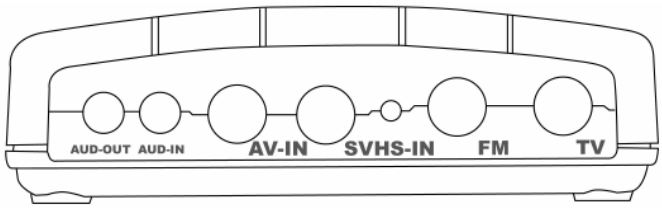


Fig. 2.13 Interface de sintonizadora de TV a través de USB

Esta tarjeta tiene muchas utilidades como la adquisición de video de cualquier dispositivo (VHS, cámaras digitales, etc...), y luego el paso a VCD o DVD

Existen también en el mercado las tarjetas que tienen como entrada 4 receptores de video, tal como se muestra en la Fig. 2.14.

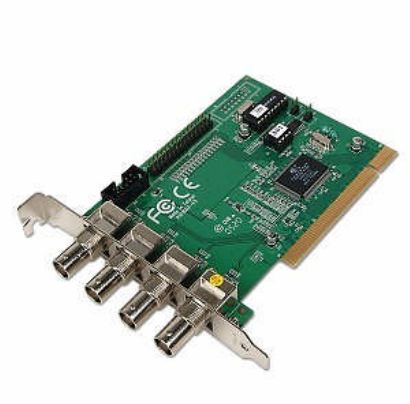


Fig. 2.14. Tarjeta receptora de video con 4 entradas RCA.

Estas tarjetas por lo general usan un chip, el BT8xx. Este CI por lo general es de bajo costo y proporciona una interface completa entre señales de video analógicas (NTSC/PAL/SECAM) y el bus PCI ya que incorpora un acondicionar de señales, un subsistema de muestreo y digitalización, y funciones del bus PCI. Además de gestionar directamente el bus, también puede realizar el control de la memoria de computador mediante DMA (Acceso directo a memoria).

No existen diferencias considerables en el hardware de distintos fabricantes que utilizan el mismo modelo de Bt8xx. Las diferencias más significativas pueden estar en el software y drives de las tarjetas. Estas tarjetas usualmente vienen con un software cerrado y sin posibilidad de obtener un entorno de desarrollo para aplicaciones basadas en la arquitectura de la tarjeta.

Se puede usar las DSK's del propio Bt8xx que es de dominio público para programar la tarjeta para tareas especificas (compresión de señales de video, decodificadores, captura de video, detección de movimiento mediante video, etc..). Para el desarrollo de las aplicaciones utilizando el SDK del Bt8xx es aconsejable emplear Visual C++ como entorno de programación, ya que Microsoft proporciona una clase AVICap que encapsula todos los detalles de streaming de video para la creación de archivos del tipo AVI.

2.1.3 Descripción en cuanto a sistemas de iluminación

Una de las consideraciones a tener en cuenta cuando se desarrolla un sistema de visión por computador es el ambiente de iluminación, muchas aplicaciones funcionan correctamente en el laboratorio sin embargo cuando se instalan en la línea de producción de una fabrica o de un lugar determinado no funciona porque se ven afectadas por la luz del ambiente. En estos casos se debe controlar el ambiente en el que se va a hacer la inspección, para una toma de fotografías es importante tener conceptos sobre el tema. Por ejemplo la iluminación es la luz que llega a los objetos y su unidad es el Lux. Un lux es equivalente a la luz de un lumen dentro en un metro cuadrado, donde lumen es el flujo luminoso.

Para una buena iluminación de un rostro, es necesario que se cumplan lo que denominamos la regla de las tres luces o tres proyectores de luz, tal como se muestra en la Fig. 2.15

Fig. 2.15. *Iluminación de un rostro mediante la regla de las tres luces.*

Para tomar las fotos se deben seguir ciertos pasos. Ajustar el diafragma automático para luego ajustar manualmente. Colocar el proyector principal de forma que

la sombra de la nariz se confunda con la de la mejilla así se consigue el triángulo en la mejilla, y el ojo iluminado. Colocar la luz de relleno para aclarar las sombras, pero sin hacerlas desaparecer. El tercer proyector o contraluz de techo sirve para "despegar" la forma del fondo. Si sólo pusiéramos dos luces, se notaría un efecto del rostro "plano" o en dos dimensiones. Con el tercer foco se consigue el efecto de fondo

Para las aplicaciones industriales es importante que los fluorescentes funcionen a alta frecuencia al menos a 25kHz. En aplicaciones de visión no pueden utilizarse fluorescente estándar debido a su efecto de parpadeo, a menos la cámara pueda funcionar esta frecuencia y cancele este efecto. Los tubos fluorescentes estándar no presentan un balance de color uniforme, incorporando longitudes de onda mayoritariamente azules. Para aplicaciones de visión artificial es necesario utilizar fluorescentes con espectro conocido. Así es habitual utilizar en según que aplicaciones, fluorescentes casi monocromáticos: ultravioletas, amarillos, verdes, azules.

Para aplicaciones donde se requiere una gran intensidad de iluminación y una larga longitud, se utilizan fluorescentes de apertura, en estos fluorescentes la luz se emite solo en una dirección y con un ángulo muy estrecho, esto permite que la intensidad lumínica pueda ser hasta 10 veces superior a la de un fluorescente estándar. La vida media de los tubos fluorescentes es algo superior a las 10000 horas.

2.1.4 Descripción en cuanto a motores paso a paso (p-p)

En cuanto a motores paso a paso, existen dos tipos de motores paso a paso de imán permanente, los unipolares y los bipolares. En los unipolares todas las bobinas de estator están conectadas en serie formando cuatro grupos, tal como se ve en la Fig. 216a. En los bipolares las bobinas del estator se conectan en serie formando solamente dos grupos, como se ve en la Fig. 2.16b.

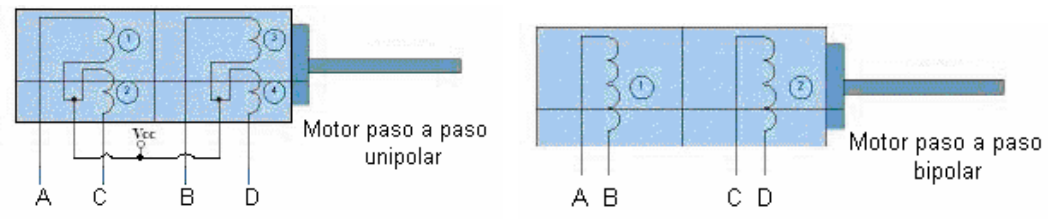


Fig. 2.16. a. *Motor paso a paso unipolar* **b**. *Motor paso a paso bipolar*

La característica principal de estos motores es de poder mover un paso a la vez por cada pulso que se le aplique. Se verá en más detalle los motores de imán permanente unipolares, ya que estas se utilizan en proyecto para mover una cámara.

Fig. 2.17. *Motores p-p Unipolares*

Estos motores p-p unipolares se diferencian porque tienen 5 o 6 salidas y se caracterizan por la cantidad de pasos que da en una vuelta, por ejemplo 48 pasos por vuelta equivalente a rotar 7.5º. Por lo general la tensión de alimentación de estos motores es de 9 a 12V y una corriente de consumo de 200 mA.

2.2 Bases de datos de rostros

Para obtener la base de datos de imágenes de rostros se busco en la red de Internet las bases de datos de imágenes de rostros utilizadas en los certámenes que realizan instituciones para reconocimiento de rostro, esto con el fin de probar nuestro algoritmo a niveles superiores. También se realizo una propia base de datos de rostros con los estudiantes del centro de Investigación y Desarrollo de la Facultad de Ingeniería Eléctrica y Electrónica de la Universidad Nacional de Ingeniería CIDFIEE-UNI, a continuación se describen estas bases de datos

2.2.1 Descripción en cuanto a la adquisición de una base de datos de rostros

Una base de datos de imágenes de rostros debe tener características especiales de manera que el sistema a probar sea robusto. Una manera de realizar esto es recurriendo a los conjuntos de datos de imágenes de rostros hechos por desarrolladores dedicados al reconocimiento de rostro.

Estos desarrolladores realizan las bases de datos de acuerdo a los criterios siguientes: gestos o expresiones de rostros, iluminación, envejecimiento entre otros.

La base de datos de la universidad de Yale [10] es la más apropiada y utilizada para estos casos. Esta base de datos contiene 165 imágenes en formato GIF de 15 rostros de personas, estas imágenes están en escala de grises y tienen una resolución

de 112x92 píxeles. Estas 165 imágenes, se dividen en grupos de 11 imágenes cada una con las siguientes características: iluminación central, con lentes oscuros, alegres, iluminación izquierda, sin lentes oscuros, normal, iluminación derecha, tristes, durmiendo, deprimidos y haciendo guiños entre otros.

La base de datos de FERET [11] (The Facial Recognition Technology Database) es la más grande base de datos de imágenes de rostros actualmente y es utilizado por los diferentes desarrolladores para testear sus algoritmos. Esta base de datos contiene 1546 conjuntos de imágenes para un total de 14,126 imágenes. Todos tomados bajo diferentes tipos de iluminación, gestos y algunos fueron tomados diariamente durante cierto tiempo. Existen otras bases de datos de imágenes [12] entre ellas AT&T, MIT, UMIS, etc...

Fig. 2.18 *Fotos de la base de datos de Yale [10]*

2.2.2 Creación de una base de datos de rostros

Realizamos una base de datos propio para nuestro interés de comprobar el sistema, para ello tomamos algunas consideraciones como por ejemplo que los rostros deberán tener diferentes gestos, iluminación y posición. Estos gestos tendrían las expresiones siguientes: Rostro alegre, la persona tendrá que sonreír naturalmente. Rostro triste, la persona tendrá que mostrar su rostro con gestos de tristeza. Rostro enojado, la persona tendrá que fungir el estrenjo naturalmente. Finalmente el rostro normal o serio, la persona deberá mostrar su rostro naturalmente.

Las tomas de las fotos de rostros se realizaran con direcciones diferentes, es decir las personas movieron su rostro de la siguiente manera:

- Mirada de frente, la persona deberá mirar directamente a la cámara.

- Mirar de costado, la persona moverá su cabeza 15º u observar el hombro del fotógrafo (izquierda y derecha)
- Mirar abajo y arriba, la persona moverá su cabeza 15° u observar el pecho y la parte superior de la cabeza del fotógrafo.

La cantidad de fotos a tomar es de 15 a 20 por persona, las fotos serán tomadas en todas las direcciones y en rostro alegre (5) y triste o normal (5), dejando a la persona un gesto particular (5) y otros (5).

Fig. 2.16. *Fotos de diferentes estudiantes de CIDFIEE-UNI*

Fig. 2.17 *Fotos de 1 persona con gestos diferentes*

Estas imágenes fueron tomadas a los estudiantes de la Facultad de Ingeniería Eléctrica y Electrónica de la Universidad Nacional de Ingeniería.

Se tomo 23 grupos de 20 imágenes cada una, haciendo un total de 460 imágenes de rostros. Las imágenes fueron tomadas mediante una cámara web de resolución de 640x480 con iluminación directa y guardada en formato bmp y jpg.

Las imágenes de rostros totales se encuentran en el anexo 1.

CAPITULO III
PRE PROCESAMIENTO DE LA IMAGEN

3.1 Introducción

Para el reconocedor de rostros la mejora de un rostro luego de su adquisición es importante. Existen métodos en el dominio espacial y en el dominio de la frecuencia que mejoran sustancialmente la imagen de rostro.

La imagen recibida al adquirir el rostro viene sin procesar y solo se confía del sensor y la iluminación a la que se ha adquirido la imagen. El formato al que recibimos la imagen es BMP sin comprimir y necesitamos llevar a un formato que se vea en escala de grises.

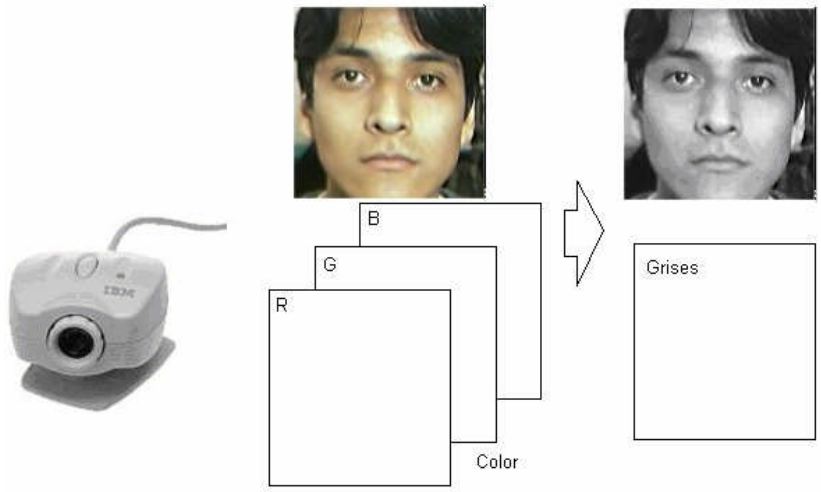


Figura 3.1. *Conversión de formato*

Luego del cambio de formato, se necesita que la imagen de rostro sea mejorada. El método que se va a utilizar se sitúa en el dominio espacial. El dominio espacial se refiere al propio plano de la imagen y las técnicas de esta categoría se basan en la manipulación directa de los píxeles de la misma imagen.

3.2 Conversión de formato

En la Fig. 3.1 se muestra un bosquejo de la conversión de RGB a escala de grises, en realidad se aplica un cambio del modelo de color RGB al modelo YIQ o YUV (creados para transmisión de video TV analógica). YIQ en el estándar americano (NTSC) y YUV en el europeo (PAL); O al modelo HSI. Estos modelos tienen en común una componente que es la llamada luminancia o brillo ("Y" en YIQ o YUV, y "I" en HSI).

El ojo humano es mucho más sensible al brillo que al color por lo que el canal de luminancia es el más prioritario (necesita más resolución). La ecuación (8) muestra la transformación del modelo RGB al modelo YIQ.

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad (3.1)$$

Solo nos interesa el canal de la luminancia Y. Esta formula fue obtenida experimentalmente de acuerdo a muchos ensayos y estudios de color. Este modelo YIQ fue diseñado teniendo en cuenta las características del sistema visual humano, en particular la mayor sensibilidad a los cambios en luminancia que a los cambios de matiz o saturación. Así pues, este estándar usa más bits (o ancho de banda) para representar Y, menos para I y Q. Así la importancia de este desacoplamiento radica en que la componente de la luminancia de una imagen puede procesarse sin afectar a su contenido cromático. Por ejemplo en la Fig. 3.2 se muestra una aplicación, con la ecualización de un rostro.

Fig. 3.2 *a) Imagen Original; b) Componente de luminancia c) Imagen ecualizada en el modelo YIQ.*

3.3 Mejora de la imagen

Los métodos en el dominio espacial son procedimientos que operan directamente con los píxeles. Las funciones de procesamiento de la imagen en el dominio espacial pueden expresarse con:

$$g(x, y) = T[f(x, y)] \qquad (3.2)$$

Donde $f(x, y)$ es la imagen de entrada, $g(x, y)$ es la imagen procesada y T es un operador que actúa sobre f, definido en algún entorno de (x,y). Además T puede operar sobre un conjunto de imagen.

Se emplea un área cuadrada centrada en (x,y), como una aproximación principal para definir un entorno alrededor de (x,y), como se muestra en la Fig. 3.3

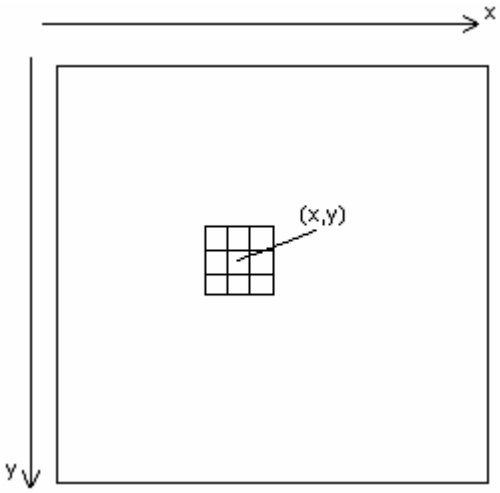


Fig. 3.3 *Representación de un píxel en el plano*

Una función de transformación del nivel de gris (También denominada correspondencia) es de la forma:

$$s = T(r) \qquad (3.3)$$

Donde r y s son variables que indican el nivel de gris de f(x,y) y g(x,y) en el punto (x,y). Esto es utilizado para el aumento de contraste también llamadas procesamiento de punto.

En entornos mayores, una de las aproximaciones principales en este tipo de formulación se basa en el empleo de las denominadas mascaras (también llamadas plantillas, ventanas o filtros). Básicamente, una mascara es una pequeña distribución bidimensional (por ejemplo: 3x3). Las técnicas de mejora basadas en este tipo de aproximación se conocen a menudo como procesamiento por mascaras o filtrado.

Tener bajo contraste en las imágenes son debidas a diferentes causas, como iluminación deficiente, falta de rango dinámico en el censor o incluso incorrecta selección de la apertura de la lente durante la captación de la imagen.

La idea subyacente en las técnicas de aumento de contraste consiste en incrementar el rango dinámico de los niveles de gris de la imagen que se esta procesando.

El histograma de una imagen digital con niveles de gris en el rango [0,L-1] es una función discreta $p(r_k) = n_k/n$, donde r_k es el k-esimo nivel de gris, n_k es el número de píxeles de la imagen con este nivel de gris, n es el número total de píxeles de la imagen con ese nivel de gris, n es el número total de píxeles de la imagen y k=0,1,2,...,L-1.

La representación grafica de la función antes mencionada para todos los valores de k que proporciona una descripción global de la apariencia de la imagen. Por ejemplo en la Fig. 3.4 muestra que los niveles de gris están concentrados hacia el extremo oscuro del rango de la escala de gris. Así este histograma corresponde a una imagen con una apariencia global oscura. Sucede justo lo contrario en el caso de la Fig.3.5.

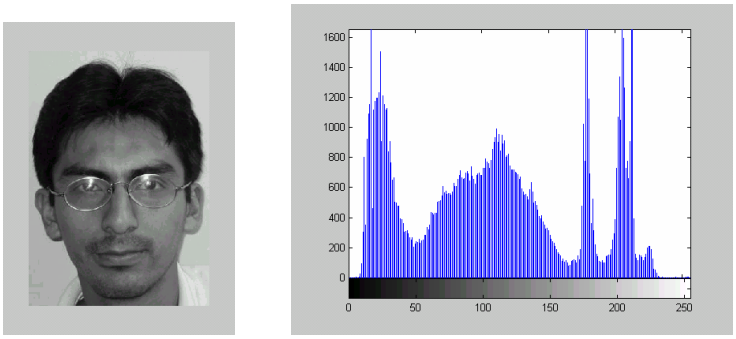


Fig. 3.4 a) Imagen b) Histograma de la imagen anterior

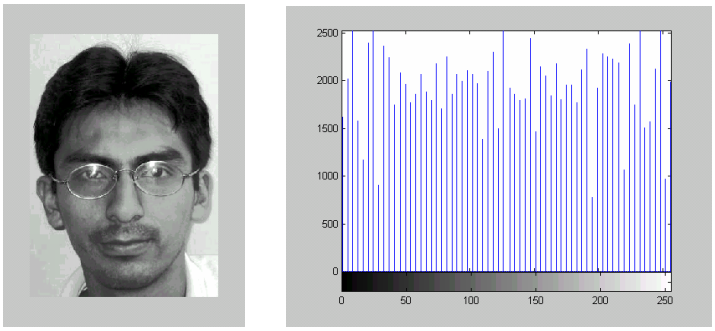


Fig. 3.5. a) Imagen b) Histograma de la imagen anterior

Si r es una variable que representa los niveles de gris de la imagen a mejorar. En la parte inicial de la representación se supondrá que los píxeles son cantidades continuas que han sido normalizadas de forma que pertenezcan al intervalo [0, 1], con r = 0 representando el negro y r=1 representando el blanco.

$$s = T(r) \qquad (3.4)$$

Para cada valor de r de este intervalo [0, 1], produce un nivel s para cada valor del píxel r de la imagen original. Esta transformación cumple con las siguientes condiciones:

T(r) es el valor único y monótamente creciente en el intervalo $0 \leq r \leq 1$; y $0 \leq T(r) \leq 1$ para $0 \leq r \leq 1$

Esta condición preserva el orden entre el blanco y negro de la escala de grises y garantiza una aplicación que es coherente con el rango de valores de píxeles permitidos.

La técnica empleada para obtener un histograma uniforme se conoce como ecualización del histograma o linealización del histograma. La ventaja de la ecualización del histograma frente a las técnicas de manipulación manual del contraste es que la primera resulta completamente automática.

CAPITULO IV

LOCALIZACIÓN Y SEGUIMIENTO DE UN ROSTRO EN UNA IMAGEN DE VIDEO

4.1 Introducción.

El diseño de una interfase para la localización de un rostro debe ser rápido y eficiente además de ser capaz de buscar la imagen en tiempo real mientras se realicen otros procesos.

Se desarrolla un detector de rostros donde el rostro debe ser representado con vista frontal con cierta variación de inclinación, además de una buena iluminación. El detector de rostros es uno de los problemas en procesamiento de imágenes ya que este es el primer paso hacia diferentes aplicaciones como codificación del rostro, reconocimiento de rostros, reconocimiento de género, rastreo del rostro, etc.

Existen métodos para la localización de un rostro en la imagen como el rastreo por contornos, métodos estadísticos o convolución de imágenes como detectores de rostros. El problema de algunos de estos casos es que son computacionalmente ineficientes en tiempo de ejecución. El primer método que se va a utilizar es la localización del rostro por medio del color y su distribución estadística. Este algoritmo usa una técnica para desarrollar gradientes de densidad y encontrar el modo de las distribuciones de probabilidad. En nuestro caso, nosotros queremos encontrar la distribución de color dentro de una escena de video.

El segundo método se basa en “cascade of boosted classifiers working with haar-like features” que es un clasificador de búsqueda en cascada que trabaja con pequeñas características propuesto por Paúl Viola y Michael Jones [14].

El método por color dio resultados de detección al 82% que fue medido en situaciones donde no hay rostros y donde los rostros fueron ligeramente tapados (lentes, gorras, barba). El método basado en “boosted” tuvo un rendimiento mejor en situaciones anteriores de 95%.

4.2 Localización de rostros por color

La Fig. 4.1 se muestra el esquema que se utilizará para ubicar el rostro por medio del color.

Para encontrar un algoritmo simple y rápido para el rastreo básico, nosotros nos hemos enfocado en el modelo de color HSV, donde H es el valor del tono de una imagen. Basado en este rastreo el tono de la carne se encuentra en una determinada región del plano H (10 a 30 aprox. de 256 valores) a comparación del modelo RGB donde esto no se puede distinguir fácilmente, ya que se necesitaría usar los tres planos RGB

Para encontrar un algoritmo simple y rápido para el rastreo básico, nosotros nos hemos enfocado en el modelo de color HSV, donde H es el valor del tono de una imagen. Basado en este rastreo el tono de la carne se encuentra en una determinada región del plano H (10 a 30 aprox. de 256 valores) mostrada en la Figura 38 a comparación del modelo RGB donde esto no se puede distinguir fácilmente, ya que se necesitaría usar los tres planos RGB.

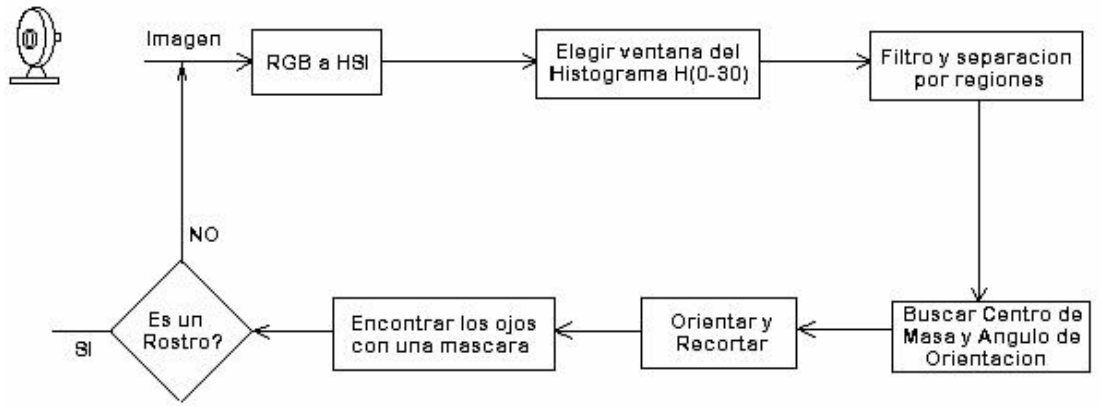


Fig. 4.1 *Esquema a emplear para la localización de un rostro por color*

La imagen RGB es convertida al modelo HSV (Tono, Saturación y Valor) y la distribución de probabilidad de color (Plano H) es mostrada por medio del histograma. Este plano H nos indica el color, por tanto es más fácil separar los colores, una equivocación común es decir que existan diferentes modelos de color para razas diferentes de personas, por ejemplo, para persona de piel blanca o negra, pues esto no es cierto excepto para albinos, ya que los histogramas muestran que el color de la piel tienden a ser igual para una variedad amplia de personas. Entonces se puede decir que todos los humanos tienen la piel del mismo color (matiz). La piel oscura simplemente es porque tiene mayor saturación.

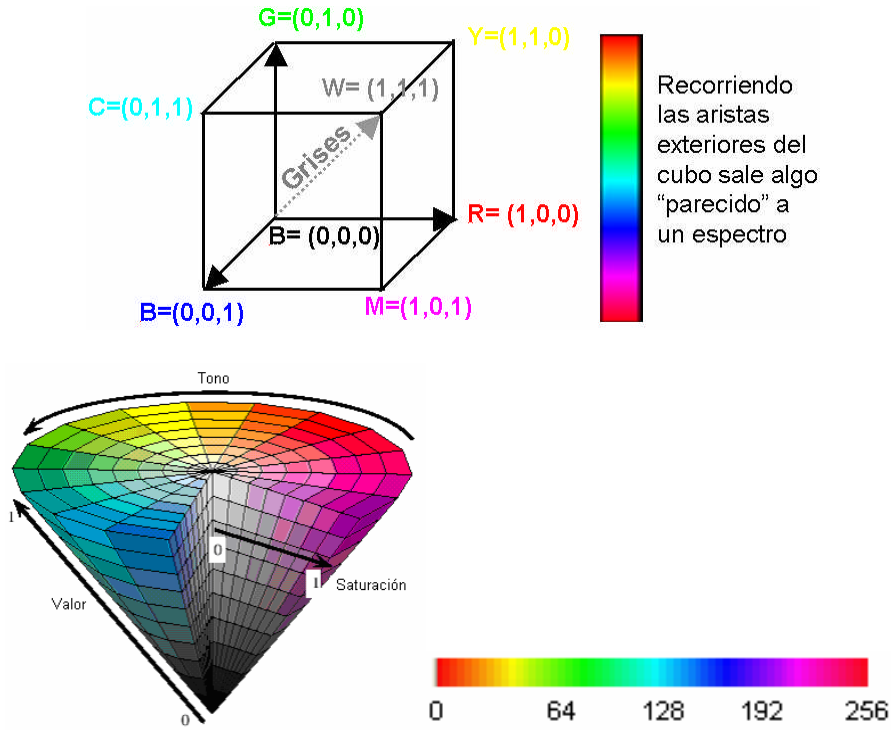


Fig. 4.2. a. *Modelo RGB* **b.** *Modelo HSI*

Luego de llevar al modelo HSV y observar el histograma del plano H, se deberá escoger una ventana (porción entre 0 y 30 de los valores del histograma del plano H) de tal manera que en esta se encuentre el color de la piel. Así la imagen resultante será el rostro y otros detalles que tengan el mismo color de la piel (mano, pared de color piel, etc.).

Fig. 4.3 a. *Imagen Original I en BMP* **b.** *Imagen binarizada del plano H de I*

Lo siguiente es binarizar la imagen y mediante un filtro morfológico se elimina los puntos no deseados.

Luego separamos las regiones halladas y tomamos la región de la imagen con mayor área.

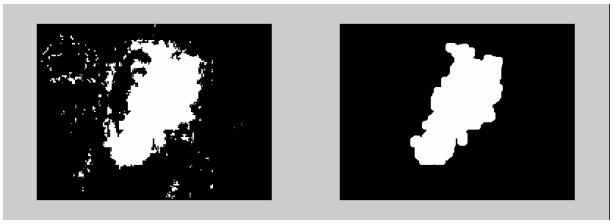


Fig. 4.4. **a.** *Imagen dilatada* **b.** *Imagen* $I(x, y)$ *luego del filtro morfológico.*

A esta imagen $I(x, y)$ le buscaremos el momento cero o la suma de todos los valores.

$$M_{00} = \sum_x \sum_y I(x, y) \tag{4.1}$$

Luego calculamos el primer momento de x e y

$$M_{10} = \sum_x \sum_y xI(x, y) \qquad M_{01} = \sum_x \sum_y yI(x, y) \tag{4.2}$$

Finalmente se busca el centro de masa de la distribución de puntos en la imagen (Centroide).

$$x_c = \frac{M_{10}}{M_{00}}; \qquad y_c = \frac{M_{01}}{M_{00}}; \tag{4.3}$$

Donde I(x,y) es el valor de la posición (x,y) en la imagen del plano H modificado y X_c e Y_c es el centro de la distribución de la imagen *I*.

Para calcular la orientación de la imagen *I*, se puede utilizar los segundos momentos de la distribución de la imagen *I*.

$$M_{20} = \sum_x \sum_y x^2 I(x, y); \qquad M_{02} = \sum_x \sum_y y^2 I(x, y); \tag{4.4}$$

Luego la orientación le da el que tiene mayor eje, este ángulo se calcula de acuerdo a la ecuación 4.5.

$$\theta = \frac{\arctan\left(\dfrac{2\left(\dfrac{M_{11}}{M_{00}} - x_c y_c\right)}{\left(\dfrac{M_{20}}{M_{00}} - x_c^2\right) - \left(\dfrac{M_{02}}{M_{00}} - y_c^2\right)}\right)}{2} \tag{4.5}$$

También el ancho y largo de la distribución es importante, para su cálculo se procede de la siguiente manera:

$$a = \frac{M_{20}}{M_{00}} - x_c^2, \quad b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right), \quad c = \frac{M_{02}}{M_{00}} - y_c^2, \tag{4.6}$$

El largo l y el ancho w se calcula como:

$$l = \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}}, \tag{4.7}$$

$$w = \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}}, \tag{4.8}$$

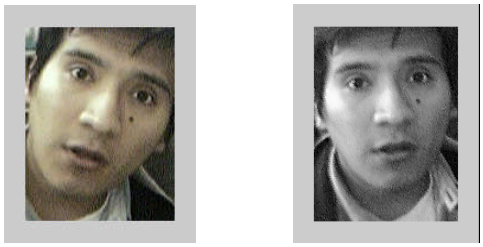


Fig. 4.5 a. *Rostro orientado 15º aprox.* **b.** *Rostro en posición normal*

La localización de los ojos, se realiza aplicando una matriz de máscara, esta matriz de máscara se pasa por cada píxel de la imagen del rostro y de acuerdo a una puntuación, que se obtiene como la suma de los productos (elemento a elemento) de la matriz máscara y una parte de la imagen total (del rostro), de esta manera se obtiene puntuaciones diferentes. Cuando la parte de la imagen coincide con la máscara se obtienen puntuaciones grandes, y en caso contrario puntuaciones pequeñas o negativas.

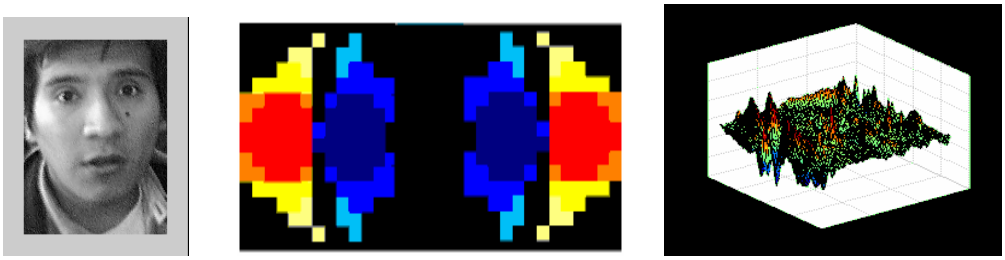


Fig. 4.6 a. *Rostro corregido* **b.** *Mascara del ojo* **c.** *Resultado de la convolución*

Acerca del tamaño de la máscara, esta debe poseer un tamaño similar del punto que se desea buscar, es decir para encontrar los ojos, se tendrá que crear una máscara con el tamaño aproximado de los ojos, ver Fig. (4.6). Esta mascara tiene que valer para distintas caras independientemente de la distancia a la que se encuentre la cara, es decir la máscara también tendrá que cambiar de tamaño.

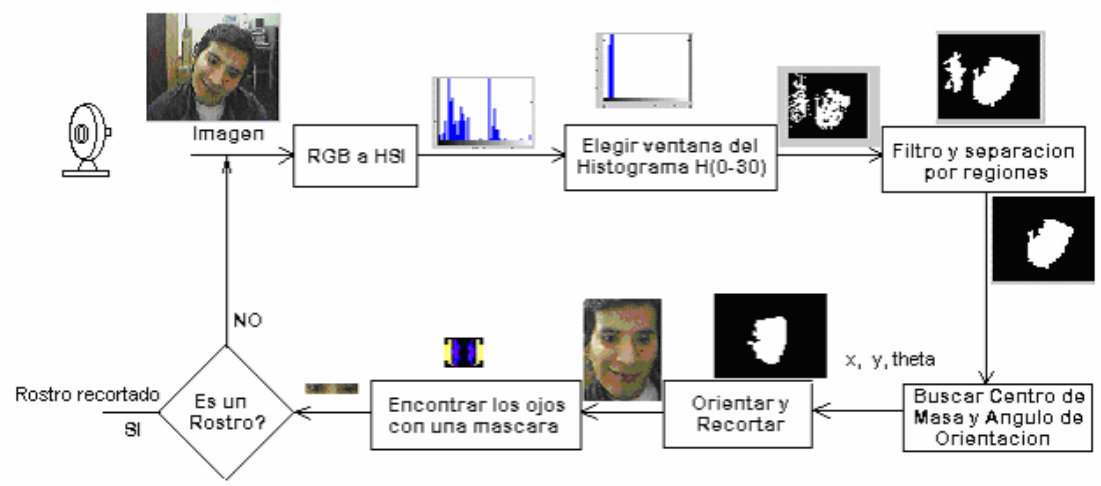


Fig. 4.7. *Resultados para la localización de un rostro por color.*

4.3 Localización de rostros por clasificadores tipo Boosting.

Uno de los sistemas más exitosos para detección de objetos en tiempo real es usando el método Boosted Cascade of Simple Features [14].

Este sistema de detección de rostros es diferente de los demás por la capacidad de detectar rostros en forma rápida. Según estudios realizados en imágenes de 384x288 píxeles, los rostros son detectados a 15 cuadros por segundo trabajando en procesadores Pentium III de 700Mhz. En otros sistemas de detección de rostros se necesita información adicional, como diferencia de imágenes en una secuencia de video, o el color de los píxeles en imágenes a color, estas tienen que ser utilizados para lograr una alta tasa de cuadros. El sistema presentado logra una alta tasa de cuadros trabajando solamente con información presente.

Este método sirve para la detección visual de objetos en general, la ventaja de este método es la rapidez al procesar una imagen y detectar rostros con baja alarma. En efecto, Paúl Viola y Michael Jones demuestran en un paper “Rapid Object Detection using boosted cascade of simple feature” [14] donde arroja un 95% de eficiencia y una falla de 14084 y una velocidad de 15 veces de la implementación de una red neuronal de en sistemas de detección construido por Rowley.

Este método tiene 4 partes:

- El primera parte es obtener un conjunto de mascaras que sean similares para la etapa de haar filter (filtro haar).
- La segunda parte es una representación de la imagen llamada “Imagen Integral” que es usado por el detector para computar rápidamente.
- La tercera parte es un algoritmo de entrenamiento basado en “Adaboost” que selecciona un número pequeño de características desde un conjunto grande y produce un efectivo clasificador.
- Por ultimo el método para combinar progresivamente más clasificadores complejos en una estructura en cascada que permite realizar menos operaciones sobre regiones alentadoras donde se encuentre el rostro.

Otro interesante aspecto de esta técnica es que puede ser usada para crear un detector de objetos que tenga una sola apariencia. Existen dos razones principales para la utilización de características en vez de usar los píxeles directamente.

La primera razón es que estas características pueden ser usadas para codificar áreas conocidas mejor que los píxeles y la otra razón es que el sistema basado en características es mucho más rápido que un sistema basado en píxeles. Se hará uso de diferentes tipos de características, estas características se dividen en varios tipos tal como muestra la Fig. 4.8.

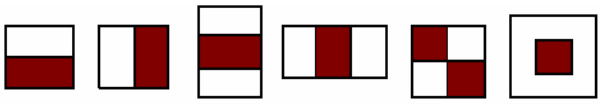


Fig. 4.8. *Características diferentes*

Si solo tomamos 3 tipos de características que son: característica de 2 rectángulos (vertical y horizontal) y característica de 4 rectángulos. El valor de una característica es la suma de todos los píxeles dentro del rectángulo en blanco menos la suma de todos los píxeles dentro del rectángulo en negro. Dado que la resolución base del detector es 24x24, el número total de características es igual a 162337.

4.3.1 Extracción de características: imagen Integral

Como se vio en el paso previo, necesitamos un método para extraer un conjunto de características grandes. Además se debe computar estas características rápidamente y en muchas escalas. La imagen integral con localización (x,y) contiene la suma de todos los píxeles de arriba e izquierda de (x,y).

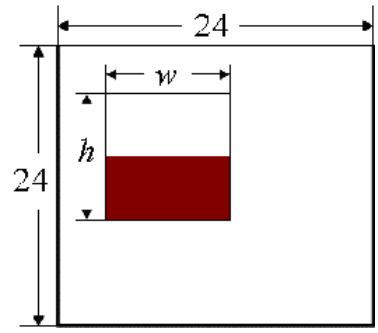


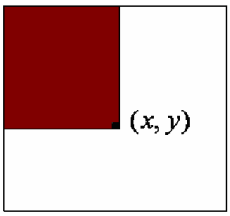
Fig. 4.9. *Característica dentro de la sub-ventana de 24x24 píxeles*

$$ii_{(x,y)} = \sum_{x'<x,y'<y} i(x', y') \tag{4.9}$$

Donde $ii_{(x,y)}$ es la imagen integral y $i_{(x,y)}$ es la imagen original. Nosotros podemos computar esto efectivamente usando las siguientes pares de recurrencias:

$$S(x, y) = S(x, y) + i(x, y) \tag{4.10}$$

$$ii(x, y) = ii(x, y) + S(x, y) \tag{4.11}$$



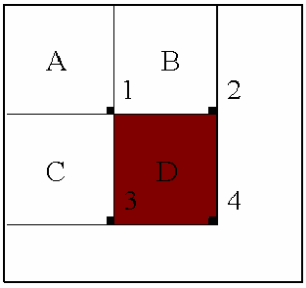


Fig. 4.10. *La suma del los píxeles del rectángulo D computado por la suma integral*

Donde $S(x, y)$ es la fila acumulativa sumada y $S(x, -1) = ii(-1, y) = 0$. La imagen integral puede ser computada en un paso encima de la imagen original. Ahora vamos a considerar todas las características que nosotros usamos. Esto es obvio, ya que ello siempre involucra computar la suma de todos los píxeles en rectángulos y esto puede ser echo fuera de la imagen, considerando solo cuatro puntos de esta suma integral. Tal como se muestra en la Fig. 46. El valor de la suma integral de la localización 1 es la suma de los píxeles del rectángulo A, el valor de la localización 2 es A+B, el valor de la localización 3 es A+C, el valor de la localización 4 es A+B+C+D. La suma en D se puede calcular como localización 4+1-(2+3).

4.3.2 Reducción de características

Para obtener un conjunto de muestras para el entrenamiento, podemos extraer una colección larga de características usando la idea del paso anterior. Nuestra hipótesis es la siguiente: un número muy pequeño de características pueden ser combinados para tener un clasificador efectivo. Esta idea proviene de P. Papegeorgiou, Michael Oren y Tomasa Poggi [16]. Ellos demuestran esto luego de seleccionar 37 características fuera del total de 1734 características, para crear un detector de rostro que produzca un buen resultado.

En este trabajo, se va a usar adaboost para seleccionar un conjunto pequeño de características y entrenar los clasificadores. El error de entrenamiento de un clasificador fuerte producido por el algoritmo Adaboost se acerca a exponencial cero a medida que se aumenta él número de rondas. Este algoritmo se usará para entrenar los clasificadores.

4.3.3 Clasificador Débil

Para cada característica, un clasificador débil es definido y debe ser incluido al conjunto de entrenamiento, para ello es necesario un umbral. Un umbral óptimo que separe las caras de las no caras se realiza en base a un histograma acumulativo. En la Fig. 4.11, θ es el umbral. El umbral óptimo será el punto que minimice $(1-c+nc)$. Donde "c" es la cantidad de caras y "nc" la cantidad de no caras. La paridad "P" indica la salida de caras que pueden estar encima o debajo del umbral. Así cada clasificador débil $h_j(x)$ tiene un valor f_j, un umbral θ_j y una paridad p_j. (Al buscar el umbral de cada clasificador débil se estará dando un paso para optimizar el algoritmo y así minimizar el error de clasificación).

Para la implementación, la salida de nuestro clasificador débil es 1 si la imagen x es clasificada como una cara, y tendrá el valor de –1 si es clasificado como no cara. Matemáticamente esto es equivalente a la ecuación 23.

$$h_{j(x)} = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ -1 & \text{en otro caso} \end{cases} \qquad (4.12)$$

El valor x es una sub-ventana de 24x24 píxeles de una ventana de una imagen.

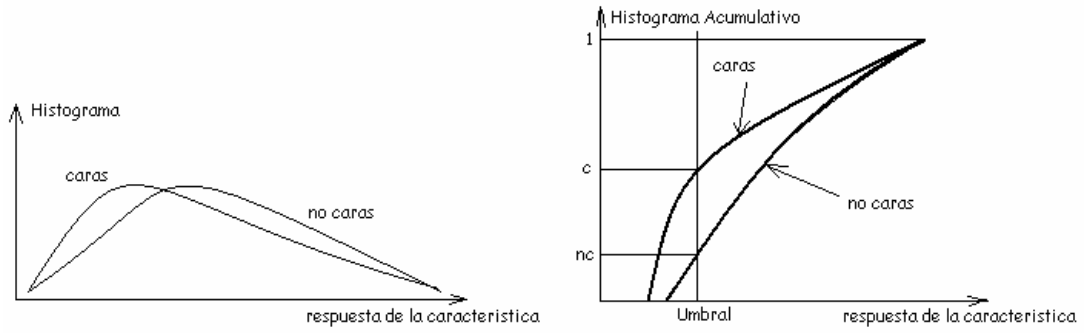


Fig. 4.11. *Histogramas del resultado del producto de los filtros con las sub-ventanas*

4.3.4 Algoritmo Adaboost

Un clasificador débil será creado para cada característica y para determinar el umbral óptimo se usará una función de clasificación. Un clasificador débil $h_j(x)$ es el resultado de la ecuación signo mostrado en la ecuación 4.13. Donde, θ_j y p_j es la característica, umbral y polaridad respectivamente.

$$h_j(x) = 1 \qquad \textit{Sí} \quad p_j f_j(x) < p_j \theta_j \tag{4.13}$$

$$= 0 \qquad \textit{otro caso}$$

El proceso es la siguiente:

- Dado las imágenes de entrenamiento $(x_1, y_1),...(x_n, y_n)$ donde $y_i = 0$ para caras o $y_i = 1$ para las no caras.

- Inicializamos pesos: $W_{1,i} = \frac{1}{2m}$ o $W_{1,i} = \frac{1}{2l}$ para $y_i = 0$ o 1 respectivamente donde m y l son las no caras y caras respectivamente.
- Para t=1,... T

1. Normalizo los pesos:

$$W_{t,i} \leq \frac{W_{t,i}}{\sum_{j=1}^{n} W_{t,j}} \tag{4.14}$$

Donde W_t es una distribución de probabilidad.

2. Para cada característica j, un clasificador h_j esta restringido a usar una característica simple. El error es evaluado con respecto a W_j.

$$\varepsilon_i = \sum_i W_i \left| h_j(x_i) - y_i \right| \tag{4.15}$$

3. Cambiamos el clasificador, h_t con un bajo error ε_t

4. Actualizamos los pesos:

$$W_{t+1,i} = W_{t,i} \beta_t^{1-e_i} \tag{4.16}$$

Donde $e_i = 0$ si x_i es un clasificador correcto y $e_i = 1$ caso contrario.

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} \tag{4.17}$$

- El clasificador final fuerte es:

$$h(x) = 1 \qquad \text{si} \quad \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \tag{4.18}$$

=0 en otro caso.

Donde:

$$\alpha_t = \log \frac{1}{\beta_t}$$

4.12 Entrenamiento del clasificador con rostros

En la Fig. 4.12 se puede ver que las características que se sobreponen en las imágenes, por ejemplo en la región de los ojos (ambos derecha e izquierda) también la región de la nariz o mejilla. La boca no se toma en consideración, esto debido al hecho de que la boca se distorsionada por las expresiones o gestos de las caras. Otra es la barba y el bigote.

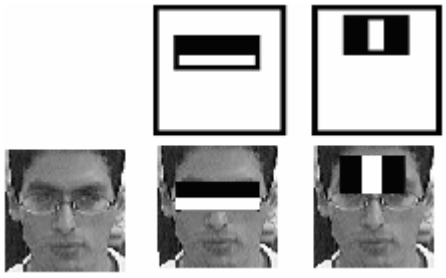


Fig. 4.12. *El primer y Segunda característica seleccionada por Adaboost*

La implementación se realizó con Matlab, teniendo en cuenta el algoritmo antes señalado. La obtención de características se realizo manualmente y la etapa de entrenamiento toma mucho tiempo en multiplicar todos las características por las sub-imágenes.

Se procedió a preparar una base de caras de rostros y no rostros. Se tomaron 250 rostros y 250 no rostros para el entrenamiento Estas fotos tienen una resolución de 24x24 píxeles. El formato utilizado fue bmp. La Fig. 49 muestra los rostros y los no rostros.

Fig. 4.13 *Rostro para el entrenamiento*

Estas imágenes lo cargamos de un archivo de imagen en formato BMP, para luego en Matlab guardarlo en formato mat, esto se realiza para poder trabajar fácilmente con Matlab.

Una vez obtenido las imágenes procedemos a normalizarla.

Estas imágenes normalizadas se guardan en un archivo de Matlab. Luego procedemos a obtener la suma integral de cada imagen de rostro.

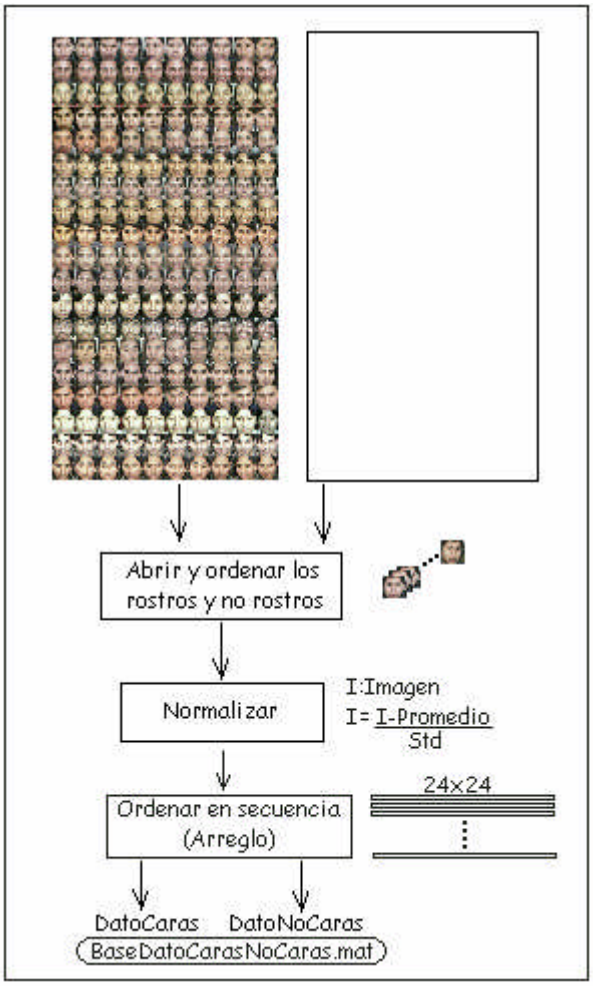


Fig. 4.14. *Normalización de las imágenes de entrenamiento*

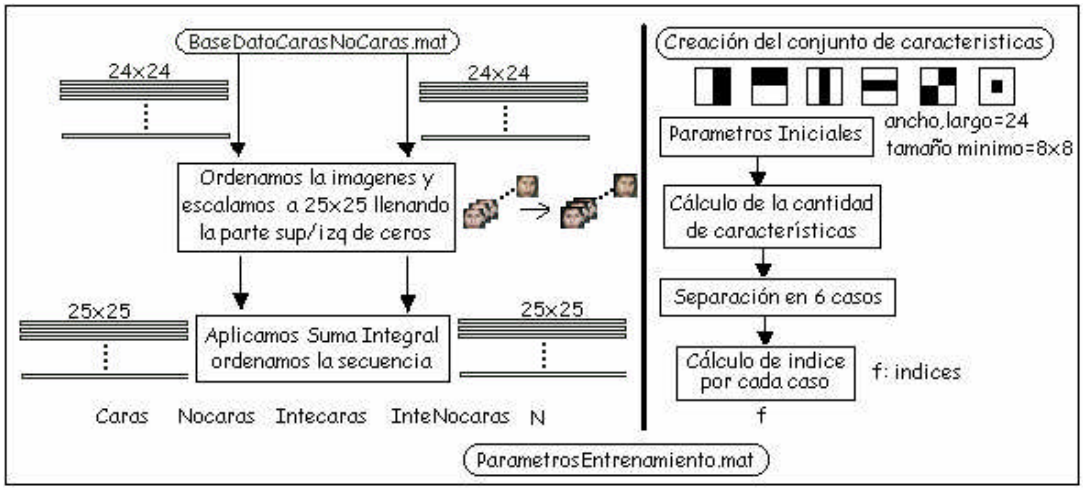


Fig. 4.15. *Parámetros de entrenamiento*

Ahora entramos a la etapa de entrenamiento, para ello utilizaremos los parámetros obtenidos anteriormente. La primera parte del entrenamiento se procede a

encontrar el umbral y la paridad que nos permitirá clasificar en primera instancia el rostro de los no rostros.

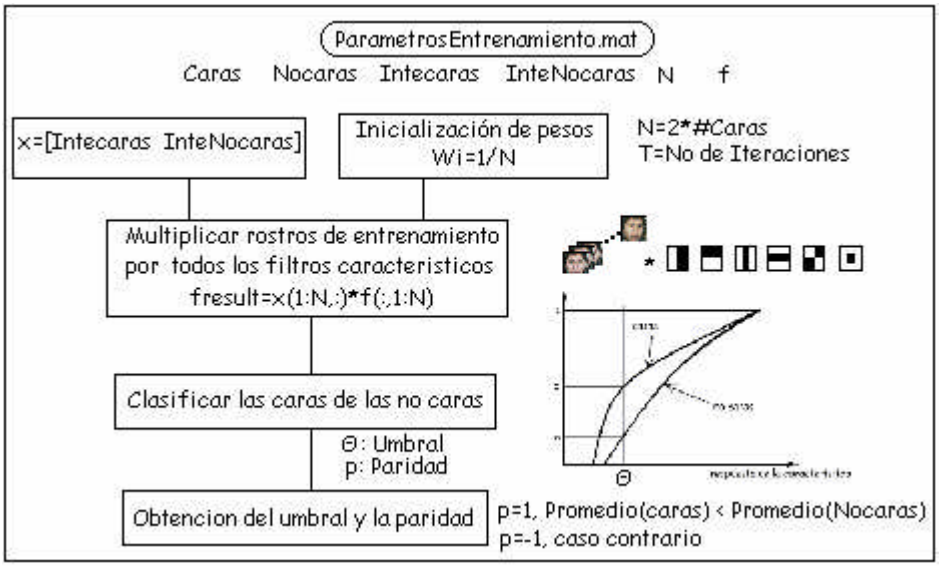


Fig. 4.16. *Umbral y paridad de la etapa de entrenamiento*

Luego de esta etapa antes del entrenamiento se procede a calcular el error de las interacciones. A partir de esto se procede a calcular el mínimo error para así poder calcular el umbral.

Una vez terminado el entrenamiento se pasa a la etapa de búsqueda del rostro, en la Fig. 4.17 se presenta al detalle la etapa de búsqueda del rostro.

4.4 Clasificación en cascada

Para aumentar la velocidad en la clasificación del rostro se crea una estructura en cascada de tal manera que el clasificador ubique el rostro mediante un umbral.

Cascada:

$Stage_1$:

$Clasificador_{11}$:

$Característica_{11}$

$Clasificador_{12}$:

$Característica_{12}$

...

$Stage_2$:

$Clasificador_{21}$:

$Característica_{21}$

...

...

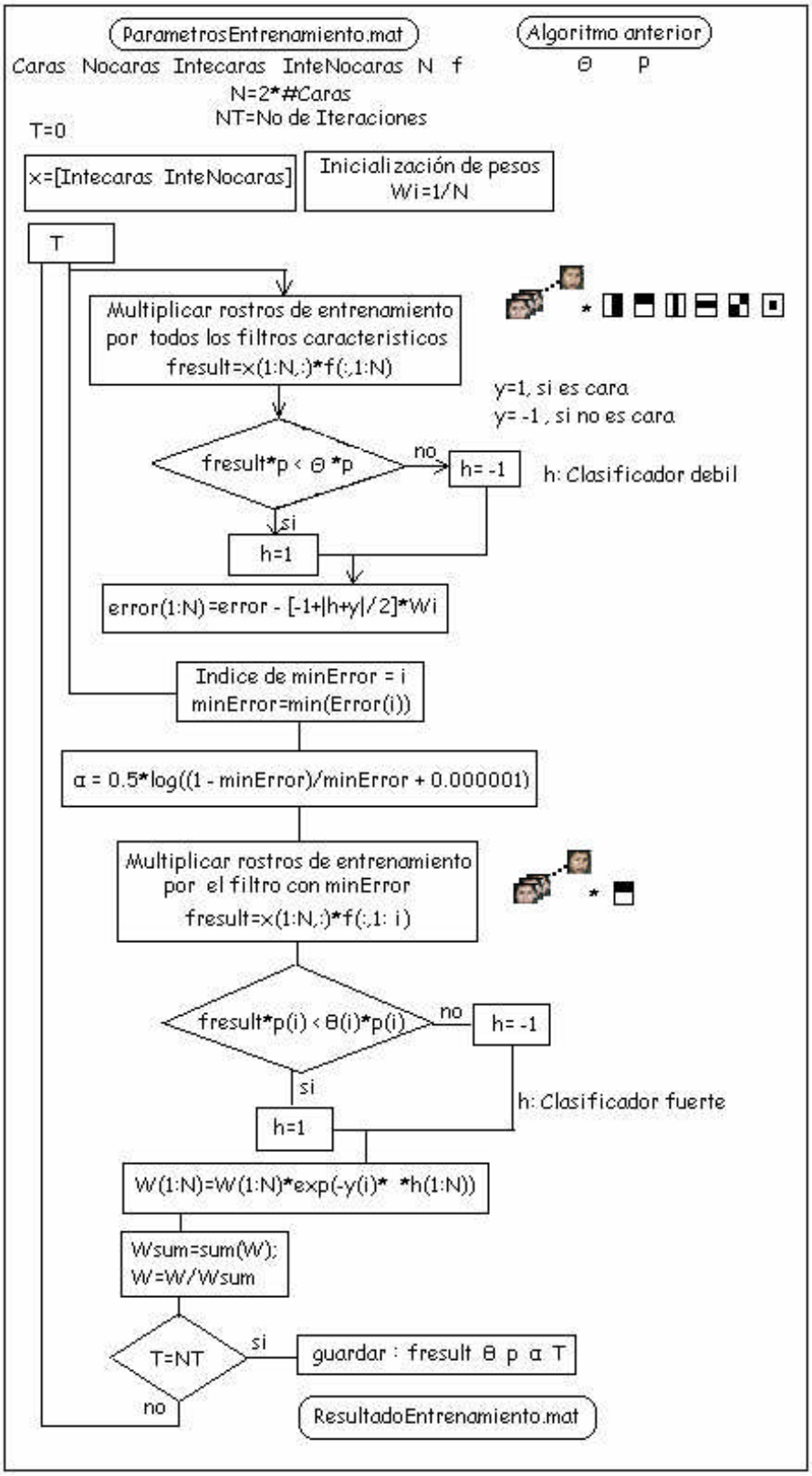


Fig. 4.17. *Entrenamiento del sistema*

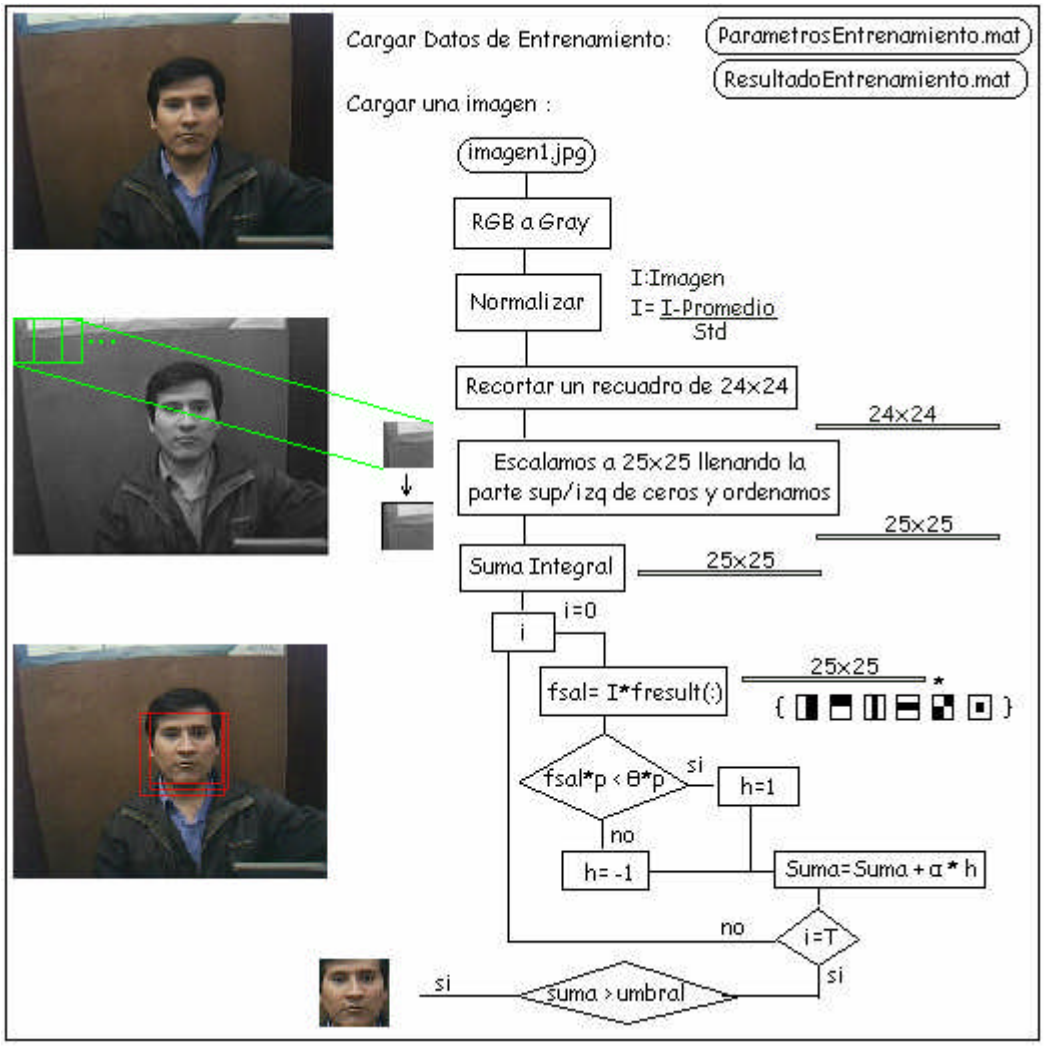


Fig. 4.18. *Diagrama de búsqueda de rostros.*

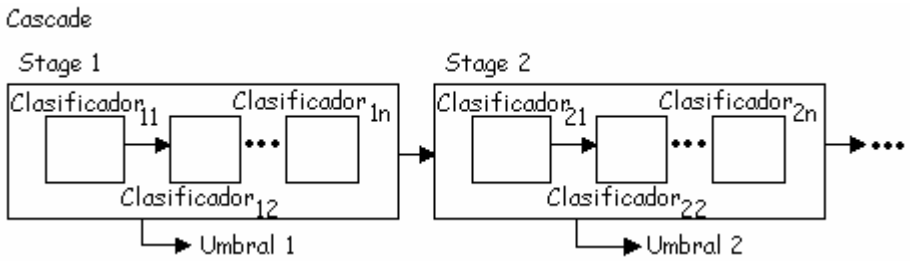


Fig. 4.19 *Clasificador en cascada*

4.6 Eficiencia de la localización de rostros

El error obtenido se calcula como la cantidad de rostros en la que la suma es mayor que cero, respecto a la cantidad de personas que se entrena. Este error se reduce al usar más características, esto se puede observar en la Fig. 4.15

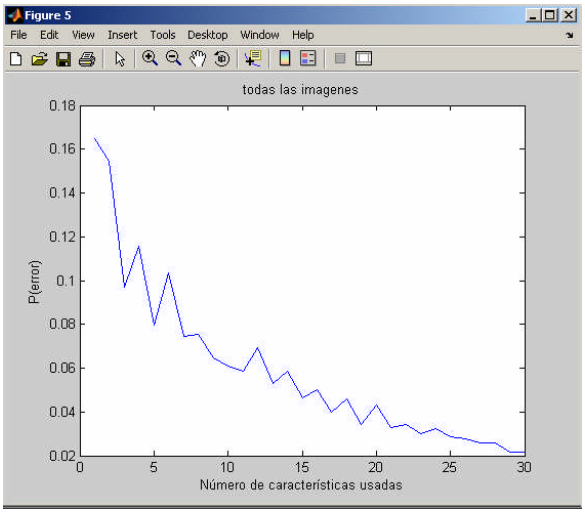


Fig. 4.20 *Porcentaje de error respecto a la cantidad de rostros*.

La forma de medir la eficiencia del clasificador es calculando la tasa de falso negativos “false negative”.

El proceso de entrenamiento con clasificadores es simple y eficiente, y puede ser utilizado como un “supervisor” que detecta la región de interés en una imagen. El termino supervisar se refiere a que el operador es entrenado para detectar objetos de una clase particular. En el dominio de la detección de falso positivo es posible obtener menos de 1% de falsos negativos y 40% de falsos positivos que usando un clasificador evaluado en 20 simples operaciones (aproximadamente 60 instrucciones del microprocesador). El efecto de este filtro es reducir más de la mitad el número de ubicaciones donde el detector debería ser evaluado.

Para la detección de rostros en cascada se empleo tiene 32 clasificadores, los cuales hacen un total de más de 80,000 operaciones. No obstante la estructura en cascada resulta en promedio rápido para la detección. La extrema rapidez para la detección de rostros tiene amplias aplicaciones prácticas. Estas incluyen interfaces de usuario, base de datos de imágenes y hasta en teleconferencias.

4.21 Detección de movimiento

El movimiento así mencionado en el libro de C. Gonzáles [3], es una herramienta utilizada para extraer objetos de interés de un fondo de detalles irrelevantes. El movimiento se deriva de un desplazamiento relativo entre el sistema de sensores y la escena que se está presentando.

El método mas básico para detectar cambios entre dos fotogramas $f(x,y,t_i)$ y $f(x,y,t_j)$ tomados en tiempos t_i y t_j, respectivamente es comparar las dos imágenes píxel a píxel. Un procedimiento para hacerlo es formar una imagen diferencia:

$$d_{ij}(x,y) = \begin{cases} 1\_si\left|f(x,y,t_i) - f(x,y,t_j)\right| > \theta \\ 0\_En\_caso\_contrario \end{cases} \qquad (4.20)$$

Donde θ es un umbral. Obsérvese que $d_{ij}(x,y)$ tiene un 1 como coordenadas espaciales (x,y) solamente si la diferencia de nivel de gris de dos imágenes es apreciablemente diferentes de estas coordenadas, según el valor determinado por el umbral θ.

En la práctica, las entradas de la matriz de valor 1 aparecen a menudo a causa del ruido, un método sencillo para eliminar las consiste en formar regiones 4 u 8 conexas de valores 1 en $d_{ij}(x,y)$ e ignorar después cualquier región que tenga menos de un número predeterminado de entradas.

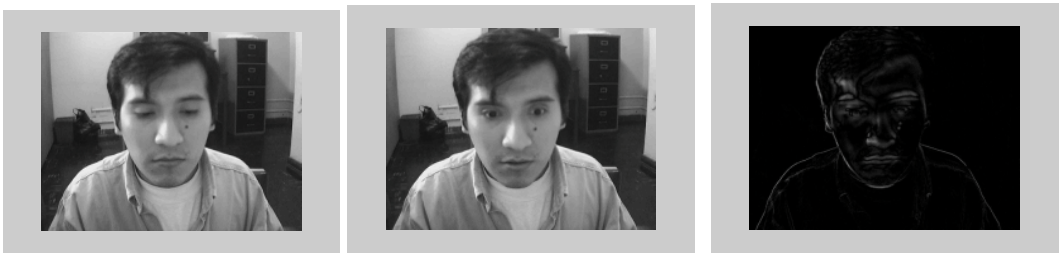


Fig. 4.21 *a) Imagen anterior b) Imagen Posterior c) Imagen Diferencia*

Los problemas anteriores se pueden solucionar con la técnica de diferencias acumulativas, al considerar los cambios en la posición de un píxel sobre varios fotogramas, introduciendo así una memoria de proceso.

En una sucesión de fotogramas y la primera es la imagen de referencia. Para formar una imagen de diferencia acumulativa se compara la imagen de referencia con cada una de las imágenes subsiguientes de la secuencia. Un contador de cada posición de píxel de la imagen acumulativa se incrementa cada vez que se produce una diferencia en dicha posición de píxel entre la referencia.

CAPITULO V
ANÁLISIS DE COMPONENTES PRINCIPALES (PCA)

5.1 Introducción

Es un método estadístico que genera una representación lineal reducida de las imágenes de rostros (matriz de proyección). De esta manera, cada rostro es proyectado en un espacio de menor dimensionalidad obteniéndose la componente que maximiza su varianza frente a otros rostros [18].

También se le conoce como descomposición de tipo “eigenspace”. El principal atractivo esta descomposiciones es que el espacio de las imágenes de rostros (dado por los vectores reducidos) tiene menor dimensionalidad que el espacio de imágenes (dado por el número de píxeles en las imágenes de rostros), y gracias a esto es posible realizar el reconocimiento rápido y eficaz.

En la Figura 5.1 siguiente se muestra un esquema general de la forma que se hará el entrenamiento y reconocimiento del rostro.

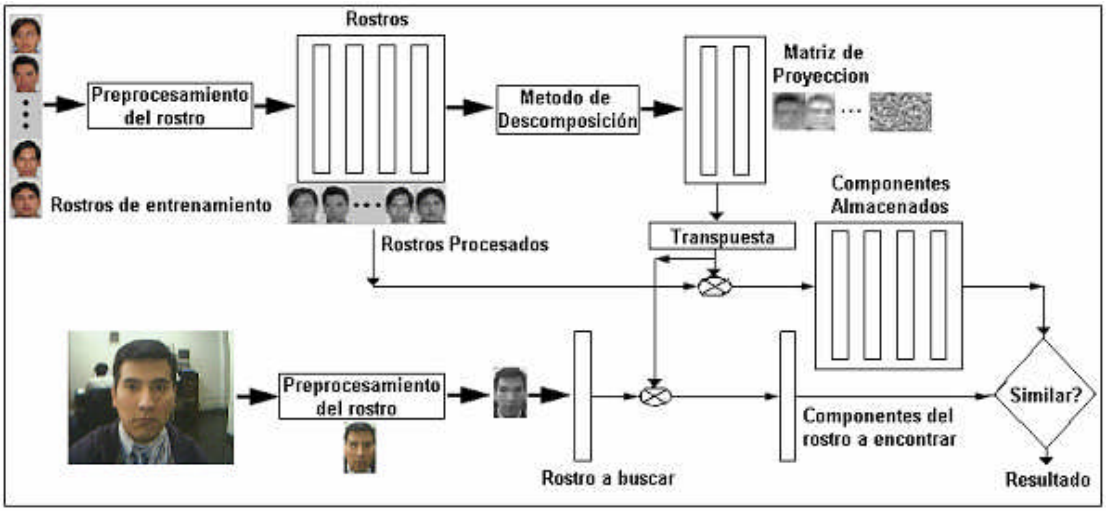


Fig. 5.1 *Proceso de entrenamiento y reconocimiento por PCA.*

Estos métodos pueden considerarse como una reducción de dimensionalidad no supervisados, puesto que no tienen en cuenta a qué clase pertenecen cada uno de los vectores de la secuencia de entrenamiento originales, previos a la reducción de su dimensión, si no que se tratan todos conjuntamente. PCA (Análisis de Componentes Principales) utiliza la transformada de Karhumen-Loeve y tiene aplicaciones diversas, para el caso de imágenes de rostros se le conoce como método de eigenfaces. En cambio LDA (Análisis Discriminante Lineal) utiliza el método de fisher y para el caso de imágenes de rostros es conocido como método de fisherfaces o método de fisher.

Para tener una idea de estos métodos, supongamos que en la figura 58a los círculos pequeños representen 5 rostros de una persona A y las aspas represente 5 rostros de una persona B. Para hacer más simple el ejemplo, estos se representan en el espacio de 2 dimensiones. Si hallamos la primera componente de PCA y proyectamos los puntos anteriores sobre ella entonces se encontrará la representación eficiente de datos, osea los puntos estarán lo mas dispersos entre sí. En cambio se hallamos la primera componente de LDA y proyectamos los puntos anteriores sobre ella entonces se favorecerá la separación entre clases tal como se muestra en la Fig. 5.2

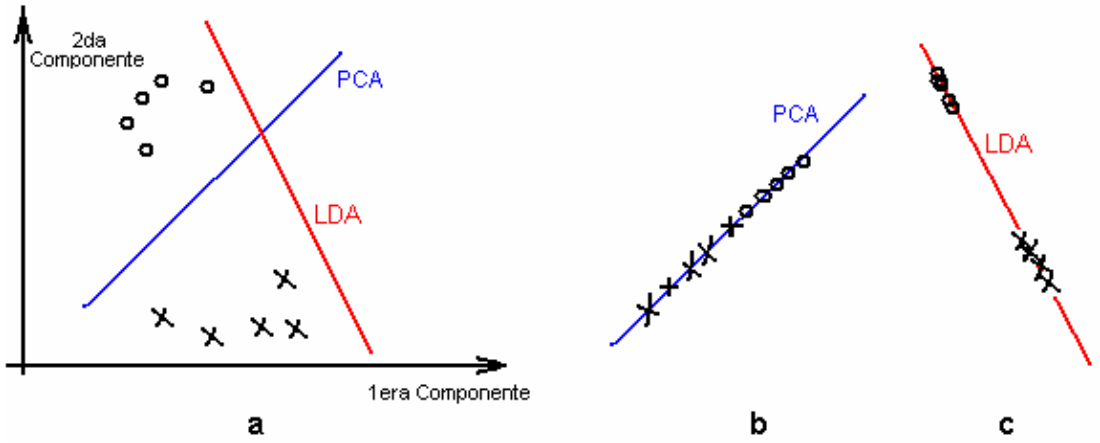


Fig. 5.2 a. *Muestras y dirección de PCA y LDA* **b.** *Proyección sobre la 1era componente PCA* **c.** *Proyeccción sobre la 1era componente LDA.*

5.2 Método de Eigenfaces.

Antes de utilizar el método de descomposición Eigenfaces, las imágenes de los rostros ya debieron haber pasado por la etapa de pre-procesamiento es decir ya debieron haber sido mejoradas, escaladas y llevadas en escalas de grises.

En este método la entrada es un conjunto de rostros diferentes ordenados de tal manera que la imagen de cada rostro es representado mediante un arreglo (vector columna) y el conjunto de los arreglos representa la matriz de rostros, la salida será un conjunto de rostros llamados *Eigenfaces* y que formarán la matriz de proyección. Las proyecciones de las caras en las *Eigenfaces* forman el vector de *Componentes Principales* de cada imagen y su dimensión se escoge igual o menor al número de imágenes de entrenamiento. De esta manera la representación original de cada rostro que es determinada por el número de píxeles, se reduce en forma considerable a través de sus componentes principales, es por ello que PCA es un método utilizado para la reducción de dimensionalidad. Estos vectores de componentes principales contienen la información de las diferencias relevantes entre las imágenes de entrenamiento.

El objetivo de este método es construir una base ortonormal en el espacio al cual pertenecen los vectores que llamaremos $\mathbf{x}$ (arreglo unidimensional de imágenes de rostros), de esta menara se busca que las proyecciones en los vectores base maximicen su varianza [19]. La varianza de la proyección de los vectores $\mathbf{x}$ sobre un vector unitario $\mathbf{u}$ cualquiera viene dada por la ecuación 5.1.

$$S^2(u^T.x) = u^T.C.u \tag{5.1}$$

Donde C será la matriz de covarianza (Cij representa la correlación entre el píxel i y el píxel j dentro del conjunto de imágenes) Y S^2 es el operador de la varianza.

$$C = \frac{1}{N}\sum_{K=1}^{M} \phi_k \phi_K^T \tag{5.2}$$

$$\phi = x_K - \varphi \tag{5.3}$$

$$\varphi = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.4}$$

La variable φ representa la media de entre todas las imágenes y ϕ es la diferencia de cada imagen con el promedio de ellos.

El objetivo es encontrar primero un vector unitario tal que la varianza de las proyecciones sobre él sea máxima. Así en el espacio ortonormal que resta por cubrir se repetirá lo anterior hasta completar el espacio de imágenes. Al finalizar este proceso se obtiene un conjunto de vectores que forman una base ortonormal y que maximizan la varianza de las proyecciones en forma sucesiva.

Como la matriz C es simétrica, sus valores y vectores propios son reales, entonces una base candidata para resolver este problema es aquella formada por los vectores propios normalizados de la matriz, para ellos veremos que:

$$C.u_i = \lambda_i.u_i \quad (u_i \text{ son vectores normalizados}) \tag{5.5}$$

$$u_i^T.C.u_i = u_i^T.\lambda_i u_i \tag{5.6}$$

$$u_i^T.C.u_i = \lambda_i.u_i^T.u_i \tag{5.7}$$

Ahora sabemos que: $u_i^T.u_i = 1$

$$u_i^T.C.u_i = \lambda_i \tag{5.8}$$

$$\lambda_i = \frac{1}{N} u_i^T \sum_{k=1}^{N} \phi_k \phi_k^T u_i \tag{5.9}$$

$$\lambda_i = \frac{1}{N} \sum_{k=1}^{N} u_i^T \phi_k \phi_k^T u_i \tag{5.10}$$

$$\lambda_i = \frac{1}{N} \sum_{k=1}^{N} (u_i^T \phi_k)^T (\phi_k^T u_i) \tag{5.11}$$

$$\lambda_i = \frac{1}{N} \sum_{k=1}^{N} (u_i^T \phi_k)^2 \tag{5.12}$$

$$\lambda_i = \frac{1}{N} \sum_{k=1}^{N} (u_i^T x_k - \frac{1}{N} \sum_{i=1}^{N} (u_i^T x_K))^2 \tag{5.13}$$

$$\lambda_i = S^2{}_k (u_i^T x_k) \tag{5.14}$$

Por lo tanto los eigenvalores (que es positivo o nulo pues es definida semi-positiva) representan la varianza de la representación de las imágenes de rostros. Los vectores entonces corresponden a las *Eigenfaces* y como pertenecen al mismo espacio de las imágenes de la base de datos, pueden ser representadas como imágenes en la representación visual. Las Componentes Principales de una imagen x de la base de datos vienen dadas por:

$$P_k = x.w_k \tag{5.15}$$

Ahora, si bien son necesarias *N* componentes principales para representar completamente cualquier imagen, dado que las imágenes que se desean representar tienen características comunes (en este caso por tratarse de rostros de personas), se espera que a partir de cierto punto los términos de la expansión sean irrelevantes por tratarse de componentes con menor varianza [17]. Las componentes irrelevantes

representan características comunes dentro de la base de datos y están consideradas en la imagen promedio. Luego, la reducción de dimensionalidad se efectúa truncando la expansión de la ecuación 5.15 hasta el término *m*-ésimo tal que *m*<*N* y cumpliendo algún criterio que asegure un grado de representación adecuado. Un criterio adecuado para este caso es establecer una cota para el error MSE (*Mean Square Error*) residual normalizado:

$$RMSE(m) = \frac{\sum_{k=m+1}^{M} \lambda_k}{\sum_{k=1}^{M} \lambda_k} \qquad (5.16)$$

Por ejemplo *m* mínimo tal que RMSE (*m*) < 1%.

Esta relación puede ser exacta si existen valores propios nulos de que no son considerados. Este caso generalmente se da en la práctica pues para calcular la matriz C se usa el estimador:

$$C = \frac{1}{M} \sum_{K=1}^{M} \phi_k \phi_K^T \qquad (5.17)$$

Donde *M* es el número de imágenes de entrenamiento. De esta forma C pasa a ser una matriz que como máximo puede tener rango igual al mínimo de N o M. En particular típicamente *M*<*N* y en estos casos el rango de C será *M* si los vectores de entrenamiento son linealmente independientes, pues toda la varianza estará proyectada en el hiperplano subtendido por estos vectores.

Las aplicaciones del algoritmo PCA a reconocimiento de rostros tienen una dificultad adicional relacionada con aspectos computacionales del mismo. Esta dificultad se basa en que C es una matriz de orden NxN. Aun si existiera memoria suficiente para hacerlo, los tiempos de ejecución del algoritmo serían demasiado grandes (sólo las operaciones para obtener los valores y vectores propios), lo cual se traduce en un método demasiado ineficiente. Sin embargo ¿Es necesario calcular los *N* valores y vectores propios, si sabemos que al menos *N*-*M* son irrelevantes?.

En efecto, el cálculo de los valores y vectores propios relevantes de puede ser efectuado sin tener que trabajar explícitamente con esta matriz. Para efectuar este desarrollo representaremos todas las imágenes de entrenamiento en la matriz $X = [X_1\ X_2\ ..\ X_{NT}]$, y de esta forma la matriz C se puede representar según la ecuación 5.18.

$$C = \frac{1}{M}\sum_{K=1}^{M} X.X^T \tag{5.18}$$

Ahora bien, a partir de lo anterior también es posible construir la matriz simétrica (equivalente a un estimador de la matriz de correlación entre las imágenes de entrenamiento) y es simple verificar que el rango de esta matriz es igual al rango de X, esto es, igual al número de vectores l.i. de la base de entrenamiento. A partir de esta matriz, factible de manejar para bases de entrenamiento no muy grandes, se puede resolver el siguiente sistema de autovalores:

$$X^T X.u^K = \lambda_K v^K \tag{5.19}$$

Con k=1 ... M, Ahora, pre-multiplicando X a (50)se tiene que:

$$XX^T X.u^K = \lambda_K X.u^K \tag{5.20}$$

Lo que define parcialmente el sistema de autovalores de C pues los vectores X.u^k corresponden a *NT* vectores propios de asociados a los valores propios λ_k, iguales para el sistema (Ecuación 50 y 51). Por último, si suponemos que al resolver la ecuación 51 se obtienen los vectores u normalizados, entonces los vectores X.u^k (correspondientes a las *Eigenfaces*) deberán ser normalizados de acuerdo a sus normas que vienen dadas por:

$$\left\| w^k \right\| = \sqrt{(Xu^k)(Xu^k)^T} = \sqrt{\lambda_k} \tag{5.21}$$

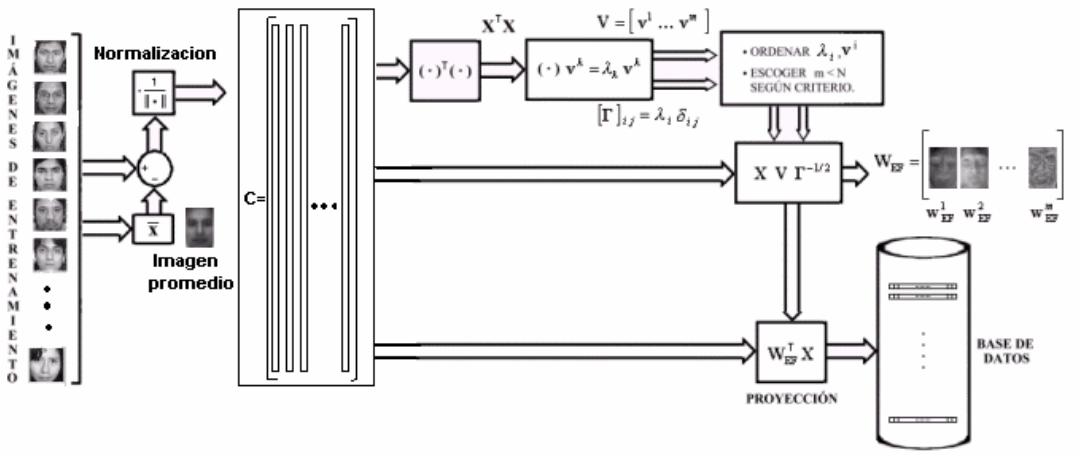


Fig. 5.3 *Esquema del entrenamiento del sistema con PCA*

5.3 Algoritmo para el cálculo de Eigenfaces

Con las deducciones realizadas anteriormente se da paso al algoritmo para el cálculo de eigenfaces.

Inicio:

M es el número de imágenes.

I_n es el conjunto de imágenes de entrenamiento n=1….M

A es el promedio de imágenes de entrenamiento

Leer M

$$X_n = \frac{I_n - A}{|I_n - A|}$$

$$K = \mathbf{X}^T \mathbf{X}$$

Calcular $Kv_n = \lambda_n v_n$ para todo n = 1hasta M

Ordenar v de acuerdo a los λ mayores

Calculo de Eigenfaces $U_n = v_n^T X$

Normalizando $U_n = \frac{U_n}{|U_n|}$

FIN

5.4 Método de Fisher

El método anterior nos permitió crear una matriz de proyección formado por un conjunto de rostros, el método de Fisher crea una matriz de proyección similar a eigenfaces que ahora tendrá el nombre de Fisherfaces. Los rostros son proyectados y de esta forma representada en un nuevo espacio de menor dimensionalidad.

En este método ya no se busca mantener la topología del espacio original. Es decir el objetivo ahora es llevar las caras a un espacio donde se pueda distinguir de mejor forma las distintas clases entre si, maximizando la distancia entre imágenes de distintas clases y a la vez minimizando la distancia entre imágenes de una misma clase. En esta nueva representación, cuando se proyecte una cara desconocida se espera que ésta quede cerca de un grupo de caras y lejos del resto, de esta manera el reconocimiento será confiable.

Para llegar a construir esta base, es necesario que las imágenes estén normalizadas, y que las proyecciones en los vectores base maximicen un indicador $\gamma(W^k)$, por tanto para definir $\gamma(u)$, con *u* un vector unitario, debemos estimar una información básica con respecto a las imágenes de entrenamiento y las clases C_i a las cuales pertenecen. Si m y $m^{(i)}$ son el valor medio de todas las imágenes de rostros y los valores medios de cada clase C_i respectivamente. Estimados como el promedio de las imágenes de entrenamiento correspondientes. Sea $P(C_i)$ la probabilidad asociada a la clase C_i dentro del espacio de rostros. Entonces se define la cantidad $\gamma(u)$ como:

$$\gamma(u) = \frac{S_b(u)}{S_w(u)} \tag{5.22}$$

Donde

$$S_b(u) = \sum_{i=1}^{NC} P(C_i)\{(m^{(i)} - m).u\}^2 \tag{5.23}$$

La ecuación 5.23 mide la separación entre los rostros promedio de cada clase con respecto al promedio global, en la proyección sobre el vector unitario *u* , y

$$S_w(u) = \sum_{i=1}^{NC} P(C_i)E[\{(x^{(i)} - m^{(i)}).u\}^2] \tag{5.24}$$

Le ecuación 5.24 mide la separación entre imágenes de una misma clase C_i en torno a su valor promedio, en la proyección sobre *u*. De esta manera al maximizar $\gamma(u)$, en las proyecciones sobre *u* se debe lograr separar las clases entre sí y juntar las imágenes de una misma clase.

Entonces primero se busca el vector unitario *u* donde se maximiza $\gamma(u)$, luego en el espacio que resta por cubrir se repite el proceso anterior sucesivamente hasta completar. Luego al terminar se habrá obtenido un conjunto de vectores que forman una base y que maximizan el indicador γ en forma sucesiva. Para encontrar la base W^k que cumple esta propiedad, primero expresamos los coeficientes (Ecuación 5.23 y 5.24) de la siguiente manera:

$$S_b(u) = u^T S_b u \tag{5.25}$$

$$S_w(u) = u^T S_w u \tag{5.26}$$

Donde:

$$S_b(u) = \sum_{i=1}^{NC} P(C_i)(m^{(i)} - m).(m^{(i)} - m)^T \quad (5.27)$$

$$S_w(u) = \sum_{i=1}^{NC} P(C_i)E[(x^{(i)} - m^{(i)}).(x^{(i)} - m^{(i)})^T]\,. \quad (5.28)$$

Estas matrices pertenecen al espacio R^{NxN} y son simétricas y definidas semi-positivas. A S_b se le denomina “matriz de dispersión de rostros entre clases” y a S_w “matriz de dispersión de rostros dentro de las clases”. Estas matrices representan la correlación de rostros entre clases y dentro de las clases, independiente del vector donde se proyecten. Por lo tanto se puede reescribir (5.28) como:

$$\gamma(u) = \frac{u^T S_b u}{u^T S_w u} \quad (5.29)$$

Ahora, siguiendo los mismos pasos de PCA, nos preguntamos si existirá un sistema de autovalores en que los vectores propios sean una base del espacio de imágenes y donde los valores propios correspondan a los coeficientes γ asociados a los vectores propios correspondientes. En efecto este sistema es fácil de encontrar, pues escribiendo (5.29) como:

$$\gamma(u) = \frac{u^T S_w S_w^{-1} S_b u}{u^T S_w u} \quad (5.30)$$

El sistema que se esta buscando está dado por:

$$S_w^{-1} S_b W^k = \lambda_k w^k\,, \quad (5.31)$$

Reemplazando (5.31) en (5.30) vemos que $\gamma(w^k) = \lambda_k$. Similar al algoritmo PCA, y actuando por contradicción, es fácil demostrar que no existe un vector distinto de W^1 con un coeficiente γ mayor. Es por eso que los vectores W^k corresponden a la base óptima, es decir, maximizan γ en forma sucesiva (con los λ_k ordenados de mayor a menor). Para demostrar lo anterior, la matriz $S_w^{-1} S_b$ no tiene por qué ser simétrica en general. Aun así, sus valores y vectores propios siguen siendo reales. A diferencia de las matrices simétricas los vectores propios de esta matriz no serán ortogonales. La ecuación (5.31) constituye un sistema conocido como sistema generalizado de autovalores de S_b y

S_w, que al involucrar matrices simétricas cumple con una propiedad similar a la ortogonalidad para el caso simétrico simple.

$$(W^n)^T S_w w^m = S_w(w^n)\delta_{nm} \tag{5.32}$$

La finalidad es identificar rostros a partir de la base w^K, entonces podemos considerar las primeras *Fisherfaces* e ignorar aquéllas con coeficientes γ pequeños (similar a PCA), el problema es que se pueden mezclar las clases y empeorar el desempeño del reconocimiento. El criterio para escoger las componentes puede ser similar al RMSE, pero el significado de los λ_k es diferente ya que se dedica exclusivamente a la diferenciación de clases.

Para poder implementar un sistema que determine las *Fisherfaces* a partir de un conjunto de imágenes de entrenamiento, necesitamos estimar la información estadística (S_b y S_w). Los estimadores para S_b y S_w están dados en ecuación (5.33) y (5.34):

$$S_b(u) = \sum_{i=1}^{NC} n_i (m^{(i)} - m).(m^{(i)} - m)^T \tag{5.33}$$

$$S_w(u) = \sum_{i=1}^{NC} \sum_{j=1}^{n} (x_j{}^{(i)} - m^{(i)}).(x_j{}^{(i)} - m^{(i)})^T \tag{5.34}$$

Donde n_i es el número de vectores en la clase C_i que idealmente debiera ser igual para todas las clases.

Para la manipulación de las matrices S_b y S_w, estas serán singulares para números de clases e imágenes de entrenamiento no muy grandes y por lo tanto la dimensionalidad original del espacio de imágenes usualmente será redundante para los cálculos. La solución a este problema es hacer una reducción de dimensionalidad de las imágenes de rostro antes de aplicar el método de Fisher. Es preferible aplicar antes PCA para que asegure una buena representación y a los vectores resultantes les aplicamos el método de Fisher. De esta manera, si la matriz de proyección PCA es $W_{EF} \varepsilon R^{Nxm}$ entonces la matriz de proyección de Fisher en el espacio reducido es $V \varepsilon R^{m.r}$ (donde $r \leq m$ es el número de componentes escogidos para una buena discriminación de clases), la matriz de proyección definitiva del método de Fisher (FF) $W_{FF} \varepsilon R^{Nxr}$ será [20]
:

$$W_{FF} = W_{EF} V \tag{5.35}$$

Un criterio adecuado para reducir la dimensionalidad y discriminar clases es establecer una cota para el error RFP (*Residual Fisher Parameter*) normalizado:

$$RFP(r) = \frac{\sum_{k=r+1}^{n} \lambda_k}{\sum_{k=1}^{m} \lambda_k} \qquad (5.36)$$

La Fig. [5.4] muestra un diagrama de bloques de todas las etapas necesarias para obtener las *Fisherfaces*.

Una vez resuelta el problema de la reducción de dimensionalidad, existe una dificultad para implementar el método de Fisher, debido a que hay que resolver un sistema de autovalores para la matriz $S^{-1}{}_w S_b$, ya que no es simétrica y el resultado podría ser inestable, es decir, con componentes complejas.

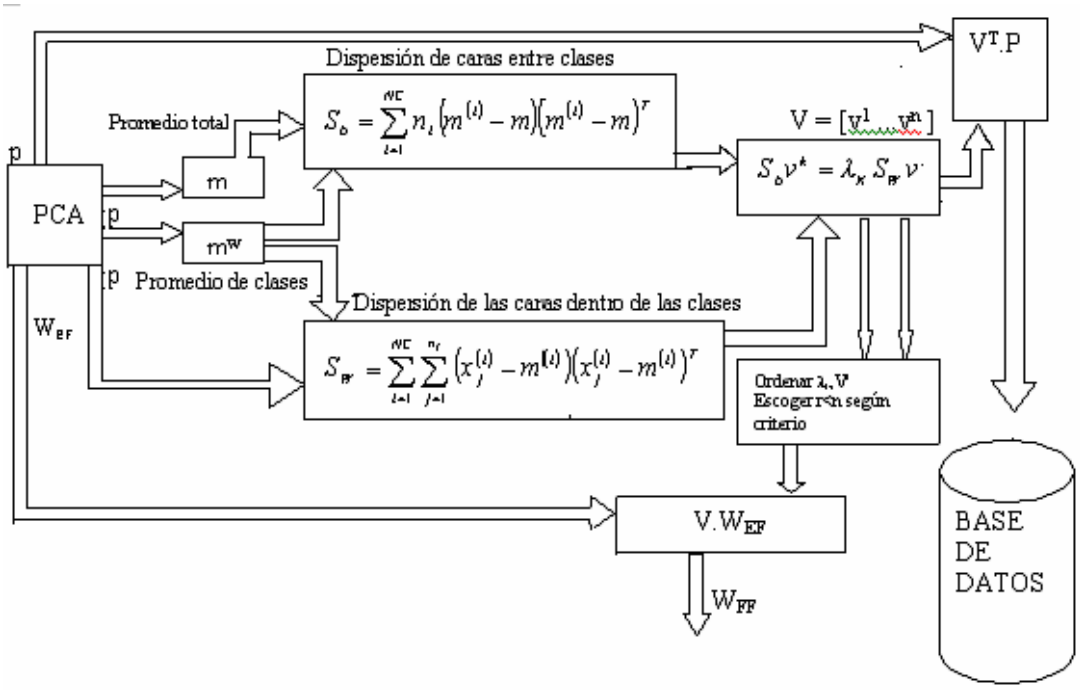


Fig.5.4. *Esquema del proceso de entrenamiento con el método de Fisherfaces*

Se sabe que las matrices S_b y S_W son simétricas y el sistema de autovalores es estable. Entonces se parte por resolver el sistema de autovalores para la matriz S_W, asi obtenemos los vectores propios H (ortonormal) y la matriz diagonal de valores propiosΛ, asi $S_w = H\Lambda H^T$. Luego, escrito de otra forma $(H\Lambda^{-1/2})^T S_W H\Lambda^{-1/2} = I$. Similarmente se

hace los mismo con la matriz S_b , $(H\Lambda^{-1/2})^T S_b H\Lambda^{-1/2} = U\Sigma U^T$. En esta última expresión, U es una matriz ortonormal y Σ es diagonal. Ambas matrices son obtenidas al resolver el sistema de autovalores para la matriz simétrica $(H\Lambda^{-1/2})^T S_b H\Lambda^{-1/2}$. Entonces:

$$S_b = H\Lambda^{1/2}U\Sigma U^T\Lambda^{1/2}H^T \qquad (5.37)$$

$$S_W = H\Lambda^{1/2}UIU^T\Lambda^{1/2}H^T \qquad (5.38)$$

Definiendo $Q = H\Lambda^{-1/2}U$, en las relaciones anteriores podemos ver que Q diagonaliza S_b y S_W al mismo tiempo. Ahora bien, como:

$$S_W^{-1} = H\Lambda^{-1}H^T \qquad (5.39)$$

de (5.38) y (5.39) se obtiene:

$$S_W^{-1}S_b = H\Lambda^{-1}H^T H\Lambda^{1/2}U\Sigma U^T\Lambda^{1/2}H^T$$

$$S_W^{-1}S_b = H\Lambda^{-1/2}U\Sigma U^T\Lambda^{1/2}H^T$$

$$S_W^{-1}S_b = Q\Sigma Q^{-1} \quad . \qquad (5.40)$$

La matriz Q corresponde a la matriz de vectores propios de $S^{-1}{}_W S_b$, y Σ a la matriz de valores propios. Como Q y Σ son reales, el sistema de autovalores de $S^{-1}{}_W S_b$ es estable.

5.5 Algoritmo para el cálculo de Fisherfaces.

En resumen, para calcular Q y Σ se deben seguir los siguientes pasos:

1 Resolver el problema de autovalores de la matriz S_W , para obtener H y Λ.

2 Calcular la matriz simétrica $(H\Lambda^{-1/2})^T S_b H\Lambda^{-1/2}$.

3 Resolver el sistema de autovalores de $(H\Lambda^{-1/2})^T S_b H\Lambda^{-1/2}$, para obtener U y Σ (valores propios de $S^{-1}{}_W S_b$).

4 Calcular $Q = H\Lambda^{-1/2}U$ (vectores propios de $S_w^{-1}S_b$).

CAPITULO VI

MODELO OCULTO DE MARKOV EMBEBIDO (HMME)

6.1 Introducción

Un proceso estocástico es un modelo matemático que describe el comportamiento de un sistema dinámico sometido a un fenómeno de naturaleza aleatoria.

Todo proceso estocástico cumple las siguientes características:

- Describe el comportamiento de un sistema variante respecto de un determinado parámetro.
- Existe un fenómeno aleatorio que evoluciona según el parámetro t, normalmente el tiempo.
- El sistema presenta estados definidos y observables
- El sistema cambia probabilìsticamente de estado
- Todo estado el sistema tiene una probabilidad asociada $P_x(t)$ el cual indica la probabilidad de encontrarse en el estado x en el instante t.

De acuerdo al anterior, un proceso estocástico queda definido por el conjunto:

X(t), $P_x(t)$, t

Un ejemplo sencillo de un proceso estocástico estaría dado por un sistema de pronostico del clima, en el cual:

t=1,2,3, ... n días

X(t) = pronostico del clima asociado al día requerido (nublado, lluvioso, soleado, etc ..)

$P_x(t)$ = probabilidad de estado asociada.

Un proceso estocástico, principalmente del tipo Marvok tiene gran importancia en el análisis y estudio de sistemas de comportamiento dinámico.

Los elementos que lo componen son:

- Conjunto de espacio de estado, que representa los diferentes estados
- Parámetro del sistema, que usualmente es t y es el tiempo

- Probabilidad condicional de transición, se define:

$$a_{ij}(\Delta t) = P\{X(t+\Delta t) = j \mid X(t) = i\}$$

- Probabilidad incondicional de estado: $P_i(t) = P_{x=i}(t)$, si t=0 entonces $\pi_i = P_{x=i}(0)$

En los procesos de tipo Markov según la naturaleza de variables

Según la naturaleza des espacio de estados, los procesos del tipo Markov pueden ser:

- Proceso de Markov, cuando la variable $X(t)$ representa una magnitud continua (fuerza, energía, presión, etc.).
- Cadenas de Markov, cuando la variable $X(t)$ representa una magnitud discreta (cantidad de clientes, número de líneas en servicio, etc...)
- Según la naturaleza del parámetro t, los procesos del tipo Markov pueden ser:
- Procesos o cadenas de Markov de parámetro continuo, cuando las observaciones al sistema se realizan en cualquier momento del continuo (t>=0)
- Procesos o cadenas de Markov de parámetro discreto, cuando las observaciones al sistema se realizan en determinados instantes del parámetro t (cada hora, minuto, día).

Todo proceso estocástico puede ser representado gráficamente, en particular las cadenas de Markov. Por ejemplo supongamos un sistema con 3 estados:

$P(X_t = j \mid X_{t-1} = i) = a_{ij}$

Sea:

$$A = \begin{bmatrix} a_{11} & . & . & a_{1N} \\ . & & & . \\ . & & & . \\ a_{N1} & . & . & a_{NN} \end{bmatrix} \quad \text{Con} \quad \begin{array}{ll} a_{ij} \geq 0 & \forall i, j \\ \sum_{j=1}^{N} a_{ij} = 1 & \forall i \end{array} \qquad \textbf{(6.1)}$$

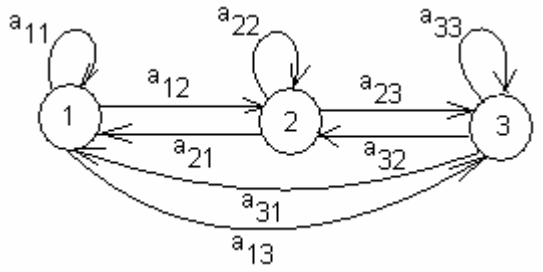


Fig. 6.1. *Modelo de estados*

6.2 Modelos Ocultos de Markov

Los modelos de Markov son modelos matemáticos de procesos estocásticos, procesos que generan secuencias aleatorias de salida de acuerdo a ciertas probabilidades. Un simple ejemplo de proceso estocástico es una secuencia de lanzamiento de moneda, los resultados son cara o cruz. Las personas usan el modelo de Markov para analizar una amplia variedad de procesos estocásticos, de stock de precios hasta la posición de genes en un cromosoma [27].

Se puede construir un modelo de Markov muy fácilmente usando 'diagramas de estado' como la que se muestra en la Fig. Siguiente.

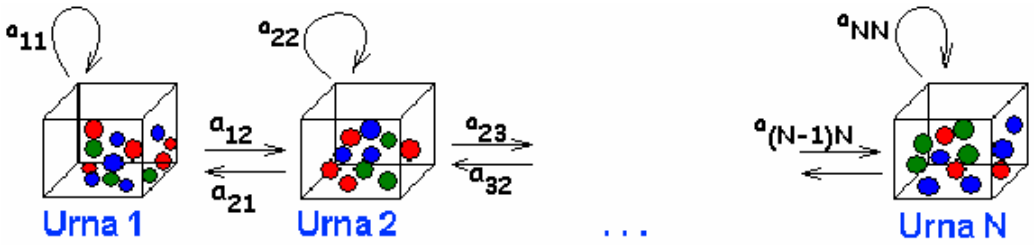


Fig. 6.2. *Un diagrama de estado para un modelo de Markov.*

Los rectángulos en el diagrama representan los posibles estados que se esta tratando de modelar y las flechas representan las transiciones entre los estados, la etiqueta en cada flecha representa la probabilidad de transición, el cual depende del proceso que se esta modelando. En cada paso del proceso, el modelo genera una salida, dependiendo dentro de que estado este, y luego hace una transición a otro estado.

Un modelo oculto de Markov es en el cual se observa una secuencia de emisiones, pero no se conoce la secuencia de estados que el modelo esta pasando para generar las emisiones. En este caso, la meta es recuperar la información de estado del dato observado.

Por ejemplo, Dos personas A y B se encuentran cada una en una habitación, Imagínese a la persona A con N urnas con M diferentes bolas de colores, él selecciona una urna inicial y escoge una bola al azar y el color es dictado a la persona B como una observación, la bola es colocada nuevamente en la urna y acorde a un proceso aleatorio asociado a la urna actual, se selecciona una nueva urna y se repite el proceso nuevamente. De esta forma la persona B solo podrá ver la secuencia R,A,V pero no podrá determinar en que forma fue realizado el experimento (oculto).

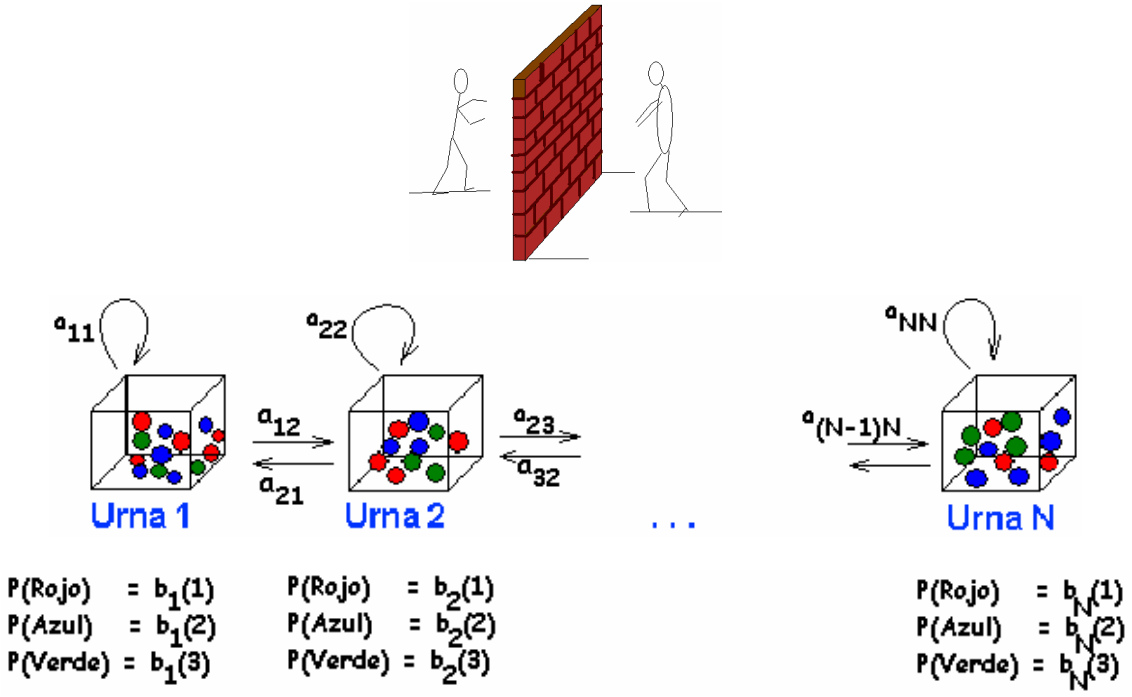


Fig. 6.3. *Ejemplo de dos personas con "N" urnas y 3 bolas diferentes.*

De este ejemplo se puede sacar un casi particular, 2 Urnas y bolas de diferentes colores (Azul, Verde y Rojo)

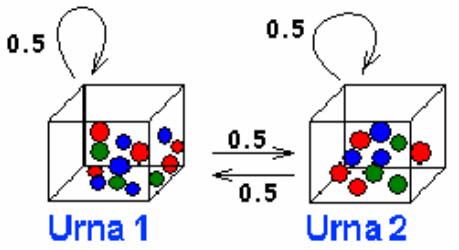


Fig. 6.4. *Ejemplo de dos personas con 2 urnas y 3 bolas diferentes.*

Para Modelar un HMM que corresponda a este escenario, los estados estarán representados por las urnas. Por el proceso aleatorio de selección de urnas existen probabilidades de transición y por el proceso aleatorio de selección de bolas y por la cantidad de bolas de cada color en cada urna existen probabilidades de salida. Así se tiene:

- Conjunto de estados Q={q1,...,qn}
- Posibles Observaciones: O={O1,...On}
- Vector de probabilidades iniciales π
- Una Matriz de probabilidad de transición A={a_{ij}}, donde $a_{ij}=P(S_t=q_j|S_{t-1}=q_j)$

- Un vector de probabilidades de Salida por cada estado (matriz) B={b_{ij}} donde,

  $B_{ij}=P(O_t=o_k|S_t=q_j)$

- En forma Compacta:

$$\lambda = (A, B, \pi) \qquad (6.2)$$

Existen tres problemas en el modelo de markov y estas son:

a) Dado el Modelo, calcular la probabilidad de una secuencia de observaciones (Evaluación o Reconocimiento). La solución esta en el Algoritmo Forward y backward.

b) Dado el modelo, obtener la secuencia de estados mas probable correspondiente a una secuencia de observaciones (secuencia óptima). La solución es el Algoritmo de Viterbi.

c) Dada una secuencia de observaciones, ajustar los parámetros del modelo (Aprendizaje). La solución es el Algoritmo Segmentación K-Menas o Baum-Welch (Reestimación de parámetros)

Algoritmo Forward

1. Inicialización: $\alpha_1(i) = P(O_1, S_1 = q_i) = \pi_i b_i(o_1) \qquad 1 \le i \le N \qquad (6.3)$

2. Inducción: $\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(o_{t+1}) \qquad 1 \le t \le T-1 \qquad (6.4)$

$$1 \le j \le N$$

3. Terminación: $P(O/\lambda) = \sum_{i=1}^{N} \alpha_T(i) \qquad (6.5)$

En cada Iteración se tiene del orden de N multiplicaciones y N sumas

Para las T iteraciones: $N^2.T <<< 2T.N^T$

Algoritmo Backward

Se define la variable "backward" $\beta_t(i) = P(O_{t+1} O_{t+2} ... O_T, q_t = S_i / \lambda)$. Es decir, la probabilidad de una secuencia parcial de observaciones y que llegue a cierto estado.

1. Inicialización: $\beta_T(i) = 1, \qquad 1 \le i \le N \qquad (6.6)$

2. Inducción: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad t = T-1, T-2, ..., 1 \qquad (6.7)$

$$1 \le i \le N$$

3. Terminación: $$P(O/\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1)\beta_1(i) \qquad (6.8)$$

De igual manera en cada iteración se tiene del orden de N multiplicaciones y N sumas, entonces para las T iteraciones: $N^2.T <<< 2T.N^T$

El algoritmo de Viterbi

El algoritmo de Viterbi estima (usando el criterio de maximización a posteriori de $P(0/\lambda)$) la secuencia más probable de estados durante la producción de una secuencia de observación, y la probabilidad final para esa secuencia de estados.

La ruta mas probable en un HMM es útil para el aprendizaje y para el alineamiento de secuencias con el modelo. La ruta mas probable $q = (q_1 q_2 \ldots q_T)$ para la secuencia de observación $O = (o_1 o_2 \ldots o_T)$ puede ser calculada utilizando el *algoritmo de Viterbi*.

Antes de proceder con el algoritmo debemos de definir otra variable:
La subsecuencia de estados óptimos hasta el tiempo t:

$$\delta_t(i) = P(S_1 S_2 \ldots S_t = q_i, O_1 O_2 \ldots O_t)$$

En forma iterativa:

$$\delta_{t+1}(i) = \left[MAX\delta_t(i)a_{ij}\right]b_j(O_{t+1}) \qquad (6.9)$$

1. Inicialización : $\delta_1(i) = \pi_i b_i(o_1) \quad 1 \le i \le N$ (6.10)

$\Psi_1(i) = 0, \quad 1 \le i \le N$ (6.11)

2. Inducción : $\delta_t(j) = \max_{1 \le i \le N}\left[\delta_{t-1}(i)a_{ij}\right]b_j(o_t), \quad 2 \le t \le T, \quad 1 \le j \le N$ (6.12)

$\Psi_t(j) = \arg\max_{1 \le i \le N}\left[\delta_{t-1}(i)a_{ij}\right], \quad 2 \le t \le T, \quad 1 \le j \le N$ (6.13)

3. Terminación : $P^* = \max_{1 \le i \le N}\left[\delta_T(i)\right]$ (6.14)

$$q_T^* = \arg\max_{1 \le i \le N}[\delta_T(i)] \qquad (6.15)$$

4. Ruta inversa : $q_t^* = \Psi_{t+1}(q^*_{t+1}), \quad t = T-1, T-2, \ldots, 1.$ (6.16)

6.3 Modelo Oculto de Markov Embebido (HMME)

A pesar del éxito que tiene HMM en reconocimiento de voz, en donde los datos son vectores de una sola dimensión (HMM-1D), este no tiene el mismo resultado en datos de dos dimensiones (2D) como son las imágenes; Una versión HMM completa de dos dimensiones es la que se muestra en la figura 3.1. Esta versión introducida por Levin y Pierracini[3.2] puede caracterizar mejor a una imagen, sin embargo este muestra una complejidad en los algoritmos de entrenamiento y reconocimiento en HMM-2D, por lo que haría que el sistema demore en procesar los datos y no sería el adecuado para aplicaciones en tiempo real.

Otra versión que se acerque al modelo de datos de dos dimensiones sin mostrar complejidad es el Metodo de Harkov Embebido o incrustado (HMME). Esta versión es introducida por los autores Kuo y hagáis para reconocimiento de caracteres [30].

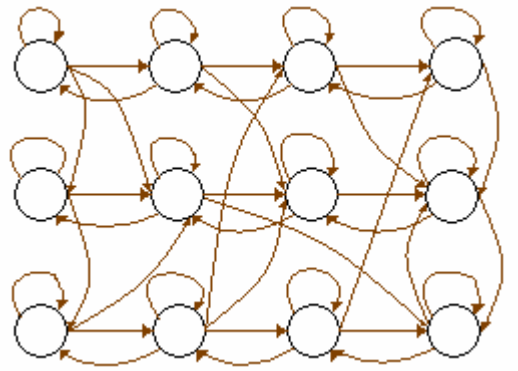


Fig. 6.5 *Versión HMM-2D*

Un HMME (Modelo Oculto de Markov Embebido) es una generalización de un HMM en donde cada estado es un HMM-1D. Es así que un HMME consiste de un conjunto de súper estados y un conjunto de estados embebidos. El modelo de súper estados tiene una dirección mientras que los estados embebidos tienen otra dirección, tal como se muestra en la Fig. 6.6. Hay que notar que un HMME no es de dos dimensiones ya que la transición entre los estados en diferentes súper estados no esta permitida.

A continuación se dará una definición formal de un HMME y se describirá los tres problemas a resolver para poder entrenar, optimizar y reconocer rostros en un HMME. Estos problemas se resuelven con el algoritmo decodificador, el algoritmo de evaluación y un algoritmo de estimación.

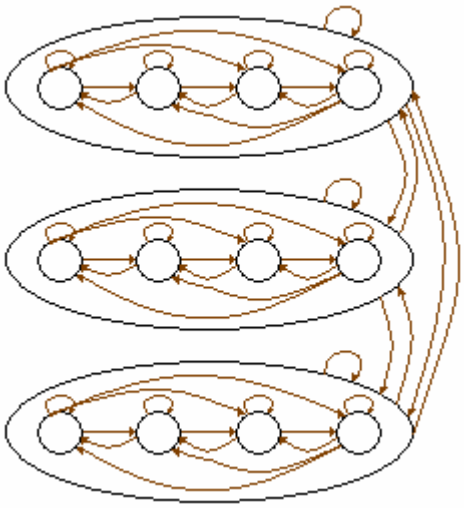


Fig. 6.6. *Versión HMME con 3 súper estados*

6.4 Estructura HMME para rostros

HMME tiene similares características que un HMM. Los elementos de HMME son los siguientes:

Un conjunto (N_0) de súper estados, $\mathbf{S}_0 = \{S_{0i}, i = 1,2,...N_0\}$, la probabilidad de distribución inicial del súper estado, $\Pi_0 = \{\pi_{0,i}\}$, donde $\pi_{0,i}$ es la probabilidad de empezar en un súper estado i en el tiempo cero y la matriz de transición entre los súper estados, $\mathbf{A}_0 = \{a_{0,ij}\}$, donde $a_{0,ij}$ es la probabilidad de transición desde el súper estado i al súper estado j.

En un HMME, cada súper estado es así mismo un HMM. Y la estructura de estos HMM es el mismo que un HMM-1D. El número de estados, la probabilidad de estado inicial, y la matriz de transición están presentes en cada súper estado, la cantidad de estados incrustados dependerá sobre que súper estado se este tratando. En un súper estado tenemos lo siguiente:

Un número de estados embebidos en el k^{th} súper estado, N_1^k, y un conjunto de estados embebidos, $\mathbf{S}_1^k = \{S_{1,i}^k, i = 1,2,..., N_1^k\}$

La probabilidad de distribución de estado inicial de estados embebidos, $\Pi_1^k = \{\pi_{1,i}^k\}$ es la probabilidad de empezar en un estado *i* del súper estado *k* en el tiempo cero.

La matriz de transición de estados para estados embebidos, $\mathbf{A}_1^k = \{a_{1,ij}^k\}$, donde $a_{1,ij}^k$ es la probabilidad de la transición desde el estado i hacia el estado j dentro de un supe estado k.

La matriz de distribución de probabilidad para las observaciones, $\mathbf{B}^k = \{b_j^k(\mathbf{O}_{t_0,t_1})\}$, donde $b_j^k(\mathbf{O}_{t_0,t_1})$ es la probabilidad de observación $\mathbf{O}_{t_0,t_1}$, dentro del estado j en el súper estado k. El vector de observación $\mathbf{O}_{t_0,t_1}$ tiene dos subíndices, t_0 y t_1.

Para un HMME discreto, se asume que $\mathbf{O}_{t_0,t_1}$ puede tener un número finito de símbolos de observación. Supongamos que P es el número de símbolos de observación y V es el conjunto de todas las posibles observaciones (También llamado Codebook del modelo). $V = v_1,...,v_P$. En este caso, **B** es la matriz de probabilidad de símbolos de observación. $\mathbf{B}^k = \{b_j^k(p)\}$, donde,

$$b_i^k(p) = P(\mathbf{O}_{t_0,t_1} = v_p \mid S_{0,k}, S_{1,i}^k, \lambda) \qquad (6.17)$$

Para un HMME de densidad continua, las observaciones son caracterizadas por una función de densidad de probabilidad continua. Al igual que HMM 1D, también se tiene una mezcla finita gausiana de la forma:

$$b_i^k(\mathbf{O}_{t_0,t_1}) = \sum_{m=1}^{M_i^k} c_{im}^k N(\mathbf{O}_{t_0,t_1}, \mu_{im}^k, U_{im}^k) \qquad (6.18)$$

Para $1 \le i \le N_1^k$, y donde c_{im}^k es el coeficiente de mezcla para la m-ava mezcla en el estado *i* del súper estado *k*, y $N(\mathbf{O}_{t_0,t_1}, \mu_{im}^k, \mathbf{U}_{im}^k)$ es una densidad gausiana con un vector promedio μ_{im}^k y una matriz de covarianza $\mathbf{U}_{im}^k$.

Una mezcla embebida HMM es tal que todos los componentes gausianos son almacenados y parte de ellas forman la distribución de estados en la salida por tanto se define para el estado i como:

$$b_i^k(\mathbf{O}_{t_0,t_1}) = \sum_{m=1}^{M} c_{im}^k N(\mathbf{O}_{t_0,t_1}, \mu_m^k, \mathbf{U}_m^k) \qquad (6.19)$$

Esta ecuación difiere de la ecuación anterior en los parámetros de la componente gausiana y también en el número de componentes mixtos que son estados independientes.

Si definimos $\mathbf{\Lambda}^k = \left\{\Pi_1^k, \mathbf{A}_1^k, \mathbf{B}^k\right\}$ como el conjunto de parámetros que define el k[th] súper estado, entonces el HMME esta definido como:

$$\lambda = (\Pi_0, \mathbf{A}_0, \mathbf{\Lambda}) \qquad (6.20)$$

Donde $\mathbf{\Lambda} = \left\{\mathbf{\Lambda}^1, \mathbf{\Lambda}^2, ..., \mathbf{\Lambda}^{N_0}\right\}$.

Dado $\mathbf{O}_{t_0}$, $1 \le t_0 \le T_0$ denotamos la secuencia $\mathbf{O}_{t_0,1}, \mathbf{O}_{t_0,2}, ..., \mathbf{O}_{t_0,T_1}$, y dado $\mathbf{O}$ denotamos la secuencia $\mathbf{O}_{t_0}, ..., \mathbf{O}_{T_0}$. Además si definimos $q_{t_0 t_1}$, tal que $1 \le t_1 \le T_1$ y que denota el estado de la observación $\mathbf{O}_{t_0,t_1}$. Si $q^0_{t_0,t_1}$ es el súper estado de $\mathbf{O}_{t_0,t_1}$ y $q^1_{t_0 t_1}$ es el estado embebido de O_{t_0,t_1}, entonces $q_{t_0,t_1} = (q^0_{t_0 t_1}, q^1_{t_0,t_1})$. Ahora dado que $\mathbf{q}^1_{t_0} = q^1_{t_0,1}, ..., q^1_{t_0,T_1}$ es una secuencia de estados embebidos y $\mathbf{q}^0 = q^0_0, ..., q^0_{T_0}$ es la secuencia de súper estados, Entonces $\mathbf{q}_{t_0}$ representa la secuencia $q_{t_0,t_1}, ..., q_{t_0,T_1}$ dado $\mathbf{q} = \mathbf{q}_0, ..., \mathbf{q}_{T_0}$.

Un modelo como el mostrado en la figura 3.5 representa un modelo HMM Embebido, donde cada súper estado esta formada por HMM 1D. Los súper estados en la imagen tienen dirección vertical, mientras que los estados embebidos tienen dirección horizontal.

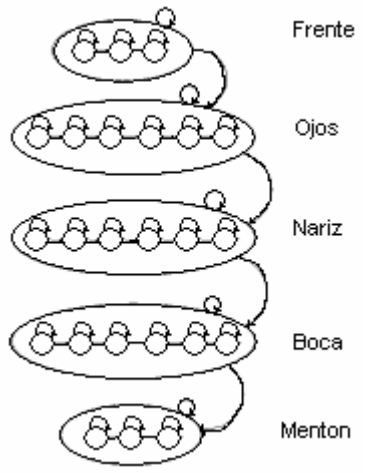


Fig. 6.7 *HMME para un rostro*

Para poder usar un HMM Embebido en el reconocimiento de rostros, necesitamos especificar la cantidad de súper estados, el número de estados embebidos en cada súper estado, la probabilidad de transición y las observaciones que son producidas por un HMM. Por ejemplo en la figura 3.5 se usa una estructura de un HMM Embebido, donde se tiene que $N_0 = 5$ súper estados que son usados para representar cinco regiones faciales que es obtenida escaneando de arriba hacia abajo de un rostro. Por ejemplo: Pelo, frente, ojos, nariz, boca, mentón. Luego el primer súper estado tiene tres estados embebidos. El número de estados embebidos fue seleccionado en base al número de regiones donde halla más información. Por ejemplo en el segundo súper estado (los ojos) aquí tenemos un estado para cada ojo y dos estados entre los ojos de igual manera entre los ojos y la cara dos estados. Así tenemos 6 estados embebidos en total para el segundo súper estado.

Ahora se considera solo 5 súper estados y no con 6 súper estados, porque esto aparte de aumentar la complejidad computacional, no tiene una eficiencia que con 5 súper estados.

Cada estado ha sido modelado por una mezcla de tres densidades gausianas. Y la matriz de covarianza es una matriz diagonal.

6.6. Vectores de observación

La secuencia de observación para una imagen de la cara es formada por bloques de tamaño $L_x x L_y$ que son extraídos en una imagen tal como se muestra en la figura ( ) los bloques tienen están sobrepuestos por P_x, P_y filas en la columna vertical y horizontal respectivamente.

Usaremos los siguientes valores: $L_x = 8, L_y = 10, P_x = 6, P_y = 8$. El vector de observación es formado por 6 coeficientes DCT de dos dimensiones (un arreglo de 3x2). O 4 coeficientes KTL correspondientes a los eigenvalores de gran tamaño.

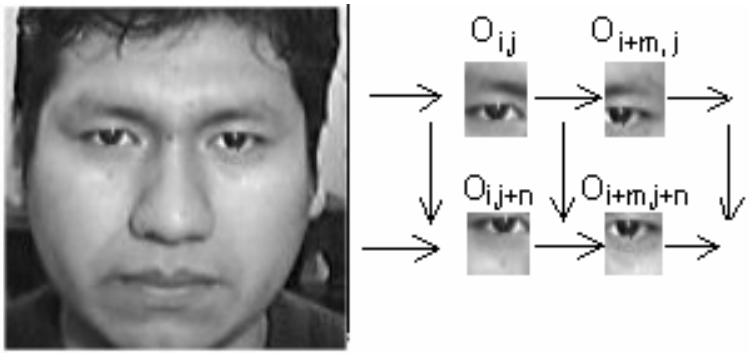


Fig. 6.8. *Generación de vectores de observación para HMME*

La técnica usada para extraer los coeficientes KTL es análoga a la extracción que se realizo en HMM [ ]. Los vectores de observación tienen una dimensión de $L_x xL_y = 80$, de estos solo usaremos 6 coeficientes DCT o 4 coeficientes KTL, de esta manera estamos reduciendo un factor cercano a 33 o 40.

Usando los coeficientes DCT reducimos la sensibilidad del HMM al ruido, rotación de la imagen y cambios de iluminación. Un cambio de iluminación solo afectan a la primera componente del DCT (componente DC).

6.3.1 Transformada Karhunen Lòeve (KTL)

KTL es una transformada adaptiva como por ejemplo las funciones básicas de esta transformada depende de la señal para el cual es aplicada. Estas funciones son los eigenvectores e eigenvalores de la matriz covarianza de todas las muestras. Estas son obtenidas de la siguiente manera:

1. Los bloques de todas las imágenes en el entrenamiento son extraídas y ordenadas en columnas. Sea $x_i^{(r)}$ el vector obtenido del ith bloque de la rth imagen.
2. El promedio u de estos vectores son calculados de la siguiente manera:

$$u = \frac{1}{\sum_{r=1}^{R} T^{(r)}} \sum_{r=1}^{R} \sum_{i=1}^{T^{(r)}} x_i^{(r)} \qquad (6.21)$$

Donde R es el número de imágenes de rostros en el conjunto de entrenamiento y $T^{(r)}$ es el número de bloques extraídos desde la rth imagen. Un nuevo conjunto de vectores son obtenidos por sustracción entre las imágenes y el promedio:

$$\hat{x}_i^{(r)} = x_i^{(r)} - u \qquad (6.22)$$

3. Los vectores $\hat{x}_i^r$ son caracterizados por promedio estadistico cero, ellos pueden ser usados para determinar un conjunto basico de funciones para KTL.
4. La matriz de covarianza de los vectores $\hat{x}_i^r$ es calculado como:

$$C = \frac{1}{\sum_{r=1}^{R} T^{(r)}} \sum_{r=1}^{R} \sum_{i=1}^{T(r)} \hat{x}_i^{(r)} (\hat{x}_i^{(r)})^T \qquad (6.23)$$

y los eigenvectores υ_k de la matriz de covarianza son determinados por $C\upsilon_k = \lambda_k \upsilon_k$ con el eigenvalor λ_k correspondiente al Eigenvector de la matriz de covarianza de C.

6.3.2 Transformada Coseno Discreto de Fourier (DCT)

La Transformada de Coseno Discreto de Fourier (DCT) utiliza funciones base sinusoidales. Estas funciones no son complejas a diferencia de las transformadas de Fourier ya que solo emplean funciones coseno. Si consideramos una imagen de dimensión NxN la transformada discreta del coseno es tal como se muestra en la ecuación (6.24).

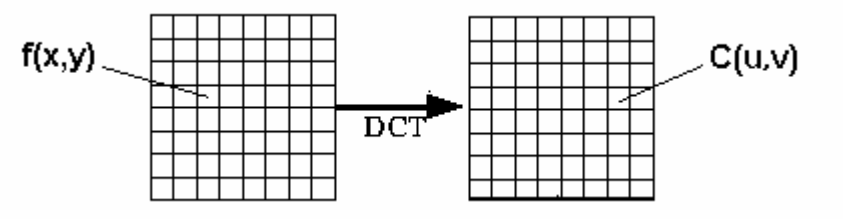


Figura 6.9. *Transformada de Coseno Discreto*

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)\pi.u}{2N}\right]\cos\left[\frac{(2y+1)\pi.v}{2N}\right] \qquad (6.24)$$

Para u,v=0,1,2, ..N-1, y donde:

$$\alpha(u),\alpha(v) = \begin{cases} \sqrt{\frac{1}{N}} & para \quad u,v = 0 \\ \sqrt{\frac{2}{N}} & para \quad u,v = 1,2,..,N-1 \end{cases} \qquad (6.25)$$

Generalmente esta transformada es usada para compresión de imágenes, especialmente en el método JPEG (Joint Photographic Experts Group) y considerado como un estándar internacional.

Las matrices bases se representan como imágenes, llamadas imágenes base donde se toma varios valores de gris para representar los diferentes valores en las matrices base.

Las imágenes base 2-D para la transformada del coseno estan dadas en la figura () para una imagen 8x8 (N=8) cada bloque consta de 8x8 elementos, correspondientes a la variación de x e y de 0 a 7. El origen de cada bloque se situa en la parte superior izquierda. Los valores máximos se muestran en blanco (nivel de gris: 255) y valores

mínimos en negro (nivel de gris: 0). El resto corresponde a los valores intermedios del nivel de gris.

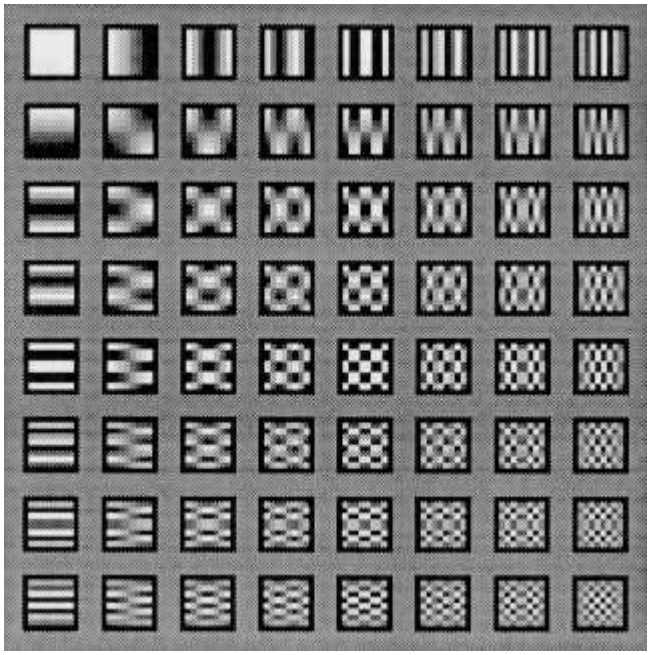


Fig. 6.10. *Imágenes base para la transformada de coseno para N=8.*

6.5 Algoritmo de Viterbi para HMME.

Si tenemos una secuencia de observaciones **O**, entonces el objetivo del algoritmo de decodificación es la de recuperar el conjunto de estados **q** mas probables que maximicen $P(\mathbf{O},\mathbf{q}\mid\lambda)$. Para describir un algoritmo eficiente que recupere el conjunto de estados, se define la siguiente variable:

$$\delta_{t_0}(i)=\max_{\mathbf{q}_{t_0}^{i}}\max_{\mathbf{q}_1,\ldots\mathbf{q}_{t0-1}} P(\mathbf{q}_1,\ldots,\mathbf{q}_{t_0-1},q_{t_0}^{0}=i,\mathbf{O}_1,\ldots,\mathbf{O}_{t_0}\mid\lambda) \qquad (6.26)$$

De la ecuación (99), $\delta_{t_0}(i)$ representa la mejor probabilidad de observación y secuencia de estado a lo largo de una sola trayectoria del estado que termina en $\mathbf{O}_{t_0}$ dentro del súper estado i. Esta ecuación muestra claramente que $P(\mathbf{O},\mathbf{q}\mid\lambda)=\max_i \delta_{T_0}(i)$. El algoritmo de decodificación será eficiente haciendo uso de la variable $\delta_{t_0+1}(i)$ del valor previo $\delta_{t_0}(i)$ de la ecuación anterior.

$$\delta_{t_0+1}(i) = \max_j \left[\max_{\mathbf{q}^1_{t_0+1}} \max_{\mathbf{q}_1,...\mathbf{q}_{t_0}} P(\mathbf{q}_1,...,\mathbf{q}_{t_0-1}, q^0_{t_0} = j, \mathbf{O}_1,...,\mathbf{O}_{t_0} \mid \lambda) a_{0,ji} P(\mathbf{q}^1_{t_0+1}, \mathbf{O}_{t_0+1}, q^0_{t_0+1} = i, \lambda) \right] \quad (6.27)$$

La ecuación anterior (3.6) puede ser escrita como:

$$\delta_{t_0+1}(i) = \max_j \left[\max_{\mathbf{q}^1_{t_0}} \max_{\mathbf{q}_1,...\mathbf{q}_{t_0-1}} \left[P(\mathbf{q}_1,...,\mathbf{q}_{t_0-1}, q^0_{t_0} = j, \mathbf{O}_1,...,\mathbf{O}_{t_0} \mid \lambda) a_{0,ji} \right] \right] \text{x}$$

$$\max_{\mathbf{q}^1_{t_0+1}} \left[P(\mathbf{q}^1_{t_0+1}, \mathbf{O}_{t_0+1}, q^0_{t_0+1} = i, \lambda) \right] \quad (6.28)$$

Entonces:

$$\delta_{t_0+1}(i) = \max_{all\_j} \left[\delta_{t_0}(j) a_{0,ji} \right] \max_{\mathbf{q}^1_{t_0+1}} P(\mathbf{q}_{t_0+1}, \mathbf{O}_{t_0+1} \mid q^0_{t_0+1} = i, \lambda) \quad (6.29)$$

Para poder evaluar $\delta_{t_0}(i)$ es necesario usar el algoritmo de Viterbi para la secuencia $\mathbf{O}_{t_0}$ y recuperar la mejor secuencia de estados embebidos correspondiente al súper estado i. Entonces, el algoritmo de Viterbi es utilizado para las secuencias $O_1,...,O_{T_0}$. Los siguientes pasos para el algoritmo de decodificación son:

Paso 1:

Calcular $P^i_{t_0} = \max_{q^1_{t_0}} P(O_{t_0}, q^1_{t_0} \mid q^0_{t_0} = i, \lambda)$ usando el algoritmo de Viterbi para cada súper estado y para cada secuencia de observación $\mathbf{O}_{t_0}$.

Paso 2:

(a) Inicialización

$$\delta_1(i) = \pi_{0,1} P^i_0 \quad (6.30)$$

$$\psi_1(i) = 0 \quad (6.31)$$

(b) Recursión

$$\delta_1(i) = \max_{j \in [1, N_0]} \left[\delta_{t_0-1}(j) a_{0,ji} \right] P^i_0 \quad (6.32)$$

$$\psi_1(i) = \arg \max_{j \in [1, N_0]} \left[\delta_{t_0-1}(j) a_{0,ji} \right] \quad (6.33)$$

(c) Terminación

$$P^* = \max_{j \in [1, N_0]} \delta_{T_0}(j) \quad (6.34)$$

$$\left(q_{T_0}^{0}\right)^{*} = \arg\max_{j\in[1,N_0]}\left[\delta_{T_0}(j)\right] \qquad (6.35)$$

P^{*} En la ecuación representa $\max_{all\mathbf{q}} P(\mathbf{O},\mathbf{q}\mid\lambda)$ y $(q_{T_0}^{0})^{*}$ es el súper estado en T_0 sobre la ruta **q** tal que maximice $P(\mathbf{O},\mathbf{q}\mid\lambda)$.

(d) Vuelta hacia atrás

La mejor ruta de **q**, que maximice $P(\mathbf{O},\mathbf{q}\mid\lambda)$ es obtenida desde el arreglo $\psi_{t_0}(i)$ como lo siguiente:

$$\left(q_{t_0}^{0}\right)^{*} = \psi_{t_0+1}\left(\left(q_{t_0+1}^{0}\right)^{*}\right) \qquad (6.36)$$

Para reducir la complejidad del algoritmo señalado y evitar un problema de desbordamiento que a menudo ocurre en los cálculos que involucran un número largo de multiplicaciones, todos estos términos pueden ser calculados con estos algoritmos mencionados. En esta representación todas las probabilidades son reemplazadas por valores logarítmicos y las multiplicaciones son transformadas en sumas. Debido para la monotonicidad de la función logarítmica, la máxima puede ser obtenida por los mismos argumentos, pero por supuesto en su forma logarítmica.

El tiempo empleado para la decodificación puede ser significativamente reducida cuando se usa una arquitectura en paralelo. Así mostramos en la figura 3.3 en la que todos los valores de Viterbi obtenidos en el primer paso del algoritmo puede ser calculada independientemente entre súper estados y por lo tanto todos esos valores pueden ser calculados al mismo tiempo usando la implementación paralela.

Por tanto el retardo total introducido para la decodificación del algoritmo puede ser reducida en aproximadamente dos veces el retardo introducido por el cálculo de un algoritmo de Viterbi Standard.

6.6. Algoritmos de evaluación

Ahora describiremos los dos algoritmos para calcular la probabilidad $P(\mathbf{O}\mid\lambda)$ de la secuencia de observación **O** dado el modelo λ. Estos son similares a los algoritmos empleados en HMM 1D.

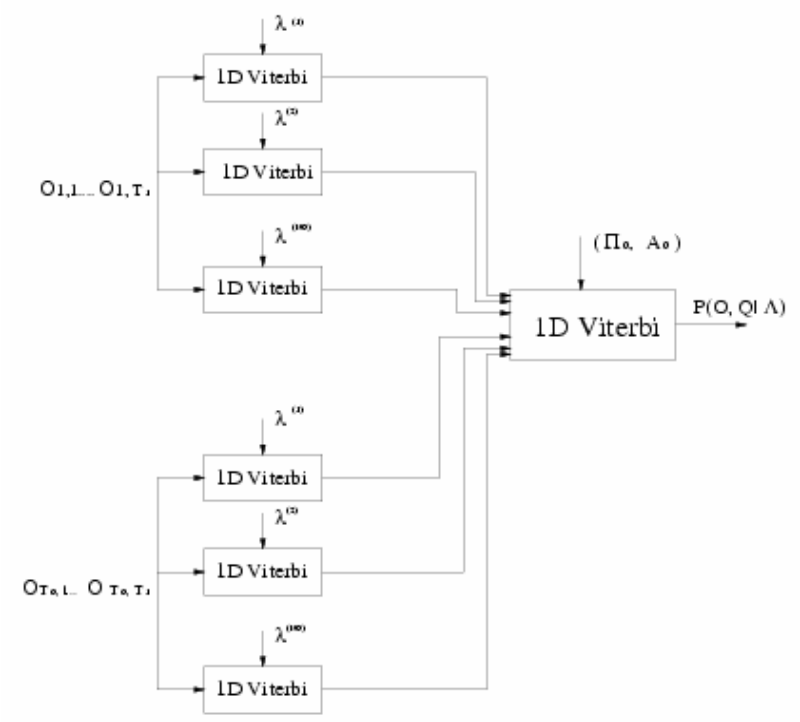


Fig. 6.11 *Uso de Viterbi en paralelo*

6.7.1 Algoritmo Forward para HMME

Vamos a denotar la variable para la secuencia $\mathbf{O}_{t_0}$ como:

$$\alpha_{t_0,t_1}(i,j) = P(\mathbf{O}_{t_0,0},...,\mathbf{O}_{t_0,t_1}, q^1_{t_0,t_1} = j \mid q^0_{t_0} = i, \lambda) \tag{6.37}$$

Similar a la ecuación anterior, la variable Backward de una secuencia $\mathbf{O}_{t_0}$ esta dada por:

$$\beta_{t_0,t_1}(i,j) = P(\mathbf{O}_{t_0,t_1+1},...,\mathbf{O}_{t_0,T_1}, q^1_{t_0,t_1} = j \mid q^0_{t_0} = i, \lambda) \tag{6.38}$$

La variable Forward y Backward son calculadas Iterativamente usando el algoritmo Forward y Backward para un HMM 1D. La probabilidad de la secuencia de observación $\mathbf{O}_{t_0}$ dado para un HMM1D correspondiente al súper estado *i* es calculada con $\alpha_{t_0,t_1}(i,j)$ y $\beta_{t_0,t_1}(i,j)$ de la siguiente manera:

$$P(\mathbf{O}_{t_0} \mid q^0_{t_0} = i, \lambda) = \sum_{t_1} \alpha_{t_0,t_1}(i,j)\beta_{t_0,t_1}(i,j) \tag{6.39}$$

Un eficiente algoritmo para calcular $P(\mathbf{O} \mid \lambda)$ se puede obtener si la variable Forward para la secuencia $O_0, O_1,..., O_{T_0}$ es definido como:

$$\alpha_{t_0}(i) = P(\mathbf{O}_0,...,\mathbf{O}_{t_0}, q_{t_0}^0 = i \mid \lambda) \tag{6.40}$$

La variable Forward $\alpha_{t_0}(i)$ describe la probabilidad de una secuencia parcial $\mathbf{O}_0,...,\mathbf{O}_{t_0}$ y súper estado i dado el modelo λ. Por tanto, $P(\mathbf{O} \mid \lambda)$ puede ser calculado como:

$$P(\mathbf{O} \mid \lambda) = \sum_{\forall i} \alpha_{t_0}(i) \tag{6.41}$$

La variable Forward $\alpha_{t_0}(i)$ es calculado iterativamente con estos valores previos y la probabilidad de $\mathbf{O}_{t_0}$ dado el súper estado *i*, $P(\mathbf{O}_{t_0} \mid q_{t_0}^0 = i, \lambda)$:

$$\alpha_{t_0+1}(i) = \left[\sum_i \alpha_{t_0}(i) a_{0,ij} \right] P(O_{t_0} \mid q_{t_0}^0 = i, \lambda) \tag{6.42}$$

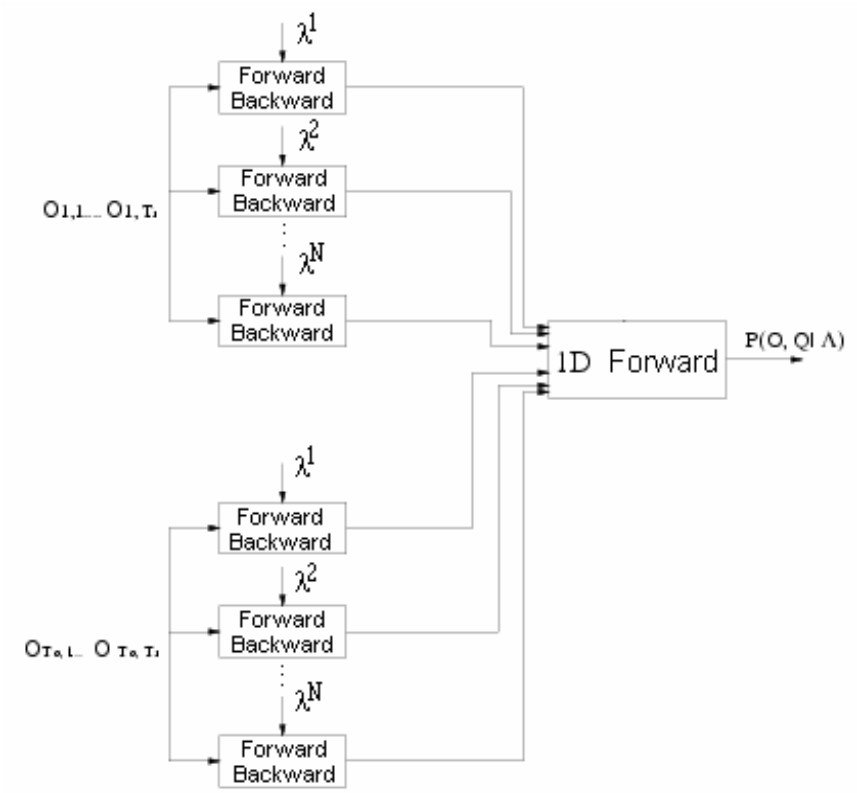


Fig. 6.12 *Forward para HMME*

El algoritmo Forward para un HMM Embebido puede ser formalmente descrito por los siguientes pasos:

Paso1:

Calcular $P_{t_0}^i = P(\mathbf{O}_{t_0,t_1} \mid q_{t_0}^0 = i, \lambda)$ para todo t_0 y súper estado i usando el algoritmo Forward-Backward de un HMM 1D.

Paso 2:

(a) Inicialización

$$\alpha_0(i) = \pi_{0,i} P_0^i \tag{6.43}$$

(b) Recursión

$$\alpha_{t_0+1}(i) = \left[\sum_i \alpha_{t_0}(i) a_{0,ij} \right] P_{t_0}^i \tag{6.44}$$

(c) Terminación

$$P(O_0, O_1, ..., O_{T_0} \mid \lambda) = \sum_i \alpha_{T_0}(i) \tag{6.45}$$

6.5.2 Algoritmo Backward para HMME

De manera similar para la variable forward definido en la ecuación ( ). Una variable Backward puede ser definida para la secuencia $\mathbf{O}_{t_0}$:

$$\beta_{t_0}(i) = P\left(\mathbf{O}_{t_0+1}, ..., \mathbf{O}_{T_0} \mid q_{t_0}^i = i, \lambda\right) \tag{6.46}$$

La variable Backward $\beta_{t_0}(i)$ define la probabilidad de una secuencia de observación parcial $\mathbf{O}_{t_0}, ..., \mathbf{O}_{T_0}$ dado que $\mathbf{O}_{t_0-1}$ en un súper estado *i.* La variable puede ser calculada en forma iterativa para valores futuros $\beta_{t_0+1}(i)$ y $P(\mathbf{O}_{t_0} \mid q_{t_0}^0 = i, \lambda)$.

$$\beta_{t_0}(i) = \sum_j a_{0,ij} P(O_{t_0+1} \mid q_{0,t_0+1} = j, \lambda) \beta_{t_0+1}(j) \tag{6.47}$$

Por tanto el algoritmo Backward consiste de los siguientes pasos:

Paso1:

Calcular $P_{t_0}^{i} = P(O_{t_0,t_1} \mid q_{t_0}^{0} = i, \lambda)$ para todo t_0 y súper estado *i* usando el algoritmo forward Backward para HMM 1D.

Paso 2:

(a) Inicialización

$$\beta_{T_0}(i) = 1 \tag{6.48}$$

(b) Recursión

$$\beta_{t_0}(i) = \left[\sum_j a_{0,ij} P_{t_0}^{i} \beta_{t_0+1}(j) \right] \tag{6.49}$$

Esto es similar al algoritmo decodificador, el tiempo requerido para la evaluación del algoritmo puede ser reducido si se usa una arquitectura en paralelo.

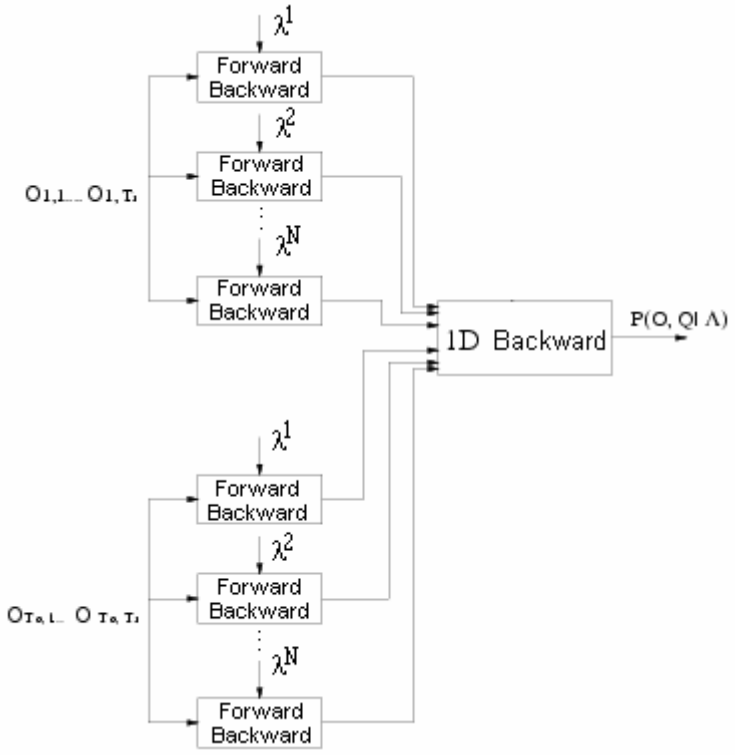


Fig. 6.13. *Backward para HMME*

6.8 Optimización del Algoritmo.

6.8.1 HMME Discreto y continuo.

El objetivo de la reestimación es encontrar un nuevo conjunto de parámetros de HMM Embebido que maximice $P(\mathbf{O}\,|\,\lambda)$. Esto es equivalente a maximizar la función auxiliar definida como:

$$Q(\lambda,\bar{\lambda}) = \frac{1}{P(\mathbf{O}/\lambda)}\left[\sum_{\mathbf{q}} P(\mathbf{O},\mathbf{q}\,|\,\lambda)\log P(\mathbf{O},\mathbf{q}\,|\,\lambda)\right] \qquad (6.50)$$

Con respecto a λ. Este resulta de la estructura HMME donde la probabilidad de la secuencia de observación **O** y la secuencia de estados **q** esta dado por:

$$P(\mathbf{O},\mathbf{q}\,|\,\lambda) = \pi_{0,q_1^0}\prod_{t_0} a_{0,q_{t_0-1}^0,q_{t_0}^0} P(\mathbf{O}_{t_0}\,|\,q_{t_0}^0,\lambda) \qquad (6.51)$$

Donde,

$$P(\mathbf{O}_{t_0}\,|\,q_{t_0}^0,\lambda) = \pi^{q_{t_0,1}^0}_{1,q_{t_0,1}^1}\prod_{t_0} a^{q_{t_0}^0}_{1,q_{t_0,t_1-1}^1,q_{t_0,t_1}^1} b^{q_{t_0,t_1}^0}_{q_{t_0,t_1}^1}(\mathbf{O}_{t_0,t_1}) \qquad (6.52)$$

Usando la el algoritmo de Baum-Welsh la ecuación de reestimación de parámetros de un HMME viene dado de la siguiente manera:

$$\pi_{0,i} = \frac{\dfrac{P(\mathbf{O},q_1^0 = i\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}}{\displaystyle\sum_{i=1}^{N_0}\frac{P(\mathbf{O},q_1^0 = i\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}} \qquad (6.53)$$

$$a_{o,ij} = \frac{\displaystyle\sum_{t_0=1}^{T_0}\frac{P(\mathbf{O},q_{t_0-1}^0 = i, q_{t_0}^0 = j\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}}{\displaystyle\sum_{t_0=1}^{T_0}\frac{P(\mathbf{O},q_{t_0-1}^0 = i\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}} \qquad (6.54)$$

$$\pi_{1,j}^{i} = \frac{\displaystyle\sum_{t_0=1}^{T_0}\frac{P(\mathbf{O},q_{t_0,1}^1 = j, q_{t_0,1}^0 = i\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}}{\displaystyle\sum_{t_0=1}^{T_0}\frac{P(\mathbf{O},q_{t_0,1}^1 = i\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}} \qquad (6.55)$$

$$a_{j,l}^{i} = \frac{\displaystyle\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\frac{P(\mathbf{O},q_{t_0,1-1}^1 = j, q_{t_0,t_1}^1 = l, q_{t_0}^0 = i\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}}{\displaystyle\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\frac{P(\mathbf{O},q_{t_0}^0 = i, q_{t_0,t_1-1}^1 = j\,|\,\lambda)}{P(\mathbf{O}\,|\,\lambda)}} \qquad (6.56)$$

$$b_j^i(k) = \frac{\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1} \frac{P(\mathbf{O}, q^1_{t_0,t_1} = j, q^0_{t_0,t_1} = i \mid \lambda)\delta(\mathbf{O}_{t_0,t_1,v_k})}{P(\mathbf{O} \mid \lambda)}}{\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1} \frac{P(\mathbf{O}, q^1_{t_0,t_1} = i, q^0_{t_0,t_1} = j \mid \lambda)}{P(\mathbf{O} \mid \lambda)}} \qquad (6.57)$$

Esta ecuación de re-estimación es obtenida para una secuencia de observación **O**, pero en general la re-estimación de parámetros debe ser para múltiples secuencias de observación, que puede ser asumido independientemente.

6.8.2 Algoritmo Baum-Welsh

Este método asume un modelo HMM inicial el cual es mejorado usando las fórmulas (dadas debajo) y de esta manera maximizar $P(O \mid \lambda)$. Un modelo inicial HMM puede ser construido un alguna manera pero se usa los 5 primeros pasos del algoritmo de segmentación de k-Medias ya que nos da una estimación razonable del modelo inicial de HMM. De aquí en adelante se asume que el modelo inicial HMM es conocida. Este algoritmo maximiza $P(O \mid \lambda)$ ajustando los parámetros de λ. Este criterio de optimización es llamado también como Criterio de máxima probabilidad. La función $P(O \mid \lambda)$ es llamada función de probabilidad. Antes se introducirá a unos conceptos importantes.

Consideramos a $\gamma_t(i)$ definida como:

$$\gamma_t(i) = P(i_t = i \mid O, \lambda) \qquad (6.58)$$

Por ejemplo la probabilidad al empezar en el estado *i* de longitud *t* dado las secuencia de observación: $O = O_1, O_2, ..., O_T$ y el modelo $\lambda = (A, B, \pi)$. Por Bayes se tiene:

$$\gamma_t(i) = \frac{P(i_t = i, O \mid \lambda)}{P(O \mid \lambda)} \qquad (6.59)$$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)} \qquad (6.60)$$

Donde $\alpha_t(i)$ y $\beta_t(i)$ fueron definidas anteriormente. $\alpha_t(i)$ da explicación sobre $O_1, ..., O_t$ y el estado i en un tiempo t, y $\beta_t(i)$ da explicación sobre $O_{t+1}, ..., O_T$ dado el estado i de tiempo t.

Se define $\xi_t(i,j)$ como:

$$\xi_t(i,j) = P(i_t = i, i_{t+1} = j \mid O, \lambda) \tag{6.61}$$

Por ejemplo la probabilidad al empezar en el estado i de longitud t y construir una matriz de transición hacia el estado j en un tiempo t+1, dado la secuencia de observación $O = O_1, O_2, ..., O_T$ y el modelo $\lambda = (A, B, \pi)$. Usando Bayes vemos que:

$$\xi_t(i,j) = \frac{P(i_t = i, i_{t+1} = j, O \mid \lambda)}{P(O \mid \lambda)} \tag{6.62}$$

Pero $O = O_1, O_2, ..., O_T$. Por lo tanto:

El Numerador = $P(i_t = i, O_1, ..., O_t, O_{t+1}, ..., O_T, i_{t+1} = j \mid \lambda)$

$$P(i_t = i, O_1, ..., O_t \mid \lambda) P(O_{t+1}, ..., O_T, i_{t+1} = j \mid \lambda)$$

(Dado que la cadena de markov es casual)

Si consideramos el segundo termino. El símbolo de observación en el tiempo t+1 es O_{t+1} a sido escogido desde el estado j. Por lo tanto tenemos:

$$P(O_{t+1}, ..., O_T, i_{t+1} = j \mid \lambda) = P(i_{t+1} = j, O_{t+1} \mid \lambda) P(O_{t+1}, ..., O_T \mid i_{t+1} = j, \lambda)$$

$$= a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \tag{6.63}$$

(Dado que $i_t = i$ es conocido)

Por lo tanto combinando las anteriores ecuaciones tenemos:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)} \tag{6.64}$$

Aquí $\alpha_t(i)$ da explicación sobre $O_1, ..., O_t$, a_{ij} da explicación sobre la transición de estado j, $b_j(O_{t+1})$ que adquirió el símbolo O_{t+1} desde el estado j, y $\beta_{t+1}(j)$ da explicación de la secuencia de observación faltante $O_{t+2}, ..., O_T$.

Si resumimos $\gamma_t(i)$ desde $t = 1$ hacia T, obtenemos una cantidad que puede ser vista como un número esperado de estados de tiempo i es visitado o si resumimos solo para T-1 entonces obtendremos el esperado número de transición de salidas del estado i

(Como no hay transición hecha en t = T. Similarmente si $\xi_t(i,j)$ ser adicionado desde t=1 hacía T-1, obtendremos el numero esperado de transiciones desde el estado i hasta el estado j. Por lo tanto:

$\sum_{t=1}^{T-1}\gamma_t(i)$ = Número esperado de transiciones desde el estado i

$\sum_{t=1}^{T-1}\xi_t(i,j)$ = Número esperado de transiciones desde el estado i hacia el estado j.

Ahora se presenta la formula de reestimación Baum-Welch

$$\pi_i = \gamma_t(i), \qquad 1 \le i \le N \tag{6.65}$$

$$a_{ij} = \sum_{t=1}^{T-1}\xi_t(i,j) / \sum_{t=1}^{T-1}\gamma_t(i) \tag{6.66}$$

$$b_j(k) = \sum_{\substack{t=1 \\ O_t=k}}^{T}\gamma_t(j) / \sum_{t=1}^{T}\gamma_t(j) \tag{6.67}$$

La formula de reestimación para π_i es la probabilidad de empezar en estado i en t. La formula para a_{ij} es la razón entre el numero de ocurrencias esperados al hacer una transición desde el estado i al estado j y el número de ocurrencias esperado al hacer una transición del estado i en todo t. La formula para $b_k(i)$ es la razón entre el número esperado de tiempos al empezar en el estado j y observar el símbolo O_k y el numero esperado en el estado j. Notar que la suma total en la formula pasada es hasta t=T.

Si denotamos el modelo inicial dado por λ y re-estimamos el modelo por $\hat{\lambda}$. Entonces tendremos dos casos:

El modelo inicial λ es un punto critico en la función de probabilidad en el caso en que $\hat{\lambda} = \lambda$ o $P(O|\hat{\lambda}) > P(O|\lambda)$, aquí encontramos un mejor modelo de la secuencia de observación $O = O_1, O_2, ..., O_T$ y es mas probable que se produzca.

Por lo tanto podemos iterativamente calcular hasta que $P(O|\lambda)$ sea máxima. Y esta seria la solución al problema.

Hemos visto que el algoritmo de Baum-Welch maximiza $P(O|\lambda)$; Cada termino en la suma y de la ecuación esta entre 0 y 1 y por lo tanto la salida (que es el producto de

muchos términos) va ser de un valor pequeño y esto se debe a los términos y al numero de posibles secuencias de estados.

Para una computadora es posible computar tales términos individualmente, pero hay que tomar en cuenta el tipo de datos a almacenar. Si T=100 donde son alrededor de 200 números (cada numero entre 0 y 1) al ser multiplicado cada termino puede ir mas allá de la precisión de la computadora. En este caso el algoritmo no ayuda ya que $P(O|\lambda)$ es la sumatoria de pequeños valores.

CAPITULO VII
ENTRENAMIENTO RECONOCIMIENTO DE ROSTROS

7.1 Entrenamiento y reconocimiento por PCA.

Para el entrenamiento se toma primero como referencia las bases de datos de imágenes de rostros externos que son utilizados para reconocedores de rostros, para ello se dispone de 10 fotos de 40 personas.

Figura 7.1 Base de Datos de Yale [10]

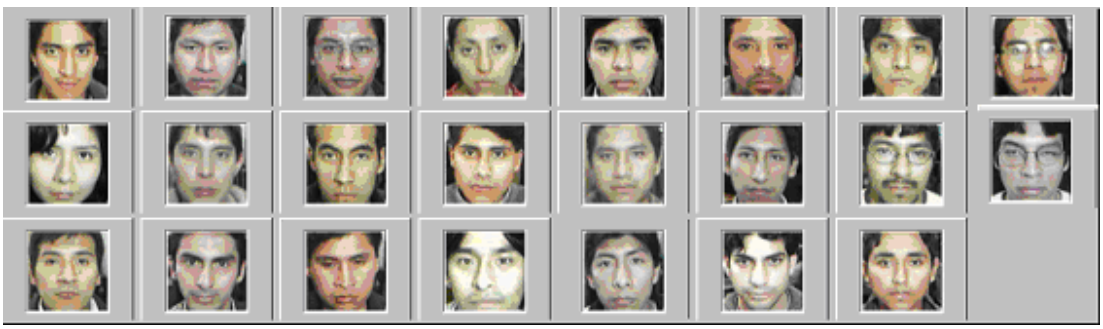


Figura 7.2 *Imágenes tomadas en la UNI*

Luego usaremos la base de datos de rostros creado con los integrantes del Centro de Investigación y Desarrollo CID-FIEE que contienen 460 fotos.

Antes de pasar a entrenar estos rostros primero se procede a implementar en Matlab la entrada de estas imágenes de rostros, la idea es que los rostros deben estar normalizados en tamaño y orientación.

La resolución de las fotos de Yale es de 112x92 en formato gif. La resolución de las fotos tomadas en la UNI es de 140x140

Las fotos de rostros tienen características diferentes. Así tenemos las de gesto normal, gestos Sonrientes o alegres, gesto serio o amargo, gesto particular o propio de la persona. La Fig. 7.3 muestra los gestos de cada persona.

Fig. 7.3 *Imágenes de rostros de una persona*

7.1.2 Formación de Clases y Patrones

Para el caso del método de Eigenfaces solo se necesita una imagen de rostro por persona, esta debe ser representativa de la misma.

Para el caso del método de Fisherfaces es necesario formar las clases y patrones. Clases se refiere a la persona y patrones por clase se refiere a los rostros de esta persona.

En el primer caso las imágenes de rostros se llevan a vector columna, tal como se muestra en la Fig. 7.4.

7.1.2 Parámetros de entrenamiento

El software para el entrenamiento se realizó en Matlab, este consistía en adquirir la foto con sus datos personales y luego ir almacenando en una matriz como un arreglo, La Fig. 7.6 muestra la interface.

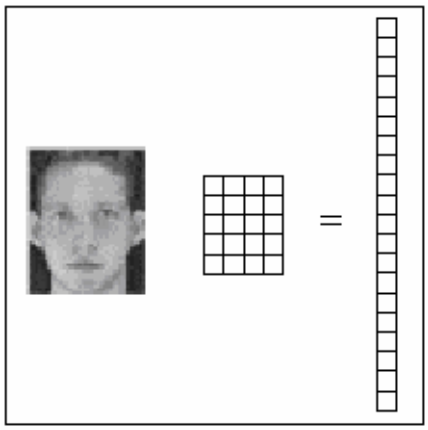


Fig. 7.4. *Representación de una imagen en vector columna*

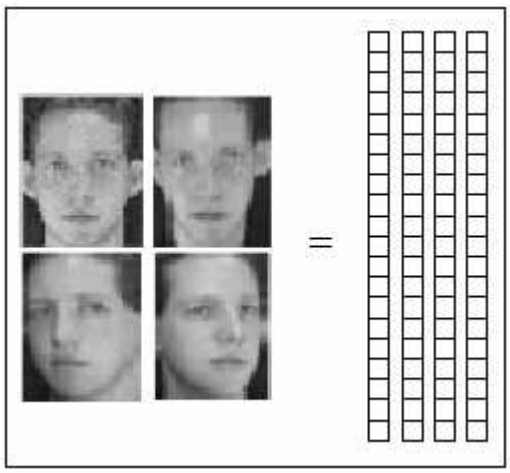


Fig. 7.5. *Representación de una clase en una matriz*

Fig. 7.6. *Etapa de Entrenamiento del sistema con Eigenfaces*

Una vez que las imágenes son almacenadas en una Base de Datos (BD), se procede a entrenar el sistema con el algoritmo para eigenfaces o fisherfaces. En los dos casos es necesario definir la cota de error residual RMSE que define la cantidad de eigenvalores que va a tomar el algoritmo.

7.1.3 Reconocimiento de Rostros por Eigenfaces

Siguiendo el esquema presentado en la figura (80), las imágenes se empiezan a entrenar siguiendo un orden. Antes de formar la matriz de imágenes se debe hallar el promedio del mismo, la Fig. 7.8 muestra el promedio de imágenes de 23 rostros del CID-FIEE

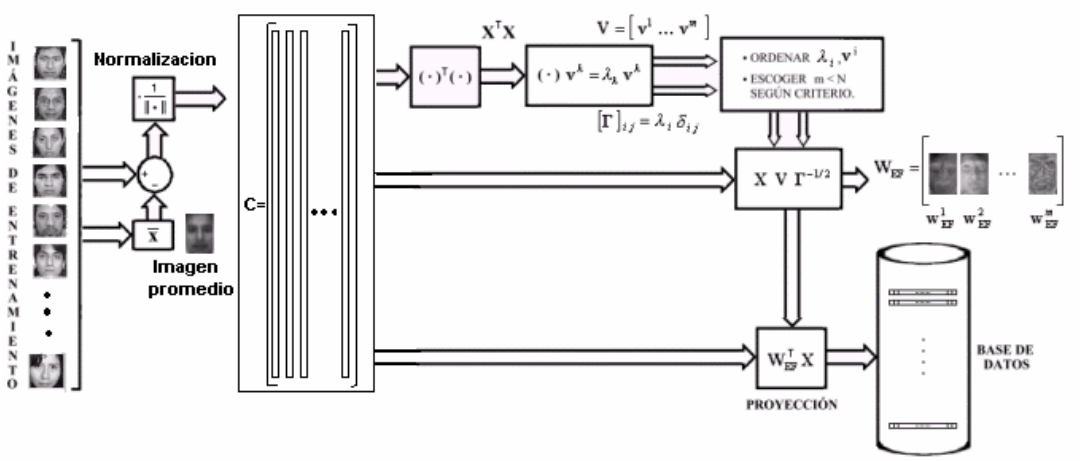


Fig. 7.7. *Esquema del algoritmo de Eigenfaces*

Fig. 7.8 *Imagen promedio de 23 rostros*

Una vez obtenida la matriz de imágenes de rostros se procede a aplicar el algoritmo de Eiganfaces, el resultado debe ser las Eigenfaces W_{EF} (Fig. 82), que son elegidas de acuerdo a los eigenvalores (Fig. 83) de mayor valor y que dependen del parámetro RMSE elegido (típicamente < 1%). Luego la proyección de los rostros sobre

esta nueva base nos dará las componentes principales que serán guardados en la base de datos y utilizados para el reconocimiento de rostros.

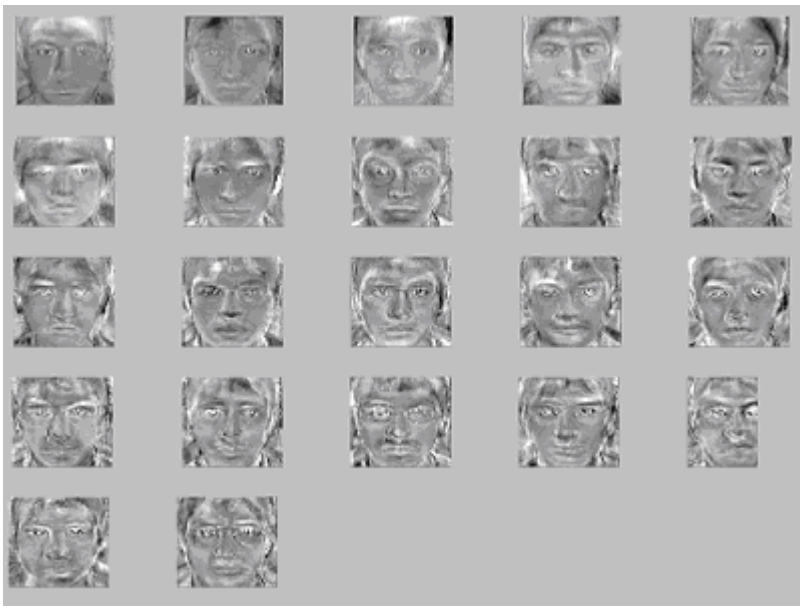


Fig. 7.9. *Eigenrostros de la base entrenada*

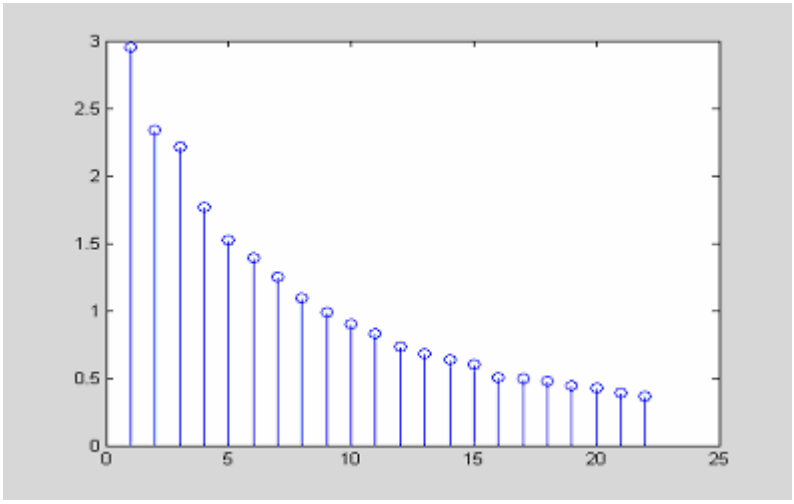


Fig. 7.10. *Eigenvalores de la base entrenada*

7.1.4 Reconocimiento de Rostros por Fisherfaces.

Para este proceso se realizó un algoritmo que satisfaga el método de Fisher en Matlab, por lo que se tiene que tener una base de datos de muchas personas con diferentes gestos y además que sean representativos de cada individuo, la Fig. 7.11 muestra la interface utilizada para el método de Fisherface.

Fig. 7.11. Etapa de entrenamiento

Después del cálculo de las Eigenfaces y luego de las Fiherfaces, el espacio del rostro ha sido poblado con rostros conocidos tomados del conjunto de entrenamiento, Cada rostro conocido es transformado al espacio de rostros y sus componentes almacenados en memoria.

Para iniciar el proceso de reconocimiento, un rostro desconocido es presentado al sistema. El rostro es identificado como el rostro más cercano en el espacio de rostros, para calcular la distancia entre vectores multidimensionales se uso una forma de la distancia Euclidiana.

Cuando se procede a realizar el entrenamiento con estos individuos, inicialmente se utilizó 5 individuos con 3 imágenes de cada uno de ellos y se obtuvo un resultado de 100% de reconocimiento. Por lo que se continúo aumentando la base de datos.

En una segunda aplicación se tomo 10 individuos con 3 imágenes de cada uno y se obtuvo un resultado de reconocimiento de 90%, luego se procedió con 15 individuos y el porcentaje de reconocimiento disminuyo a 84%. Y si aumentábamos mas individuos tomando 3 imágenes de cada uno, el porcentaje de reconocimiento disminuye. Pero si aumentábamos las imágenes de cada individuo (mas patrones por cada clase) esto hacia que el reconocimiento sea eficiente, ya que se probo con 10 individuos con 5 imágenes de cada uno, esto nos dio un resultado de 98%, entonces se realizo con 6 imágenes de cada uno, donde se obtuvo un porcentaje de 100% de reconocimiento.

Fig. 7.12 Etapa de Reconocimiento

7.2 Entrenamiento y reconocimiento por Modelos Ocultos de Markov Embebidos.

Para el entrenamiento con HMM se realizó un programa en Matlab que permite cargar un archivo con nombre de la persona para ser entrenada, este archivo contiene los nombres de los archivos de imágenes a ser entrenadas (de la misma persona). Una vez cargado, el sistema efectúa la segmentación (número de bloques por imagen) y calcula la media nula y varianza unitaria de los bloques para luego obtener mediante KTL los coeficientes que a la vez será parte del vector de observación.

Para proceder a entrenar con el modelo de markov, nosotros podemos decidir con cuantos estados, el número de coeficientes KTL y el número de mezcla.

El entrenamiento en Matlab toma 20 segundos, dependiendo de la cantidad de imágenes a tratar.

Para la etapa de entrenamiento se realizó un programa en matlab que permite cargar un archivo de texto que contiene los modelos entrenados. Una vez cargados los modelos, empezamos a abrir los archivos de imágenes de rostros y buscamos los rostros que estén más cercano a los modelos mediante el algoritmo de viterbi. Al final tendremos probabilidades de coincidencia en cada modelo.

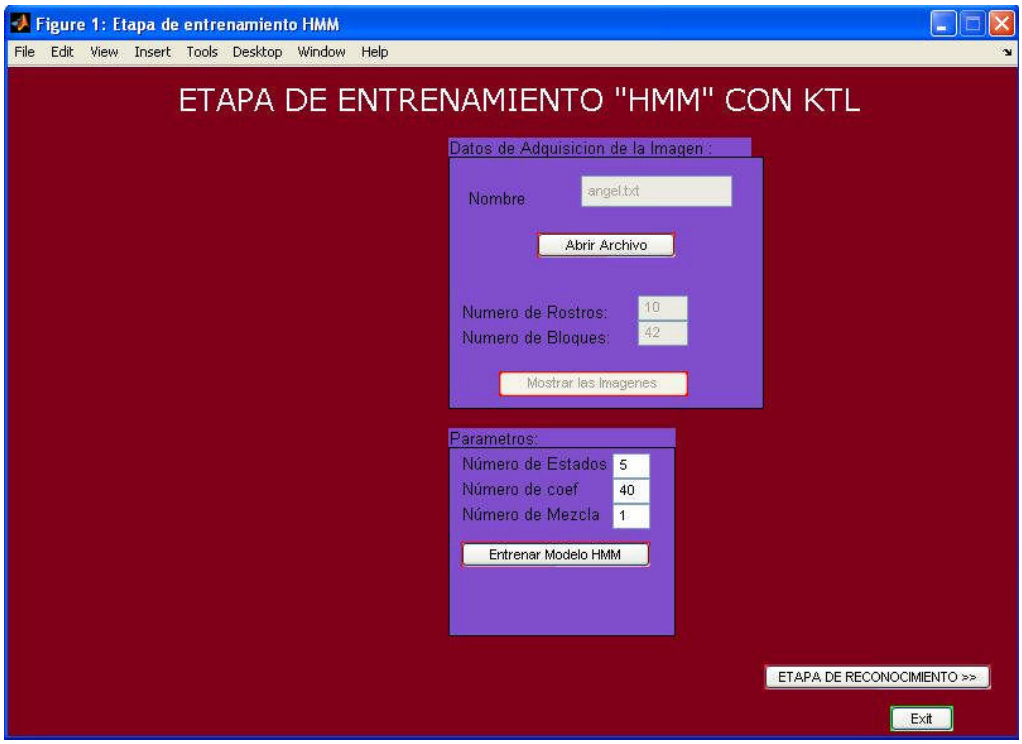


Fig. 7.13. *Etapa de entrenamiento*

Fig. 7.14 *Etapa de reconocimiento*

Para el entrenamiento con HMME se procedió a hacerlo en C++ ya que la idea era hacerlo en tiempo real, para ellos se utiliza librerías de OpenCV (Open Computer Vision) y la capacidad de DirectX, especialmente la componente de DirectShow que nos permite un mejor manejo de video.

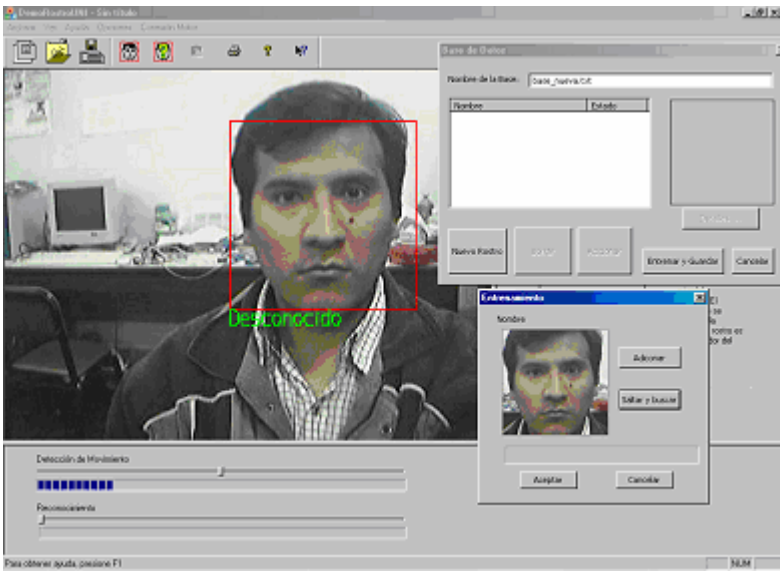


Fig. 7.15. *Etapa de entrenamiento con HMME utilizando Visual C++*

7.2.1 Formación de Clases y Patrones

Para la formación de clases y patrones, no es necesario que todos las imágenes esten en una matriz para poder entrenar, tal como sucedia en el caso de PCA. Aquí cada clase es conformada por patrones (rostros de una persona) luego del entrenamiento se crea un modelo. Este modelo esta conformada de acuerdo a la estructura dada con los parametros HMME.

Cada patrón es llevado como un vector de observación, este vector de observación es divido en grupos pequeños de la imagen. A estos grupos se les aplica la transformada discreta de Fourier y se toman los coeficientes de mayor valor de acuerdo a un orden establecido. De esta manera el vector de observación por ejemplo es formado por 6 coeficientes DCT (dos dimensiones en un arreglo de 3x2) de los 80 coefientes iniciales. Con los coeficientes DCT reducimos la sensibilidad del HMME al ruido, rotación de la imagen y cambios de iluminación. Un cambio de iluminación solo afecta a la primera componente del DCT (componente DC).

Las imágenes de entrada serán las tomadas del centro de investigación de la FIEE. Estas imágenes tienen un tamaño de 140x140.

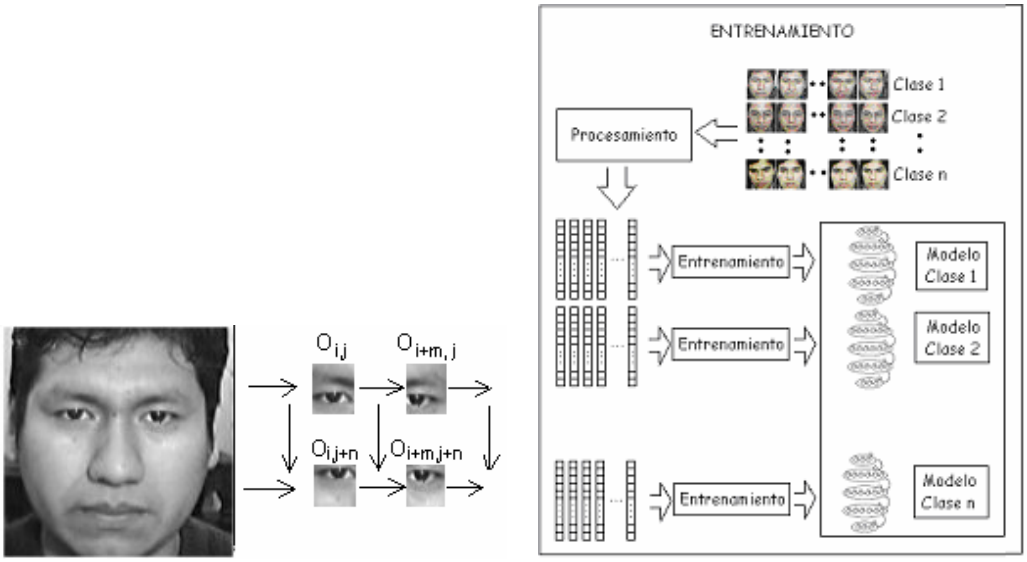


Fig. 7.16 a. Generación de vectores de observación **b.** Etapa de entrenamiento con HMME.

7.2.2 Parámetros del Entrenamiento

Los parámetros de entrenamiento se tomaron lo siguiente:

Número de Súper estados: 5

El primer súper estado tiene 3 estados, el segundo, tercero y cuarto súper estado contiene 6 estados y el último súper estado posee 3 estados. La Figura 90 muestra la arquitectura del HMME.

Los coeficientes de los vectores de observación de cada bloque fueron obtenidos por la transformada del coseno de Fourier DCT. De estos bloques de 8x10 píxeles mostrados en la Fig. 16a, se consideró 6 coeficientes.

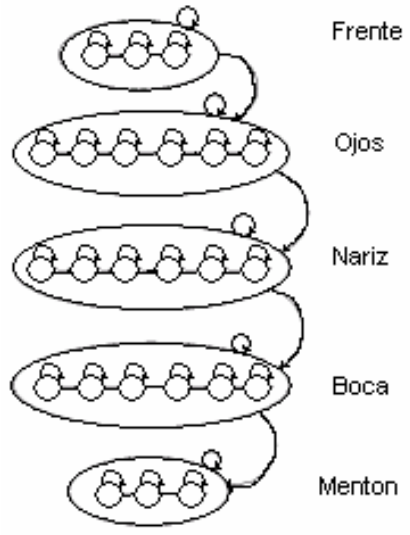


Fig. 7.17 *HMME para un rostro.*

7.2.3 Proceso del Entrenamiento

Para entrenar el HMM Embebido se usa el criterio de máxima probabilidad. El entrenamiento es similar a HMM 1D salvo la segmentación de Viterbi, procedimiento Forward y Backward y la reestimación Baum Welch. Estos algoritmos son reemplazados por la doble segmentación embebida, y los algoritmos explicados anteriormente.

Las imágenes de rostros tienen la misma características que las entrenadas por HMM 1D. Cada persona será representada en la base de datos mediante su modelo HMME.

Para entrenar se procede de la siguiente manera:

1. Los vectores de observación serán extraídos de cada individuo del conjunto de imágenes de entrenamiento.

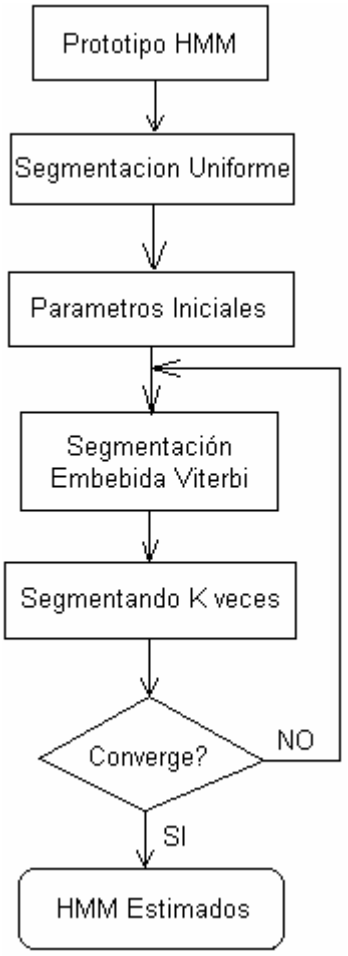


Fig. 7.18. *Entrenamiento de HMME*

2. En adición, el promedio estimado $\bar{\boldsymbol{\mu}}^{i}_{jm}$, la covarianza $\bar{\mathbf{U}}^{i}_{jm}$ de la mezcla gausiana, y el coeficiente de mezcla $\bar{c}^{i}_{jm}$ para m mezclas en el estado j del súper estado i son obtenidos como sigue:

$$\bar{\mu}^{i}_{jm} = \frac{\sum_{r=1}^{R}\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\psi^{i,r}_{jm}(t_0,t_1)\mathbf{O}(t_0,t_1)}{\sum_{r=1}^{R}\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\psi^{i,r}_{jm}(t_0,t_1)} \qquad (7.1)$$

$$\bar{\mathbf{U}}^{i}_{jm} = \frac{\sum_{r=1}^{R}\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\psi^{i,r}_{jm}(t_0,t_1)(\mathbf{O}(t_0,t_1)-\bar{\mu}^{i}_{jm})(\mathbf{O}(t_0,t_1)-\bar{\mu}^{i}_{jm})^{T}}{\sum_{r=1}^{R}\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\psi^{i,r}_{jm}(t_0,t_1)} \qquad (7.2)$$

$$\bar{c}^{i}_{jm} = \frac{\sum_{r=1}^{R}\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\psi^{i,r}_{jm}(t_0,t_1)}{\sum_{r=1}^{R}\sum_{t_0=1}^{T_0}\sum_{t_1=1}^{T_1}\sum_{m=1}^{M}\psi^{i,r}_{jm}(t_0,t_1)} \qquad (7.3)$$

Donde $\psi^{i,r}_{jm}(t_0,t_1)$ es igual a uno si $O(t_0,t_1)$ esta asociado con la componente m en el estado j del súper estado i, y es cero en otros casos.

3. La iteración se detiene y el HMM Embebido es inicializado esto cuando el doble Viterbi embebido alcanza un umbral.

4. Los parámetros del HMM Embebido son luego re-estimados usando el procedimiento Forward y Backward para maximizar la probabilidad de las observaciones dado el modelo del rostro. Esta estimación se detiene cuando llega a un cierto umbral.

7.2.3 Reconocimiento de rostro usando HMM Embebido

En la Fig. 7.19 se muestras las etapas del entrenamiento y reconocimiento:

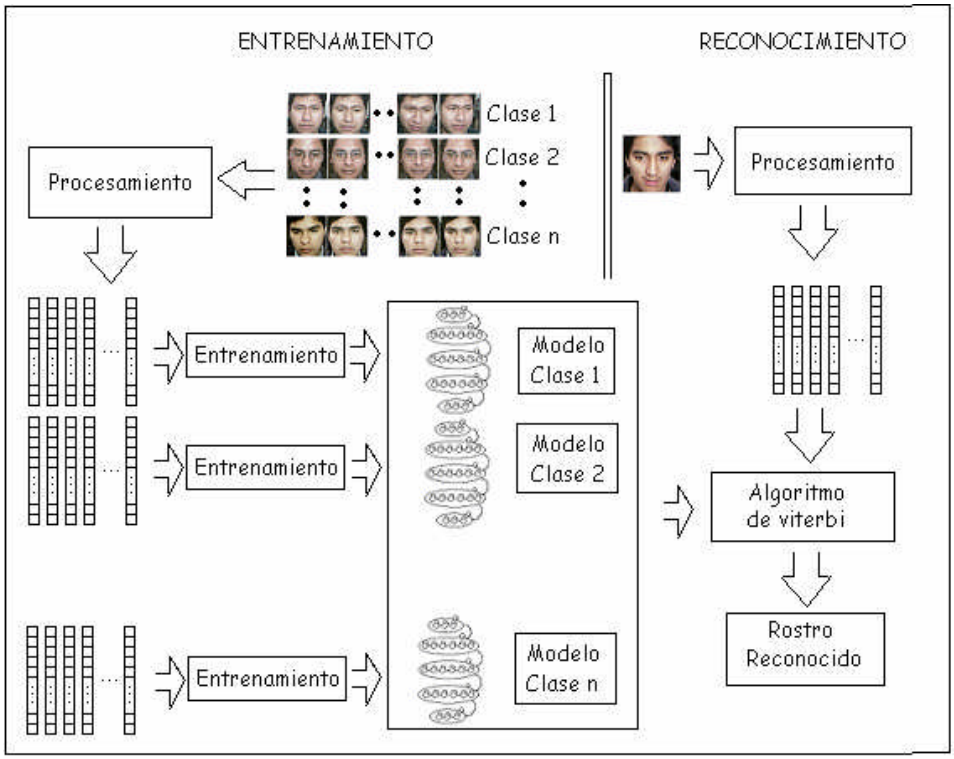


Fig. 7.19 *Reconocimiento de rostros utilizando HMME*

Una vez que el modelo a sido entrenado, un conjunto de imágenes de rostros es usado para el reconocimiento, para ello los vectores de observación son extraídos de la nueva imagen de rostro, luego la probabilidad de la secuencia de observación dado el Modelo HMM. Embebido de caras de una persona es calculado usando el doble algoritmo de viterbi. Finalmente el modelo que haya obtenido la máxima probabilidad es el rostro reconocido.

7.3 Resultados Obtenidos

Para el entrenamiento del sistema se utilizo una base de datos de los participantes de los proyectos en el Centro de Investigación y Desarrollo CID-FIEE. Estas imágenes de rostros se guardaron en formato comprimido jpg con una resolución de 140x140.

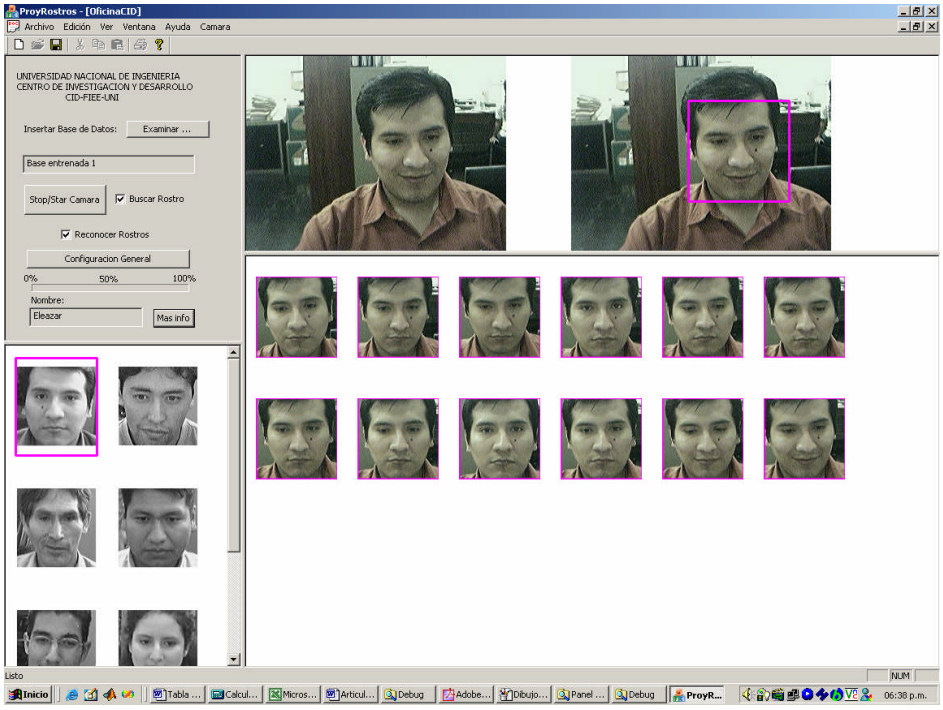


Fig. 7.20. *Interface para el reconocimiento de rostros utilizando HMME*

7.3.1 Resultados obtenidos por el método PCA

La Fig. 7.21a muestra la interface utilizada para el reconocimiento de rostros en el caso de acertar una imagen. La Fig. 7.21b muestra cuando un rostro no es encontrado en la base de datos.

Fig. 7.21. a) *Reconocimiento acertada b) Reconocimiento sin acierto por video.*

Realizamos diferentes pruebas para buscar la mejor eficiencia del sistema, y encontramos que las imágenes de entrenamiento deben ser representativas del

individuo, además de que estas deben ser normalizadas de acuerdo a la sección anterior. Los Resultados realizados se muestran en la tabla I.

Tabla I *Resultados del algoritmo PCA*

# de rostros Entrenadas	# de rostros a Reconocer	# de Aciertos	# de Desaciertos	Porcentaje de reconocimiento
2	20	20	0	100%
4	40	39	1	97.5%
6	60	57	3	95%
8	80	72	8	90%
10	100	88	12	88%
15	150	130	20	86.6%
18	180	155	25	86.1%
19	190	164	26	86.3%
20	200	171	29	85.5%

Se continuó con la etapa de reconocimiento, en la que se tomó solo una foto por persona en la etapa de entrenamiento, las imágenes a comparar los resultados fueron de 10 por imagen entrenada, es decir los que pasaron por la etapa de pre-procesamiento antes mencionada. Los resultados tienen una eficiencia por encima de 80% pero a medida que aumentemos mas rostros el sistema ira perdiendo eficacia. Por otro lado si aumentamos más fotos en la que podríamos incluir más fotos por persona el sistema incrementa su eficiencia.

Si comparamos este resultado con lo obtenido en la tabla anterior se observa un incremento de 3%. Pero el sistema aun con este resultado todavía no alcanza un nivel óptimo. Se tiene que entrenar con mas imágenes por rostro para un buen desempeño, aun así los resultados no serian lo deseado. La inclusión de un parámetro de rechazo, también influye en los resultados.

A continuación se muestran los resultados obtenidos con el método de Fisher después del reconocimiento en tabla II.

Tabla II. *Resultado con el método de PCA*

Numero de Clases	Patrones por clase	Reconocimiento (10 por cada Clase)		Eficiencia
		Reconoce	No reconoce	
10	3	90	10	90%
10	4	92	8	92%
10	5	98	2	98%
10	6	100	0	100%
10	7	100	0	100%

Numero de Clases	Patrones por clase	Reconocimiento (10 por cada Clase)		Eficiencia
		Reconoce	No reconoce	
3	3	30	0	100%
5	3	50	0	100%
10	3	90	10	90%
15	3	126	24	84%
20	3	148	52	74%

Numero de clases	Patrones por clase	Reconocimiento (10 por cada clase)		Eficiencia
		Reconoce	No reconoce	
3	5	30	0	100%
4	5	40	0	100%
5	5	50	0	100%
6	5	60	0	100%
7	5	69	1	98.5%
8	5	78	2	97.5%
9	5	87	3	96.6%
10	5	96	4	96%

El algoritmo de Fister corrige la deficiencia de PCA de forma que agrupa al conjunto de rostros de una misma persona tomando la misma cantidad de imágenes por persona. De esta manera se obtiene mejores resultados que el algoritmo PCA.

7.3.2 Resultados obtenidos por el método HMME

Con el método HMM con coeficientes KTL en el primer entrenamiento se hizo con 9 personas, cada persona con 5 gestos diferentes.

Los parámetros considerados son los siguientes:

5 estados

40 coeficientes KTL de cada bloque.

1 mezclas gausianas

TABLA III. *Resultados 9 personas, 5 gestos diferentes*

Personas	Total	Aciertos
Amilcar	14	14
Carlos	20	16
Enrique	20	16
Fernando	20	19
Flora	20	19
Gabriel	20	17
Giancarlo	21	17
Giover	21	19
Gustavo	21	13
TOTAL	177	150
		84.7% reconocimiento

Entrenamiento con 30 personas, 10 fotos cada una.

Los parámetros considerados:

5 estados

40 coeficientes KTL de cada bloque

3 mezclas gausianas

TABLA IV. *Resultados 30 personas, 10 gestos diferentes*

Personas	Total	Aciertos
Alain	14	14
Amilcar	20	19
Angel	20	18
Antonio	20	16
Carlos	20	17
Cristian	15	14
Doris	20	19
Eleazar	20	15
Enrique	20	18
Fernando	20	20
Flora	20	19
Gabriel	20	18
German	20	20
Giancarlo	21	19
Gino	12	12
Giover	21	19
Gustavo	21	19
Israel	20	16
Jorge	20	18
Jose	20	20
Juan	20	19
Luis	20	17
Maria	20	19
Percy	20	18
Philip	20	15
Piero	20	19
Renato	20	20
Ricardo	20	17
Ronald h	20	19
Ronald m	20	19
Total	584	532
		91.01% reconocimiento

Respecto al número de estados empleados, si tomamos 6 estados que representaría el rostro (pelo, frente, ojos, nariz, boca, mentón). La eficiencia no es tan buena que si empleamos 5 estados (sin considerar el pelo), ya que hay una diferencia de 5% tomando la misma base de entrenamiento.

Respecto al número de coeficientes utilizados por KTL, la mayor parte de todos los Eigenvalores están en los primeros coeficientes con mayor valor. A medida que este aumenta disminuye notablemente el valor de lo mismo, es por ello que tomamos 40 coeficientes que representan el 95% del total.

A medida que aumentamos mas rostros por personas la eficiencia aumenta pero llega a su tope, el máximo porcentaje encontrado es de 91.01% con 30 personas con 10 imágenes cada una y comparada con una base de datos de 584 imágenes de rostros.

Se procedió a optimizar el sistema, añadiendo algunos parámetros y analizar los resultados obtenidos, el sistema se realizo en Matlab. Las imágenes de entrenamiento son las imágenes tomadas en el centro de investigación de la facultad. Se procede a entrenar las fotos de las diferentes personas, inicialmente 3, 4, 5... de cada persona. También se describe y fundamenta las etapas de implementación del software de reconocimiento en tiempo real, basado en el modelo de desarrollo orientado a objetos, utilizando para ello el lenguaje C/C++ mediante el compilador Visual C++. Este enfoque nos permitirá contar con módulos reutilizables para poder usar y adaptar a diferentes algoritmos de reconocimiento (PCA, LDA, HMM, etc).

Para utilizar el método de HMME, se realizo el software en Visual C++, ya que se requiere que el procesamiento se haga en tiempo real y a mayor velocidad. El Reconocimiento se realiza una vez que hayamos insertado una base de datos ya entrenada. Para esto el software realizado busca los rostros en una imagen de video, los encuentra y tiene la opción de que estas fotos se almacene en una carpeta, a la vez se ira comparando con la base de datos de rostros y los estará mostrando en pantalla cada vez que reconozca un rostro, este proceso se realiza en tiempo real, de acuerdo a la capacidad y procesamiento de las PC.

Tabla V. *Resultado con el método HMME*

Número de Clases	Patrones por Clase	Reconocimiento (~20 por cada clase)		Eficiencia
		Reconoce	No Reconoce	
3	3	54	0	100%
3	6	54	0	100%
3	9	54	0	100%
3	12	54	0	100%
3	15	54	0	100%
5	3	94	0	100%
5	6	94	0	100%

5	9	94	0	100%
5	12	94	0	100%
5	15	94	0	100%
10	3	192	3	98.46 %
10	6	192	3	98.46 %
10	9	194	1	99.48%
10	12	195	0	100%
10	15	195	0	100%
15	3	287	12	95.98%
15	6	293	6	97.99%
15	9	298	1	99.96%
15	12	299	0	100%
15	15	299	0	100%
20	3	381	19	95.25%
20	6	391	9	97.75%
20	9	397	3	99.25%
20	12	400	0	100%
20	15	400	0	100%

Para los resultados se considero en general que el número de clases representa a la cantidad de personas entrenadas y el número de patrones por clase se refiere a la cantidad de rostros por persona entrenada. Se realizo una tabla con el método utilizado, La tabla 1 muestra los resultados del método de HMME, el reconocimiento se realiza por un conjunto aproximado de 20 rostros por persona incluyendo las entrenadas.

El desempeño que se obtuvo con HMME, muestra una eficiencia mínima de 95.98% donde se considero solo 3 rostros por persona haciéndola esta en el peor de los casos.

El Método de HMME puede ser utilizado para una mayor cantidad de personas, además es robusto para imágenes con menor escala y no es necesario que tengan una resolución constante.

Para el mejor entrenamiento de rostros, esta se debe hacer en el lugar donde se realizará el reconocimiento con ellos se consigue una mayor efectividad.

7.3.3 Comparación de resultados

Los métodos utilizados para el reconocimiento de rostros en este trabajo fueron: PCA, Fisher, HMM 1D. Estos 3 métodos tienen un porcentaje aceptable mayor al 90%.

En todos los métodos la entrada del sistema es la imagen del rostro, estas imágenes son normalizadas antes de entrar al sistema, si se trata de entrenar al sistema estos rostros deben ser representativos de cada persona. Si tenemos unos datos bien definidos, el algoritmo actúa eficientemente ya sea por el método de PCA y de Fisher (extensión del PCA).

El Algoritmo de PCA nos muestra que es eficiente para un determinado número de imágenes (1 imagen representativa por persona), pero si aumentamos el número de imágenes por persona desordenadamente, la eficiencia disminuye según lo visto en la fig. 7.22 esto se debe a que el algoritmo trata a los rostros por separado, sin vincular o agrupar los rostros de una misma persona.

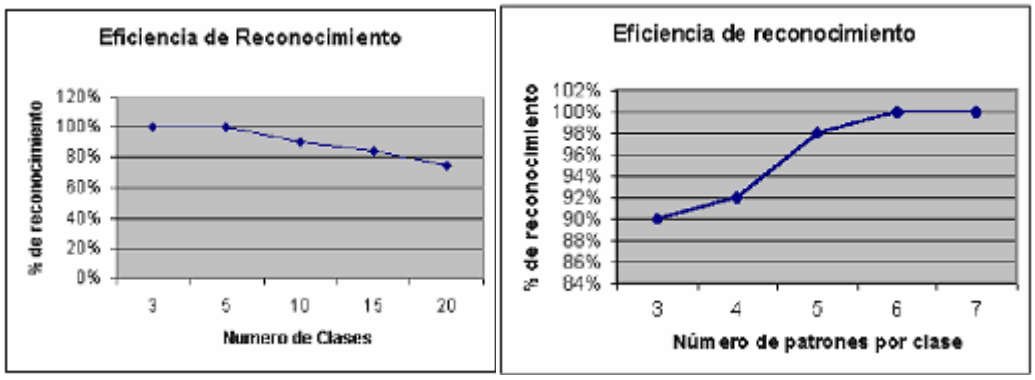


Fig. 7.22. *Eficiencia según el algoritmo PCA*

El Algoritmo de Fisher corrige la deficiencia de PCA de forma que agrupa al conjunto de rostros de una misma persona tomando la misma cantidad de imágenes por persona. La convergencia de los estimadores involucrados en este método es mucho más lenta que el método de PCA. Por tanto en la etapa de entrenamiento con el método Fisher el tiempo de entrenamiento es grande. Hay que notar que el tiempo que tarda un sistema de identificación genérico en dar un resultado es proporcional al número de imágenes de entrenamiento. Por tanto, si la base de datos es muy grande, estos métodos no serán eficientes para aplicaciones de tiempo real.

Con el método HMM, según el gráfico de la Fig. 7.23 la eficiencia aumenta a medida que aumentamos más imágenes a nuestra base de entrenamiento rostros por persona. Si

se emplea 5 estados en vez de 6 estados la eficiencia es similar. La diferencia esta en los tiempos de ejecución de los mismos, es decir, que a medida que se aumenta mas estados al sistema este tarda en entrenarse y se obtiene resultados similares. Para la formación del vector de observación se usaron los coeficientes KTL de cada imagen.

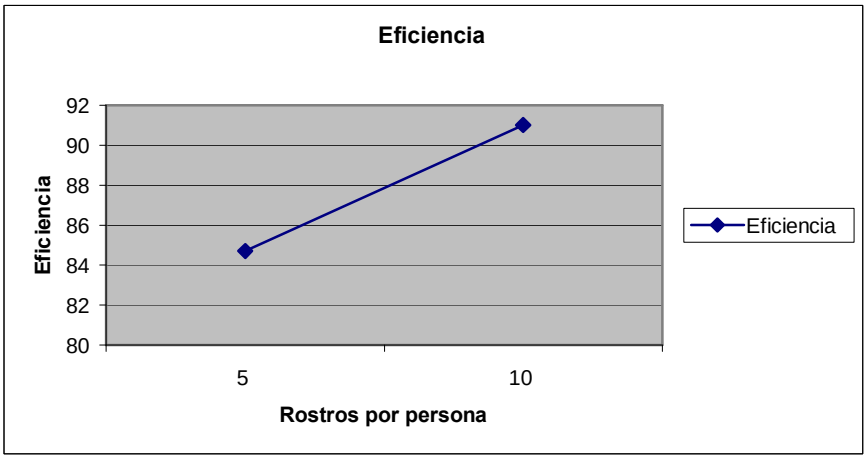


Fig. 7.23. *Eficiencia según el Algoritmo HMME*

Con respecto a la cantidad de rostros por persona, a medida que aumentamos mas rostros la eficiencia aumenta pero llega a su tope, el máximo porcentaje encontrado es de 91.01% con 30 personas con 10 imágenes cada una y comparada con una base de datos de 600 imágenes de rostros.

9.2 Comparación frente a sistemas actuales

Los sistemas reconocedores de rostros a nivel mundial buscan siempre el mejor algoritmo que tenga una buena eficiencia, en la rapidez y eficacia en reconocer rostros con menor grado de error. Para ello, existe una base de datos (ORL Face Database) Olivetti Research Lab, Cambridge, que es tomada como modelo de reconocedores de rostros.

Nosotros también usamos esta base de datos, y obtenemos los resultados mostrados en la Fig. 7.24

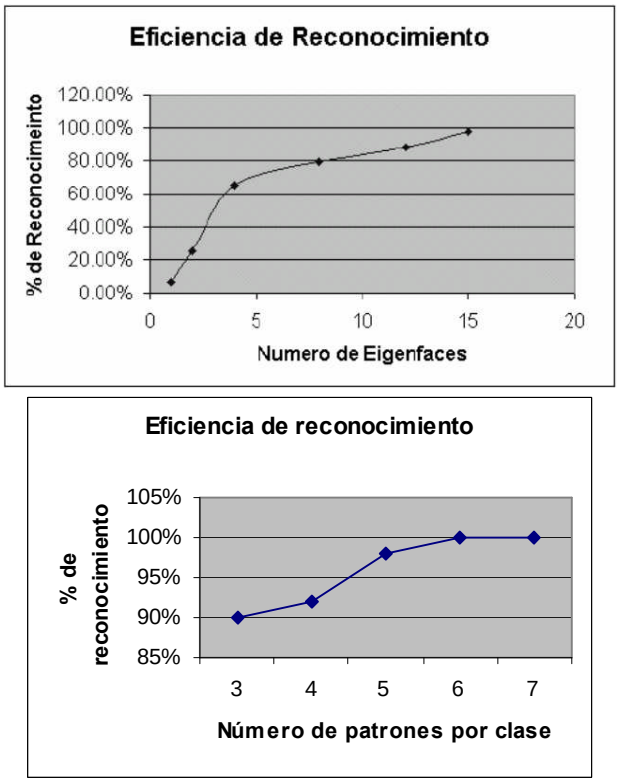


Fig. 7.24. *Resultados con la base de Datos (ORL FACE Database)*

Los resultados son similares a lo empleado con la base de datos propios.

Respecto al reconocimiento de rostros el desempeño que se obtuvo con HMME, muestra una eficiencia mínima de 95.98% donde se considero solo 3 rostros por persona suponiendo esta el peor de los casos. El Método de HMME puede ser utilizado para una mayor cantidad de personas, además es robusto para imágenes con menor escala y no es necesario que tengan una resolución constante. Para que el reconocimiento de rostros sea eficiente, el entrenamiento se debe realizar en el mismo lugar donde se efectuara reconocimiento, además se sugiere que la iluminación deba ser constante para conseguir una mayor efectividad.

CAPITULO VIII
IMPLEMENTACION DEL RECONOCEDOR DE ROSTROS

8.1 Adquisición de video

La adquisición de video se hará a través del computador, para ello se utilizará los drivers de las cámaras preinstaladas. Estos dispositivos de video serán manipulados por software, el software debe tener la capacidad de realizar las operaciones de manera rápida y eficaz.

Así tenemos VFM (Video por Matlab), VFW (Video por Windows) y DirectX. Este último realiza operaciones a través del hardware, siendo el adecuado para el manejo a gran velocidad de los datos.

8.1.1 VFM (Video for Matlab) para adquisición de video

VFM (Video for Matlab) [34] usa un modelo driver de Microsoft's Video for Windows (VFW), de esta manera puede acceder a dispositivos que tienen fuente de video sobre un computador personal. Con este modelo se pueden conectar hasta 10 dispositivos de captura de video diferentes, y asi modificar sus propiedades de captura de imagen y llevar en un formato apropiado para Matlab.

Para poder utilizar este driver se procede a copiar vfm.m y vfm.dll (que son archivos compilados) a una carpeta de Matlab y luego se procede a dar la ruta de estos archivos en Matlab.

VFM graba perfectamente cualquier fuente de imagen de video por Windows (ejemplo: Cámara WebCam, Cámaras digitales, etc.), las funciones siguientes son útiles con el primer parámetro, la invocación a esta sub-funciones, como por ejemplo ‘grab’ es de la siguiente forma:

```
>> Vfm(‘grab’,...Parámetros...);
VFM(‘grab’,n)
```

Graba los cuadros de video según el valor de n, por default n=1, esta sentencia retorna una matriz de MxNx3xNumero de cuadros de 8 bits, donde M y N son el ancho y alto de la matriz respectivamente, las imágenes están en el formato RGB.

VFM(‘preview’,n);

Es un interruptor de video, la imagen de video queda estática, n puede tomar los valores 1 o 0, 0 indica que la imagen debe estar estática en el preciso momento en que se invoca a esta función. 1 es el valor por defecto.

VFM(‘show’,n);

Muestra el video en una ventana, n puede tomar los valores 1 o 0, 1 indica que el video debe ser mostrado, 0 indica que el video debe estar oculto. 1 es el valor por defecto.

VFM('configsource')

Muestra un cuadro de dialogo sobre la configuración de video.

VFM('configformat')

Muestra un cuadro de dialogo sobre el formato de video

VFM('configdisplay')

Muestra un cuadro de dialogo sobre la visualización de video.

VFM('drivername', index)

Retorna el nombre del dispositivo conectado a la PC, index puede tomar 10 valores, según la cantidad de drives existentes en dicho sistema.

VFM('setdriver', index)

Se puede efectuar modificaciones en los parámetros del dispositivo conectado a la PC, index puede tomar 10 valores, según la cantidad de drives existentes en dicho sistema.

En Matlab la adquisición se realiza de la siguiente manera:

>> A = vfm(‘grab’,1);

Con esta sentencia tomamos una imagen de la WebCam y lo guardamos en A como una matriz de M x N x 3.

8.1.2 VFW (Video for Windows) para adquisición de video

VFW (Video for Windows) es la antigua API de video para Windows. Estos codecs tienen la extensión .DLL o, raramente, .DRV. A medida que las capacidades gráficas de las computadoras personales iban mejorando existía la posibilidad de ver imágenes en movimiento. Los primeros sistemas eran normalmente lentos, de baja calidad y además las imágenes almacenadas ocupaban mucho espacio. El primer sistema fue el formato

FLI de Autodesk. Este formato en sus orígenes era una secuencia de imágenes GIF a 320x200 y 256 colores sin sonido. El sistema de compresión usado era muy simple y consistía en a partir de la primera imagen (fotograma) almacenaba solo las variaciones que existían en las siguiente. Con esto se conseguía un ratio de compresión de 1:2 aproximadamente. Este formato posteriormente evolucionó admitiendo distintas resoluciones y más profundidad de color, manteniendo el mismo sistema de compresión sin perdida.

Los archivos de video digital son muy grandes, requiriendo gran velocidad de transferencia de datos en la lectura y reproducción. Para referirnos a Video For Windows, nos referimos a un “AVI” debido a que las extensiones .avi son las que utiliza VFW de forma especifica. Sus codecs están desarrollados como controladores para ACM (Audio Compression Manager) y VCM (Video Compression Manager), y también pueden ser usados por algunas otras arquitecturas, incluidas DirectShow y Windows Media.

AVI significa AudioVideo Interleave o Audio y Video Intercalado y fue desarrollado por Microsoft. “Intercalado” significa que en un fichero AVI los datos de audio y video son almacenados consecutivamente en capas (un segmento de datos de vídeo es seguido inmediatamente por otro de audio). Es el formato más extendido para el manejo de datos de audio/video en un PC. Video for Windows (VfW) versión 1.0 fue lanzada en noviembre de 1992 para el sistema operativo Windows 3.1 y fue optimizado para capturar imagen en movimiento a disco. Desde entonces, las posibilidades de la captura han crecido espectacularmente debido al uso de nuevos buses PCI, nuevos controladores de bus, nuevos formatos Wide SCSI, y transferencia directa del vídeo capturado de la memoria del adaptador al disco y por supuesto las tarjetas firewire, la tecnología DV (video digital) y USB. Por diversos motivos técnicos y algunas deficiencias de la arquitectura del sistema sacadas a la luz en especial con la aparición de la videoconferencia y la convergencia PC/TV, se hizo necesario el desarrollo de una nueva tecnología de captura de vídeo.

La arquitectura VfW fallaba en puntos importantes referidos a la videoconferencia, visionado de televisión, captura de campos de vídeo y otros referidos a los flujos de datos como el intervalo de refresco vertical. Algunos fabricantes han añadido extensiones propietarias para evitar estas limitaciones pero, sin interfaces estandarizadas, los programas que usan estas posibilidades deben incluir códigos específicos para el hardware. La estrecha relación entre los drivers de captura de VfW y los de visionado significa que un cambio realizado en el driver de captura implica un cambio en los de visionado también.

Los drivers VfW continúan apareciendo en los sistemas operativos con dispositivos que son principalmente usados para capturar vídeo debido a la larga lista de aplicaciones que aún lo utilizan. Para apoyar a estos dispositivos, VfW sigue presente en Windows 98, ME, Windows 2000, y Windows NT.

8.1.3 Componente DirectShow de DirectX para adquisición de video

Microsoft DirectX es un conjunto de APIs(Application programming interfaces) de bajo nivel para aplicaciones multimedia de alto rendimiento. Trae soporte para gráficos de 2 y 3 dimensiones (2D y 3D). Efectos de sonido y música asi como entrada de dispositivos. DirectX trae varios componentes como son: Direct3D, DirectInput, DirectPlay, DirectSound, DirectMusic, DirectShow entre otros.

La componente DirectShow es una arquitectura media-streaming sobre la plataforma de Microsoft Windows. Esta provee una captura de video de alta calidad y soporta una variedad de formatos como ASF(Advanced Systems format), MPGE(Motion Picture Experts Group, AVI(Audio Video Interleaved, MPEG Audio layer-3 (MP3) y archivos de sonido (WAV), también soporta tarjetas capturadoras usando Windows Driver Model (WDM) o el antiguo VFW (Video for Windows). Mediante DirectShow se puede detectar y usar automáticamente la aceleración del hardware de video o sonido si estos se encuentran disponibles. DirectShow esta basado en COM (Component Object Model).

8.1.4 Especificación de la cámara usada para el reconocimiento

Las cámaras utilizadas para las pruebas son, una cámara WebCam-USB con sensor CMOS de 640x480 de resolución marca Creative-PRO. Y una cámara Domo analógica AVC523 LN/F36 [9] que tiene una lente de 3.6mm, la cámara esta diseñada para vigilancia y su ángulo de visión es de mas de 72º, utiliza la norma NTSC . La cámara Domo es a color y utiliza una fuente DC de 12V regulado para su funcionamiento, sus características de fabricante son las siguientes:

Sensor.....1/4 CCD
Numero de píxeles..... 500 X 582
Alimentación.....12 V
Min. Iluminación... 3 lux /F1.2
Lente.....F 2.0 3.6 mm
Ángulo de Visión....72º {H) X 57(V)
Resolución..... 320 líneas

Relación Señal / ruido.... Más de 48 dB
Corrección Gama.....0.45
Control Ganancia....Automática
Salida de vídeo.....1 Vpp
Consumo.....2,4 máximo
Temperatura de uso....-10ºC a 50ºC
Dimensiones.....96 x 70
Peso.....135 gr.

Usamos estas cámaras anteriores porque existen en el mercado y tienen un rendimiento suficiente para nuestro requerimiento. Si hablamos de las cámaras que se utilizan en visión artificial, son más sofisticadas que las convencionales, ofreciendo un completo control de los tiempos y señales, de la velocidad de obturación, de la sensibilidad y de otros factores fundamentales tanto en aplicaciones científicas como industriales.

Los sensores de cámaras modernos son mayoritariamente CCD ya que utilizan material sensible a la luz para convertir los fotones en carga eléctrica. Aunque los CMOS están reemplazando a estas gracias a su bajo consumo de energía y por ende al bajo precio. Cada vez se puede encontrar en el mercado cámaras CMOS con mas eficiencia.

Fig. 8.1 *Cámara WebCam Creative modelo PD1030*

Fig. 8.2 *Cámara Domo utilizada en el sistema*

Estas cámaras tienen la posibilidad de cambio de lentes, asi tenemos dos lentes, una de 8mm y otra de 3.6mm. Fig. 8.3

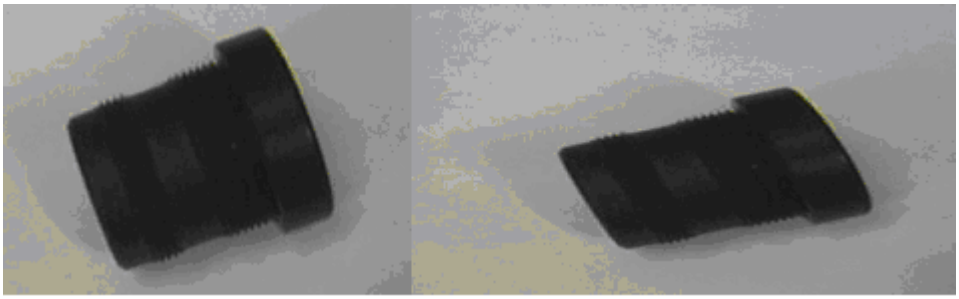


Fig. 8.3 *Lentes de 3.6mm y 8 mm respectivamente*

La lente de 3.6 mm tiene una visión de mas de 70º, por lo que es utilizado mas para sistemas de vigilancia, ya que ellas requieren una visión completa del panorama general de la imagen. La lente de 8 mm tiene una visión menor de 30º, y es utilizado para dar una mayor resolución de objetos cercanos, tal como el caso de un rostro.

8.2 Desarrollo del software en C++

Esta etapa es la que se ocupó más tiempo, se buscó una librería que permitiera un eficiencia en el procesamiento de la imagen, esta librería fue la OpenCV de Intel, para la adquisición de video se utilizó la componente DirectShow que maneja eficientemente los dispositivos de video. Se hicieron 3 versiones de este software.

8.2.1 DirectShow.

Los Elementos básicos de DirectShow son los filtros y los grafos de Filtros, estos elementos se implementan siguiendo el modelo orientado a objetos COM.

El Filtro es un componente modular que realiza una función concreta sobre el flujo de datos multimedia. Estos tienen un punto de entrada y de salida llamados pins.

Fig. 8.4. *Filtro X*

El grafo de filtros es un conjunto de filtros que a través de los pins estos pueden conectarse entre sí formando un camino en la cual los datos multimedia se van procesando.

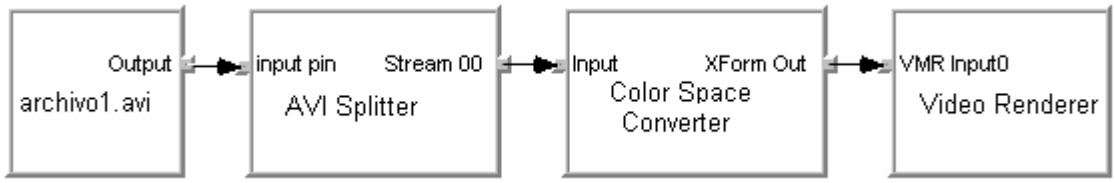


Fig. 8.5. *Grafo de Filtros*

Entre los tipos de filtros se encuentran filtros de origen, filtro parser, filtro de transformación, filtro render.

Los filtros de origen suministran los datos multimedia a través de archivos de discos (.mpg, .avi, etc.), URLs de Internet, cámaras Web, cámaras analógicas, etc. Actúan como punto de entrada de datos en el Grafo de Filtros

Fig. 8.6. *Filtros de origen*

Los filtros parser interpretan el significado de datos que les llegan y saben como interpretarlos y moverse a través de ellos.

Fig. 8.7. *Filtro Parser*

Los filtros de transformación Procesan los datos que les llega transformando su contenido, su formato, o no lo modifican.

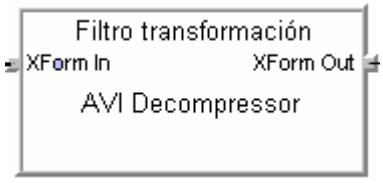


Fig. 8.8 *Filtro transformación*

Los filtros de render se encargan de pasar los datos a los dispositivos hardware (Adaptador de Gráficos, de sonido, archivos en disco, etc.).

Fig. 8.9. *Filtro Render*

Como se vio anteriormente, cada filtro se encarga de una operación muy concreta, desde la fuente hasta la salida en dispositivos hardware. El Filter Graph Manager es el componente de DirectShow que permite crear y gestionar grafos de filtro, así como el paso de datos a través de ellos que construye automáticamente la configuración mínima para reproducir archivos multimedia. Utiliza también filter mapper para ver los filtros instalados en el sistema y los tipos de datos que admite cada uno de ellos a través de sus pins.

Los componentes básicos de DirectShow, el Filter Graph Manager y los propios filtros, están implementados como clases COM.

Tabla 8.1. *Interfaces que COM implementa*

Componente DirectShow	GUID del la Clase COM	Interfaces COM que implementa
Filter Graph Manager	CLSID_FilterGraph	IFilterGraph IID_IFilterGraph IGraphBuilder IID_ImediaBuilder IMediaControl IID_ImediaControl IMediaSeeking IID_ImediaSeeking IMediaPosition IID_ImediaPosition IMediaEventEx IID_ImediaEventEx IVideoWindow IID_IvideoWindow
Filtro	Depende del filtro	IBaseFilter IID_IBaseFilter ISpecifyPropertyPages IID_ISpecifyPropertyPages

Gran cantidad de la funcionalidad de un filtro esta implementada para ser accedida desde el grafo de filtros, con lo cual que da mas o menos oculta. La Interface IFilterGraph es la que nos permite manipular la estructura del grafo de filtros. A través de ella podemos insertar filtros, crear y destruir conexiones entre ellos, o consultar los filtros que hayan instanciados. La consulta se hace a través de un enumerador, que es un objeto que implementa la Interface **IEnumFilters**.

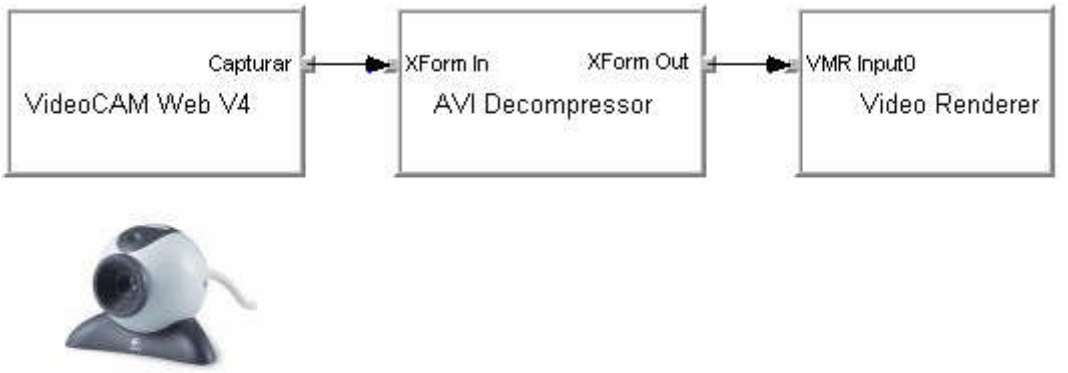


Fig. 8.10. *Grafo de Filtros para una Cámara web*

En la figura 8.10 se muestra el grafo de filtros simple para la vista de video de una cámara web cualquiera. El primer Filtro es la Cámara web, quien captura la imagen por su sensor sea CMOS o CCD. Luego pasa a su descomprensor que solo hace la conversión de formato de color de video de I420 320x240 de 12bits hacia YUY2 320x240 de 16 bits (codificación) para finalmente pasar al último filtro quien mostrara por el hardware (monitor) la secuencia de video. Los grafos de filtros para las diferentes cámaras son similares.

Antes de la implementación hay que tener presente que nuestra aplicación debe tener la ultima versión de DirectX instalados en nuestro sistema. Esto se obtiene gratuitamente de:
http://www.microsoft.com/windows/directx/downloads/default.asp

Y para compilar nuestra aplicación necesitamos de el kit de desarrollo de DirectX, esto se puede encontrar en:
http://www.microsoft.com/downloads/Popular.aspx?DisplayLang=en&categoryId=2

Preferible es bajar DirectX 9.0 SDK Update - (Summer 2004) , con el cual realizo la aplicación.

Una vez instalado, se procede a compilar baseclasses que se encuentra generalmente en C:\Archivos de programa\Microsoft DirectX 9.0 SDK (Summer 2004)\Samples\C++\DirectShow\BaseClasses. Con ello crearemos strmbasd.lib que es el

resultado de la compilación, y lo copiamos en C:\Archivos de programa\Microsoft DirectX 9.0 SDK (Summer 2004)\Lib.

Abrimos un proyecto en MFC y la llamaremos VideoDirectX, Luego elegimos la aplicación basada en dialogo

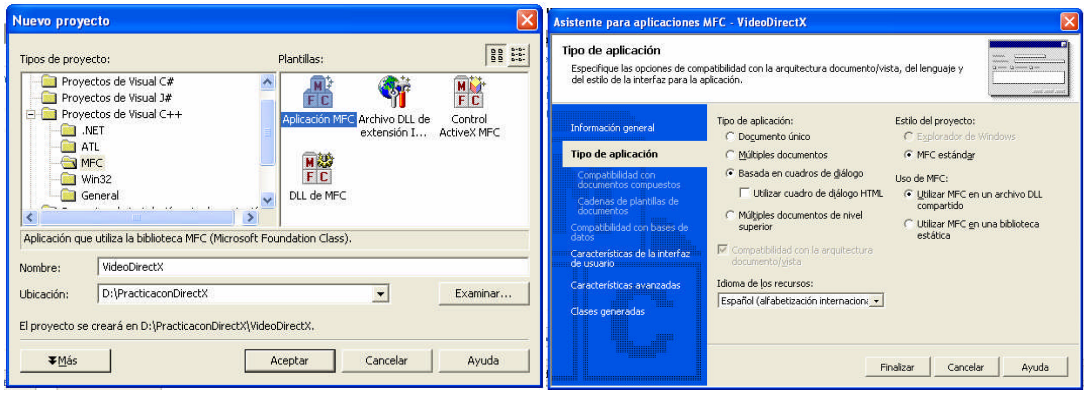


Fig. 8.11. *Inicialización del programa basado en dialogo*

Ahora abrimos en Proyecto – Propiedades de VideoDirectX... y nos vamos a la sección de vinculador mostrado en la figura, luego en Dependencias adicionales colocamos strmbasd.lib (que fue compilado anteriormente) y winmm.lib

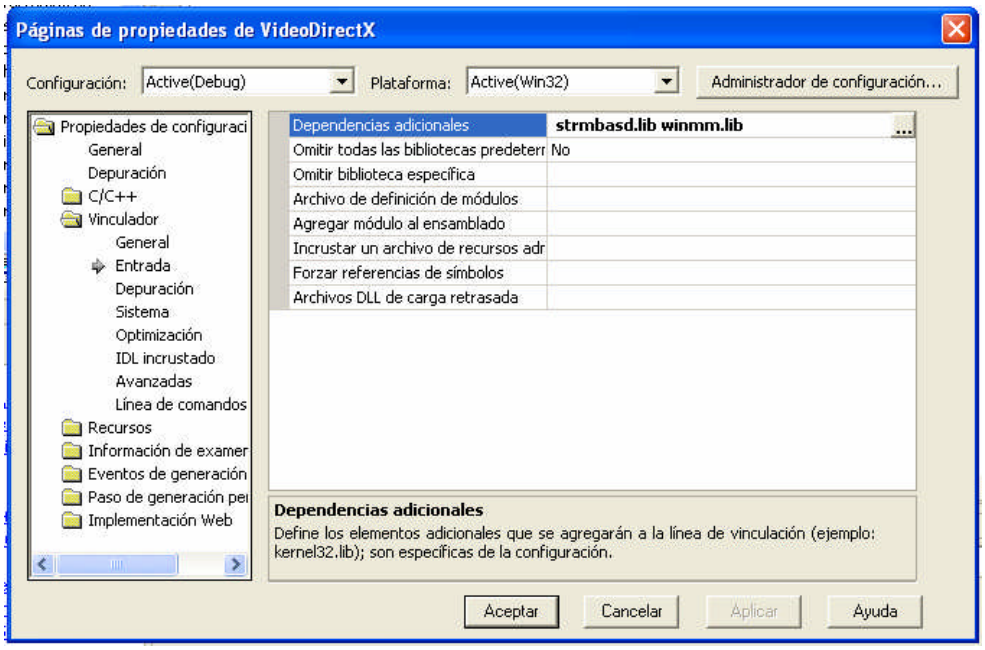


Figura 8.12. *Inclusión de librerías de DirectShow en nuestro programa*

Para la creación del grafo de filtro antes de todo, primero incluiremos los archivos de cabecera a necesitar como son:

En stdafx.h agregar:

```
#include <qedit.h>  //para muestras, qedit.dll debe estar registrado
```

En VideoDirectXDlg.cpp agregar:

```
#include "dshow.h" // cabecera propia de DirectShow
#include "mtype.h" //para el tipo e video
```

Para realizar la salida de video de un dispositivo por pantalla se deberá crear el grafo del Filtros, entonces Primero debemos declarar la función InitVideoGraph(); que será del tipo HRESULT. Todas las funciones COM y métodos de objetos COM devuelven un valor de tipo HRESULT con un código de error. Si todo a funcionado correctamente, este valor es S_OK. En cambio, si ha habido algún error, este código dependerá de la función o método que sea, ya que cada una puede generar tipos diferentes de errores. Esto lo haremos en VideoDirectXDlg.h

```
class CVideoDirectXDlg : public CDialog
{
...
// Implementación
protected:
        HICON m_hIcon;
        HRESULT InitVideoGraph( );
```

Antes de la implementación vamos a iniciar la captura gráfica de la imagen al iniciar la aplicación, para ello en VideoDirectXDlg.cpp

```
BOOL CVideoDirectXDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
....
// TODO: agregar aquí inicialización adicional
        // Empezar la captura grafica de la imagen
        HRESULT hr = InitVideoGraph( );

        return TRUE;  // Devuelve TRUE  a menos que establezca el foco en un control
```

Ahora procedemos a implementar la función InitVideoGraph() en VideoDirectXDlg.cpp

```
HRESULT CVideoDirectXDlg::InitVideoGraph( )
{
}
```

En nuestro programa primero hay que indicarle al sistema que nuestra aplicación va a ser usuaria de objetos COM, entonces escribimos:

```
HRESULT hr=CoInitialize(NULL);
        if( FAILED(hr))
        {
                Error(TEXT("No pudo iniciarse la Librería COM");
        return E_FAIL;
        }
```

La función Error viene definida de la siguiente manera

```
void CVideoDirectXDlg::Error( TCHAR * pText )
{
   ::MessageBox( NULL, pText, TEXT("Error!"), MB_OK | MB_TASKMODAL |
MB_SETFOREGROUND );
}
```

Ahora empezaremos a construir el Grafo de Filtros, El Filter Graph Manager es el encargado de gestionar la construcción del árbol de los filtros, este es un objeto COM. Estos objetos se crean mediante la función CoCreateInstance, y se accede a ellos a través de punteros a Interfaces, antes obtendremos un puntero a una interfaz a raíz IGraphBuilder.

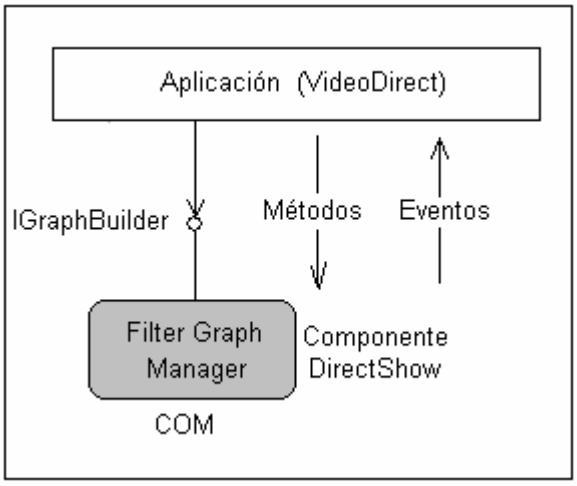


Fig 8.13. *Objeto COM Filter: Graph Manager.*

```
IGraphBuilder *m_pGraph;
hr=CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
IID_IGraphBuilder,(void**)&m_pGraph);
        if( FAILED(hr))
        {
                Error(TEXT("No se pudo crear el Filter Graph Manager"));
            return E_FAIL;
        }
```

Cada DLL tiene una identificación única que se llama CLSID y que se almacena en el registro de Windows. Como se puede ver, el identificador de clase (CLSID) es CLSID_FilterGraph. CLSCTX_INPROC_SERVER indica que el contexto de ejecución es el de la propia aplicación. De aquí se puede utilizar m_pGraph->QueryInterface para el resto de interfaces DirectShow

Otra manera de construir el Grafo de Filtros es la siguiente:

```
CComPtr< IGraphBuilder > m_pGraph;
HRESULT hr;
hr = m_pGraph.CoCreateInstance( CLSID_FilterGraph );
        if( !m_pGraph )
        {
        Error(TEXT("No se pudo crear el Grafo de Filtro"));
        return E_FAIL;
        }
```

Donde la Clase CComPtr viene definida como la siguiente
Class CComPtr Es una clase elegante para manejar los punteros de las interfaces.
template < class *T* > class CComPtr
Donde **T** es una interface COM especificando el tipo de puntero para ser guardado

Estas son las dos formas de crear el Grafo de Filtros, en este caso nos quedaremos con la segunda forma por la facilidad de manejar los punteros.

Ahora procedemos a capturar el dispositivo de video presente, ya sea cámara web o analógica. Se continuará agregando el siguiente código en el programa:

```
CComPtr< IBaseFilter > pCap;
GetDefaultCapDevice( &pCap );
        if( !pCap )
    {
                Error( TEXT("No se ha detectado una capturadora de video en tu
                sistema.\r\n\r\n")
        TEXT("Esto requiere un dispositivo de video, tal\r\n")
        TEXT("como una cámara USB.") );
      return E_FAIL;
    }
```

Aquí se obtuvo un puntero a una interfaz a raíz IBaseFilter. Y luego se paso por la función GetDefaultCapDevice(&pCap) quien capturará y tendrá la información del dispositivo instalado. Para no salirnos de la función actual, dejaremos para el final la explicación de la función GetDefaultCapDevice.

Una vez encontrado el dispositivo de video del sistema se adiciona el filtro de captura en el grafo con las propiedades de dispositivo mencionado.

Fig. 8.14 *Filtro de Origen (source filter)*

```
hr = m_pGraph->AddFilter( pCap, L"Cap" );
if( FAILED( hr ) )
   {
     Error( TEXT("no se puede colocar el dispositivo de captura en el
          grafo"));
     return E_FAIL;
   }
```

Con esto ya tenemos la primera parte del grafo de filtros.

Filtro de Transformación que obtiene una muestra y lleva a 24 bits

Ahora obtendremos una muestra de las imágenes de video.

```
// creamos un muestra
   CComPtr< ISampleGrabber > m_pGrabber;
   hr = m_pGrabber.CoCreateInstance( CLSID_SampleGrabber );
   if( !m_pGrabber )
   {
      Error( TEXT("No se puede crear un Grabber (esta qedit.dll registrado?)"));
      return hr;
   }
   CComQIPtr< IBaseFilter, &IID_IBaseFilter > pGrabBase(m_pGrabber);
```

El Sample Grabber filter provee una manera para recuperar muestras de aquellos que pasan por el Filtro gráfico. Este es un filtro de transformación de un pin de entrada y un pin de salida, la interface de este filtro es IsampleGrabber.

Ahora se va a forzar que el video sea de 24 bits, para ello definiremos a VideoType del tipo de clase de CMediaType. Es aquí donde tenemos la muestra y lo que haremos será transformado a una formato de 24 Bits.

```
   CMediaType VideoType;
   VideoType.SetType( &MEDIATYPE_Video );
   VideoType.SetSubtype( &MEDIASUBTYPE_RGB24 );
   hr = m_pGrabber->SetMediaType( &VideoType );
   if( FAILED( hr ) )
   {
      Error( TEXT("no se puede colocar media type"));
      return hr;
   }
```

con hr = m_pGrabber->SetMediaType(&VideoType); forzamos a que nuestro formato sea el adecuado.

Ahora adicionaremos el grabber en el grafo.

```
hr = m_pGraph->AddFilter( pGrabBase, L"Grabber" );
if( FAILED( hr ) )
{
Error( TEXT("No se puede poner una muestra de grabber en graph"));
return hr;
}
```

Con ello ya tenemos los dos bloques deseados.

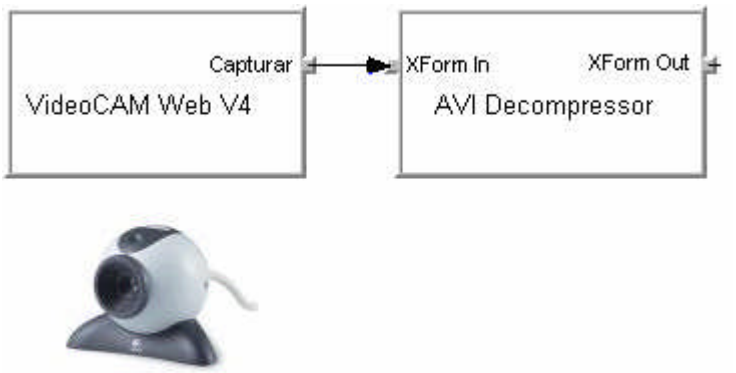


Fig. 8.15. *Filtros de fuente y transformación*

Ahora deseamos obtener la salida del Filtro de Trasformación, para ellos debemos trasladar nuestro Grafo a un Grafo general que contenga lo anterior, para ellos utilizamos un puntero de la interface ICaptureGraphBuilder2.

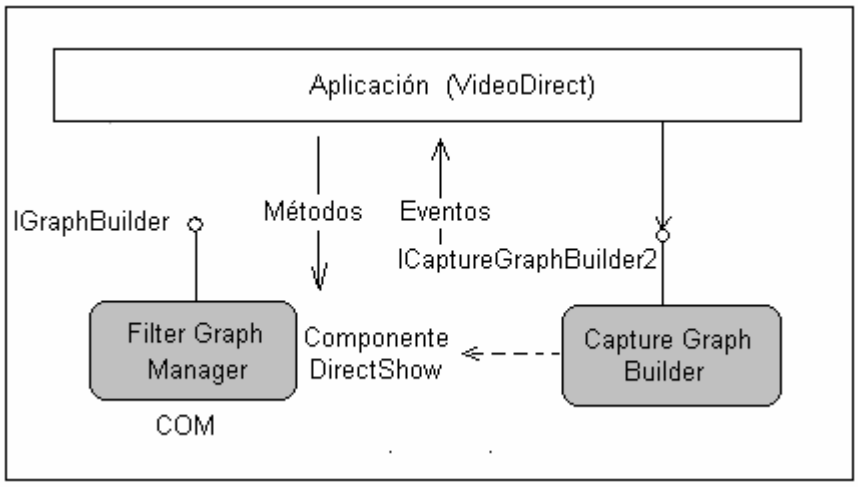


Fig. 8.16. *Capture Graph Builder*

```
   CComPtr<ICaptureGraphBuilder2> pCGB2;
   hr = pCGB2.CoCreateInstance (CLSID_CaptureGraphBuilder2, NULL,
CLSCTX_INPROC);
   if (FAILED( hr ))
   {
      Error(TEXT ("No se puede Conseguir una referencia a ICaptureGraphBuilder2"));
      return hr;
   }
```

Luego de crear el Grafo, procedemos a reemplazar lo anterior en pCGB2

```
hr = pCGB2->SetFiltergraph( m_pGraph );
   if (FAILED( hr ))
   {
      Error(TEXT("Fallo SetGraph"));
      return hr;
   }
```

Procedemos a definir un PIN que será pVPPin que será la salida del Grafo.

```
   CComPtr<IPin> pVPPin;
   hr = pCGB2->FindPin(
                  pCap,
                  PINDIR_OUTPUT,
                  &PIN_CATEGORY_VIDEOPORT,
                  NULL,
                  FALSE,
                  0,
                  &pVPPin);
```

Ahora Crearemos el Ultimo Filtro con un puntero pRenderer de la interfase IBaseFilter.

Si existe un VP pin, pondremos el renderer sobre NULL Renderer.

```
   CComPtr<IBaseFilter> pRenderer;
         if (S_OK == hr)
```

```
{
    hr = pRenderer.CoCreateInstance(CLSID_NullRenderer);
    if (S_OK != hr)
    {
        Error(TEXT("no es capaz de hacer un NULL renderer"));
        return S_OK;
    }
    hr = m_pGraph->AddFilter(pRenderer, L"NULL renderer");
    if (FAILED (hr))
    {
        Error(TEXT("No puede adicionar el filtro para el graph"));
        return hr;
    }
}
```

Con hr = m_pGraph->AddFilter(pRenderer, L"NULL renderer"); ya tenemos adicionado a nuestro Grafo el filtro de salida pRenderer.

```
        hr = pCGB2->RenderStream(
                    &PIN_CATEGORY_CAPTURE,
                    &MEDIATYPE_Video,
                    pCap,
                    pGrabBase,
                    pRenderer);
        if( FAILED( hr ) )
{
    Error( TEXT("No se puede construir el Graph") );
    return hr;
}
```

Con esto conectamos la salida al filtro y adicionalmente a un tercer filtro, como es el caso de pGrabBase.

Nos disponemos a mostrar el video por la cámara, para ello:

```
        CComQIPtr< IVideoWindow, &IID_IVideoWindow > pWindow = m_pGraph;
    if( !pWindow )
```

```
    {
        Error( TEXT("Could not get video window interface"));
        return E_FAIL;
    }
```

Haremos que nuestra cámara tenga las siguientes propiedades:

```
HWND hwndPreview = NULL;
    GetDlgItem( IDC_CAM, &hwndPreview );
            RECT rc;
    ::GetWindowRect( hwndPreview, &rc );

    hr = pWindow->put_Owner( (OAHWND) hwndPreview );
    hr = pWindow->put_Left( 0 );
    hr = pWindow->put_Top( 0 );
    hr = pWindow->put_Width( rc.right - rc.left );
    hr = pWindow->put_Height( rc.bottom - rc.top );
    hr = pWindow->put_Visible( OATRUE );
    hr = pWindow->put_WindowStyle( WS_CHILD | WS_CLIPSIBLINGS );

            Ahora procedemos a correr el archive por medio de esta interface

    CComQIPtr< IMediaControl, &IID_IMediaControl > pControl = m_pGraph;
    hr = pControl->Run( );
    if( FAILED( hr ) )
    {
        Error( TEXT("Could not run graph"));
        return hr;
    }

            return 0;

            Esta función nos permite elegir el dispositivo de salida del filtro.
void CVideoDirectXDlg::GetDefaultCapDevice( IBaseFilter ** ppCap )
{
}
```

Primero ponemos al puntero ppCap en NULL,

```
        HRESULT hr;
        ASSERT(ppCap);
  if (!ppCap)
    return;
        *ppCap = NULL;
```

Ahora nos dispondremos de crear el enumerador de los dispositivos. Creamos el objeto con el identificador CLSID_SystemDeviceEnum que enumera los dispositivos y los filtros instalados en el sistema.

```
        CComPtr< ICreateDevEnum > pCreateDevEnum;
        pCreateDevEnum.CoCreateInstance( CLSID_SystemDeviceEnum );
        ASSERT(pCreateDevEnum);
  if( !pCreateDevEnum )
    return;
```

La interface ICreateDevEnum apunta a pCreateDevEnum que crea un enumerador para diferentes dispositivos, como puede ser dispositivos de captura de video, captura de audio, video comprensores, y mas. Luego se crea el objeto con el identificador CLSID_SystemDeviceEnum.

Una vez creada el objeto, ahora se procede a enumerar los dispositivos de captura de video, para ello la interface IEnumMoniker apunta a pEm, y se procede a crear la clase del Enumerador, en este caso deseamos solo tener los dispositivos de la categoría de entrada de video.

```
        CComPtr< IEnumMoniker > pEm;
        pCreateDevEnum->CreateClassEnumerator( CLSID_VideoInputDeviceCategory,
&pEm, 0 );
        ASSERT(pEm);
  if( !pEm )
    return;
        pEm->Reset( );
```

Una vez ya sabido que dispositivos se van a mostrar, Iremos al primer dispositivo encontrado, para ello una interface IMoniker que apunta a pM, será creada para abarcar el primer dispositivo.

```
while(1)
{
        ULONG ulFetched = 0;
        IMoniker *pM;
hr = pEm->Next( 1, &pM, &ulFetched );
if( hr != S_OK )
  break;
```

Con el método Next se encuentra el primer dispositivo, generalmente es de acuerdo al un orden alfabético. Una vez encontrada se procede a buscar las propiedades del dispositivo de video, para ello la interface IpropertyBag apunta a pBag.

```
        CComPtr< IPropertyBag > pBag;
hr = pM->BindToStorage( 0, 0, IID_IPropertyBag, (void**) &pBag );
if( hr != S_OK )
  continue;
```

Ahora buscamos el nombre del dispositivo y lo almacenamos en la variable var de la clase CComVariant.

```
  CComVariant var;
var.vt = VT_BSTR;
hr = pBag->Read( L"FriendlyName", &var, NULL );
if( hr != S_OK )
  continue;
```

Para que se visualice en nuestro proyecto el nombre de la cámara se añade a nuestro dialogo un control de texto (static text). Las propiedades para este control será:

ID = IDC_CAPOBJ

Ahora Procedemos a colocar el nombre del dispositivo de la cámara en nuestro Static text.

```
  USES_CONVERSION;
SetDlgItemText( IDC_CAPOBJ, W2T( var.bstrVal ) );
```

Y ahora preguntamos por el filtro actual utilizado, para que sea la salida de esta función.

```
        hr = pM->BindToObject( 0, 0, IID_IBaseFilter, (void**) ppCap
    if( *ppCap )
      break;
```

A continuación el resultado de nuestra aplicación en la figura 8.17. Esta cámara es una cámara de video analógica, con tecnología CCD.

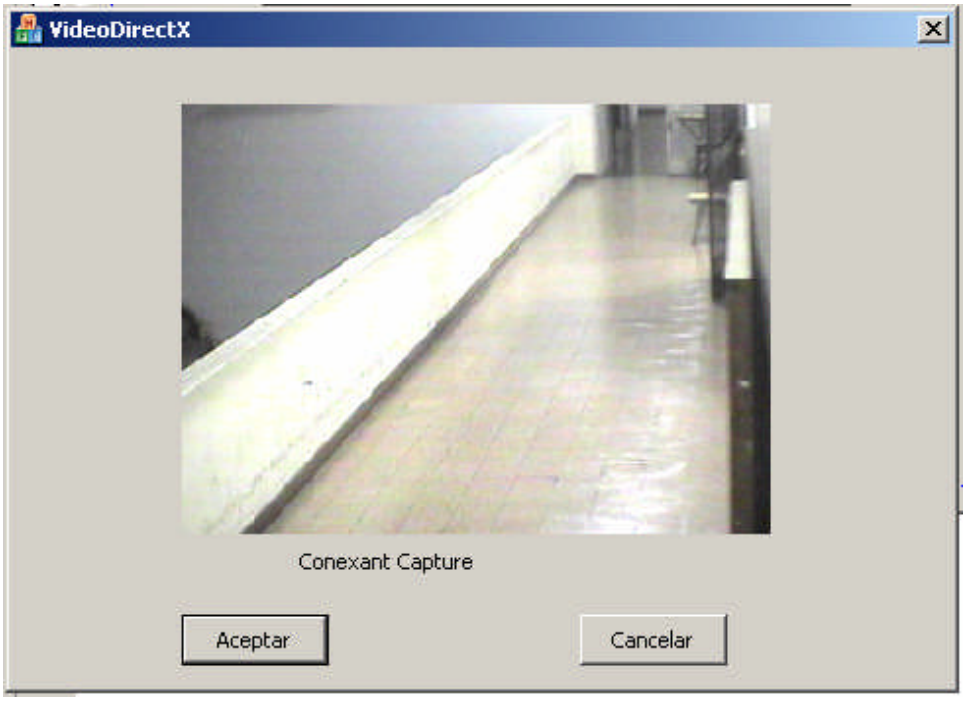


Fig. 8.17. *Vista de una cámara, resultado de nuestra aplicación*

Para realizar esta toma, el grafo de filtro esta mostrada en la figura 8.18. Aquí se puede ver que ya no es necesario un filtro de transformación, ya que la tarjeta sintonizadora de TV instalada tiene su propio filtro de transformación por hardware, con esto la PC ya no procesa dicha información, como en el caso de un webcam que necesitaba de un filtro de transformación.

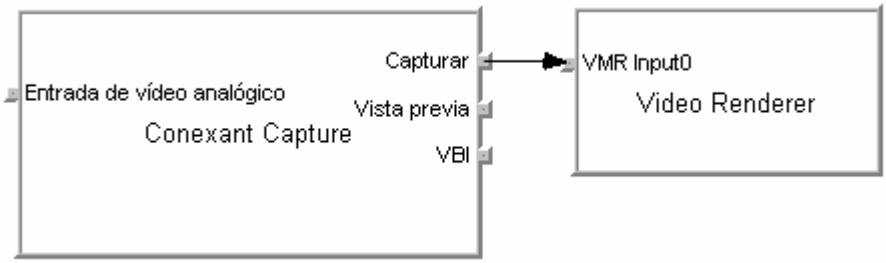


Fig. 8.18. Grafo de filtro para una cámara analógica por medio de una tarjeta sintonizadora de TV.

8.2.2 Librería OpenCV de Intel.

Para implementar el sistema se ha recurrido a una librería de código abierto, respaldada por una gran empresa y utilizada en la actualidad por muchos investigadores independientes. Esta librería no se limita a ofrecer la operación de calibración de cámaras, si no que es una librería genérica de funciones básicas para la implementación de aplicaciones de Visión por computador. La librería se llama Open Source Computer Vision Library (OpenCV en adelante) y ha sido desarrollada con el soporte de Intel.

El grupo de Interactividad Visual en el laboratorio de investigación de microprocesadores de Intel inició el desarrollo de esta librería en 1999 y ha ido liberando versiones durante este tiempo. La versión actual se puede obtener de []. El objetivo de los desarrolladores es establecer una comunidad de visión “Open Source” y dar un lugar donde se puedan consolidar los esfuerzos distribuidos de sus colaboradores. La librería está pensada tanto para uso de investigación como uso comercial.

Intel ha reclutado un comité de expertos en visión para coordinar la aceptación de nuevos contenidos en la librería y para decidir la línea general de la librería. Además, también contribuye con versiones en ensamblador altamente optimizadas para el código más costoso en términos de complejidad computacional. Por ello, la librería está optimizada para la familia de procesadores Intel, aunque funciona en procesadores compatibles con Intel (por ejemplo, AMD). Está disponible para las versiones de 32 bits de Microsoft Windows y Linux.

La librería tiene tres posibles áreas de uso:

Comercial: código optimizado para uso en aplicaciones comerciales que es de libre uso, modificación y redistribución (sujeto a la licencia)

Investigación: un substrato común para diseminar nuevos trabajos que, además, están optimizado

Docente: el código está documentado y los algoritmos se pueden utilizar en la enseñanza.

En cuanto al requerimiento del sistema, los requerimientos de hardware en general son una PC con un microprocesador basado en Pentium MMX, Pentium Pro, Pentium III o Pentium 4 y la cantidad de memoria deberá ser apropiado al tamaño de las imágenes con que se trabaje. Aunque cabe resaltar que el hardware que se recomienda es un microprocesador basado en Pentium III o Pentium IV, puertos USB para las cámaras y 256 Mb de Memoria RAM.

En cuanto a los requerimientos de Software son Win98, WinNT4.0 o Win2000 y Microsoft Visual C++ 6.0 (con el ultimo servicie pack). Las librerías de IPL deben estar

instaladas ya que el manejo de imágenes que realiza OpenCV lo realiza a través de las definiciones de esta librería.

Instalación de OpenCV sobre Windows La pagina inicial de la librerías OpenCV (Open Computer Vision library) es la siguiente:

http://www.intel.com/technology/computing/opencv/index.htm. Para instalar las librerías la página es: http://sourceforge.net/projects/opencvlibrary/. Descargamos el modulo Opencv_core_b3_5 o superior (Actualmente es la beta5 de aprox. 16Mb para Windows). Una vez descomprimido, ejecutamos setup.exe (cerrar los programas anteriores antes), luego aceptar la licencia (donde te indica que debes instalar librerías ipl). Por defecto nos mostrara el directorio donde será colocado (c:/Archivos de programa/OpenCV), caso contrario tendrás que darle esta dirección para tener un orden. Luego se procede con la instalación.

Para la implementacion utilicé OpenCV beta 3.1, Visual C++ 6.0 Servicie Pack 5 bajo Windows 2000 luego con Visual Net (c++ 7.0) bajo Windows XP.

Se debe adecuar estas librerías en el sistema (Setup) de Visual C++.

Primero seleccionaremos la opción del menú: Tools|Options. Seleccionar la lengüeta Directories y luego añadir los archivos siguientes:

En executables files:
c:\windows\system o c:\windows\system32
c:\archivos de programa\opencv\bin

En include files:
c:\archivos de programa\opencv\cv\include
c:\archivos de programa\opencv\cvaux\include
c:\archivos de programa\opencv\otherlibs\cvcam\include
c:\archivos de programa\opencv\otherlibs\highgui

En Library files:
c:\archivos de programa\opencv\lib
c:\archivos de programa\opencv\otherlibs\cvcam
c:\archivos de programa\opencv\otherlibs\highgui

De esta manera (Global Settings), solo tenemos que seleccionar category General en la lengüeta Link, y adicionar las siguientes librerías a Object/library modules:
cv.lib highgui.lib

Por ultimo se debe copiar las dll que vienen en la carpeta BIN de la carpeta principal OpenCV , estos archivos dinámicos dll se deben copiar en Sistem32 de la carpeta de Windows.

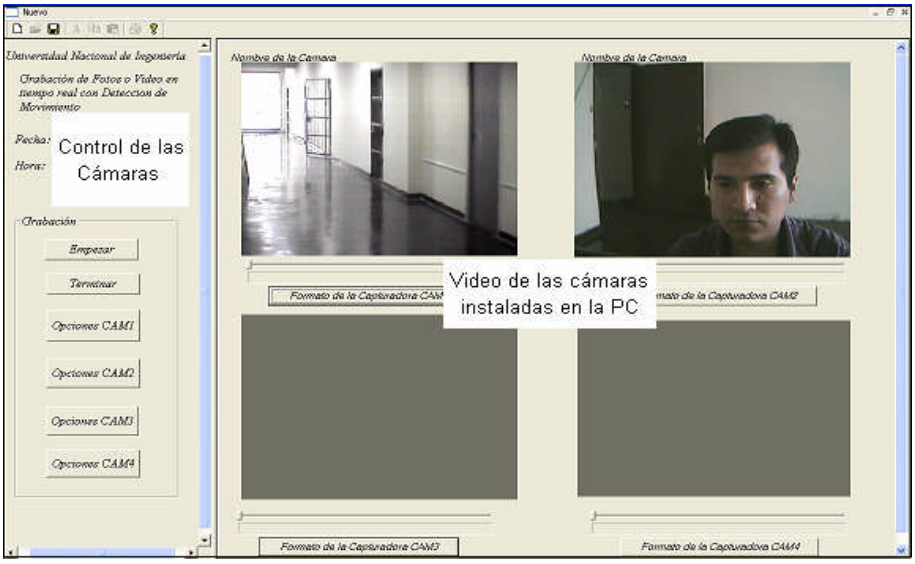
8.2.3 Arquitectura del desarrollo de software.

El prototipo inicial del software se basa en la utilización de VFM (Video for Windows) para la captura de la imagen de video. Las librerías utilizadas fueron las de Intel (OpenCV) para el procesamiento del mismo.

La idea es tener la etapa de entrenamiento y reconocimiento en una misma aplicación además de tener la opción de poder ver video de varias cámaras conectadas a la PC.

Para ello se emplea Visual C++ con la ayuda del MFC (MDI). El esquema de presentación del software se presenta en la Fig. 8.19

(a)



(b)

Fig. 8.19 *(a) Elección reconocimiento de rostro. (b) Grabación de Video*

La estructura del programa para el reconocedor de rostros se realizó utilizando como base el algoritmo del HMME (Modelo Oculto de Markov Embebido) y para ello se utilizó la librera OpenCV.

La grabación de video en tiempo real se hizo utilizando el kit de desarrollo de DirectX, en este caso DirectShow, y se realizo con los codec's que este soportaba.

El Sistema se desarrollo en las plataformas Window2000/XP y tiene como entrada una base de datos de imágenes almacenadas en carpetas personales donde cada una de ellas contendrá las imágenes de un individuo particular.

Estructura del desarrollo de la Clase Base de Rostros

<Carpeta de la Base>

<Persona 1>

<imagen1_1>

<imagen1_2>

<imagen1_3>

...

info.txt

<Persona 2>

<imagen2_1>

<imagen2_2>

<imagen2_3>

...

info.txt

...

<indice_archivo>

Esta fue la estructura que se realizo para adicionar las personas en nuestra base. Por Ejemplo:

Base_FIEE

José Huamán

Jose1.bmp

Jose2.bmp

Jose3.bmp

Indice.txt

Amilcar Mescco

Amilcar1.bmp

Amilcar2.bmp

Amilcar3.bmp

Indice.txt

Índice Base_FIEE

El Índice Base_FIEE contiene el nombre de la Base, en este caso Base_FIEE, y una lista de los nombre de cada persona. El formato es el siguiente:

Línea	Contenido	
1	Base_FIEE	//carpeta de la Base
2	Nombre y/o Descripción de la Base	
3	<Línea vacía>	
4	Persona 1	
5	Persona 2	
6	...	

Indice.txt que aparece en cada persona, contiene el nombre de la persona y una lista de los archivos de cada imagen de la persona. El formato es el siguiente:

Línea	Contenido
1	Persona 1 //Nombre de la persona
2	Nombre y/o Descripción de la Persona
3	<Línea vacía>
4	Nombre de la imagen1
5	Coordenada1
6	Nombre de la imagen2
7	Coordenada2
6	...

Coordenada1 o Coordenada2, es la coordenada de la imagen cortada, es decir Coordenada estará representada por 4 números: coordenada súperior izquierda y el ancho y largo de la imagen. Si no se especifica nada en este parámetro, se considera entonces que toda la imagen a sido seleccionada.

Para realizar lo anterior se crearon 3 Clases, estas fueron:
CPersonaImagen, donde cada individuo esta incluido en una carpeta personal, que define a la persona y para este caso se define la clase **CPersona**. Y el conjunto de personas (carpetas personales) es una base de datos que se definirá en la clase **CBaseRostro**.

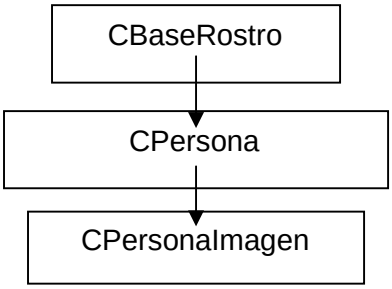


Fig. 8.20. *Estructura de las clases a desarrollar*

Class CBaseRostro
Class CPersonaImagen
Class CPerson

CBaseRostro, contiene métodos importantes como son**:**
void EntrenarTodo(int flag)

```
void ColocarArchivo(const CString& nombrearchivo)
void ColocarNombre(const CString& nombre)
CPersona* AdicionarPersona(const char* nombre,
                           const char* folder,
                           bool impotar_datos)
void RemoverPersona(POSITION pos)
void ColocarTamanioImage(CSize tamanio)
CImage& ConsigueImagenEntrenada()
.
.
```

CPersona, contiene métodos importantes como son:

```
void ColocarNombre(const CString& nombre)
void ColocarFolder(const CString& folder)
void AdicionarImagen(const char* nombrearchivo,
                     CImage* importar_imagen,CRect roi)
void RemoverImagen(POSITION pos)
CPersonImgList& ConsigueListaImagen()
.
.
bool Leer()
bool Salvar()
```

CpersonaImagen, contiene métodos importantes como son:

```
void ColocarArchivo(const CString& nombrearchivo )
void ColocarCoordenada(CRect r)
CImage& ConsigueImagen()
bool Leer()
.
.
bool Salvar()
```

Clase CList , Aquí se realizara la implementación de las clases como elementos de una lista doblemente enlazada tanto para las imágenes de cada persona como para cada persona. Para ello es necesario utilizar la clase CList , esta clase necesita dos parámetros: el tipo de datos de la lista y la referencia base.

Una lista y un vector se diferencian en las inserciones, pues en una lista son mucho menos costosas en tiempo de ejecución y esto se debe a que solamente añade un elemento y no recompone todo el contenido. Por otro lado, el tiempo necesario para encontrar un elemento dentro de la lista depende de la posición del elemento dentro de ella, ya que el acceso es secuencial.
Otra diferencia es que en un vector se utiliza la posición de los elementos para acceder a cada uno de ellos. En cambio en una lista utiliza elementos del tipo POSITION (que realmente son punteros) y el acceso a cada elemento, ya no es secuencial, sino que podemos movernos directamente hasta el.

De acuerdo a esto se define lo siguiente:
Typedef CList<CPersonaImagen *, CPersonaImagen *> CPersonaImagenLista
Typedef CList<CPersona *, CPersona *> CPersonaLista

Ahora aprovecharemos las librerías de OpenCV para utilizar el algoritmo de HMM embebido, para ello primero definiremos la estructura que representan estos modelos:
CvHMM, es la estructura del HMM Embebido

```
typedef struct _CvEHMM     //Define la estructura del modelo
{
    int level;
    int num_states;
    float* transP;
    float** obsProb;
    union
    {
        CvEHMMState* state;
        struct _CvEHMM* ehmm;
    } u;
} CvEHMM;
```

level

Es el nivel del HMM embebido. Si *level ==0*, HMM es lo mas externo. En 2DHMM estos son de dos tipos de HMM: un externo y un embebido severo. HMM Externo tiene el level==*1* y los HMMs embebidos tienen *level ==0* .

num_states

Numero de estados en 1D HMM.

transP

Matriz de transición de probabilidad, Matriz cuadrada *(num_state×num_state)*.

obsProb

Matriz de probabilidad de Observación

state

Arreglo de estados de HMM. Para el Ultimo nivel de HMM, esto es, un HMM por fuera embebido, Los estados de HMM son reales.

ehmm

Arreglo de HMMs embebidos. Si el HMM no es del Ultimo nivel, entonces los estados de HMM no son reales real y ellos son HMMs.

Para representar las observaciones, se define la siguiente estructura:

CvImgObsInfo, Estructura de la Observación de una Imagen

```
typedef struct CvImgObsInfo
{
    int obs_x;
    int obs_y;
    int obs_size;
    float** obs;
    int* state;
    int* mix;
} CvImgObsInfo;
```

obs_x

Número de observaciones en la dirección horizontal.

obs_y

Número de observaciones en la dirección vertical.

obs_size

Longitud de cada vector de observación.

obs

Puntero para el vector de observación guardado por consiguiente. El número de vectores es: *obs_x*obs_y* .

state

Arreglo de índices de estados, asignado para cada vector de observación.

mix

índice del componente mixto, corresponde al vector de observación dentro de un estado asignado.

Create2DHMM, Crea HMM 2D embebido.

CvEHMM* cvCreate2DHMM(int* stateNumber, int* numMix, int obsSize);

stateNumber

Arreglo, el primer elemento en el cual especifica el número de súper estados en HMM. Todos los subsiguientes elementos especifican el numero de estados en cada HMM embebido, correspondiente a cada súper estados. De esta manera, la longitud del array es stateNumer[0]+1.

numMix

Arreglo con números de componentes mixtas gausianas por cada estado interno. El Número de Elementos en el arreglo es igual al numero de estados internos en el HMM, esto es, Los súper estados no se cuentan aquí.

obsSize

Tamaño del vector de observación para ser usado con el HMM creado

La función cvCreate2DHMM retorna la estructura creada del tipo CvEHMM con sus parámetros especificados.

Para el desarrollo de la primera versión del software se utilizo 3 clases principales la primera de ellasla clase CBaseRostros y donde se definió las siguientes funciones:

SetFileName, Obtiene la ruta de la base de datos

SetName, Obtiene el nombre de la base de datos

GetRootFolder(char* root_folder, int* root_path_len), Nos da la ruta del fólder y su longitud Ejemplo C:\Entrenado , construye la raíz de la ruta del fólder a partir de la ubicación de rostros.txt.

GetPersonFolder (const char* root_folder, int root_folder_len, const char* person_folder, char* folder), Retorna la ruta del nombre del fólder personal (C:\\Entrenado\\rostros) a partir de la ruta de la base de datos (C:\\Entrenado)

GetPersonSubFolder(const char* folder, char* subfolder), A partir de un archivo del folder personal obtiene el archivo, ejemplo: hmm.txt

Load(), Cargo las imágenes de cada carpeta personal de acuerdo a lo leído del archivo rostros.txt

Unload(), Descarga las imágenes hasta que queden vacías de la lista.

Save(), Guarda toda la información personal y la carpeta. (rostros.txt, CFG.txt)

AddPerson(const char* name, const char* folder, bool import_data), Adhiero personas de acuerdo al nombre o al folder al que pertenecen y lo coloco en la lista.

RemovePerson(POSITION pos), Remueve una persona de la base de datos

SetImageSize(CSize size), Coloca el tamaño de las imágenes.

GetPerson(int index), Retorna una persona de la lista, de acuerdo a un índice.

FindPersonByName(const CString& name), Si name es igual al nombre de la persona, entonces se retorna la persona.

GetPersonIndex(CPerson* person), Retorna un índice de la lista dada la persona

Draw(int index, CImage& img, SIZE win_size, int y_pos, SIZE base_size, SIZE delta), Dibuja la imagen en una vista predeterminada por el tamaño a elegir.

UpdateTrainedImage(), Actualiza la ventana del cliente de window m_base_view

DeleteHMMInfo(), Borro información de entrenamiento de todas las carpetas personales hmminfo.txt

La segunda clase **Cperson** en donde se definio las siguientes funciones:

GetPersonFullImageName(const char* root, int root_len, const char* image, char* full_image_name), A partir de la imagen y la carpeta personal root se obtiene la ruta completa de la imagen.

ExtractPersonImageName(const char* full_image_name, char* image),
Extraigo el nombre de la imagen(persona) de toda la ruta de ella.

Load(), Carga en el buffer (Lista) la primera imagen (leídas del archivo info.txt) de la carpeta personal y los datos entrenados previamente, si no lo están los entrena.

Unload(), Descargo del Buffer (Lista de Imágenes personales) todas las imágenes.

Save(), Guarda parámetros en carpeta personal hmminfo.txt info.txt

LoadRest(), Lee el resto. lee todas las imágenes de caras comenzando desde el segundo(el primero es siempre leído).

GenerateFileName(const char* base, char* filename), Genera el nombre del archivo en la carpeta, ejemplo: Amilcar0000, Amilcar0001 cambiando la ultima posición.

NormalizeIntensity(IplImage* ipl_image, int level), Normaliza la intensidad de la imagen dado un cierto nivel.

ExtractDCT(float* src, float* dst, int num_vec, int dst_len), Extraer el DCT de una imagen.

AddImage(const char* filename, CImage* import_image, CRect rect), Colocar una nueva imagen a una base ya entrenada y guardarla.

RemoveImage(POSITION pos), Remueve la imagen de la persona de la carpeta personal.

MoveToTop(POSITION pos), Moverse una imagen de la lista a la parte superior de la lista de imágenes.

DeleteHMMInfo(), Elimino información de entrenamiento.

LightingCorrection(IplImage* ipl_image), Corrección de la iluminación.

Para entrenar el sistema, calcula los parámetros de entrenamiento y guardar en archivos personales correspondientes hace falta leer.

El número máximo de interacciones será de 80, este bucle puede ser no convergente.

Para el entrenamiento se procedió a crear la estructura del HMM embebido, luego se creo el HMM2D para ello se necesita saber:

- La cantidad de súper estados que existen y también la cantidad de estados por cada súper estado.
- Las componentes mixtas gausianas por cada estado.
- Tamaño del vector de observación

Entonces esto retorna la estructura del HMM embebido, para el entrenamiento se procede a alimentar al sistemas con rostros, inicialmente debemos tener una carpeta donde estarán la base de datos, en esta carpeta se pondrá un archivo de texto nos indique la cantidad de personas que entrara para el entrenamiento. Ejemplo:

FACE DATABASE
Base de 5 rostros por 5 personas
Amilcar
Angel.Cucho
Blanca
Carlos
Enrique.Llanos

El sistema almacenara las imágenes en varias carpeta con el nombre de cada persona. Por ejemplo Amilcar; Cada carpeta una archivo como este:

PERSONAL INFO
Amilcar
Amilcar0000.bmp
Amilcar0001.bmp
Amilcar0002.bmp
Amilcar0003.bmp
Amilcar0004.bmp

Y también tendrá los parámetros iniciales de entrenamiento en un archivo: untitled_baseCFG.

**********Parámetros de Muestreo**********

WindowWidth 12

WindowHeight 12

DeltaX 4

DeltaY 4

DCTcoeffX 3

DCTcoeffY 3

**********Parámetros HMM **********

SuperStates 5

States 3 6 6 6 3

NumMixtures 3

********** Otros Parámetros **********

SUPPRESS_INTENSITY 1

Una vez obtenido la base de datos y los parámetros iniciales (se pueden modificar los parámetros iniciales). Se procede al entrenamiento. El entrenamiento creara en cada carpeta personal un archivo como: hmm.txt

Esta contendrá el entrenamiento del sistema así como los valores de la matriz de transición, el vector termino medio, la desviación inversa, y el LogVarVal.

Se desarrollo el software donde se muestra el video en nuestra aplicación utilizando VFW.

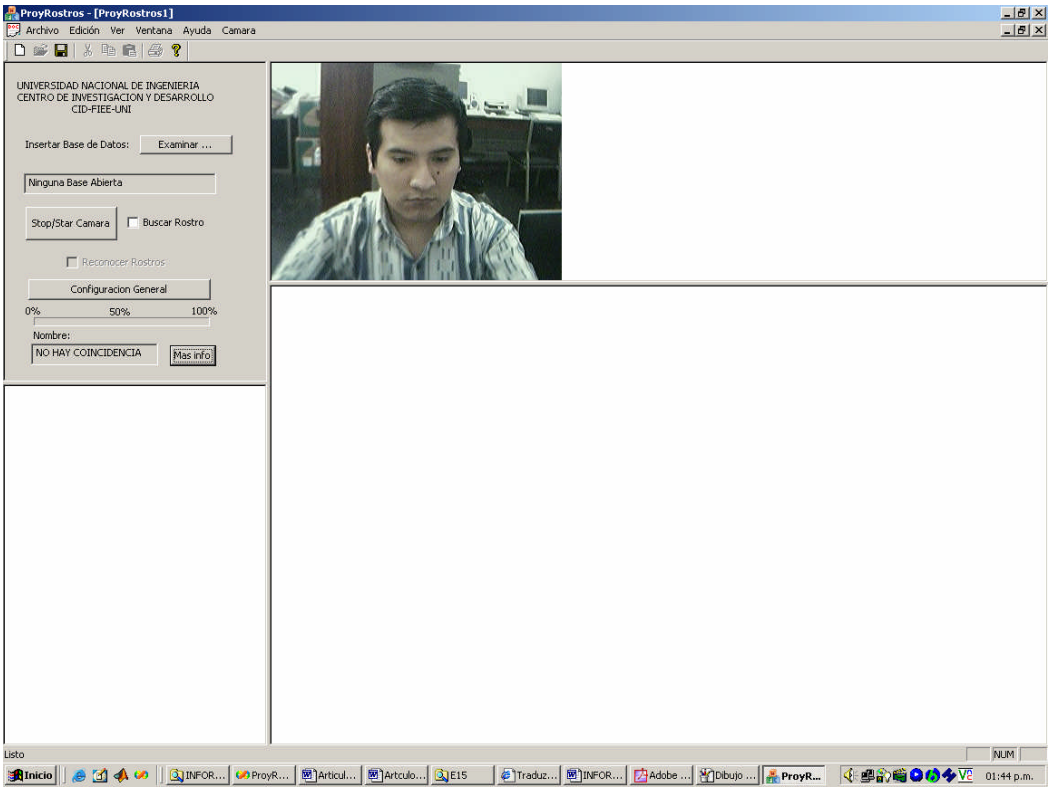


Fig. 8.21. *Adquisición de imagen de video*

Para la localización del rostro se procedió a utilizar el clasificador (*cascade of boosted classifiers)* que consiste en tener plantillas de rostros de diferentes tamaños (en cascada), y aplicar algún método (boosted), que clasifica cada etapa de la cascada. La función llamada cvHaarDetectObjects que viene en la librería de OpenCV busca la región rectangular en una imagen dada, y retorna los rostros como una secuencia de rectángulos. Esta función escanea la imagen en varios tiempos y con diferentes escalas.

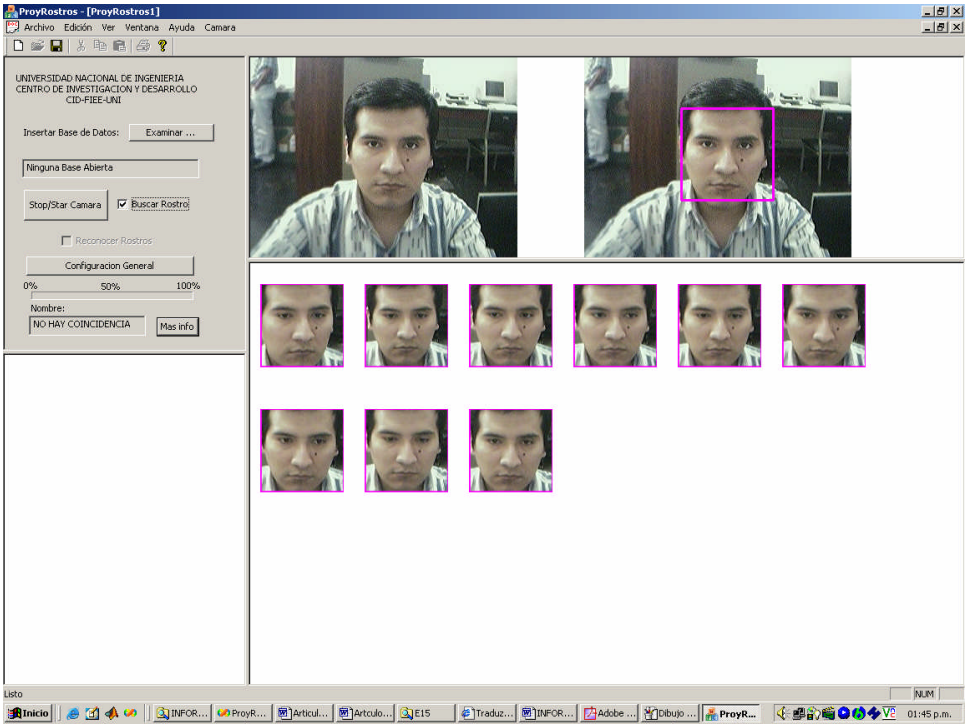


Figura 8.22. *Adquisición de rostros y captura automática de los rostros en una imagen de video en tiempo real.*

8.2.4 Diagramas de Bloques de los algoitmos utilizados

Como ahora el objetivo es llevar todo el código a componentes, con el fin de que el sistema sea mas robusto y sea aplicable con la nueva tecnología de DirectShow. Anteriormente se definio la arquitectura del desarrollo del software, ahora se pretende optimizar.

Aquí se muestra el diagrama de la figura 8.23 del primer objetivo. El primer bloque a diseñarse muestra todos los dispositivos de captura de video que se encuentran en el sistema operativo. Con ellos se podrá facilitar la elección de la cámara que nosotros deseamos manejar. Este bloque también utilizara componentes COM, o sea también hará uso de los filtros de DirectX.

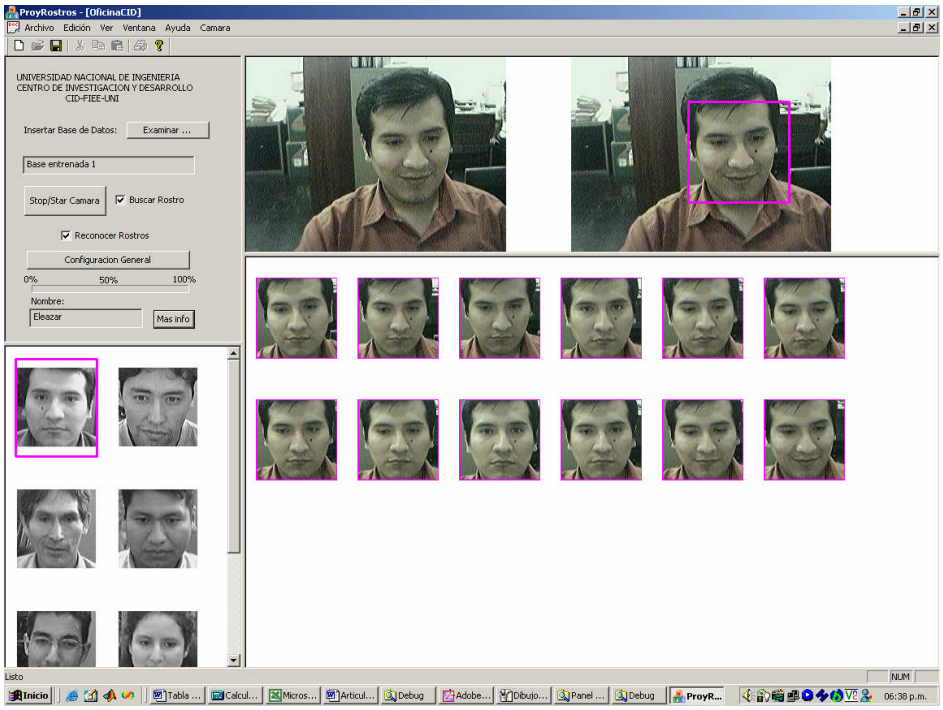


Fig. 8.23. *Reconocimiento de Rostros de acuerdo a una base de datos entrenada por HMME.*

El segundo Gran Bloque es el Filter Graph Manager que es el encargado de usar los bloque del filtro, transformación y Traducción (renderer) , para el proceso del mismo a través de las interfaces que maneja. Aquí los pines de conexión cumple su papel de concatenar los filtros.

El tercer bloque es la presentación en el monitor, esta ocultamente hace uso del componente DirectDraw. Una forma de saber como funcionan estos filtros se puede encontrar en la aplicación de GraphEdit que viene con el kit de desarrollo de DirectX. En DirectShow cada bloque es un filtro, y el conjunto de bloques se le suele llamar Diagrama de grafos. Todos los bloques tienen una función específica.

El primer bloque es el filtro fuente (Filter Source) y es la fuente de donde saldrá nuestro video. El segundo bloque es el filtro de transformación que sirve para cambiar el formato de video.

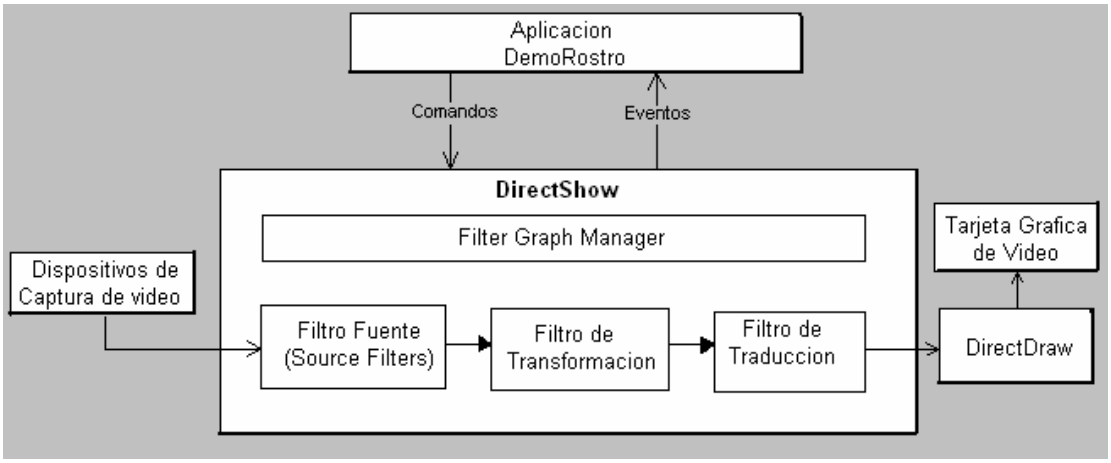


Fig. 8.24. *Diagrama de la aplicación DemoRostro*

El tercer bloque es el filtro de salida a hardware figura 8.25, este filtro traduce el tipo de formato de video y lleva las imágenes al componente DirectDraw y este al monitor de acuerdo a la configuración de resolución del mismo. Esta es una manera de comprender mejor la función de cada filtro.

Fig. 8.25. *Diagrama que origina GraphEdit, como ejemplo una WebCam.*

El propósito es poder utilizar todos los dispositivos capturadoras de imágenes de video, específicamente Cámaras WebCam, tarjetas de adquisición de imágenes, etc .. Instaladas en el computador a través de un listado. Para realizar todo ello se hará uso de la tecnología de DirectX y de algunos controles de Visual C++.

Al hacer clic en el botón “Mostrar Cámaras”, automáticamente aparecen los dispositivos de video instalados en nuestra PC.

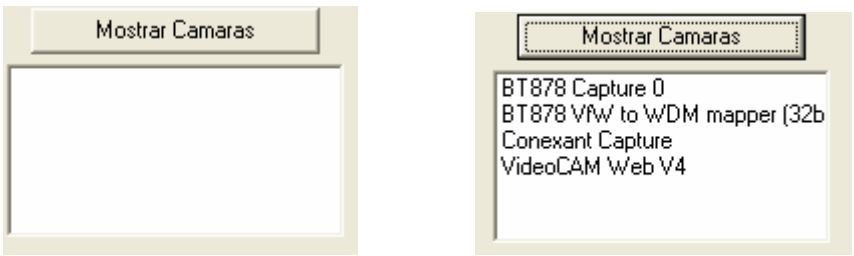


Fig. 8.26. *Cuadro de lista que muestra los dispositivos capturadores de video*

Para poder hacer esto crearemos una función que será empleada en la clase CMainFrame quien recibirá un mensaje de control de un botón, esta a la vez originara que se liste los dispositivos en una lista de control.

```
void CMainFrame::enumFilters(REFCLSID CLSIDcategory, std::vector<CString>& names,
std::vector<CLSID>& clsidFilters)
{…
…
…
}
```

El primer parámetro es el tipo de dispositivo que se quiera enumerar, el segundo parámetro incluirá el nombre del dispositivo y el tercer parámetro se incluye su identificador GUI en el sistema operativo, cada dispositivo tiene un GUI diferente.

Esta función implementada que se encuentra en el anexo, será llamada mediante el siguiente código:

```
std::vector<CString> fname;
std::vector<CLSID> fclsid;
mainframe->enumFilters(CLSID_VideoInputDeviceCategory, fname, fclsid);
```

Primero se define a fname y fclsid del tipo “Vector” (Array) que incluirá datos CString y CLSID respectivamente, este tipo se encuentra implementado en una librería al que incluiremos en la cabecera:

```
#include <vector>
```

Luego se lama a EnumFilters que recibe como primer parámetro a CLSID_VideoInputDeviceCategory que mostrara los dispositivos de captura de video.

Una vez obtenido el nombre y el identificador, pasaremos a mostrar estos mediante el siguiente código:

```
m_list.ResetContent();
for (int i=0; i<fname.size(); i++)
m_list.AddString(fname[i]);
```

Con esto incluiremos los nombres de los dispositivos que se encontraron tal como se muestra en la Fig. 8.26.

Ahora se procede a instanciar un dispositivo y mostrarlo en parte de la pantalla. Primero se crea el objeto Filter Graph Manager mediante CoCreateinstance, esto a través de un puntero a IGraphBuilder.

```
IGraphBuilder *m_pGraph;
IMediaControl *m_pMediaControl;
IVideoWindow*   m_VideoWindow;
```

Y se Inicializa de esta manera:

```
CoInitialize(NULL);
m_pGraph= NULL;
m_pMediaControl= NULL;
if (!FAILED(
  CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC,
  IID_IGraphBuilder, (void **)&m_pGraph)))
 {
        m_pGraph->QueryInterface(IID_IMediaControl,
        (void **)&m_pMediaControl);
        m_pGraph->QueryInterface(IID_IVideoWindow, (void**)&m_VideoWindow );
}
```

Con este código ya inicializamos el grafo de filtro y a la vez tenemos los controles de m_pMediaControl y m_VideoWindow.

Una vez realizada esto, el botón de OnFileNew lo utilizaremos para inicializar las imágenes de video de acuerdo al dispositivo elegido.

```
void CMainFrame::OnFileNew()
{
        m_pMediaControl->Stop();
        if(m_pGraph)
        removeAllFilters(m_pGraph);
        createFilterGraph();//creamos los objetos necesarios
        VerVideo();
}
```

La primera línea indica que si hubiera un video inicializado, lo tendríamos que parar medio el control indicado. La segunda y tercera línea implica que si m_pGraph a sido inicializado anteriormente, entonces que se destruyan los objetos anteriores

mediante removeAllFilters(m_pGraph). La cuarta línea creara todos los filtros necesarios para inicializar el video, osea quedar a punto de mostrar. Esta función se muestra en el anexo.

Ahora en el código anterior VerVideo() realiza lo siguiente:

```
CMainFrame::VerVideo();
{
if( m_pMediaControl )//si esta listo procedemos a mostrar
{
CWnd* av = GetDemoRostroView();
RECT rc;
::GetWindowRect( av->m_hWnd, &rc );

        m_VideoWindow->put_Owner((OAHWND)av->m_hWnd);
        m_VideoWindow->put_Left( 0 );//los 4 parámetros siguientes para
        ocupar la pantalla completa
        m_VideoWindow->put_Top( 0 );
        m_VideoWindow->put_Width( rc.right - rc.left );
        m_VideoWindow->put_Height( rc.bottom - rc.top );
        m_VideoWindow->put_WindowStyle(WS_CHILD|
        WS_CLIPSIBLINGS|WS_CLIPCHILDREN);
        m_VideoWindow->put_MessageDrain((OAHWND)av->m_hWnd);
    m_pMediaControl->Run();//mostramos el video
}
}
```

Una vez listo, se procede a mostrar. Primero se obtiene la ventana donde queramos que se muestre el video en este caso elegiremos GetDemoRostroView() que recoge la primera ventana izquierda. Una vez obtenido dicha ventana se procede a capturar sus dimensiones mediante un rectángulo.

Realizado esto se procede a colocar la ventana con las funciones de m_VideoWindow. Luego con m_pMediaControl->Run() procedemos a mostrar la imagen de video del dispositivo elegido.

Finalmente se procede a destruir las interfaces utilizadas mediante release(). En caso de otras interfaces es necesario saber si estan o no activas, para ellos se define una función de la siguiente manera:

```
#define SAFE_RELEASE(p)
```

```
{ if( (p) != 0 )
        {
                (p)->Release();
                (p)= 0;
        }
}
```

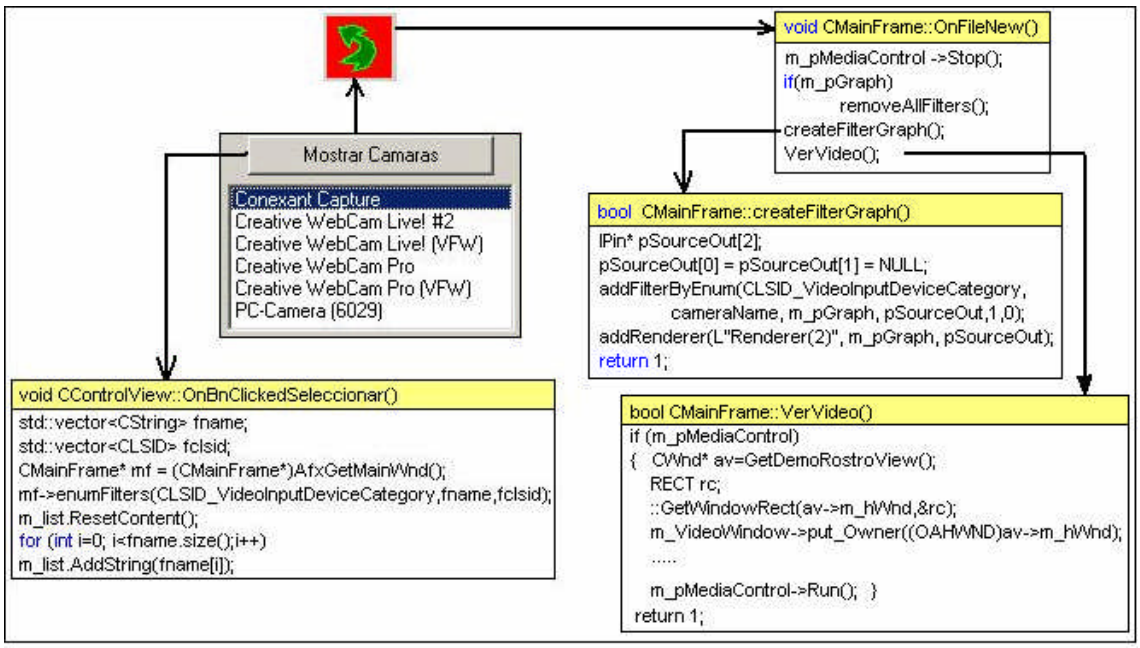


Fig. 8.27. Funciones Utilizadas para mostrar el video.

Con el presente programa se alcanzo el objetivo de lograr enumerar todos los dispositivos de captura de video de nuestra PC, además de poder visualizar las cámaras que están instaladas y conectadas. El siguiente objetivo será procesar dichas imágenes de video siguiendo con la idea de hacerlo en tiempo real.

8.2.5 Prueba y corrección de errores

Nuestra PC, Pentium IV de 2.4Ghz con memoria de 512MB y Tarjeta de Video de 64MB contiene una tarjeta sintonizadora de TV, que a la vez puede tener a su entrada video compuesto. Es por esta interface donde nosotros aprovechamos para poner una cámara analógica de entrada, tal como se muestra en la Fig. 8.28.

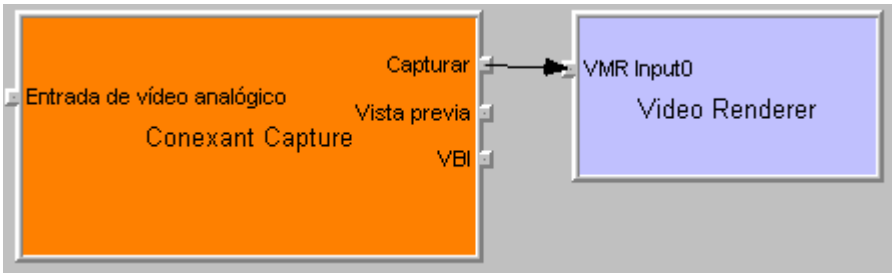


Fig. 8.28. Tarjeta sintonizadora de TV que actúa como entrada de video Analógico

Además utilizamos 3 cámaras WebCam que tienen como salida el puerto USB. Estas cámaras WebCam tienen internamente una tarjeta digitalizadora que procesa la imagen proveniente de los sensores CMOS de las cámaras. El barrido es progresivo para estos tipos de cámaras.

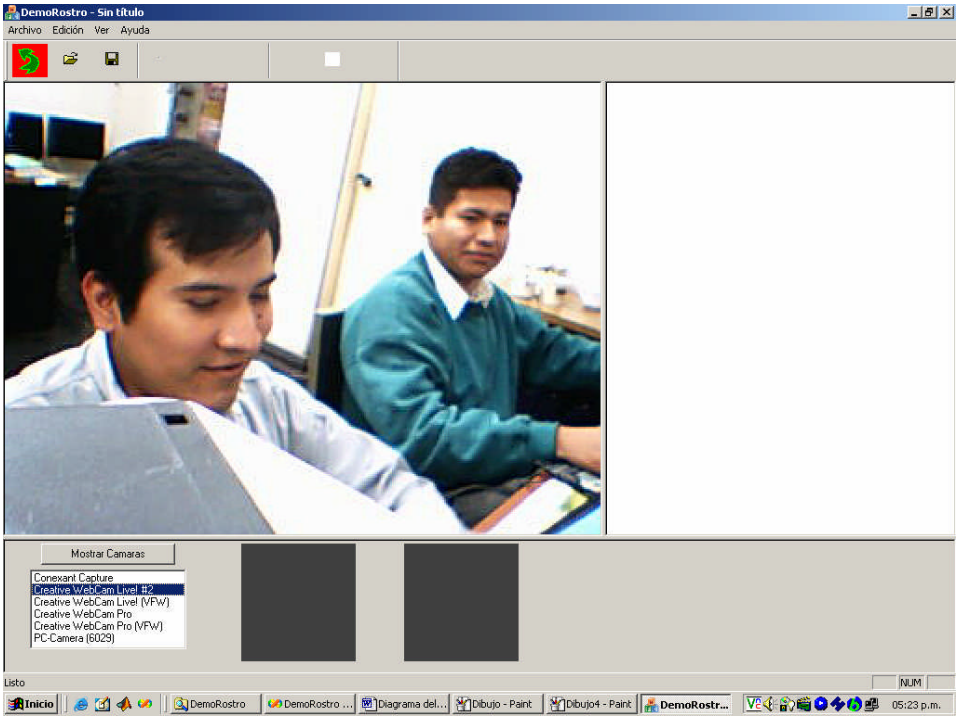


Fig. 8.29. *Vista desde una WebCam Creative Live*.

El software debe ser capaz de integrar rápidamente nuevos rostros para su entrenamiento para ello se rediseña la forma de presentación.

Inicialmente se diseña de tal manera que se pueda cambiar de cámara instalada en el computador y luego proceder a la calibración de los parámetros de brillo, contraste, saturación y matiz. (Mínimo de parámetros de cada cámara de video). Algunas cámaras incluyen la luz de fondo, nitidez, Exposición, etc . Esto se puede ver en la Fig. 8.32.

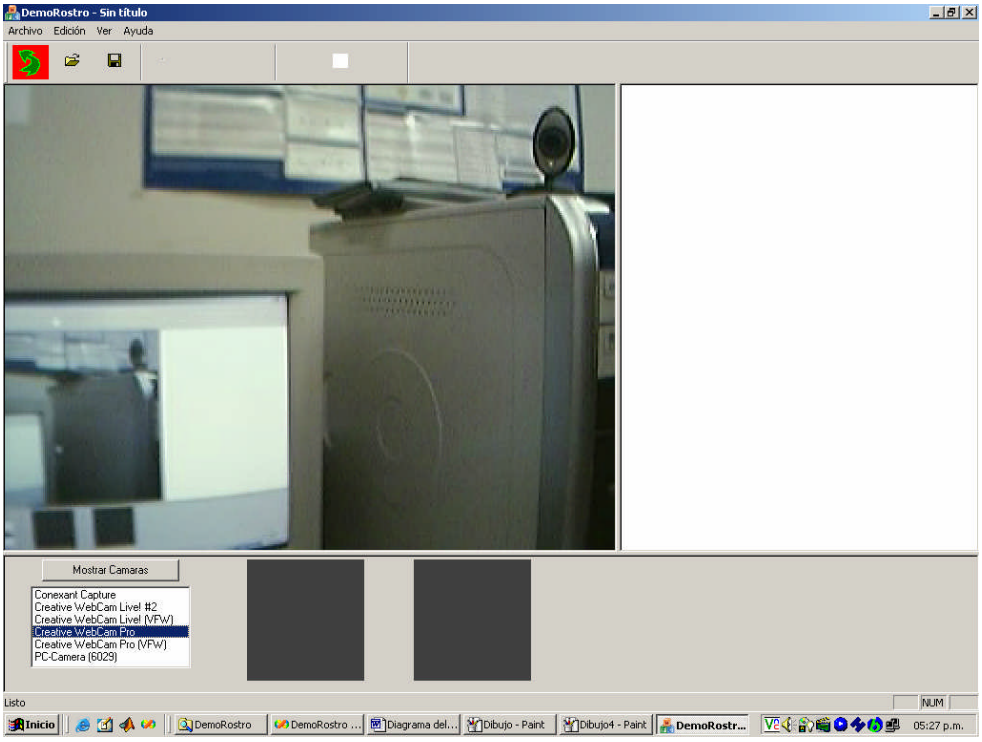


Fig. 8.30. *Vista desde una WebCam Pro*

Fig. 8.31. *Vista desde una Cámara Analógica AVC523LN/F36 color DOMO.*

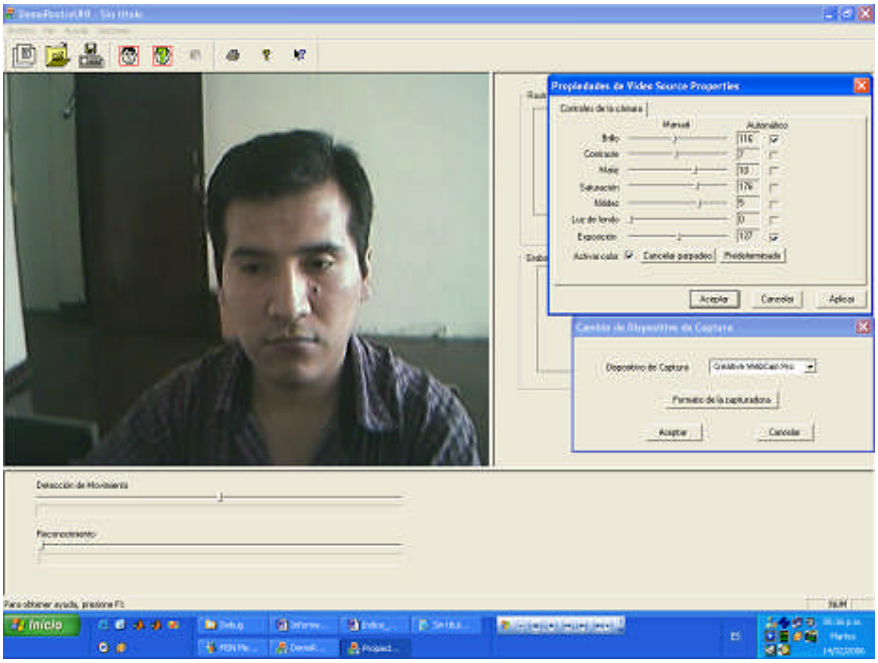


Fig. 8.32. *Propiedades de la camara WebCam*

La etapa de entrenamiento es capaz de entrenar un rostro, solamente incluyendo el nombre tal como se muestra en la Fig. 8.33.

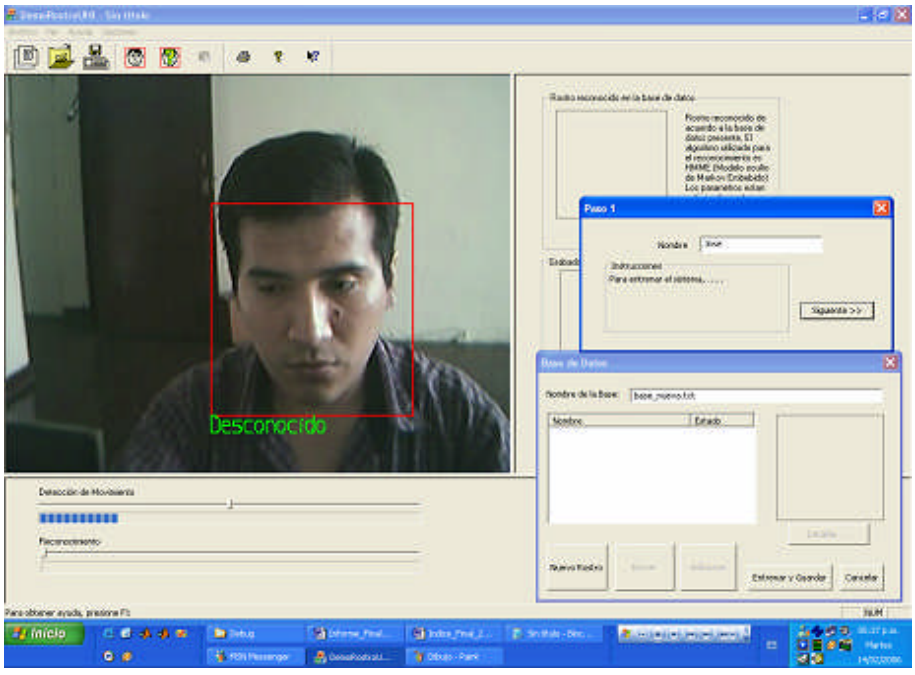


Fig. 8.33. *Procedimiento para el entrenamiento I*

Luego se escoge los rostros que mejor representen al individuo. Estas fotos pasan a formar parte del entrenamiento. De esta manera se puede incluir más individuos en la base de datos, tal como se muestra en la Fig. 8.34.

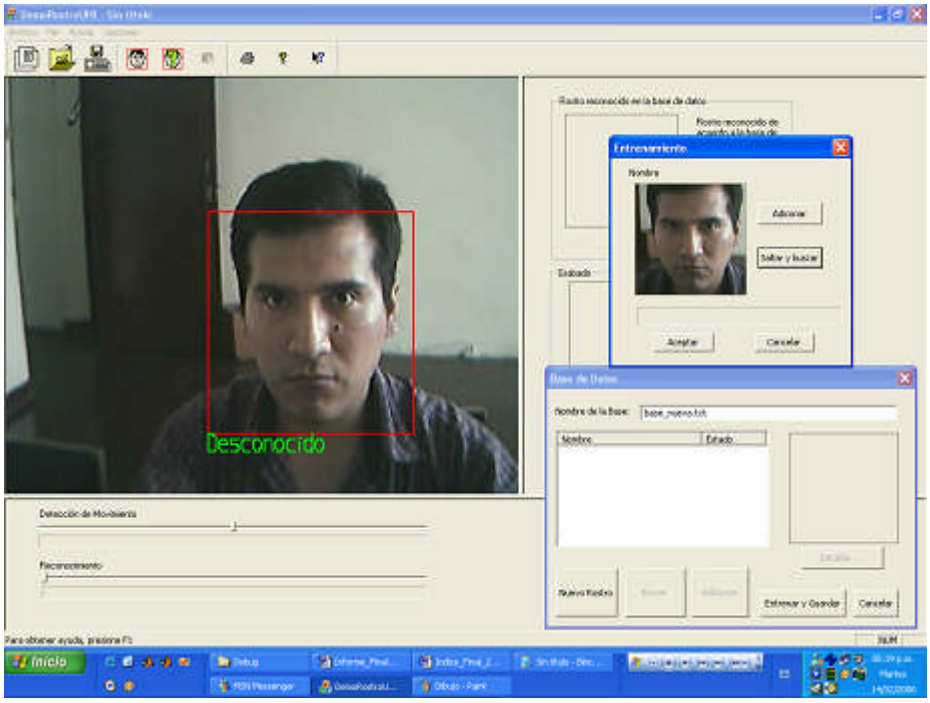


Fig. 8.34. *Procedimiento para el entrenamiento II*

Una vez que se tiene a los individuos a reconocer, se procede a entrenar el sistema, tal como se muestra en la figura 8.35.

Una vez entrenada, se procede a reconocer el rostro que se encuentra en la base de datos. Para ello solo basta con presionar el boton de reconocimiento para que el sistema proceda a grabar las imágenes reconocidas y mostrarlas en el entorno, tal como se muestra en la Fig. 8.35

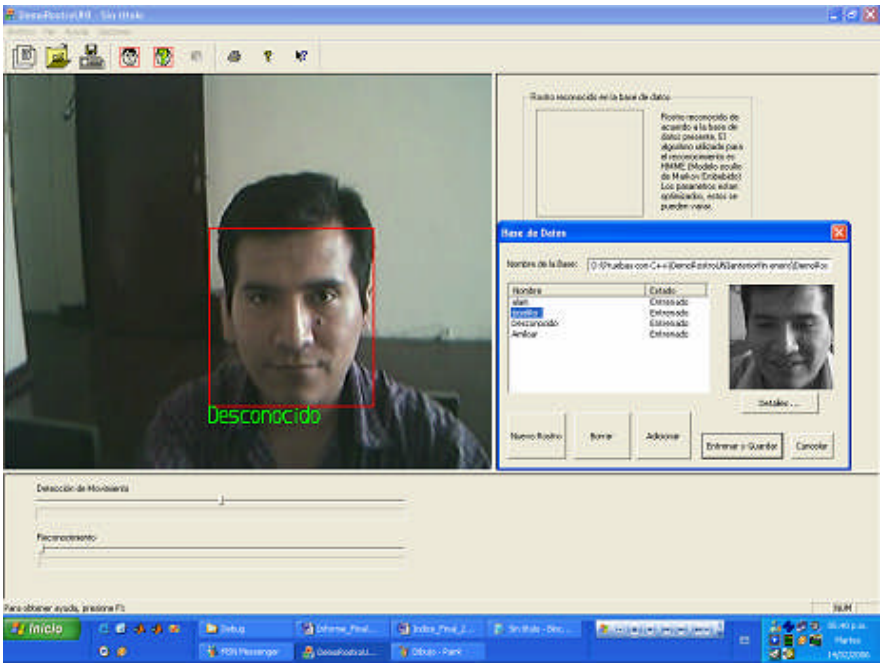


Fig. 8.35. *Procedimiento para el entrenamiento III*

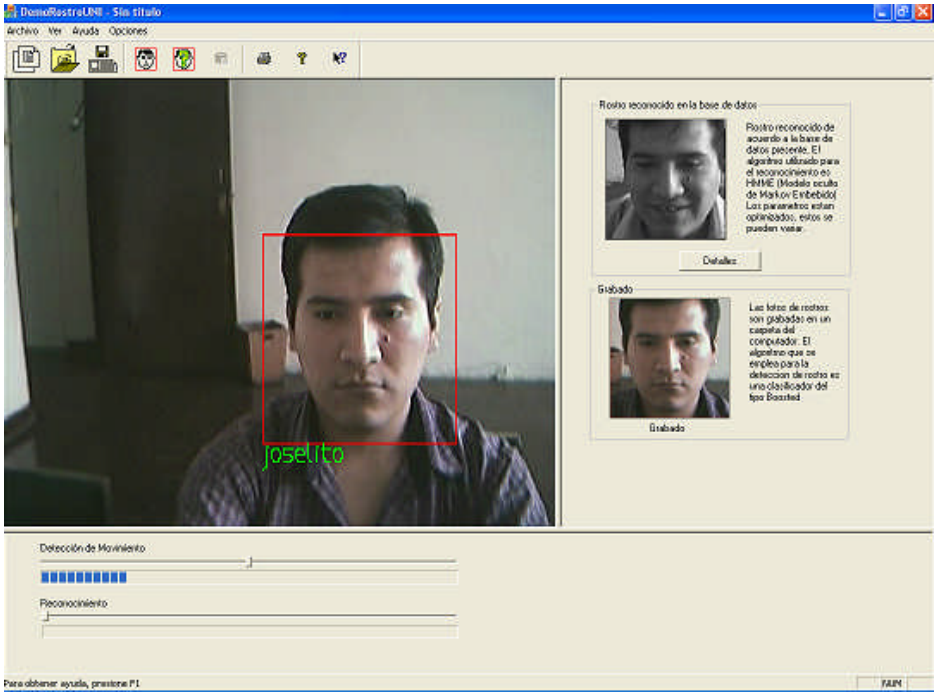


Fig. 8.36. *Procedimiento para el entrenamiento IV*

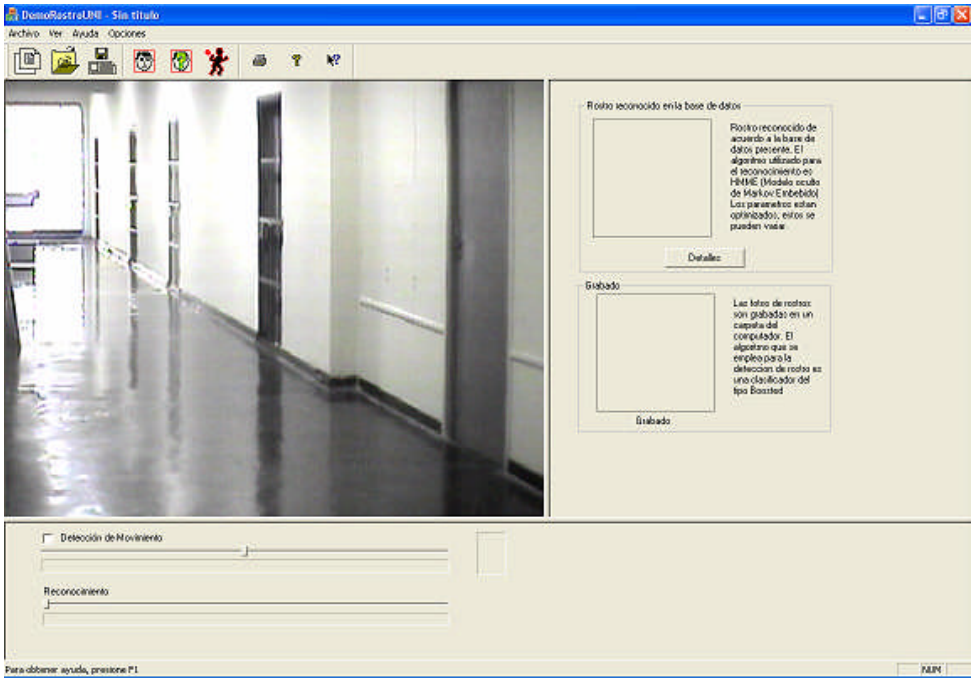


Fig. 8.37. *Puesta en funcionamiento*

Se incluye un modulo de grabación de video al sistema reconocedor de rostros, para ello primero se hace una comprobación de los bloques a usar con el GraphEdit. La figura 8.38 muestra la forma en que los bloques están dispuestos.

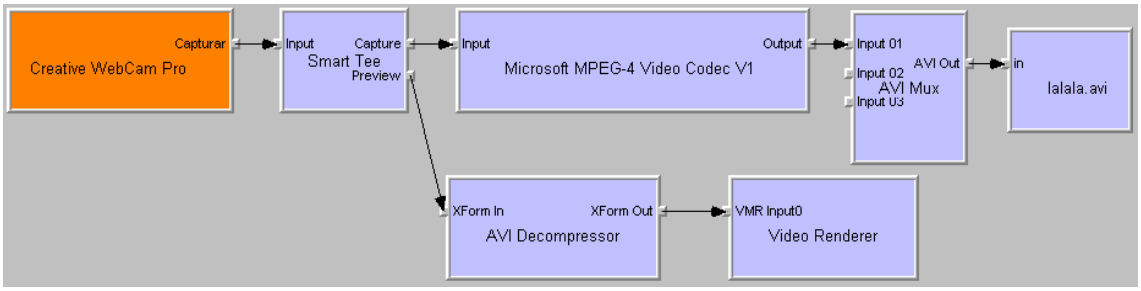


Fig. 8.38. *GraphEdit*

Este bloque se integra en el reconocedor de rostros, a manera que tambien este actué como un detector de movimiento, la idea es que al haber movimiento en la imagen, esta grabe por lo menos 10 segundos y si existe aun movimiento que siga grabando. Según las pruebas realizadas, un video de 10 segundos pesa entre 700Kb a 1.2Mb en formato mpeg-4.

8.3 Seguimiento de personas u objetos

Se ha realizado un algoritmo para la detección de movimiento en Matlab, utilizando la WebCam. La idea consiste en que si una imagen nueva aparece en la vision de la cámara, esta automáticamente procederá a mostrar la imagen en vivo.

Inicialmente nuestro programa haga un muestreo cada segundo el espacio que muestra la cámara, la imagen nueva será diferenciada con la imagen anterior, si esta diferencia sobrepasa un umbral definido (esto fue definido estadísticamente), el programa conmutará a un muestreo normal (1/30 segundos) y mostrara la imagen en una pantalla. Fig. 8.39.

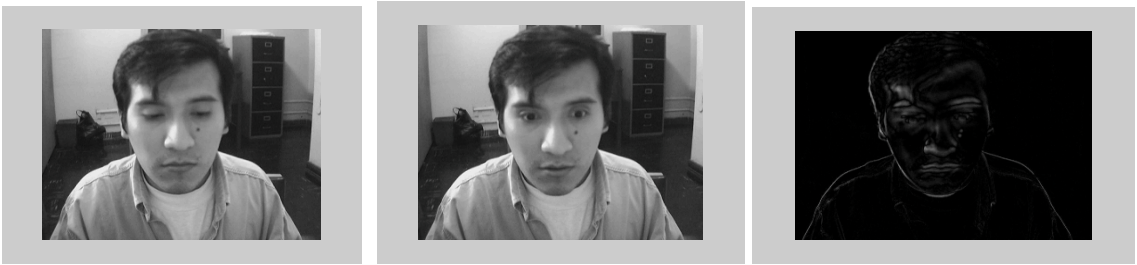


Fig. 8.39 *a) Imagen anterior b) Imagen Posterior c)Imagen Diferencia*

Para capturar un solo cuadro de imagen en vivo, la tarjeta capturadora debe tener suficiente memoria para almacenar temporalmente la imagen capturada, teniendo posibilidad que el usuario pueda ver la imagen capturada antes de que sea salvada en el

disco duro, usando el display adaptor que puede ser usado como display memory, si es digitalizado en tarjeta aparte la imagen temporal debe ser movida sobre el bus al display adaptor para poder ser visualizada. Para ser almacenada debe adicionársele la información de cabecera y el formato escogido. La frecuencia de muestreo para el ADC debe estar sincronizada con el video analógico de entrada, se escoge tal que un numero apropiado de píxeles horizontales sean generados durante el intervalo de exploración de video analógico según la ecuación siguiente: $f_S = (F_H R_H) C_H$, donde RH es la resolución deseada (píxeles), f_H es la frecuencia de exploración horizontal y C_H es la fracción de exploración activa, p.e. para una resolución de 640x480 y video NTSC la frecuencia de muestreo es:

$$f_S = (15{,}734 \times 640)/0.84 = 11.988 \text{ MHz}$$

Los píxeles verticales están limitados por el número de líneas activas en el video entrante. Para exploración entrelazada, uno o dos campos pueden ser capturados, si dos campos son capturados se consigue un cuadro completo, se puede tener errores cuando la fuente esta en movimiento.

Los adaptadores digitalizadores de video no son satisfactorios para la captura de video en movimiento, porque capturan en la memoria RAM, si es superior a algunos segundos es necesario almacenar los segmentos en el disco duro. El problema en los discos duros es la data rate porque no son lo suficiente rápidos para manejar las tasas de video sin comprimir. En el cuadro siguiente muestra valores de tamaño de ventana versus tasas de video sin comprimir.

Tabla 8.2 Velocidad según la resolución de la pantalla

Tamaño (en píxeles)	Cuadros/s	Data rate (MB/s)
640x480	30	27.6
320x240	30	6.9
320x240	15	3.5
160x120	15	0.9

La velocidad de muestreo de datos es de 10 MB/s y se manejan en sistemas de edición no lineal de buena calidad con PCs, siendo este ultimo necesario ya que la edición no lineal debe utilizar poca comprensión para preservar la calidad de imagen.

Una solución al almacenamiento de los datos es realizar una comprensión limitada en el momento de la captura, usando algoritmos simples, tienen la desventaja porque estos algoritmos pudiesen introducir mucha degradación en la calidad de la imagen, se tiene que buscar un compromiso entre lo que exista en el mercado. Lo mejor es capturar

en HD con la mayor calidad posible y luego ejecutar la comprensión en una etapa separada, la desventaja es que toma mucho tiempo.

8.3.1 Diseño del control a usarse.

El control de movimiento de la cámara se hará mediante un circuito electrónico que ira conectado al puerto serie del computador. El funcionamiento será el siguiente, mediante el programa del sistema se conectará el puerto serie. Cuando la cámara detecte un rostro o movimiento alguno de la escena, la cámara centrará el objeto y se moverá de acuerdo al movimiento del objeto (Horizontalmente) hasta un rango determinado (120º libre). Durante la detección el sistema sabrá si el objeto es un rostro o no, en caso afirmativo buscará en su base de datos para el reconocimiento y grabará el rostro, en caso de no reconocer grabará el video de dicha escena durante 10 segundos, si el movimiento continúa seguirá grabado sucesivamente.

El esquema del circuito implementado se divide en 3 partes, la primera parte se muestra en la Fig. (8.40) y es usado para el control de un motor paso a paso, en su diseño se implemento con una compuerta OR Exclusivo 74LS86 [40] y un flip-flop 7473 [41] para la parte lógica y unos transistores BD-135 para que actúen en corte y saturación para la parte de potencia. Este circuito tiene dos entradas, una de ellas es para el giro que tomara el motor (dirección) y la otra es la cantidad de pasos que dará dicho motor. La Fig. 8.41 muestra el diagrama de tiempos de la simulación del circuito.

La segunda parte es el detector de reinicio que hará volver al motor a su estado original. Esta implementado mediante un optoacoplador ps-2501-1 que detecta siempre una señal, si esta señal es obstruido por algun agente externo, el motor volvera a su posición original.

La tercera parte es la implementación del PIC quien se comunicará con una computadora mediante la interfaz max232 al puerto serie. La computadora será la que mande sobre el PIC mediante comandos.

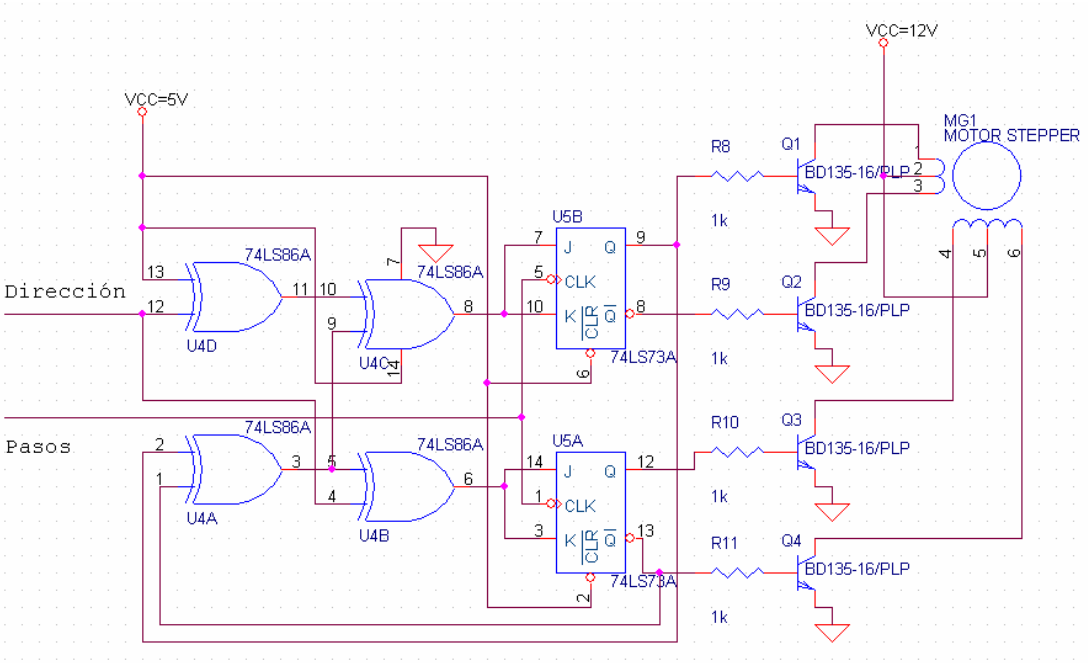


Fig. 8.40. *Control del motor paso paso*

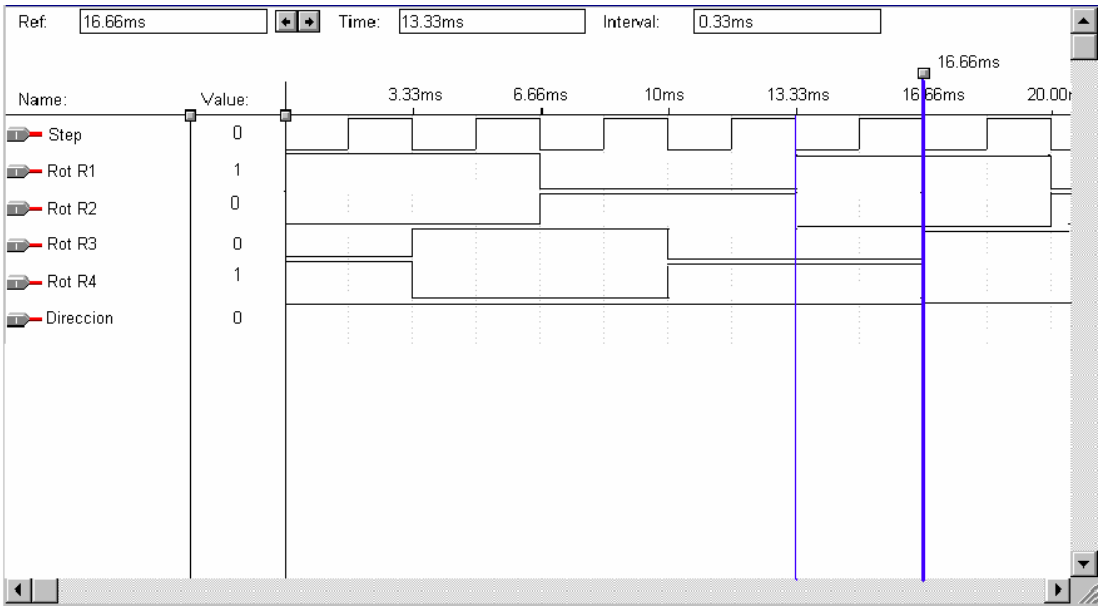


Fig. 8.41 *Diagrama de tiempos*

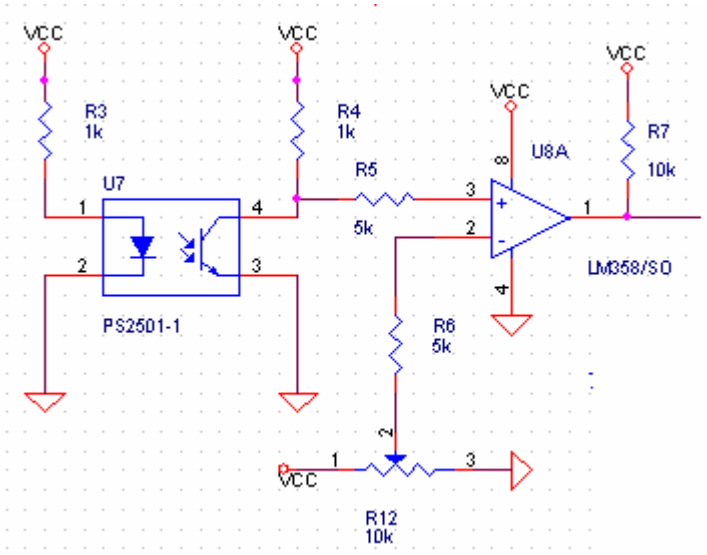


Fig. 8.42 *Detector de reinicio*

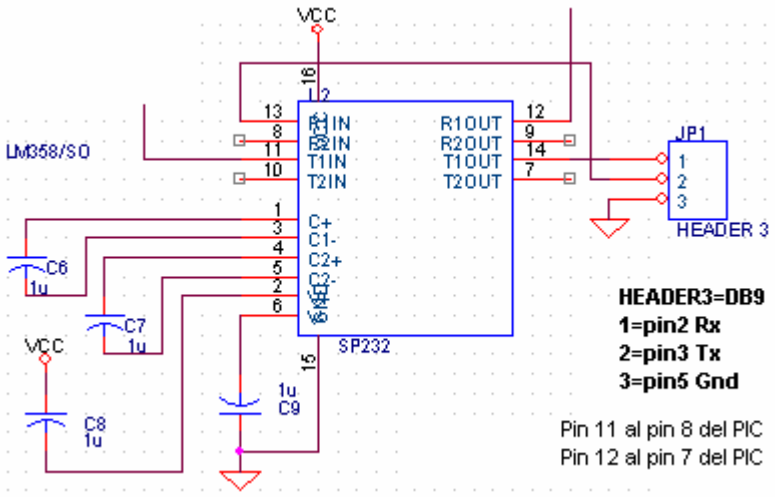


Fig. 8.43 *Interface PIC –RS232 de la PC.*

El microcontrolador usado será un PIC16f84A [42] , sus caracteristicas son lo siguiente:

Memoria de programa: 1Kx14 Flash

Memoria de Datos RAM: 68 bytes

Memoria de Datos EEPROM: 64 Bytes

Pila (Stack): 8 niveles

Interrupciones: 4 Tipos diferentes

Juego de Instrucciones: 35

Encapsulado: Plastico DIP de 18 patitas

Frecuencia de trabajo: 10Mhz máxima.

Temporizadores: Solo uno, TMR0, también tiene WDT.
Líneas de E/S Digitales: 13 (5 Puerta A y 8 Puerta B)
Corriente máxima absorbida: 80mA Puerta A y 150mA puerta B
Corriente máxima suministrada: 50 mA Puerta A y 100mA puerta B.
Corriente máxima absorvida por línea: 25mA
Corriente máxima suministrada por linea: 20mA
Voltaje de Alimentación (VDD): De 2 a 6V DC
Voltaje de Grabación (Vpp): 12 a 14V DC.

La elección de este PIC fue por la cantidad de memoria de programa (1Kbyte) y la cantidad de lineas de entrada/salida digital. Solo utilizaremos 5 líneas digitales. Dos para de Salidas RB1, RB2 (pasos y dirección), dos entradas RB3, RB4 (Rx y Tx al max232) y un entrada RB0 para el reinicio.

En la Fig. 8.44 se muestra el PIC16f84A implementado, El circuito que entra a la patita 4 del PIC es el circuito de reset y se activa cuando se aplica un nivel lógico bajo, de esta manera el PIC reinicializa su estado. La frecuencia de trabajo del microcontrolador será de 4 Mhz, o sea el periodo de 250ns, pero como cada instrucción tarda en ejecutarse 4 periodos, entonces un ciclo de instrucción dura 1ms.

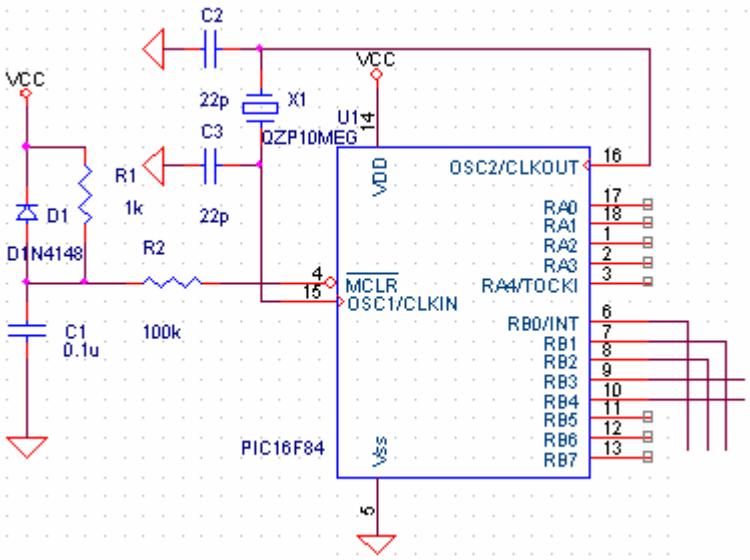


Fig. 8.44 *Implementación con el PIC16f84A*

En la Fig. 8.45 se muestra el esquema general del circuito de control, inicialmente se probó en un protoboard para luego implementar el circuito en una placa de fibra de vidrio. Para ello se procede a realizar el Layout del circuito con el programa ORCAD [43]. Tal como se muestra en la Fig. 8.46, la placa salio en doble cara.

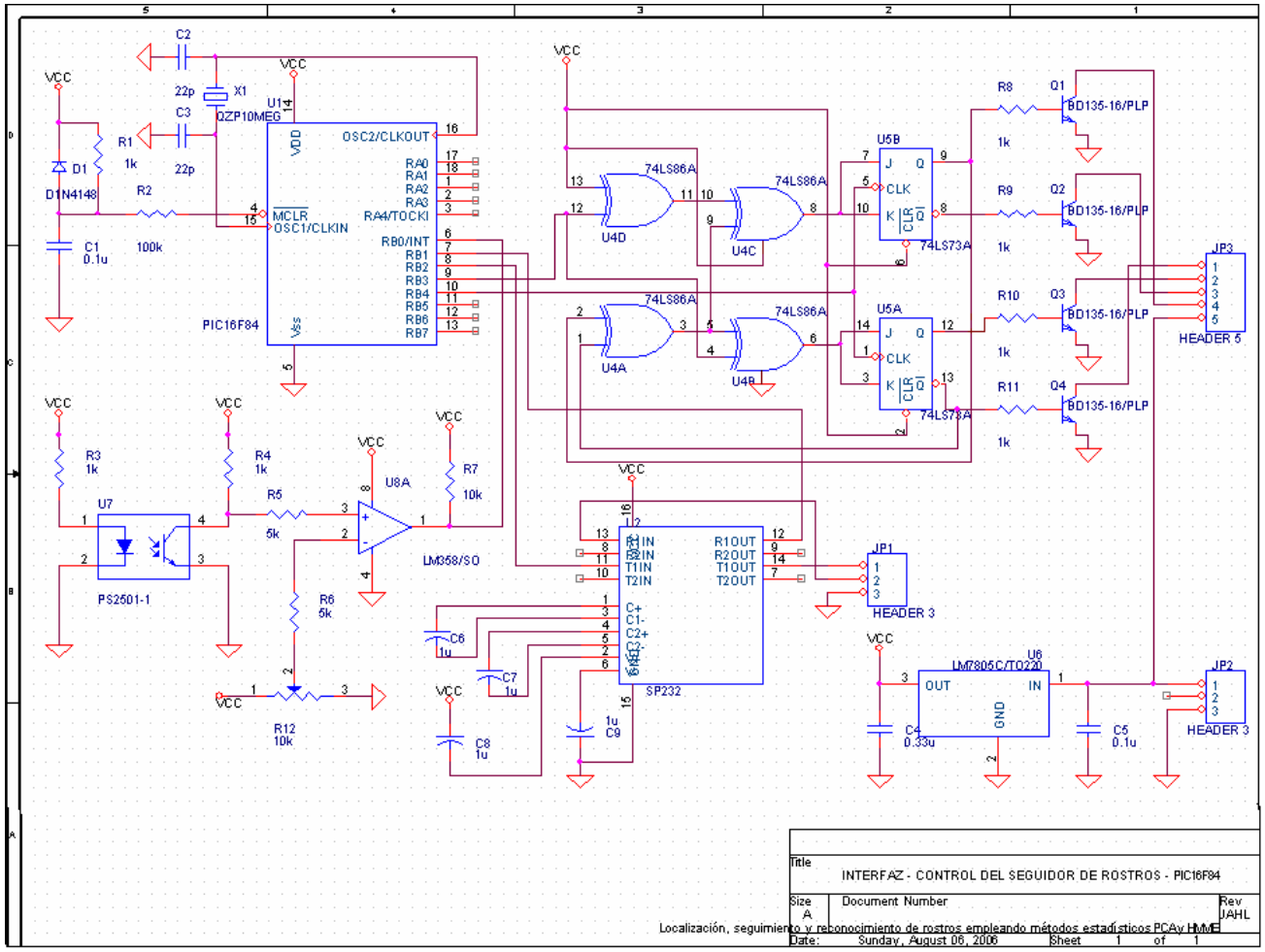


Fig. 8.45. *Esquema del circuito de control del motor paso a paso*

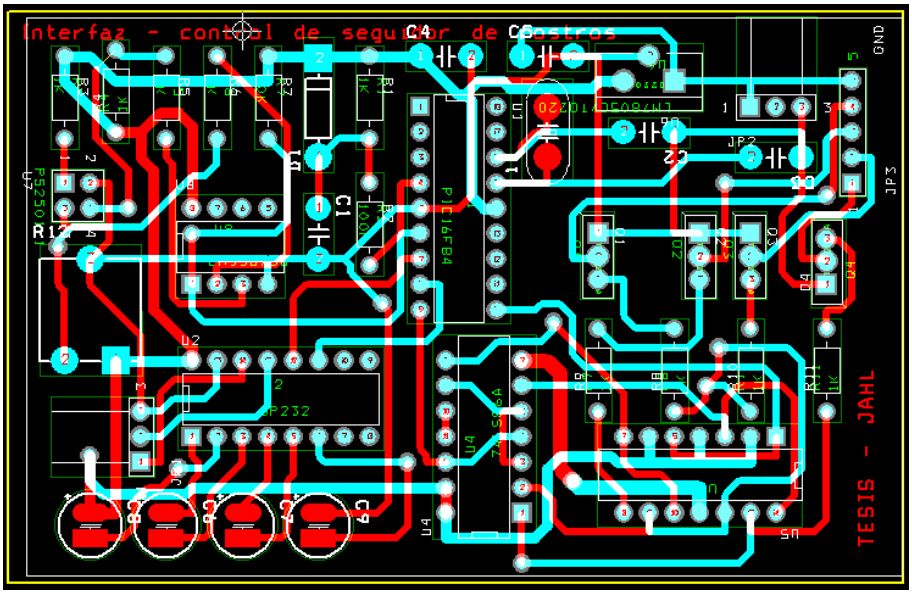


Figura 8.46. Layout del circuito de control del motor paso a paso.

8.3.2 Programación del Microcontrolador PIC 16F84A de Microchip

Una vez hecha la interface con la PC a través del puerto serial, se procede a programar el PIC, para ello se utiliza el entorno Delphi [44]. La programación se realizo con el lenguaje Pascal, luego se uso el compilador para crear el archivo asm.

```
//Programa para control de motor paso a paso para cuadrar posicion de camara
//El programa funciona de manera integrada con programa de deteccion de rostros
//autor:
//        Bach. Jose A. Huamán  email: jhuaman_cidfiee@uni.edu.pe
//        Centro de Investigación y Desarrollo FIEE UNI
//        Lima, Setiembre  2006

program TEST;
#INCLUDE   "C:\Archivos de programa\MPLAB\P16F84a.INC"
//RS232 settings
#pragma RS232_TXPORT PORTB
#pragma RS232_RXPORT PORTB
#pragma RS232_TXPIN  2
#pragma RS232_RXPIN  1
#pragma RS232_BAUD 9600
```

```
//Timing settings
#pragma CLOCK_FREQ 4000000
var actual,i,anterior: char;
 procedure InputText;
  begin
  putchar( 10 );
  putchar( 13 );
  putchar( 'I' );
  putchar( 'n' );
  putchar( 'i' );
  putchar( 'c' );
  putchar( 'i' );
  putchar( 'a' );
  putchar( 13 );

  end;
 procedure OutputText( var num: char );
  begin
  putchar( 'X' );
  putchar( '0' + num/100 );   //ascii 48+numero decimal
  putchar( '0' + (num mod 100)/10 );
  putchar( '0' + num mod 10 );
  end;
begin
//Inicio
    disable_interrupt( 'GIE' );
    set_bit( STATUS, 'RP0' );
    set_tris_a(255);
    set_tris_b(3);
    clear_bit( STATUS, 'RP0' );
    output_port_a( 0);
    output_port_b( 4);
                        //busqueda de posicion inicial
    for i:=1 to 12 do
        begin
```

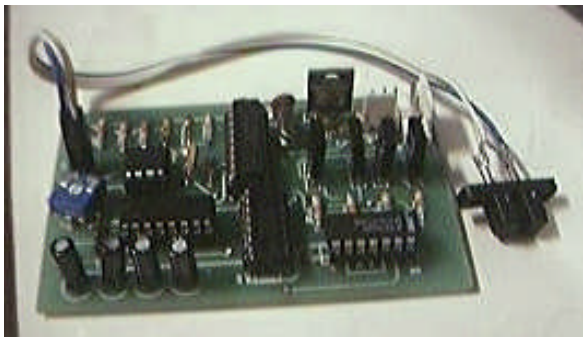
```
        if input_pin_port_b( 0 ) then
        begin
        delay_ms(1);
        end;
      else
        begin
        output_low_port_b( 3 );{derecha  0/izquierda 1}
        output_high_port_b(4);{pulso de rotacion}
        delay_ms( 100 );
        output_low_port_b(4);  //pulso de rotacion
        end;
      end;
   InputText;

//Ciclo de control de posicion de camara
   while 1 do
   begin
      //InputText;
      actual := getchar;
      actual :=actual- 48;
      //OutputText( actual );
      if  actual<>0 then
      begin
      anterior:=actual & 0x0F;
        for i:=1 to anterior  do
        begin
          if input_pin_port_b( 0 ) then
          OutputText(0);
          else
          begin
          if  actual<=13 then
            begin
            output_high_port_b( 3 );{derecha  0/izquierda 1}
            output_high_port_b(4);{pulso de rotación}
            delay_ms(100);
            output_low_port_b(4);  //pulso de rotacion
```

```
                end;
                else
                begin
                output_low_port_b( 3 );{derecha  0/izquierda 1}
                output_high_port_b(4);{pulso de rotacion}
                delay_ms( 100 );
                output_low_port_b(4);  //pulso de rotacion
                end;
             end;
          end;
          OutputText(actual); //comprobación de valor enviado
       end;
    end;
end.
```

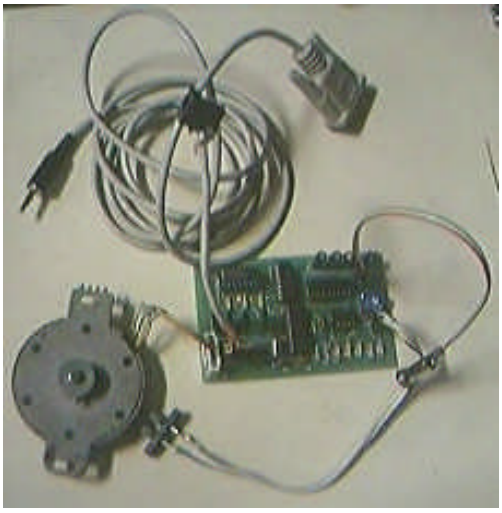
La compilación se realizo con pascal y se obtuvo el archivo .asm y .hex, el archivo .hex sirvió para la grabación del PIC mediante el programa EPICWIN [45].

8.3.3 Control del posicionamiento de la cámara.

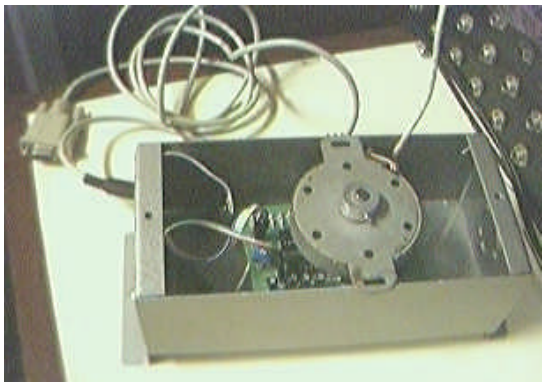
Una vez que el sistema se encuentre en marcha, la cámara deberá rotar al lugar donde hay movimiento. El control se realizará mediante software indicando cuantos grados debe girar el motor para ubicar al objeto en movimiento.



(a)



(b)



(c) **(d)**

Fig. 8.47 a, b, c,d *Partes del sistema de localización de movimiento*

Para poder dar ordenes al PIC mediante el computador se utilizo un control ActiveX en el software del programa, específicamente se uso el “Microsoft Communication Control” versión 6.0 [46]. El código para usar este ActiveX se encuentra en el anexo.

.

8.4 Implementación final

Se realizó el diseño de la implementación del sistema reconocedor de rostros en el Centro de Investigación y Desarrollo CIDFIEE y en la oficina del centro de Investigación de la facultad.

En el centro de investigación y desarrollo CIDFIEE se instaló el sistema tal muestra en la Fig. 8.49.

En la oficina de investigación y desarrollo se implemento el sistema de reconocimiento de rostros y se tuvo que tomar varias consideraciones y elegir los materiales necesarios para dicho implementación, para ellos se busco una cámara y una tarjeta de adquisición de video.

a) Cámara Domo CCD

Cámara color CCD avanzado de 1/4" en domo miniatura, de fácil instalación, su amplio giro nos permite colocarla tanto en pared como en techo además gracias su nuevo sistema de ajuste de ángulo nos permite mover la cámara sin necesidad de quitar la burbuja, integra una cámara de color con óptica de 3'6 mm 380 líneas de Tv. Con control de ganancia, auto balance de blancos, una relación señal ruido de mas de 48 db y una iluminación mínima de 3 lux. Dentro de una carcasa de plástico con burbuja oscura, alimentación 12v.

También fueron necesarias para la instalación los siguientes accesorios, como los cables para la cámara, al cable coaxial RG6 diseñado para cámaras, también se tubo que adquirir un cable monoral para la transmisión de audio, ya que la cámara adquirida tenia la opción de adquirir sonido.

Fig. 8.50. Cámara DOMO Adquirida

Fig. 8.51 *Accesorios necesarios para la instalación de la cámara*

De acuerdo la cámara, esta tiene una lente de f3.6mm/F2.0 y el ángulo que abarca la lente es de 70°.

También la Altura de una persona varía de 80cm 200cm. De acuerdo a esto la distancia horizontal desde la puerta hasta la cámara es aproximadamente 90 cm a 100 cm.

La altura de la cámara respecto al suelo es aproximadamente de 2.00 m. Se estima que la longitud total será de 10 a 15 metros. Aquí se muestra un bosquejo inicial de proceso de diseño:

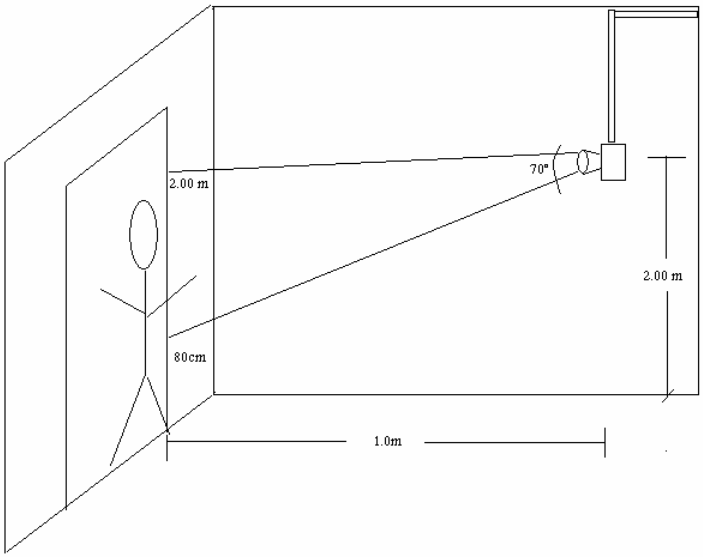


Fig. 8.52 *Bosquejo inicial para la colocación de la cámara*

b) Iluminación

El problema de iluminación en la instalación del sistema no afecto al reconocimiento de rostro, ya que en el ambiente existía un fluorescente que nos permitía observar al individuo con iluminación continua. Aquí se muestra un bosquejo de la instalación del sistema Fig. 8.53

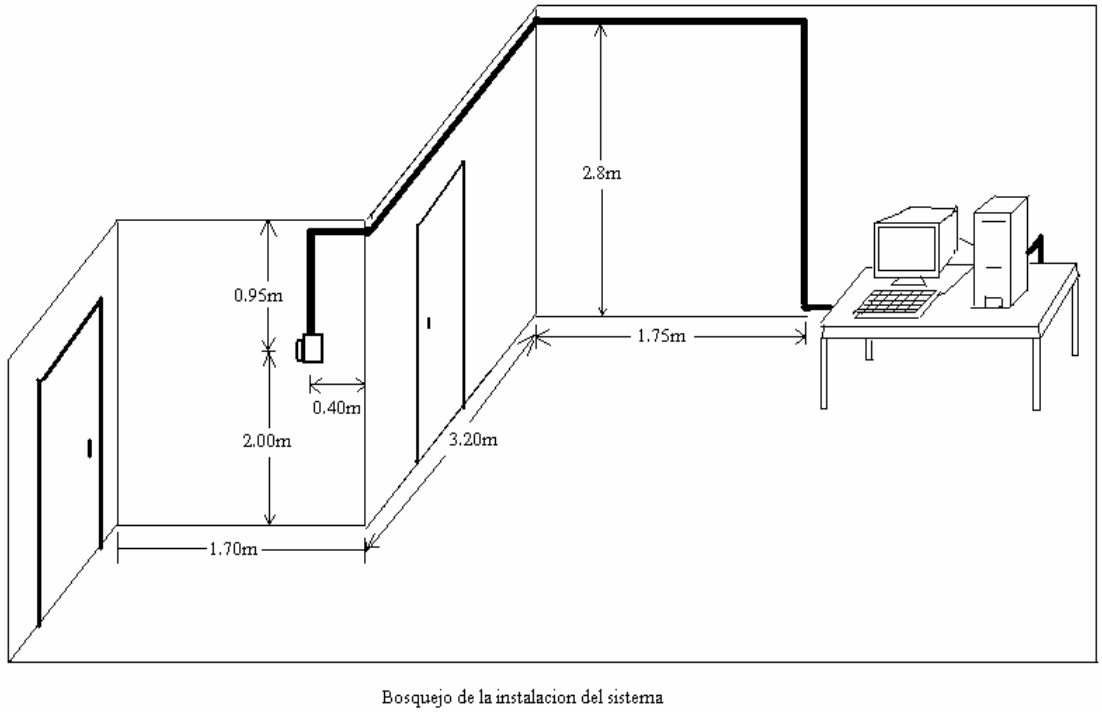


Fig. 8.53 *Bosquejo 2 de la instalación del sistema en una oficina*

C) Tarjeta de Adquisición de Video

La tarjeta de Adquisición adquirida fue una sintonizadora de video MpegTV. De Kworld Computer. Esta tarjeta tiene muchas utilidades como la adquisición de video de cualquier dispositivo (VHS, cámaras Digitales, etc..), y luego el paso a VCD o DVD.

Fig. 8.54. *Tarjeta de Adquisición de Video*

Fig. 8.55 *Accesorios de la tarjeta de Adquisición de Video*

Información Adicional:

MARCA: KWORLD Computer
MODELO: USB TV BOX
COLOR: Dorado con Gris
APLICACION: Para ver la TV en la PC
INTERFASE: USB Externo
CONECTORES
- RCA para salida de video compuesto
- Tipo minidin de S-Video
- Audio In
- Audio Out
- Coaxial para antena de TV
- RCA para antena de radio FM

REQUERIMIENTOS
- Puertos USB
- Pentium III 700 Mhz o mayor
- 128 MB en RAM
- 10 MB libres en disco duro
- Directx 8.0
- Tarjeta de sonido
- Bocinas
- Microfono

ACCESORIOS
- Manual de usuario ver: 8.10
- Cable de audio plug - plug
- Antena de TV
- CD de instalacion

8.5 Instalación del sistema en la Oficina del centro de investigación CID-FIEE

La implementación también requirió de cable coaxial RG-6, cable monoral y conectores de los mismos para video y audio respectivamente. Para que la cámara este segura, se realizo una preformación de pared, interior a la colocación de la cámara para proteger el cable. Y se instalo soportes para las cámaras, en caso de robo de la misma.

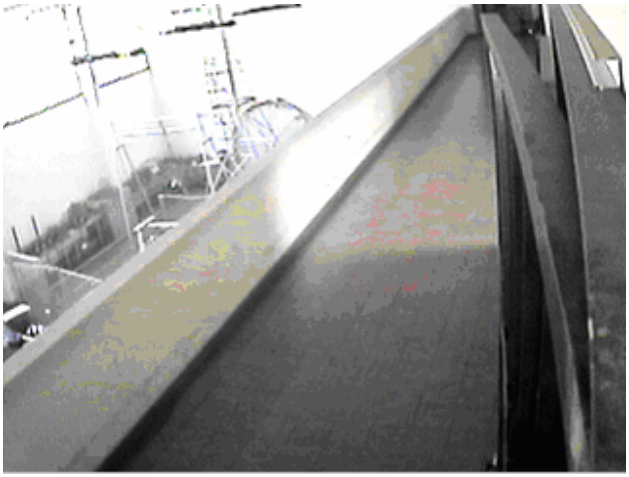


Fig. 8.56 *cámara vista con una lente de 3.6mm*

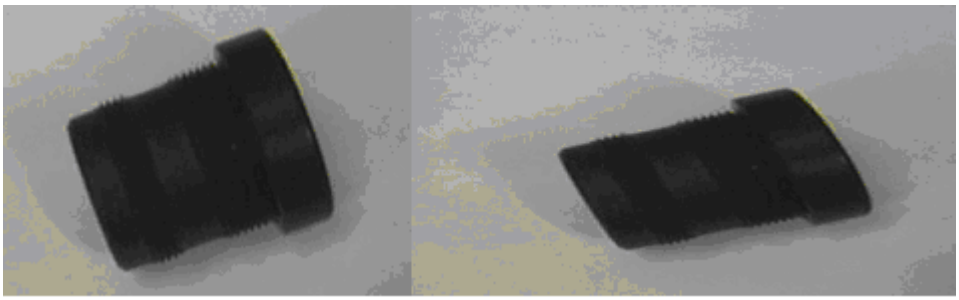


Figura 8.57 *Lentes de 3.6mm y 8 mm respectivamente*

La lente de 3.6 mm tiene una visión de mas de 70º, por lo que es utilizado mas para sistemas de vigilancia, ya que ellas requieren una visión completa del panorama general de la imagen.

La lente de 8 mm tiene una visión menor de 30º, y es utilizado para dar una mayor resolución de objetos cercanos, tal como el caso de un rostro.

Fig. 8.58 Cámara con los soportes para su protección.

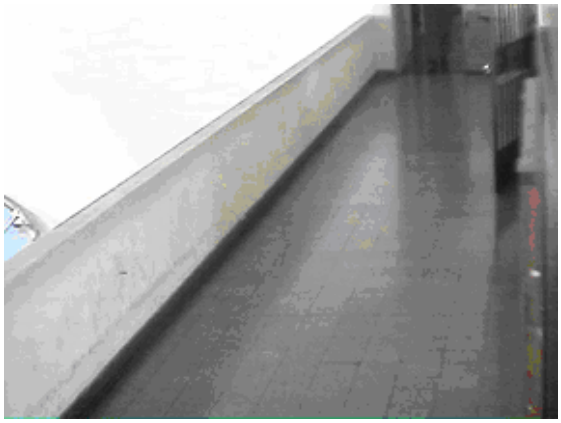


Fig. 8.59 *Instalación final de la Cámara y vista del área principal con la cámara con una lente de 8 mm.*

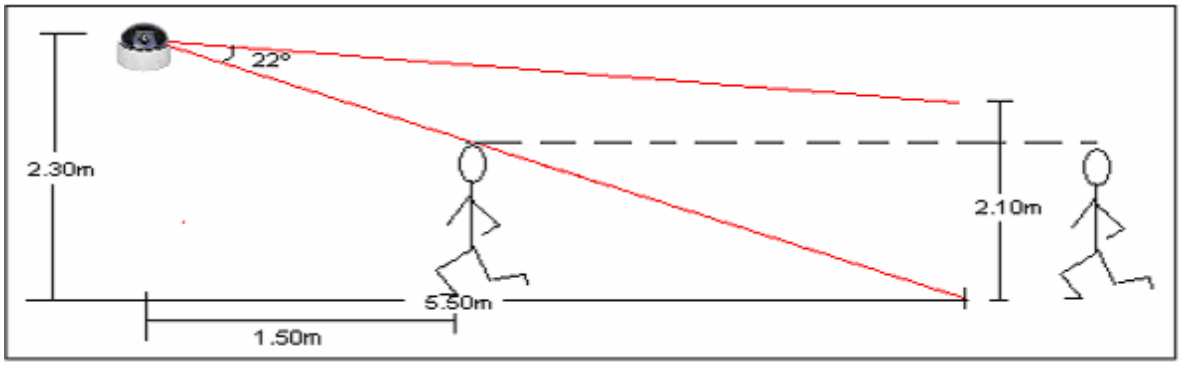


Fig. 8.60 *Bosquejo de la cámara con la lente de 8mm*

Fig. 8.61 *Cámara DOMO con sensor CCD instalada en el ambiente de imágenes de CID-FIEE a modo de prueba*

La instalación de la cámara se realizo en varias posiciones de tal manera de obtener el mejor desempeño.

Inicialmente estaba a una altura de dos metros tal como lo muestra la Figura 8.62, La ventaja que podíamos sacar de ello es la seguridad de la cámara, mas no así la eficiencia que se deseaba obtener, ya que el rostro no se notaba siempre. Por ejemplo si una persona pasaba sin mirar la cámara era una situación perdida, para ello solo grababa el instante en que pasaba la persona.

Luego de estudiar la mejor posición de la cámara, esta se podía situar a 1.7 metros de altura Figura 8.62. Que es la altura promedio de una persona, de esta manera se puede encontrar el rostro de la persona de manera más eficiente.

Fig. 8.62 instalación en otro ambiente del centro de investigación y desarrollo a modo de prueba.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Para el reconocimiento de rostros se utilizan dos etapas que son el entrenamiento y el reconocimiento de rostros propiamente dicho. En el entrenamiento la entrada debe ser un conjunto de imágenes representativas de uno o varios rostros según el método utilizado y la salida debe consistir en parámetros característicos que definan un rostro. Los métodos utilizados para el reconocimiento de rostros en este trabajo de tesis fueron: PCA, Fisher, HMM, HMME. Estos 4 métodos tienen un porcentaje promedio aceptable de reconocimiento del 90% de acuerdo a la base de datos utilizada en el presente trabajo.
2. Con el algoritmo de PCA la eficiencia de reconocimiento para 6 personas (1 imagen de rostro representativa por persona) de entre un conjunto de 60 rostros es de 95%, pero si aumentamos a 20 el número de personas y realizamos el reconocimiento de entre un conjunto de 200 rostros, la eficiencia disminuye a 85.5%. Esto se debe a que el algoritmo PCA trata a los rostros por separado, sin vincular o agrupar los rostros de una misma persona.
3. El Algoritmo de Fisher corrige la deficiencia de PCA de forma que agrupa al conjunto de rostros de una misma persona (Se debe tener la misma cantidad de imágenes por persona). La convergencia de los estimadores involucrados en este método es mucho más lenta que el método de PCA. Por tanto en la etapa de entrenamiento con el método Fisher el tiempo de entrenamiento es grande. Hay que notar que el tiempo que tarda un sistema de identificación genérico en dar un resultado es proporcional al número de imágenes de entrenamiento. Por tanto, si la base de datos es muy grande, estos métodos no serán eficientes para aplicaciones de tiempo real.
4. A diferencia del método de Fisher, el método HMM puede tener diferente cantidad de imágenes por persona y la eficiencia aumenta a medida que aumentamos más imágenes de rostros por persona a nuestra base de entrenamiento. Si se emplea 5 estados en vez de 6 estados la eficiencia no varia, la diferencia esta en los tiempos de ejecución de los mismos, es decir, que a medida que se aumenta más estados al sistema este tarda en

entrenarse y se obtiene resultados similares. Por lo que concluimos que 5 estados (frente, ojos, nariz, boca y mentón) son suficientes para el entrenamiento.

5. El desempeño que se obtuvo con el algoritmo HMME, muestra una eficiencia mínima de 95.98% donde se considero solo 3 rostros por persona haciéndola esta en el peor de los casos. El Método de HMME puede ser utilizado para una mayor cantidad de personas, además es robusto para imágenes con menor escala y no es necesario que tengan una resolución constante. Para el mejor entrenamiento de rostros, esta se debe hacer en el lugar donde se realizará el reconocimiento con ello se consigue una mayor efectividad.

6. Los sistemas reconocedores de rostros a nivel mundial buscan siempre el mejor algoritmo que tenga una buena eficiencia, en la rapidez y eficacia en reconocer rostros con menor grado de error. Para ello, luego de la experimentación respectiva con la base de datos (ORL FACE Database) Olivetti Research Lab. Cambridge, que es tomada como modelo de reconocedores de rostros a nivel mundial y con la base de datos propias de la Universidad Nacional de Ingeniería, se concluye que las eficiencias de los algoritmos no son los mismos en ambos casos debido a que las fotos son tomadas a diferentes condiciones de iluminación.

7. La mejor manera de encontrar y segmentar un rostro de una imagen es utilizando un clasificador de tipo Boosting, este tiene menos errores al detectar el rostro, además de ser mas rápido frente a otros métodos experimentados (método por color, por movimiento, transformada de Hough), debido a su bajo falso negativo que posee y a la velocidad. El clasificador utilizado es Adaboost específicamente “Boosted Cascade of simple features” que es el método que usa este clasificador. Este clasificador se puede usar en detección de objetos en general previo entrenamiento.

8. Las librerías OpenCV de Intel para imágenes, a diferencia con otras librerías experimentadas nos permite utilizar funciones a las PC’s que tengan poca memoria RAM y aprovecha aquellas que posean memoria suficiente para el proceso de entrenamiento y reconocimiento, estas librerías van de la mano con el procesador utilizado.

9. La iluminación juega un papel importante en el reconocimiento de rostros. Las pruebas realizadas en laboratorio con iluminación constante difieren cuando se implementa en un lugar determinado ya que se ven afectadas por la luz del ambiente no controlado. Por tanto para que el sistema trabaje correctamente la iluminación debe ser en lo posible constante.

10. Los sistemas biométricos no confirman la identidad real de una persona, únicamente reconocen si es la misma persona la que ha pasado previamente por el sistema. En el

caso del reconocimiento de rostros, si no se tienen fotografías externas de buena calidad de la persona al que se quiere identificar, de poco servirá.

11. DirectX nos proporciona estrategias, tecnologías y utilidades que ayudan a construir aplicaciones multimedia. Uno de los componentes de DirectX con el que se trabajo fue DirectShow que a diferencia de la tradicional programación con VFW (Video For Windows) es muy efectivo y potente. Estos componentes son independientes de cualquier aplicación y residirán en el sistema en forma de DLL's. Las aplicaciones con VFW tienden a fallar con capturadoras de video análogo, es por ello que se trabajo con DirectShow que es más robusto.

12. Acerca de los equipos necesarios para la implementación, se concluye que con una webcam de buena resolución (640x480 mínimo) CMOS es suficiente para pruebas cercanas al computador. Pero si se quiere hacer pruebas en campo es necesario tener una cámara analógica CCD de una mayor resolución (480 líneas horizontales mínimo) y una tarjeta de adquisición de video que soporte SDK DirectX.

13. Es necesario hacer un estudio del lugar donde se desee instalar el equipo ya que las condiciones de iluminación no es la misma en cada lugar, con ello se busca una cámara y/o equipos adecuados para tal situación.

RECOMENDACIONES

1. Fomentar la participación en área de procesamiento de imágenes, así este proyecto crea un precedente para el área mencionada.

2. Estar a la par con la tecnología, ya que a medida que aumenta el poder computacional, también se evalúan algoritmos pesados que tienes la posibilidad de tener éxito frente a algoritmos utilizados actualmente ya que siempre esta en pleno estudio.

3. El proyecto esta orientado al área de la biometría, utilizando pareas ello sensores específicamente cámaras de video y lo que se espera es poder incrementar la eficiencia de este reconocedor en futuros trabajos, aplicando algoritmos genéticos, otros tipos de redes neuronales, wavelets, etc.

4. La seguridad es otro gran problema en la actualidad, este proyecto de tesis trata de contribuir al desarrollo de sistemas de seguridad en la región, ya que vemos cada día más cámaras de video instaladas en todas las ciudades para usarlas en vigilancia. Estas cámaras carecen de software orientado al reconocimiento automático de personas.

5. El sistema puede ser empleado como reloj biométrico, es decir la entrada a determinada empresa o negocio puede ser agilizada solo mostrando el rostro, así de esta manera se evita el fraude que pueden ocasionar otros sistemas marcadores de relojes.

ANEXO A

MANUAL DEL PROGRAMA RECONOCEDOR DE ROSTROS (IMÁGENES FACIALES)

1. Diagrama del sistema

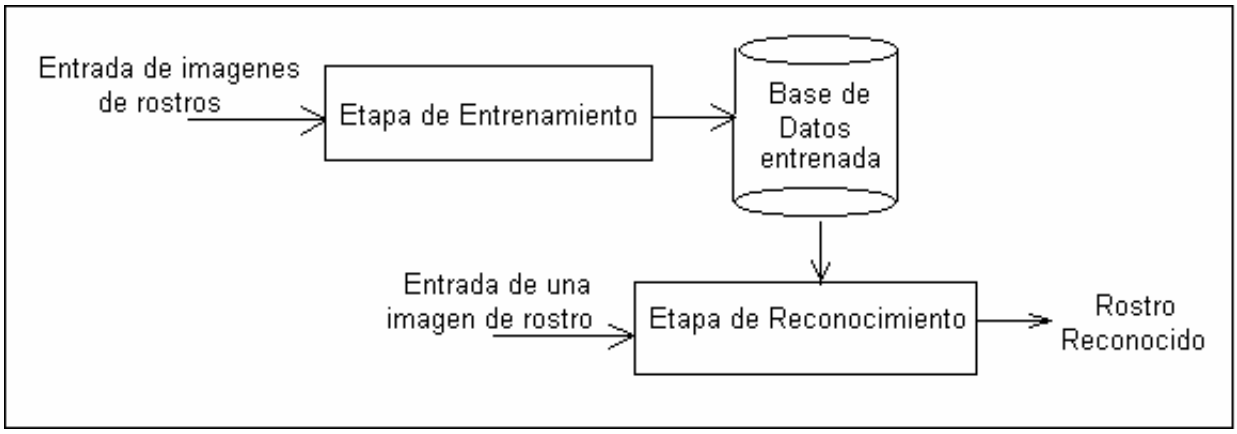


Fig. A1. *Diagrama del sistema reconocedor de rostros*

2. Algoritmo de Eigenfaces

2.1Etapa de Entrenamiento

La Etapa de entrenamiento se realizo en Matlab, la interfaz gráfica fue diseñada con las herramientas de Matlab, donde se considera a cada control como un evento, cada control tiene un nombre asignado y su proceso a realizar, por ejemplo tomando en consideración la figura 2, el control de abrir imagen / archivo llama al archivo de programa abrir_imagen.m, donde contiene los parámetros y funciones de Matlab para realizar dicho proceso.

2.1.1 Descripción de cada Archivo empleado

Entrenamiento.m Esta el Entorno general mostrado en la figura 1, donde esta contenido las llamadas a los siguientes archivos:

Fig. A.2 *Entorno general de la Etapa de Entrenamiento*

abrir_imagen.m

Se usa para abrir una imagen de rostro desde un archivo del disco al sistema.

Fig. A3 y A4: Abrir una imagen de rostro

anadira.m

Añade el rostro a la base de Datos, incluye sus datos personales, como nombre, apellido, código.

ARCHIVOS .m	Descripción de cada archivo
guardar_base.m	Este archivo de programa permite guardar la base de datos cruda, sin entrenar
abrir_base.m	Este archivo de programa permite abrir una base de datos cruda, sin entrenar. La idea de estos archivos es añadir mas rostros a estas bases.
Imágenes.m	Este archivo de programa permite ver las imágenes de rostros de la base de datos utilizada.
entrenarlo.m	Este Archivo de programa permite entrenar al sistema con el Método de Eigenfaces, tomando como parámetro el RMSE (Cota de Error Residual Normalizada)
pcasave.m	Luego de Entrenar al sistema, este archivo de programa permite guardar la base entrenada, donde solo se incluye a la matriz de proyección y las direcciones y datos de cada rostro, en un archivo .mat
Promedioimagen.m	Permite ver la imagen promedio generada por los rostros de entrenamiento.
eigenrostros.m	Permite ver los Eigenrostros de la base entrenada.
mostrar_eigenval.m	Permite ver los Eigenvalores de la base entrenada.
abrirbase_ent.m	Este Archivo de programa permite abrir la base entrenada por el sistema.

2.2 Etapa de Reconocimiento

2.2.1 Descripción de cada Archivo empleado

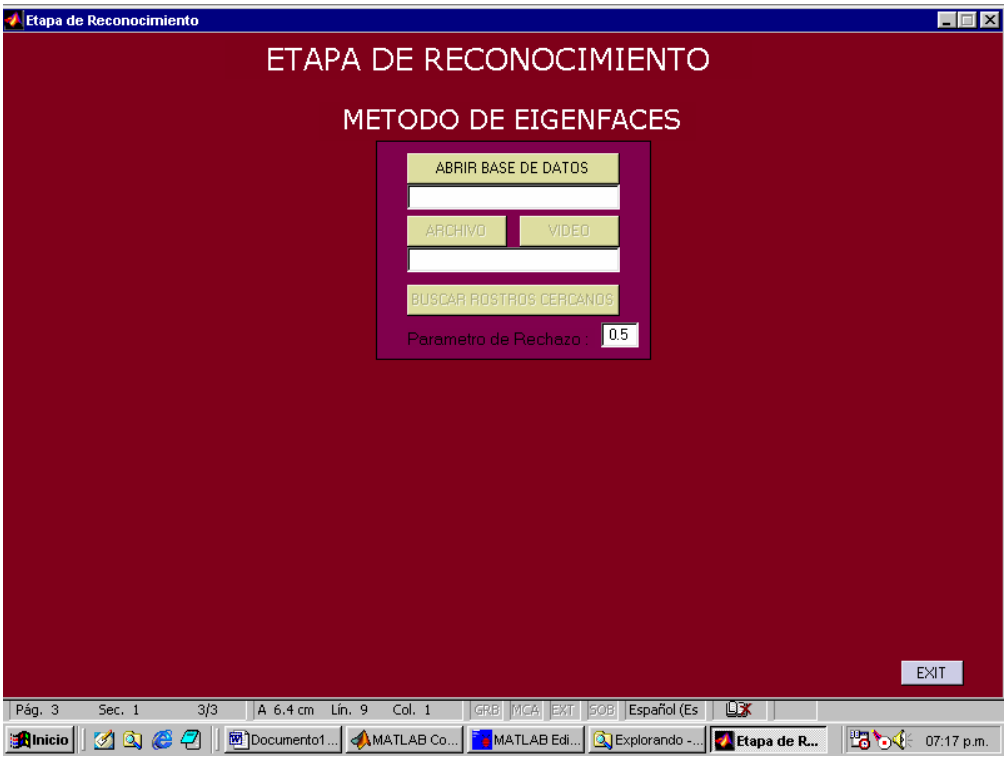


Fig. A6. Etapa de reconocimiento

reconocera.m

Este Archivo de Programa es el Entorno gráfico de la Etapa de Reconocimiento, donde esta contenido las llamadas a los siguientes archivos:

Tabla. A1. *Descripcion de cada archivo por el metodo Eifgenfaces*

ARCHIVOS .m	**Descripción de Cada Archivo**
abrirbase_ent.m	Este archivo permite abrir una base entrenada de la etapa de entrenamiento.
recoabrir.m	Este archivo de programa permite abrir una imagen de rostros desde archivo, para pasar a la etapa de reconocimiento

Adquisicion.m	Este archivo de programa es un entorno gráfico que permite capturar una imagen de video
adquirir.m	Este archivo de programa permite Adquirir una imagen de video desde una cámara, para luego poder hacerle un corte de la imagen a reconocer
rec_antes.m	Este archivo de programa permite pasar la imagen capturada a la etapa de reconocimiento.
buscara.m	Este archivo de programa busca la imagen de rostro dentro de la base de datos entrenada, si no lo encuentra el sistema nos muestra una imagen y un sonido de negación.

3. Algoritmo de Fisherface

3.1 Etapa de Entrenamiento

Para la etapa de entrenamiento del algoritmo de Fisher, se procede primero a indicarle al sistema la cantidad de personas a entrenar y el número de imágenes por persona. De esta manera se empieza la etapa de entrenamiento: fmetodo_fisher.m

3.1.1 Descripción de cada Archivo empleado

fmetodo_fisher.m.-

Este archivo de Programa permite declarar la cantidad de rostros (patrones) de las personas (clases) a entrenar.

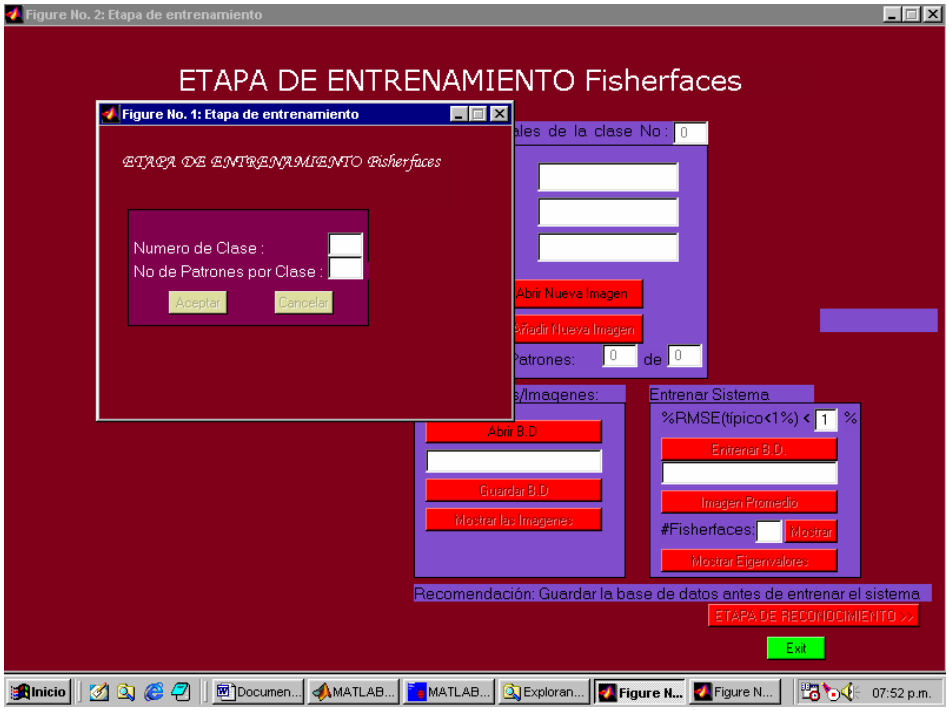


Fig. A7 *Etapa de entrenamiento Fisher*

Fentrenamiento.m

Esta el Entorno general mostrado en la figura, donde esta contenido las llamadas a los siguientes archivos:

Tabla A2. Descripción de los archivos con método Fisher – entrenamiento

ARCHIVOS .m	Descripción de cada archivo
f_abrir_imagen.m	Este Archivo de Programa permite abrir una imagen de rostro
f_anadir.m	Añade el rostro a la base de Datos, incluye sus datos personales, como nombre, apellido, código.
f_abrir_base.m	Permite abrir una base de datos sin entrenar, para seguir añadiendo mas datos
f_guardar_base.m	Permite guardar una base de datos sin entrenar, para su posterior utilización.
f_imagenes.m	Permite mostrar todas las imágenes de la base de datos
f_entrenarlo.m	Este archivo de programa permite entrenar al sistema, tomando como parámetro de entrenamiento en RMSE (Error residual normalizado)
f_promedioimagen.m	Permite mostrar la imagen promedio de todas las imágenes entrenadas
f_pcasave.m	Guarda la base entrenada del sistema en un archivo .mat

fisher_rostros.m	Permite mostrar los Fisher_rostros o fisherfaces de la etapa de entrenamiento (la matriz de proyección)
f_eigenval.m	Permite mostrar los eigenvalores de la etapa de entrenamiento.

3.2 Etapa de Reconocimiento

Fig. A8 interfaz metodo fisher

3.2.1 Descripción de cada archivo empleado

F_reconocera.m .-

Este archivo permite mostrar la interfaz gráfica del sistema (ver figura anterior), donde esta contenido las llamadas a las siguientes funciones:

Tabla A2. Descripción de los archivos con metodo Fisher – Reconocimiento

ARCHIVOS .m	Descripción de Cada Archivo
f_abrirbase_ent.m	Este archivo del programa permite abrir la base de datos entrenada por el sistema.
f_recoabrir.m	Este archivo de programa permite abrir una imagen.
f_buscara.m	Este archivo de programa permite buscar la imagen de entrada en la base de datos.

4. Ejemplos de Uso:

Para poder utilizar el programa primero se debe entrenar el sistema con una serie de fotos que puede ser de archivos o en tiempo real (video), Para ello se debe disponer lo siguiente:

4.1 Requerimiento mínimo:

Computadora Pentium III 1Ghz
Memoria 128Mb
Windows 98 o superior
Software Matlab version 5.3 o superior
Cámara web cam
Base de datos de rostros

4.2 Pasos para Utilizar el software:

Abrir el Matlab disponible en el sistema, luego configurar el path de Matlab, para que lea la carpeta de trabajo de PCA_versión_1.0 y la carpeta de trabajo Fisher_Versión_1.0, para ello en Matlab ir al menú File, elegir la opción SetPath.. , Luego en el, menú Path elegir add to path, y luego fijar la carpeta de trabajo.
Es preferible que las imágenes de entrenamiento se ubiquen en misma carpeta de los archivos fuente.

4.3 Ejemplo de Uso:

Podemos ir a entrenar una base de datos o ir a la etapa de reconocimiento, ya que se posee varias bases de datos entrenadas.

4.3.1 Entrenar la base de datos de imágenes de rostros

Para poder Entrenar el sistema lo primero que debe de hacer es abrir el Matlab, luego escribir en la línea de comandos: >> Entrenamiento, esto hará que se muestre una interfaz gráfica del método de PCA, esta interfaz nos permite entrenar al sistema. Para entrenar una base de datos, lo primero es ingresar las imágenes de rostro con sus respectivos datos, luego añadir de a uno a la base de datos.
Se puede guardar la base de datos de entrada presionando: “Guardar base de datos”, y se puede seguir añadiendo mas datos.

También existe la función de seguir añadiendo mas datos a una base de datos anterior, presionando: “abrir base de datos”. Las bases de datos se guardan con extensión .mat.

Luego de tener nuestra base de datos procedemos a entrenarlo, presionando: “Entrenar el sistema”, esto demorara dependiendo cuantas imágenes fueron entrenadas.

El sistema pedirá el nombre de la base de datos entrenada y de esta forma hemos entrenar una base de Datos, para pasar a la etapa de reconocimiento se puede presionar: “Etapa de Reconocimiento”, donde rápidamente iremos a la siguiente etapa.

4.3.2 Reconocer una imagen de rostro

Para poder reconocer una imagen de rostro, abrir Matlab y escribir en la línea de comandos: >>reconocera, luego se mostrará una interfaz grafica del sistema. Lo primero será ahora cargar al sistema con una base de datos entrenada (extesión .mat), para ello presionamos “Cargar base de datos”. Luego de ello procedemos a cargar una imagen, ya sea de un archivo “Abrir Archivo” o de una imagen de video “Imagen de video”. Para la imagen de video, la persona debe estar frente a una cámara, ya sea WebCam u otro, y cuando presione adquirir imagen, se tendrá que cortar el rostro que se quiere reconocer.

Luego de cargar la imagen se procederá a reconocer el rostro, presionando “Buscar rostro cercano”.

5. BASES DE DATOS

Las bases de datos de rostros fueron adquiridas por medio de Internet y también se realizo una base de datos de los estudiantes de la FIEE.

5.1 Bases de datos de imágenes desde Internet

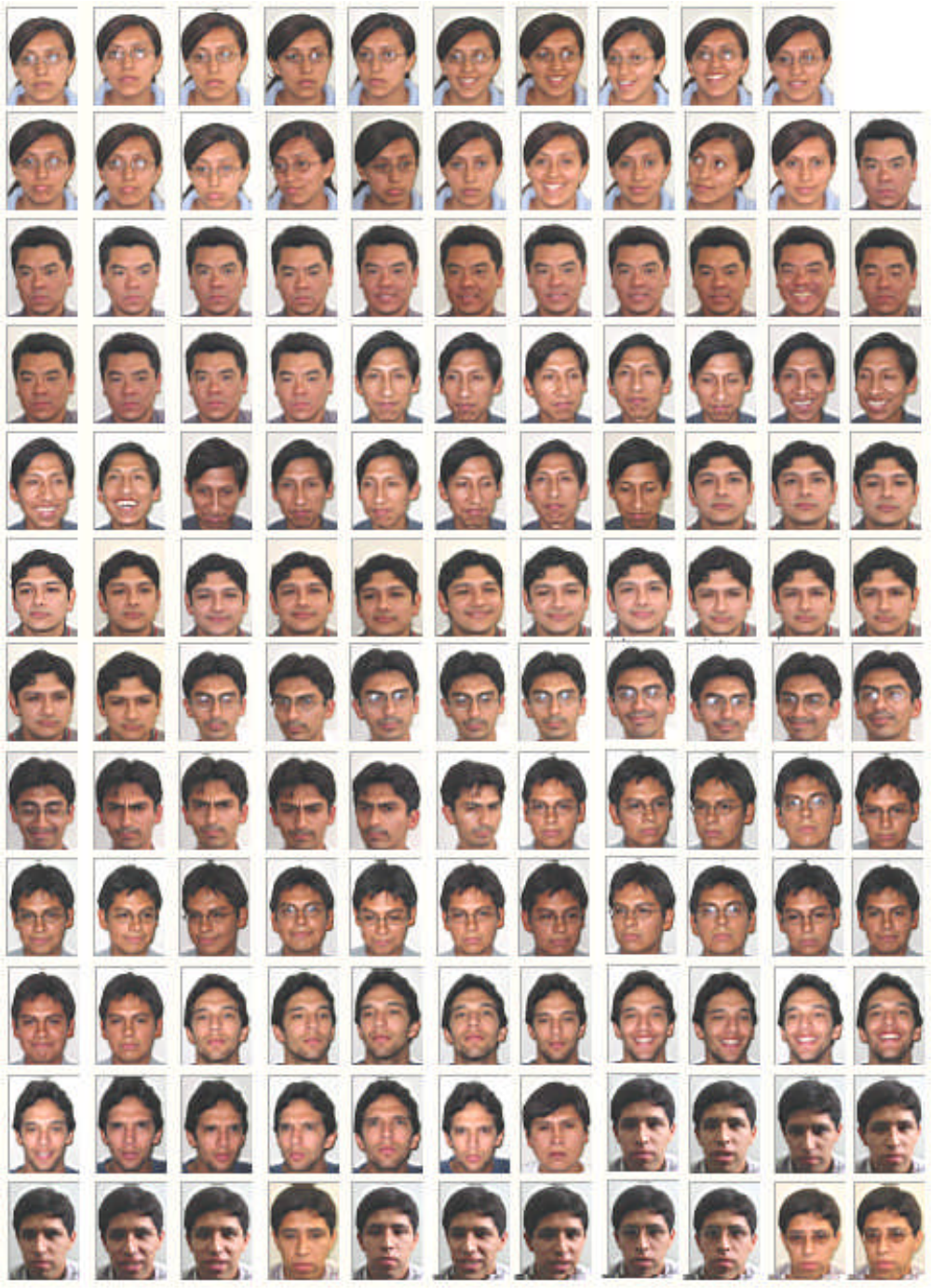
Estas bases fueron tomadas de la página de la universidad de Cambridge, face Database, estas fotos son tomadas como referencia para validar reconocedores de rostros, de diferentes métodos. Estas fotos están en formato PNG, con una resolución de 112x96, con una profundidad de BIT de 8 bits (Escala de grises). Esta Base de datos posee 400 Imágenes de rostro.

A continuación las fotos de esta base en las siguientes figuras.

Figura A9.*Base de datos de imágenes de la Universidad de Cambridge*

También nosotros poseemos nuestra base de datos propios de imágenes de rostros. Fueron tomados en un ambiente de investigación, las fotos tomadas fueron con una

cámara powershot A4,(digital), con una resolución de 300x200, con una profundidad de bit de 24 bits (imágenes a color), y en formato Jpeg. Nuestra base de datos posee 200 imágenes. A continuación las fotos de nuestra base en la figura A10.



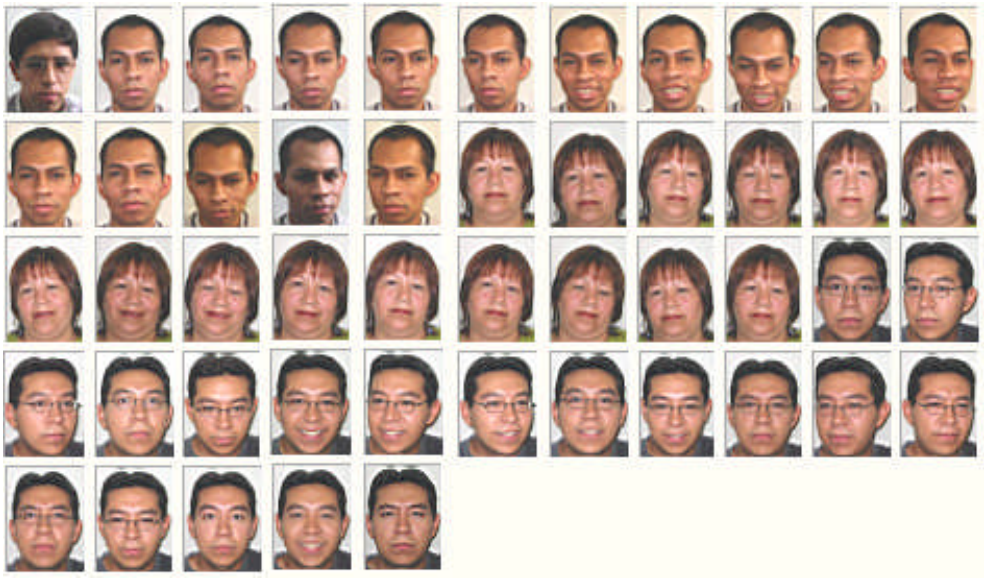


Fig. A10. *Base de datos de imágenes de la Universidad Nacional de Ingeniería*

BIBLIOGRAFÍA

[1] Torres Gemelas Atentado 11 de Setiembre 2001 http://www.september11.com.ar/

[2] Identix FaceId “Frequently Asked Technical Questions 2005”. pp 8.

[3] Rafael C. Gonzáles, Richard E. Woods “Tratamiento digital de imágenes” pp. 11-12 Editorial Addison-Wesley Iberoamericana S.A. 1996

[4] Gonzalo Pajares, Jesús M. de la Cruz. “Visión por Computador”. Editorial ALFAOMEGA grupo Editor S.A. 2002.

[5] Anil k. Jain “Fundamentals of digital image processing” Editorial Prentice-hall. 1986.

[6] Davis A. Forsyth, Jean Ponce “Computer Vision” Editorial Prentice Hall, Person Education, Inc. 2003

[7] Marino Tapiador, Juan A. Sigüenza. “tecnologías biométricas aplicadas a la seguridad” Editorial RA-MA 2005.

[8] Sharp Worl 2006
www.sharp-world.com/products/device/catalog/pdf/ccd/**ccd2003_e**.pdf

[9] Sego-Seguridad Optima “Productos Cámaras” http://www.sego.com.pe/

[10] OmniVision technologies Inc. Data Sheet rev. 1.2 Advance Camera to USB Bridge http://www.ovt.com

[11] Point Grey Research “Imaging Products”
http://www.ptgrey.com/products/imaging.asp

[10] Universidad de Yale “The Yale Face Database”
http://cvc.yale.edu/projects/yalefaces/yalefaces.html

[11] National Institute of Standards and Technology “The Facial Recognition Technology Database”
http://www.itl.nist.gov/iad/humanid/colorferet/home.html

[12] Face Recognition HomePage Databases. http://www.face-rec.org/databases/

[13] Paul Viola, Michael Jones “Robust Real-time Object Detection” International Journal of Computer Vision, 2001.

[14] Paul Viola, Michael Jones “Rapid Object Detection using a Boosted Cascade of Simple Features” International Journal of Computer Vision, 2001.

[15] Rowley, Baluja and Kanade “Neural Network-Based Face Detection. PAMI, Copyright 1998 IEEE.

[16] C. Papageorgiou, M. Oren and T. Poggio. “A general framework for detection, International Conference on Computer Vision, Bombay, India, pp. 555-518. 2001.

[17] Sirovich L., and Kirby M. A low-dimensional procedure for the characterization of human faces, *J. Opt. Soc. Amer. A*, vol. 4, no. 3, pp. 519-524. (1987)

[18] Davis A. Forsyth, Jean Ponce “Computer Vision” Editorial Prentice Hall, Person Education, Inc. pag 507-515 – 2003

[19] Matthew A. Turk, Alex P. Pentland “Face Recognition Using Eigenfaces” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3-6 June 1991, Maui, Hawaii, USA, pp. 586-591.

[20] Swets D.L., and Weng J.J. “Using Discriminant Eigenfeatures for Image Retrieval”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, pp. 831-836. August 1996.

[21] W. Zhao, R.Chellappa, A. Krishnaswamy. “Discriminant Analysis of Principal Components for face Recognition” Center for Automation Research University of Maryland. 1998.

[22] Peter N. Belhumeur, Joao P. Hespanha, David J. Kriegman. “Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection” IEEE trnsactions on pattern analysis and machine intelligence, vol.19,no.7, July 1997.

[23] Alex Pentland, Baback Moghaddam, Thad Starner. “View-Based and Modular Eigenspaces for Face Recognition” Vision and Modeling Group, The Media Laboratory. Massachusetts Institute of technology. 1994.

[24] W. Zhao, A. Krishnaswamy, R. Chellappa, D.L. Swets, J. Weng, Discriminant Analysis of Principal Components for Face Recognition, Face Recognition: From Theory to Applications, H. Wechsler, P.J. Phillips, V. Bruce, F.F. Soulie, and T.S. Huang, eds., Springer-Verlag, Berlin, 1998, pp. 73-85

[25] A.M. Martinez, A.C. Kak, PCA versus LDA, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 23, No. 2, 2001, pp. 228-233

[26] A.T. Bharucha-Reid “Elements of the theory of Markov Processes and Their Applications” Editorial McGraw Hill 1960.

[27] Rakesh Dugal, U.B. Desai. A Tutorial on hidden markov models. Indian Institute of Technology – Bombay Powai.

[28] L.R. Rabiner, “A Tutorial on Hidden Markov Models and selected Applications in Speech Recognition. L.R. Rabiner, Proceedings of the IEEE, Vol 77, Nº 2 Febrero 1989: pp. 257-286.

[29] E. Levin and R. Pieraccini, “Dynamic planar warping for optical character recognition, “ en la conferencia internacinoal de acustica, habla y procesamiento de señales. pp 149-152.

[30] S. Kuo, O.Agazzi, “Keyword spotting in poorly printed documents using pseudos 2-D Hidden Harkov models,” IEEE Trnaslations on Pattern Analisis and Machina Intelligence. Vol 16. pp 842 848.

[31] Ara V. Nefian, Monson H. Hayes,”Face Detection and Recognition Using Hidden Markov Models”

[32] Ara Nefian, “A Hidden Markov Model-Based Approach for Face Detection and Recognition” IEEE Conference on Audio and Video-based Biometric Person Authentication.

[33] Ara V. Nefian, Monson H. Hayes, An Embedded HMM-Based Approach for Face Detection and Recognition

[31] The Matworks Matlab “VFM video for Matlab” http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=247

[40] Texas Instruments Incorporated. Dual J-K Flip-Flops with clear, SDLS118-Diciembre 1983 – revisado en marzo de 1988.

[41] SN74LS86 Quad 2-Input Exclusive OR Gate Semiconductor Components Industries, LLC, 2001. October, 2001 – Rev. 7. 1.

[42] José M. Angulo U., Ignacio Angulo M. Microcontroladores PIC – Diseño práctico de Aplicaciones McGraw Hill.

[43] Orcad PCB Design Tools. Orcad Layout. http://www.cadence.com/orcad/

[44] Borland IDE: Delphi – Windows.NET Application Development Tool http://www.borland.com/delphi

[45] MicroEngineering Labs, Inc. EPIC for Windows 95/98/ME/NT/2000/XP version2.43 beta http://www.melabs.com

[46] Microsoft Corporation 2007. Microsoft Visual Studio C++. Net 2003. hht://www.microsoft.com/en/us/default.aspx