Universitat
Oberta de Catalunya

Proyecto Final de Carrera.

Reconocimiento de puntos de interés en rostros.

Alumno: Santi Valverde Montseny.

Consultor: David Masip Rodo.

Titulación: Enginyeria Informàtica. Junio 2011.

Resumen.

Este trabajo pertenece al área de Visió per Computador y nos introduce en el mundo del reconocimiento de patrones mediante ordenador. Actualmente la visión por computador se utiliza para reconocer la matrícula de nuestro coche al entrar a un parking, nuestra huella dactilar para abrir una puerta, reconocimiento de imágenes topográficas, o incluso reconocer la información biométrica de nuestra cara para ser utilizada como contraseña.

Este trabajo centra su estudio en el reconocimiento de puntos de interés de un rostro humano. En concreto vamos a intentar reconocer 20 puntos de interés. Para ello será necesario realizar dos acciones:
La primera, recurriendo al concepto de clasificador de la inteligencia artificial, llevaremos a cabo un aprendizaje sobre un conjunto de imágenes. Con la información obtenida crearemos un clasificador para cada punto de interés.
La segunda acción será, dada una imagen y con la información de cada clasificador obtenida, buscar y situar cada punto de interés sobre un rostro humano.

Para poder realizar esta aplicación utilizaremos una librería libre llamada OpenCV y el código será escrito en C, C++.

Índice.

Proyecto Final de Carrera. 1
Reconocimiento de puntos de interés en rostros. 1
Resumen. 2
Índice. 3
Índice Figuras. 4
1 Introducción. 5
1.1 Justificación. 5
1.2 Objetivos. 5
1.3 Método a seguir. 5
1.4 Planificación. 6
1.5 Productos obtenidos. 6
1.6 Entorno de desarrollo. 6
2 Módulo de entrenamiento. 7
2.1 Qué es un Fern. 7
2.2 Ventana de entrenamiento. 7
2.3 Entrenador. 8
3 Módulo de prueba de aciertos. 9
3.1 Ventana de desplazamiento. 9
3.2 Máxima probabilidad. 10
3.3 Evitar valores cero. 11
3.4 Detector de objetos. 12
3.5 Test de aciertos. 13
4 Interface gráfico. 16
4.1 Descripción. 16
4.2 Funcionamiento. 17
5 Imágenes de prueba. 19
5.1 Formato. 19
6 Algoritmo entrenamiento. 21
6.1 Random. 21
6.2 Regular. 22
6.3 Random Versus Regular. 22
7 Resultados. 24
7.1 Ensayo 1. 24
7.2 Ensayo 2. 25
7.3 Ensayo 3. 26
7.4 Ensayo 4. 27
7.4 Pruebas de rendimiento. 28
8 Conclusiones. 29
9 Glosario. 30
10 Bibliografía. 31
11 Anexos 32
11.1 Estructura directorios. 32
11.2 Instalación aplicación. 32
11.3 Configuración. 34

Índice Figuras.

Ilustración 1. Diagrama de planificación. 6
Ilustración 2 . Proceso entrenamiento. 7
Ilustración 3. Ejemplo Clasificador 8
Ilustración 4. Pseudocódigo entrenador. 8
Ilustración 5. Rastreo para localizar los ojos. 9
Ilustración 6. Fórmula probabilidad. 10
Ilustración 7. Conversión multiplicación en suma de logaritmos. 10
Ilustración 8. Fórmula para evitar los valores cero. 11
Ilustración 9. Detección rostro. 12
Ilustración 10. Función detección cara. 13
Ilustración 11. Pseudocódigo test de aciertos. 14
Ilustración 13. Código función desplazamiento. 15
Ilustración 12. Estructura donde almacenamos el resultado del test para cada punto. 15
Ilustración 14. Interface gráfico. 16
Ilustración 15. Resultado con la opción ver caras activada. 17
Ilustración 16. Resultado con algún punto fuera de tolerancia. 18
Ilustración 17. Fichero resultados.txt 18
Ilustración 18. Detalle fichero punto de interés. 19
Ilustración 19. Orden de los puntos de interés. 20
Ilustración 20. Diagrama entreno Random. 21
Ilustración 21. Código algoritmo Random. 21
Ilustración 22. Estrategia algoritmo Regular. 22
Ilustración 23. Tabla comparativa Ferns y algoritmos. 22
Ilustración 24. Ensayo 1. 24
Ilustración 25. Ensayo 2. 25
Ilustración 26. Ensayo 3. 26
Ilustración 27. Comparación tolerancia. 27
Ilustración 28. Prueba rendimiento. 28
Ilustración 29. Directorios datos. 32
Ilustración 30. Ejecución CMake. 33
Ilustración 31. Esquema solución. 34
Ilustración 32. Configuración librerías. 34
Ilustración 33. Directorios VC++. 35
Ilustración 34. Dependencias adicionales. 35

1 Introducción.

En este apartado voy a detallar los motivos por los que he elegido este tema así como de que manera se ha estructurado la realización de este trabajo.

1.1 Justificación.

La elección del trabajo final de carrera siempre es una decisión difícil. Por un lado se busca un tema de actualidad y que pueda ser nos útil en nuestra vida profesional, por otro lado se busca un tema nuevo, desconocido y que nos motive en el aprendizaje de alguna disciplina nueva.
Yo he conseguido aglutinar estos dos motivos, ya que por un lado nunca había realizado ninguna actividad en visión por computador y por otro, en mi vida profesional, vamos a introducir la visión por computador en el reconocimiento de patrones de imágenes de televisión.

Una vez elegimos el área del trabajo de fin de carrera, no tenemos todavía muy claro en que consistirá nuestro objetivo. En mi caso, ha sido una sorpresa que el trabajo propuesto por el consultor, no solamente consistía en obtener unos productos de software, sino que a partir de una idea vamos a descubrir si es posible obtener unos resultados aceptables en el reconocimiento de rostros. Esta pequeña dosis de investigación e incertidumbre a provocado una gran satisfacción en la realización de este trabajo.

1.2 Objetivos.

El principal objetivo es disponer de una aplicación que nos permita, por un lado, realizar un clasificador para cada punto de interés a partir de una colección de imágenes y por otro lado, otra aplicación que dada una imagen sea capaz de rastrearla píxel a píxel y situar los puntos de interés. También obtendremos el tanto por ciento de aciertos para cada punto de interés.
Finalmente uniremos estas dos aplicaciones en una interficie gráfica para hacer su uso más ágil y agradable.

1.3 Método a seguir.

A partir de la información inicial proporcionada por el consultor se ha concretado el objetivo e idea del trabajo. Como que necesitamos librerías especializadas y entorno de desarrollo adecuado lo primero será instalar tanto las librerías como el entorno de desarrollo para poder familiarizarnos.
El siguiente paso será la realización de la parte de entrenamiento, en la cual obtendremos los clasificadores de los puntos de interés.
Seguidamente crearemos la parte de rastreo de una imagen y reconocimiento de los puntos de interés.
Finalmente integraremos las dos aplicaciones en una interficie gráfica.

1.4 Planificación.

El siguiente diagrama de Gantt nos muestra la planificación prevista.

Id		Nombre de tarea	Duración	Comienzo	Fin
1					
2					
3	✓	**Definición Proyecto**	**7 días**	**mié 02/03/11**	**jue 10/03/11**
4	✓	Estudio documentación inicial	4 días	mié 02/03/11	lun 07/03/11
5	✓	Acordar líneas generales	3 días	mar 08/03/11	jue 10/03/11
6	✓	Plan de proyecto definido	0 días	jue 10/03/11	jue 10/03/11
7	✓	**Entorno desarrollo**	**7 días**	**vie 11/03/11**	**lun 21/03/11**
8	✓	Instalación software	4 días	vie 11/03/11	mié 16/03/11
9	✓	Pruebas software	1 día	jue 17/03/11	jue 17/03/11
10	✓	Análisis librerías OpenCV	2 días	vie 18/03/11	lun 21/03/11
11	✓	Entorno desarrollo listo y probado.	0 días	lun 21/03/11	lun 21/03/11
12		**Módulo entrenamiento**	**12 días**	**mar 22/03/11**	**mié 06/04/11**
13		Especificación	4 días	mar 22/03/11	vie 25/03/11
14		Codificación	4 días	lun 28/03/11	jue 31/03/11
15		Prueba	4 días	vie 01/04/11	mié 06/04/11
16		**Módulo Test**	**9 días**	**jue 07/04/11**	**mar 19/04/11**
17		Especificación	3 días	jue 07/04/11	lun 11/04/11
18		Codificación	2 días	mar 12/04/11	mié 13/04/11
19		Prueba	4 días	jue 14/04/11	mar 19/04/11
20		**Interface Gráfico**	**8 días**	**mié 20/04/11**	**vie 29/04/11**
21		Diseño	2 días	mié 20/04/11	jue 21/04/11
22		Codificación	4 días	vie 22/04/11	mié 27/04/11
23		Prueba	2 días	jue 28/04/11	vie 29/04/11
24		Módulos creados y verificados	0 días	mié 27/04/11	mié 27/04/11
25		**Integración**	**4 días**	**jue 28/04/11**	**mar 03/05/11**
26		Pruebas Integración	4 días	jue 28/04/11	mar 03/05/11
27		Integración finalizada y probada	0 días	mar 03/05/11	mar 03/05/11
28		**Documentación**	**30 días**	**mié 04/05/11**	**mar 14/06/11**
29		Redacción memoria	15 días	mié 04/05/11	mar 24/05/11
30		Presentación Virtual	15 días	mié 25/05/11	mar 14/06/11
31		Final PFC	0 días	mar 14/06/11	mar 14/06/11

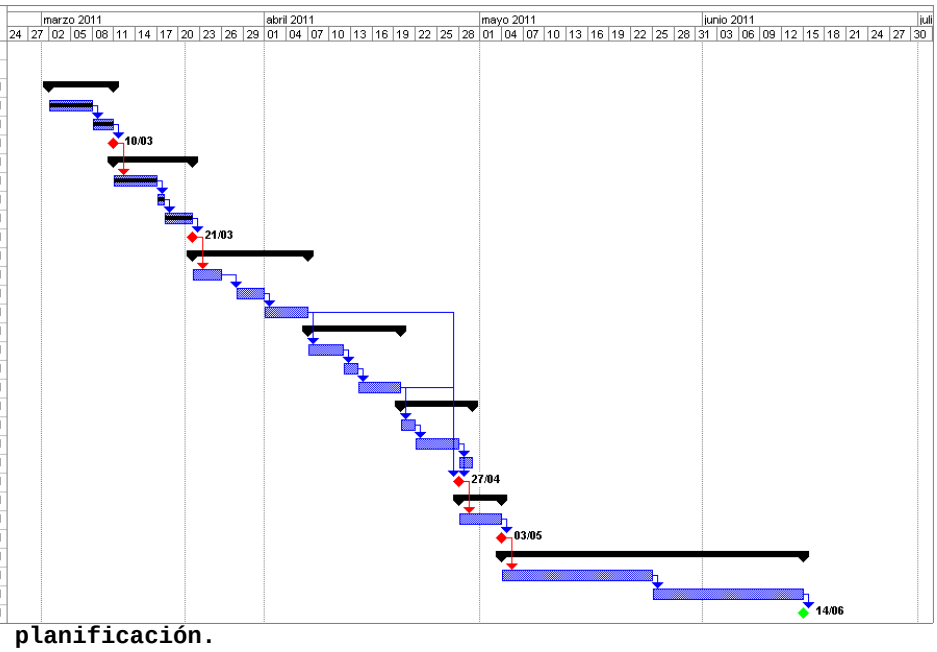


Ilustración 1. Diagrama de planificación.

Una vez finalizado el proyecto ha quedado patente que la planificación realizada ha sido totalmente equivocada. Esto se debe a que este proyecto contenía una parte de investigación y experimentación que no se tuvo en cuenta al inicio.
La planificación se realizó como si el objetivo, fuera solamente, obtener un programa que entrenará y luego realizará el test. Pero nada más lejos de la realidad, para obtener los resultados ha sido necesario realizar un gran número de pruebas que tardaban un tiempo considerable de máquina.

1.5 Productos obtenidos.

Durante la creación de este proyecto se han generado los siguientes productos:

- Plan de trabajo Microsoft Project.
- Aplicación de entrenamiento y prueba de aciertos.
- Memoria proyecto.
- Anexo creación estructura directorios para funcionamiento del programa.
- Anexo instalación librería OpenCV.

1.6 Entorno de desarrollo.

Las herramientas utilizadas para realizar este proyecto han sido:

- Notebook con un procesador Intel Pentium a 1,7GHz y 1GB de memoria.
- El sistema operativo ha sido Microsoft Windows XP.
- El entorno de programación ha sido Visual C++ 2010 Express.
- La librería de visión utilizada es OpenCV v2.1.
- La redacción de la memoria se ha realizado con Microsoft Word.

2 Módulo de entrenamiento.

Para introducirnos es este mundo necesitamos conocer algunos conceptos fundamentales que intentaré explicar a continuación.

2.1 Qué es un Fern.

Cuando queremos reconocer un objeto o área en una imagen lo primero que debemos hacer es situar una ventana sobre el punto de interés y tomar muestras que nos permitan más tarde su reconocimiento por probabilidad. La forma de tomar muestras, si tenemos imágenes en escala de grises, es comparando el nivel de gris entre dos pixeles y guardar esta información. Para que el reconocimiento sea posible necesitamos realizar unas $N \approx 300$ muestras, lo que nos lleva a tener que estimar y guardar 2^N entradas para cada clase (punto de interés) .
Par poder hacer este problema tratable hemos recurrido al trabajo de Mustafa Ozuysal, Michael Calonder , Vincent Lepetit y Pascal Fua denominado *Fast Keypoint Recognition using Random Ferns* (http://cvlab.epfl.ch) .
En nuestro caso la idea ha sido convertir $N \approx 300$ en $M \times 2^S$ con M entre 20 y 40 y S=10 lo que nos lleva a un orden de 30000 entradas.
Los autores del estudio anteriormente mencionado llaman a las subdivisiones creadas Ferns, por lo que a partir de ahora M y Fern será lo mismo.

2.2 Ventana de entrenamiento.

Ya nos hemos introducido un poco en materia, pero supongo que todavía no está muy claro como se realiza esta toma de muestras para realizar el clasificador.
Lo primero que debemos hacer es crear una ventana de entrenamiento que centraremos sobre el punto de interés. En nuestro caso hemos elegido una ventana de 24×24 pixeles.
Una vez tenemos centrada la ventana, el proceso es el siguiente:

- Obtenemos aleatoriamente dos pixeles A y B del interior de la ventana.
- Leemos sus valores de gris.
- Si A > B el valor es 1.
- En caso contrario el valor es cero.
- Colocamos este valor en una variable.
- Desplazamos la variable 1 bit a la izquierda.
- Repetimos estas acciones S veces.

Ilustración 2 . Proceso entrenamiento.

Una vez realizado este proceso, obtenemos un valor que representa el índice de una tabla, como S=10 y puede valer 0 o 1 nuestra tabla tendrá $2^S = 1024$ entradas.
Cogeremos la entrada obtenida e incrementaremos su valor en una unidad, esta acción la repetiremos Ferns (M) veces.

Entradas	Fern 1	Fern 2	...	Fern 30
1	0	1		2
2	1	0		0
...				
1024	0	1		0

Ilustración 3. Ejemplo Clasificador

Una vez terminado este proceso, guardaremos a disco la información obtenida y será nuestro clasificador para el punto de interés estudiado.
Realizaremos el mismo proceso para el resto de puntos de interés.

2.3 Entrenador.

En este apartado mostraremos el pseudocódigo del entrenador que hace posible la creación de los clasificadores.

```
Var caras=100;
Var puntos=20;
Var ferns=30;
Var test=10;
Para C=0 hasta caras hacer
        Leemos datos imagen C.
        Leemos puntos de interés de C.
        Para P=0 hasta puntos hacer
                Situamos la ventana sobre el punto de interés.
                Para F=0 hasta ferns hacer
                        Para  T=0 hasta test hacer
                                Si A>B entonces
                                        valor=1;
                                sino
                                        valor=0;
                                Fin Si
                                Indice=valor;
                                Indice << 1;
                        Fin Para
                        TablaFerns[indice]=incremento;
                Fin Para
        Fin Para
Fin Para
GuardarTabla(TablaFerns);
```

Ilustración 4. Pseudocódigo entrenador.

3 Módulo de prueba de aciertos.

Para poder comprender como se realiza la prueba de aciertos tenemos que introducir algunos conceptos nuevos como pueden ser la máxima probabilidad, evitar los valores cero, ventana de desplazamiento o detector de objetos.

3.1 Ventana de desplazamiento.

Para poder rastrear una imagen en busca de los puntos de interés es necesario desplazar la ventana de entrenamiento (24×24 pixeles) píxel a píxel y comprobar si dicha ventana es candidata a ser la elegida. Esto puede llevar un tiempo considerable ya que las imágenes que hemos utilizado para este experimento tienen un tamaño de 384×286 pixeles.
Por este motivo es indispensable acotar las zonas de rastreo a las zonas donde es más probable encontrar el punto de interés.

Ilustración 5. Rastreo para localizar los ojos.

3.2 Máxima probabilidad.

Una manera formal de explicarlo sería

$$P(f_1,f_2,...f_N \mid C = c_i) = \Pi_{k=1\rightarrow M}\, P(F_k \mid C = c_i)$$

Ilustración 6. Fórmula probabilidad.

Donde F_k representa el valor del índice obtenido de los test que realizamos dentro de la ventana y c_i es la clase del punto de interés que buscamos. Entonces la fórmula se expresa como el producto de las probabilidades obtenidas de cada fern (M).

La máxima probabilidad la buscaremos de la siguiente manera:

- Situamos una ventana a analizar.
- Realizamos el test de comparación de pixeles (A>B).
- Obtenemos el valor del índice de la tabla.
- Calculamos el $\log_{10}$(valor índice).
- Realizaremos esta acción para cada Fern y sumaremos los logaritmos obtenidos.
- Desplazamos la ventana un pixel en x y repetimos las acciones anteriores.
- Realizaremos las mismas acciones desplazando la ventana un pixel en y
- La ventana con la probabilidad más alta será la candidata a punto de interés.

La utilización de logaritmos se debe a que al multiplicar valores de probabilidad muy pequeños desbordamos la resolución de la máquina y se hace imposible su cálculo, para evitar esto convertimos los valores a logaritmos y de esta manera tenemos que

$$Log(A\times B\times....\times N) = Log(A)+Log(B)+...+Log(N)$$

Ilustración 7. Conversión multiplicación en suma de logaritmos.

3.3 Evitar valores cero.

Nuestra tabla de ferns tiene con toda seguridad muchos valores de índice (1 a 1024) con valor cero, esto se debe a que durante el entrenamiento no todos los valores de indice encuentran un acierto.
Durante el rastreo de la imagen si el valor de un índice es cero nos va a provocar un error al calcular el logaritmo, por este motivo debemos encontrar una solución a este problema.
En este punto utilizaremos la fórmula empleada en el trabajo anteriormente mencionado *Fast Keypoint Recognition using Random Ferns* para evitar dicho problema.

A cada valor leído del clasificador le aplicaremos

$$P_{k,ci} = N_{k,ci} + N_r / N_{ci} + K \times N_r$$

Ilustración 8. Fórmula para evitar los valores cero.

Donde $P_{k,ci}$ es la probabilidad del fern k y e l clasificador c_i .
$N_{k,ci}$ es el valor del indice k del clasificador c_i .
N_r es un factor de regularización que debe ser mayor de 0, en nuestro caso utilizaremos 0,01.
N_{ci} es el número de muestras (caras) del entrenamiento.
K es el número de posibles entradas en la tabla 2^S, en nuestro caso 1024.

3.4 Detector de objetos.

OpenCV lleva incorporada algunas librerías específicas para poder realizar acciones determinadas como la detección de objetos. Esta característica nos irá muy bien para limitar una imagen a solamente la cara de la persona que aparece y de esta manera reducir el área de rastreo.

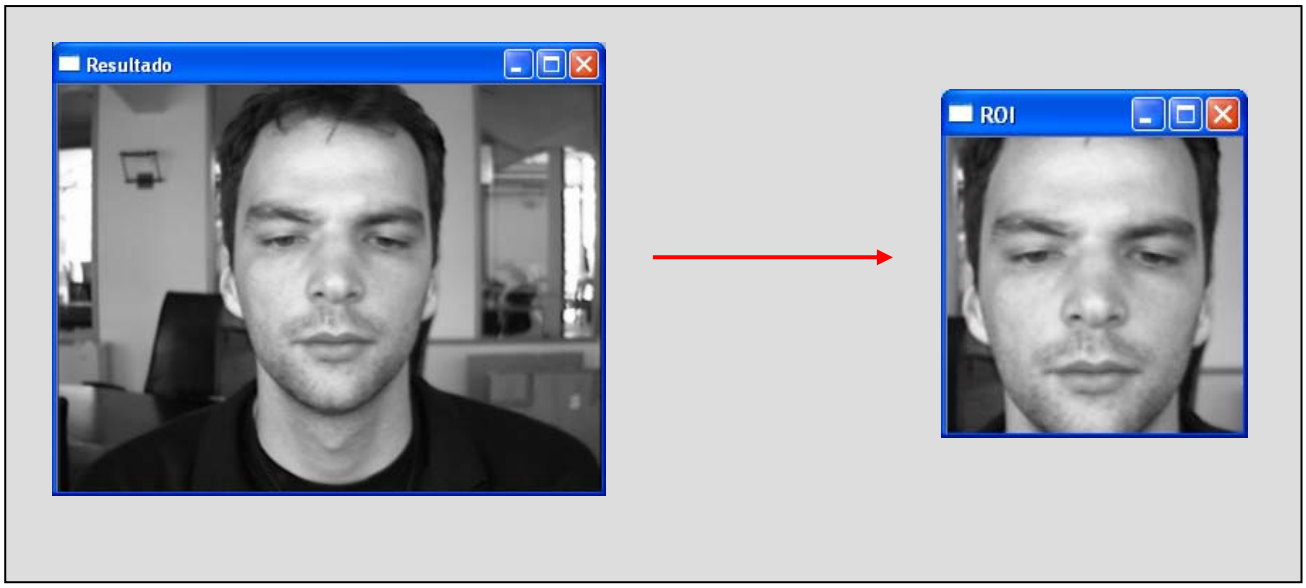


Ilustración 9. Detección rostro.

Para implementar esta funcionalidad hemos necesitado el clasificador de la cara que tiene incorporado OpenCV llamado *haarcascade_frontalface_alt.xml*.

```
(CvHaarClassifierCascade*)cvLoad("c:/DataPFC/haarcascades/haarcascade_frontalface_alt.xml");
```

y el método de detección de objetos

```
cvHaarDetectObjects(img,pcascade,pstorage,1.1,1,CV_HAAR_DO_CANNY_PRUNING,cvSize(100,100));
```

La función para realizar la detección, nos devolverá una variable tipo CvRect que define el área donde se encuentra el rostro y nos permitirá asignar esta área al ROI (Region of Interest) de la imagen. La función es la siguiente

```
//*************************************************************************
// Algoritmo Haar-like
//*************************************************************************
CvRect Prueba::DetectorCara(){
	CvSeq* pfaceRectSeq;
	CvRect r;
	pstorage=cvCreateMemStorage(0);
	pfaceRectSeq=cvHaarDetectObjects(img,pcascade,pstorage,1.1,1,
		CV_HAAR_DO_CANNY_PRUNING,cvSize(100,100));
	if(pfaceRectSeq->total>0){
		r=*(CvRect*)cvGetSeqElem(pfaceRectSeq,0);
	}else{
		r.height=0;
	}
	cvClearMemStorage(pstorage);

	return r;
}
```

Ilustración 10. Función detección cara.

3.5 Test de aciertos.

Una vez vistos los aspectos más relevantes que van a intervenir en el test de aciertos vamos a centrarnos en la estrategia para intentar localizar los puntos de interés.
Una posibilidad sería rastrear toda la imagen para cada punto de interés, aunque salta a la vista que esta opción no parece muy inteligente, ya que por ejemplo, el ojo izquierdo nunca se puede encontrar a la derecha del ojo derecho.
Con el anterior razonamiento y buscando siempre el mínimo gasto de tiempo de computación, he elegido la siguiente estrategia:

- Primero, localizar los dos centros de los ojos.
- Luego obtenemos la distancia entre ojos y la utilizamos para crear incrementos porcentuales en función de la distancia encontrada.
- Estos incrementos los utilizamos para irnos situando en los diversos puntos de interés.

El pseudocódigo para el test de aciertos es el siguiente:

```
Var carasTest=100;
CvRect imgR;
Var p=0; // punto de interés
Para C=0 hasta carasTest hacer
        Leemos datos imagen C.
        Leemos puntos de interés de C.
        ImgR=Detectamos cara();
        cvSetImageROI(img,imgR); // definimos la ROI donde se encuentra el rostro

        Definimos area de rastreo ojo derecho ini_x, fin_x, ini_y, fin_y
        P=0; // ojo derecho
        Leemos tabla ferns(p);
        DesplazamosVentana(p,ini_x,fin_x,ini_y,fin_y);

        Definimos area de rastreo ojo izquierdo ini_x, fin_x, ini_y, fin_y
        P=1; // ojo izquierdo
        Leemos tabla ferns(p);
        DesplazamosVentana(p,ini_x,fin_x,ini_y,fin_y);

        Distancia_entre_ojos= distancia_ojo_iz.x – distancia_ojo_der.x;
        Calculamos incrementos porcentuales inc5, inc20,inc30,....

        A partir de aquí buscamos el resto de puntos utilizando los incrementos y la lógica de
        situación.

Fin Para
GuardarResultadosTest();
```

Ilustración 11. Pseudocódigo test de aciertos.

El código de la función de desplazamiento nos va a permitir rastrear la zona delimitada por las variables ini_x, fin_x, ini_y, fin_y. Una vez encontrada la ventana con máxima probabilidad, almacenaremos la información en la estructura mostrada a continuación.

```
struct T_TEST{CvPoint esquina;double prob;bool PER;int tol_x;int tol_y;};
```

Ilustración 12. Estructura donde almacenamos el resultado del test para cada punto.

```
//*************************************************************************
// Desplaza una ventana pixel a pixel hasta encontrar la máxima // //
//probabilidad los límites vienen fijados por ini_y,fin_y,ini_x,fin_x
//*************************************************************************
void Prueba::DesplazaVentana(int p,int ini_y,int fin_y,int ini_x,int
fin_x){
	double proba=0.0;
	ven_test[p].prob =-1000.0;
	for (int desp_y=ini_y ;desp_y<fin_y ;desp_y++){
		for (int desp_x=ini_x;desp_x< fin_x;desp_x++){
			// calculamos la probabilidad de la ventana
			proba=ProcesaTest(desp_x,desp_y);
			if(proba>ven_test[p].prob ){
				ven_test[p].prob=proba;
				ven_test[p].esquina.x=desp_x;//
				ven_test[p].esquina.y=desp_y;//
			}
		}//desp_x
	}// desp_y
	ven_test[p].tol_x= ven_test[p].esquina.x+ventana/2 -
puntos_clave[p][0];
	ven_test[p].tol_y= ven_test[p].esquina.y+ventana/2 -
puntos_clave[p][1];

	if(abs(ven_test[p].tol_x )>tolerancia ||
abs(ven_test[p].tol_y)>tolerancia){
		ven_test[p].PER=true; // el punto encontrado supera la
tolerancia fijada
	}
	else{
		ven_test[p].PER=false; // el punto se encuentra dentro de la
tolerancia
		aciertos[p]=aciertos[p]+1;
	}
}
```

Ilustración 13. Código función desplazamiento.

4 Interface gráfico.

Para poder realizar un uso práctico de los dos módulos anteriormente descritos (entrenamiento y test de aciertos) he creado una interface que considero una ayuda a la ejecución de las diversas pruebas.

4.1 Descripción.

En la siguiente imagen se describen las características más relevantes.

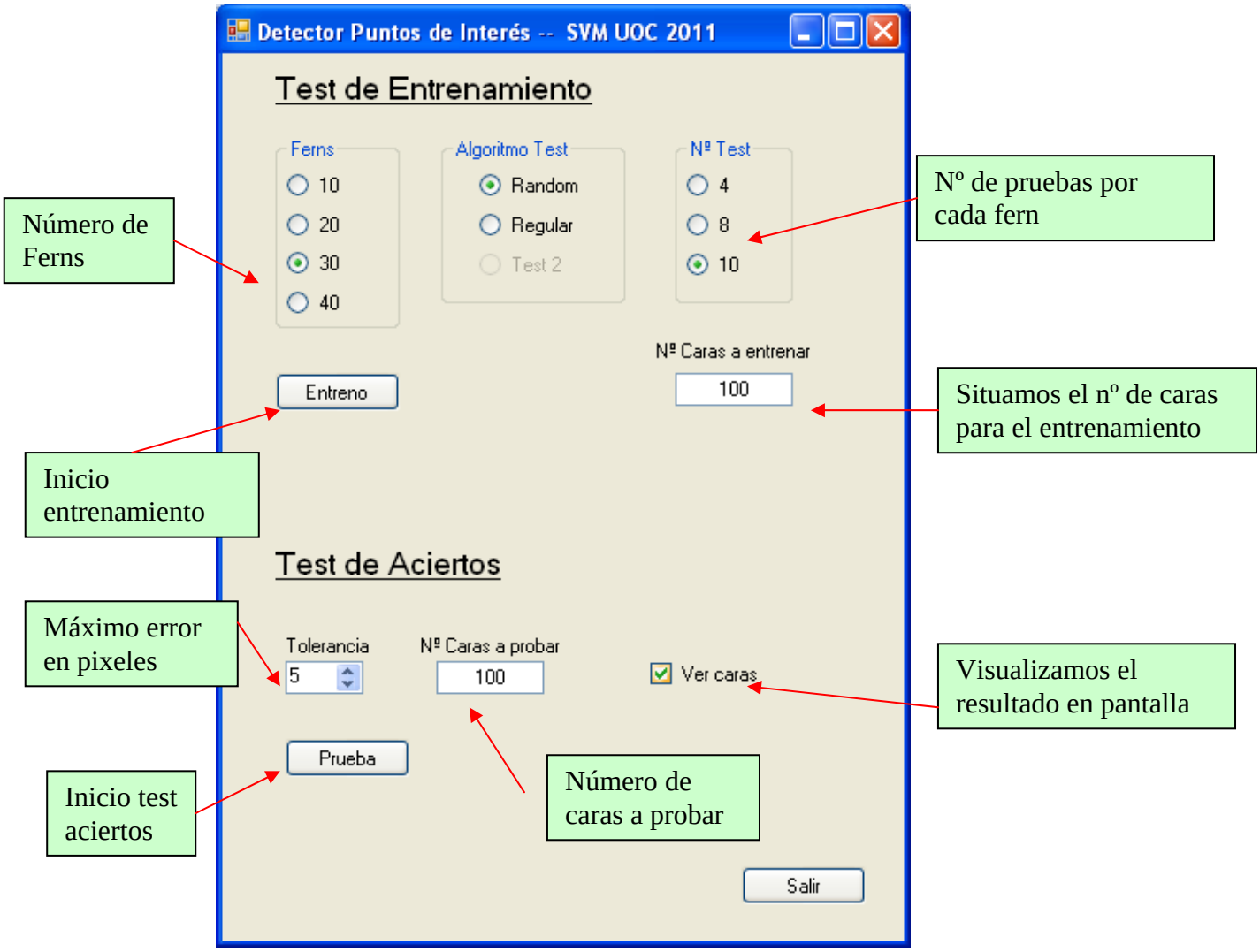


Ilustración 14. Interface gráfico.

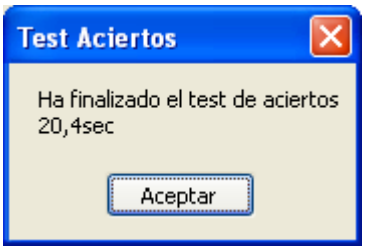
4.2 Funcionamiento.

Una vez puesta en marcha la aplicación su funcionamiento es el siguiente:

- Seleccionamos los Ferns, el nº de test, tipo algoritmo y las caras para el entrenamiento.
- Pulsamos sobre el botón Entreno.
- Al cabo de unos segundos aparece el mensaje de finalización de entreno.

Ahora pasamos al test de aciertos.

- Seleccionamos la tolerancia deseada (entre 2 y 10 pixeles de error).
- Elegimos el número de caras a probar (máximo 1520).
- Marcamos si deseamos ver el resultado de cada imagen probada.
- Esperamos el mensaje de finalización de test de aciertos (el test puede tardar varios minutos).



Si hemos elegido ver caras, el resultado aparecerá en dos ventanas en la que una nos mostrará el resultado de aciertos y la otra el ROI detectado.

Ilustración 15. Resultado con la opción ver caras activada.

Cuando algún punto supera la tolerancia fijada los puntos aparecerán en color rojo sobre la imagen.

Ilustración 16. Resultado con algún punto fuera de tolerancia.

Una vez finalizado el test de aciertos se crea un fichero llamado resultados.txt en el cual han quedado registrados los porcentajes de aciertos de cada punto de interés así como los parámetros sobre los que se ha realizado la prueba.

```
Parámetros Test Entreno
Ferns:30
Caras:100

Parámetros Test Aciertos
Tolerancia:5
Caras a Probar:100

88%
90%
53%
49%
76%
40%
66%
86%
79%
87%
87%
88%
87%
86%
89%
90%
78%
90%
38%
58%
```

Ilustración 17. Fichero resultados.txt

5 Imágenes de prueba.

En este apartado voy a describir las características más importantes sobre el conjunto de imágenes de prueba utilizadas.

5.1 Formato.

Las imágenes utilizadas tienen una dimensión de 348×286 pixeles, son en escala de grises con una profundidad de 8 bits. El formato del fichero es PGM y es uno de los que soporta OpenCV. La colección de imágenes e información relacionadas con ellas para realizar este trabajo han sido descargadas de http://www.bioid.com.

La colección de imágenes está identificada de la siguiente manera:
BioID_0000.pgm
...
BioID_1520.pgm

Los puntos de interés asociados a cada cara se denominan
Bioid_0000.pts
...
Bioid_1520.pts

El formato del fichero se muestra a continuación (fichero obtenido de http://www.bioid.com).

```
version: 1
n_points: 20
{
159.128 108.541
230.854 109.176
164.841 179.633
223.237 178.998
132.469 93.9421
183.883 94.5768
211.177 95.2116
254.974 91.4031
129.295 109.176
144.529 109.811
176.901 107.272
216.89 107.272
246.088 110.445
261.957 109.811
196.578 139.009
184.518 147.261
207.369 145.991
195.943 175.189
193.404 193.597
192.769 229.143
}
```

Ilustración 18. Detalle fichero punto de interés.

Cada línea representa el valor X,Y del punto de interés.

El orden de los puntos de interés se describe en la siguiente relación (relación obtenida de http://www.bioid.com).

```
There are 20 manually placed points on each of your 1521 images.

The markup scheme is as follows:

  0 = right eye pupil
  1 = left eye pupil
  2 = right mouth corner
  3 = left mouth corner
  4 = outer end of right eye brow
  5 = inner end of right eye brow
  6 = inner end of left eye brow
  7 = outer end of left eye brow
  8 = right temple
  9 = outer corner of right eye
 10 = inner corner of right eye
 11 = inner corner of left eye
 12 = outer corner of left eye
 13 = left temple
 14 = tip of nose
 15 = right nostril
 16 = left nostril
 17 = centre point on outer edge of upper lip
 18 = centre point on outer edge of lower lip
 19 = tip of chin
```

Ilustración 19. Orden de los puntos de interés.

6 Algoritmo entrenamiento.

Para poder realizar el entrenamiento, ya se ha comentado que situamos una ventana en el centro del punto de interés y capturamos una serie de muestras. Pero, ¿ qué pixeles utilizar para capturar las muestras?

6.1 Random.

Uno de los dos algoritmos que se han probado lo he llamado Random y consiste en buscar de manera aleatoria pixeles dentro de la ventana de entrenamiento.

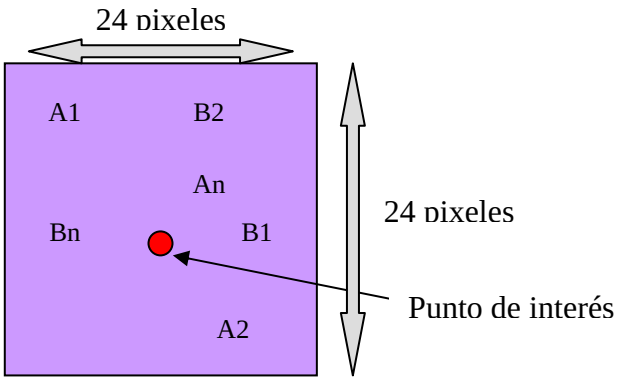


Ilustración 20. Diagrama entreno Random.

Para obtener estos valores se calculan 4 valores (A_x, A_y, B_x, B_y) en cada iteración y los valores se eligen entre 1 y 24, estos valores son almacenados en una tabla.
El código que realiza esta acción se muestra a continuación.

```
//*************************************************************************
// crea los puntos aleatorios
//*************************************************************************
void Training::CreaPuntosRand(){
      srand(1);
      for(int cnt_fila=0;cnt_fila<base_test*ferns;cnt_fila++){
            for    (int cnt_col=0;cnt_col<4;cnt_col++){
                   puntos_test[cnt_fila][cnt_col] = rand() % ventana + 1;
            }
      }

}
```

Ilustración 21. Código algoritmo Random.

6.2 Regular.

El segundo algoritmo que he probado le he llamado regular, porque consiste en comparar pixeles vecinos e ir avanzando línea a línea.

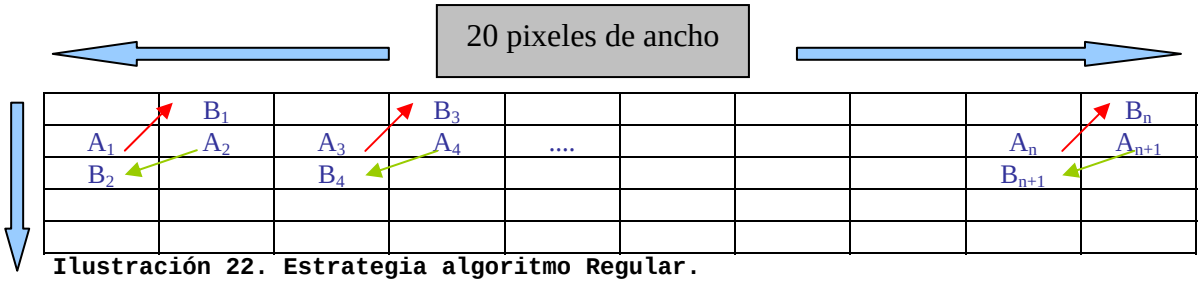


Ilustración 22. Estrategia algoritmo Regular.

Si se alcanza el límite de la ventana se desplaza un píxel en dirección del eje - X y se repite el proceso desde el inicio.

6.3 Random Versus Regular.

Par poder comparar la efectividad de los dos algoritmos se ha realizado un entrenamiento y un test de aciertos con las siguientes características:

- Caras de entrenamiento: 100
- Caras a probar: 100
- Ferns: 20, 30, 40.
- Tolerancia: 5 pixeles.

Los resultados se muestran en la siguiente tabla.

	Random			Regular		
100 muestras entreno	**20**	**30**	**40**	**20**	**30**	**40**
pupila derecha	91%	93%	95%	84%	91%	92%
pupila izquierda	94%	95%	94%	92%	89%	93%
esquina derecha boca	82%	85%	85%	76%	79%	81%
esquina izquierda boca	81%	85%	89%	74%	76%	79%
exterior ceja derecha	92%	93%	95%	82%	88%	89%
interior ceja derecha	74%	75%	76%	78%	75%	79%
interior ceja izquierda	74%	81%	77%	62%	65%	65%
exterior ceja izquierda	89%	92%	92%	89%	90%	93%
sien derecha	75%	82%	86%	83%	90%	91%
esquina exterior ojo derecho	85%	88%	87%	78%	81%	87%
esquina interior ojo derecho	87%	92%	93%	78%	86%	90%
esquina interior ojo izquierdo	89%	93%	92%	85%	85%	88%
esquina exterior ojo izquierdo	90%	92%	91%	76%	76%	78%
sien izquierda	91%	92%	93%	92%	92%	92%
punta nariz	91%	93%	93%	85%	91%	92%
orificio nariz derecho	93%	95%	95%	85%	91%	93%
orificio nariz izquierdo	80%	83%	84%	65%	74%	76%
punto central labio superior	93%	95%	95%	83%	91%	92%
punto central labio inferior	81%	81%	79%	80%	78%	81%
punta barbilla	49%	59%	57%	69%	75%	76%
Media	83	87	87	79	83	85

Ilustración 23. Tabla comparativa Ferns y algoritmos.

Los resultados que se aprecian, es que cuando aumentamos el número de ferns aumenta claramente el porcentaje de aciertos. Por otro lado tiene un comportamiento algo mejor el algoritmo Random, en valor medio, que el Regular, con la única excepción del último punto (punta barbilla) que presenta un mejor comportamiento el algoritmo Regular.
La conclusión que podemos obtener es que el último punto presenta unas características muy homogéneas, y en este caso el comportamiento del algoritmo Regular mejora la tasa de aciertos casi un 20%.

7 Resultados.

Para poder obtener resultados y poder contrastar el comportamiento se han realizado una serie de ensayos con diferentes parámetros de entrenamiento y test de aciertos.

7.1 Ensayo 1.

Para este ensayo se han tenido en cuenta solamente dos puntos de interés, pupila derecha y pupila izquierda. El entreno se ha realizado con 40 Ferns y 500 caras de entreno. Luego se han pasado por el test de aciertos 1500 caras y hemos observado su comportamiento.

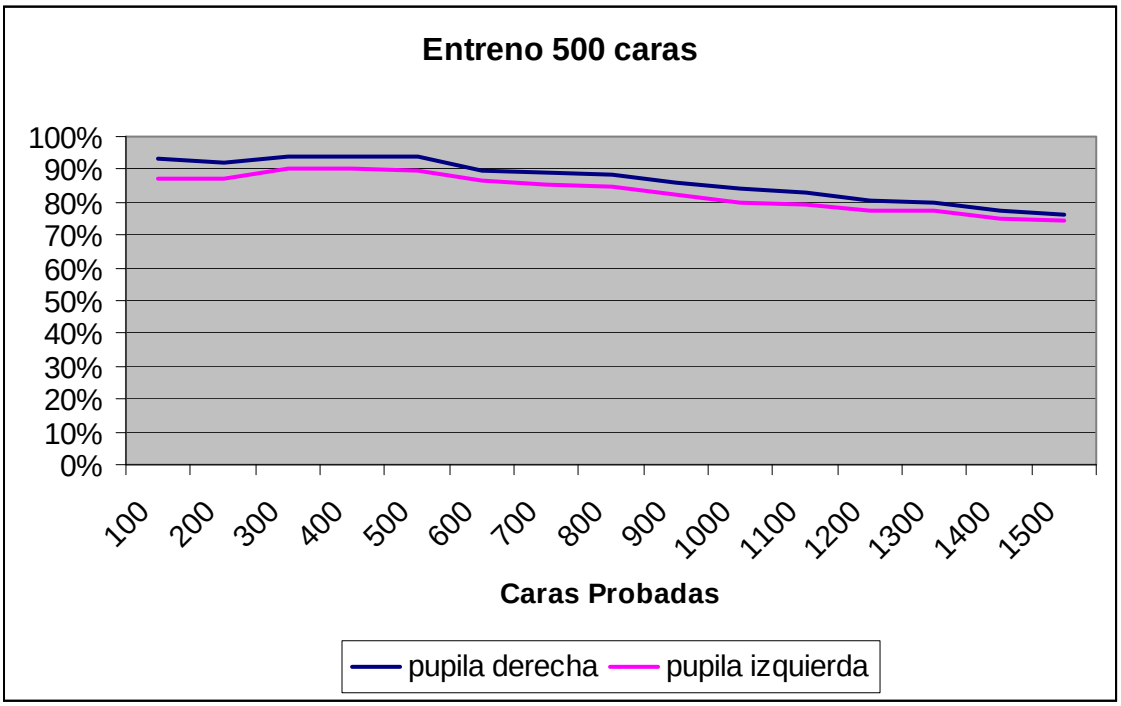


Ilustración 24. Ensayo 1.

El resultado ha sido, que a partir de las caras de entreno (500) la tasa de aciertos disminuye pero sigue conservando un nivel bastante aceptable (sobre el 75% de aciertos). Esto quiere decir que el patrón de entreno sigue funcionando con caras que no han participado en dicho entreno.

7.2 Ensayo 2.

Ahora vamos a utilizar todas las caras de las que disponemos para hacer el entrenamiento (1500). Las condiciones serán las mismas, mismo algoritmo y misma cantidad de Ferns.

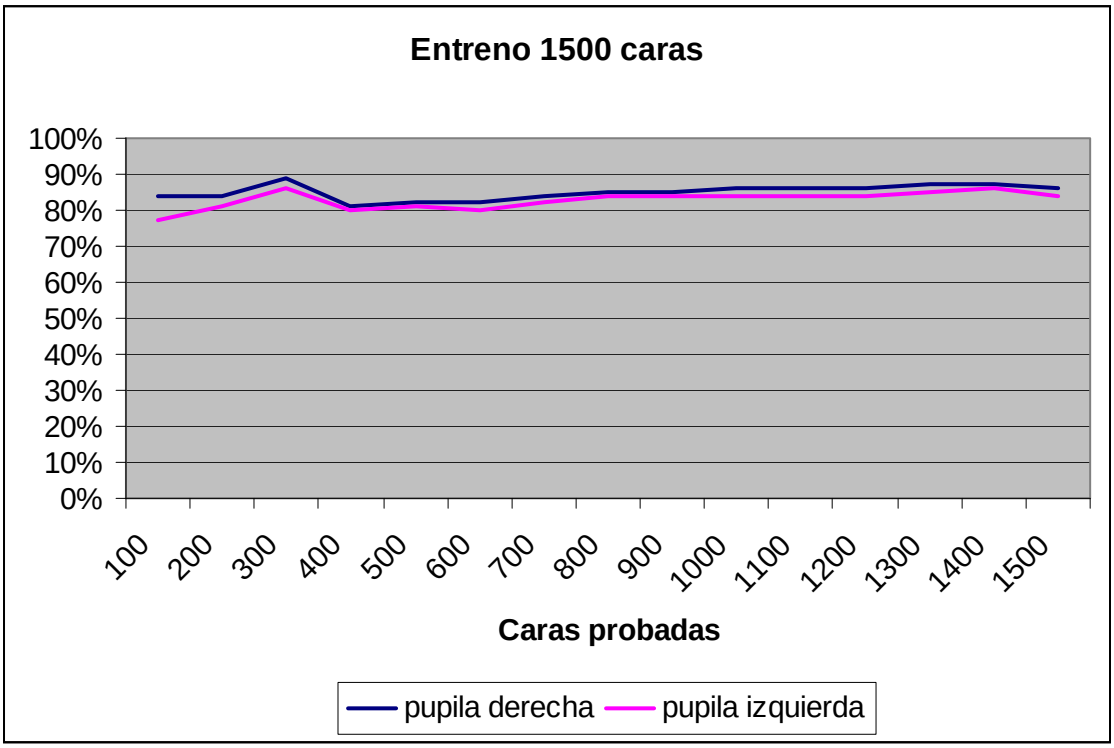


Ilustración 25. Ensayo 2.

El resultado es que han disminuido los puntos máximos de aciertos pero se ha suavizado el comportamiento obteniendo una tasa de aciertos del 80% durante casi todas las pruebas.

7.3 Ensayo 3.

Así como en los casos anteriores en que hemos restringido el ensayo a dos puntos de interés, ahora vamos a realizarlo sobre todos los puntos de interés y ordenaremos los resultados de mayor a menor para poder apreciar de una forma más clara que puntos son más fáciles de detectar.

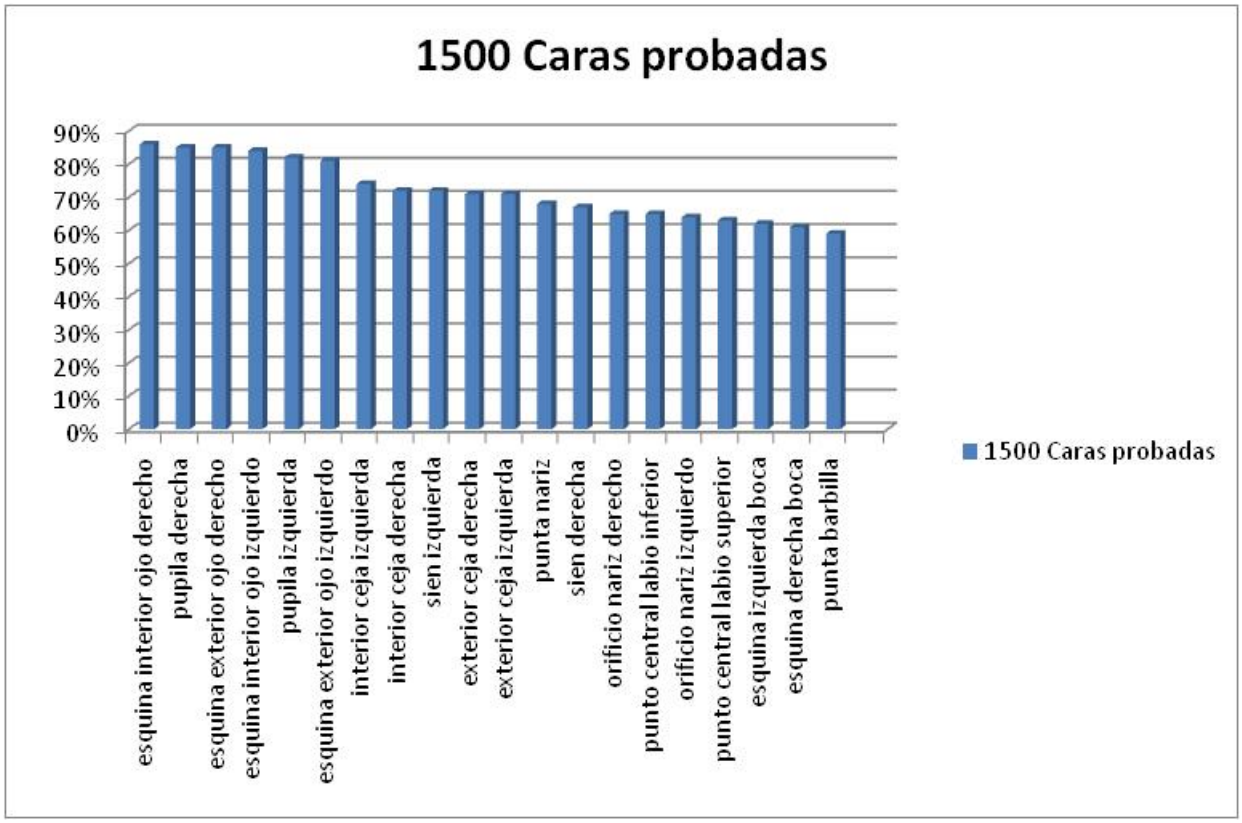


Ilustración 26. Ensayo 3.

Según se observa en el gráfico, los puntos con una mejor tasa de aciertos son las pupilas y los extremos de los ojos, es posible que se deba a que una vez localizados los centros de las pupilas el área de rastreo se restringe mucho alrededor de estos y por este motivo la tasa de aciertos disminuye mínimamente.

7.4 Ensayo 4.

En la siguiente tabla podemos observar como mejora la tasa de aciertos si aumentamos la tolerancia de errores a 8 pixeles.

	Tolerancia		
	5 pixeles	8 pixeles	Diferencia
pupila derecha	85%	86%	1%
pupila izquierda	82%	83%	1%
esquina derecha boca	61%	74%	13%
esquina izquierda boca	62%	74%	12%
exterior ceja derecha	71%	82%	11%
interior ceja derecha	72%	82%	10%
interior ceja izquierda	74%	83%	9%
exterior ceja izquierda	71%	82%	11%
sien derecha	67%	74%	7%
esquina exterior ojo derecho	85%	88%	3%
esquina interior ojo derecho	86%	89%	3%
esquina interior ojo izquierdo	84%	87%	3%
esquina exterior ojo izquierdo	81%	86%	5%
sien izquierda	72%	83%	11%
punta nariz	68%	78%	10%
orificio nariz derecho	65%	74%	9%
orificio nariz izquierdo	64%	75%	11%
punto central labio superior	63%	70%	7%
punto central labio inferior	65%	74%	9%
punta barbilla	58%	72%	14%
Total	71	80	6

Ilustración 27. Comparación tolerancia.

Podemos observar como los puntos que tienen una tasa más baja su valor aumenta por encima del 10%.

7.4 Pruebas de rendimiento.

Para ver como se comporta la aplicación y el consumo de tiempo de procesador se han comparado tres tipos de máquinas, cuyos resultados se muestran a continuación.

	NoteBook	PC de Escritorio	PC de Escritorio
Procesador	Pentium	Pentium 4	Pentium Dual Core
Velocidad	1,7GHz	3GHz	2,76GHz
Memoria	1GB	1GB	2GB
S.O.	Windows XP	Windows XP	Windows 7
Entrenamiento (1500 caras)		43sec	8sec
Test de aciertos (1500 caras)		3188sec	1661sec

Ilustración 28. Prueba rendimiento.

8 Conclusiones.

Este trabajo me ha permitido introducirme en la visión artificial así como en algunas técnicas utilizadas para el reconocimiento de objetos. Una vez he conseguido redactar esta memoria, todavía pienso en algunos algoritmos que podría haber probado, o corregir la inclinación de los rostros para el entrenamiento, o ajustar el tamaño de las imágenes para que todas tuvieran las mismas proporciones. Todo esto son ideas que iban apareciendo día a día pero que ha sido imposible llevar a cabo su implementación y que considero con toda probabilidad habrían contribuido a mejorar los resultados.

Creo que el resultado de los ensayos realizados demuestra que es posible la utilización del concepto de Fern para el reconocimiento de objetos con una tasa de aciertos más que aceptable y de esta manera reducir la cantidad de información a almacenar y a procesar.

No tengo dimensión de si los tiempos empleados en la ejecución se encuentran dentro de parámetros adecuados, con la máquina más potente conseguimos reconocer los 20 puntos de interés de una cara en 1,8sec.

Las líneas futuras de este proyecto continuarán con la implementación de imágenes en movimiento para el entrenamiento y posterior identificación de las mismas.

9 Glosario.

- **Clase:** Representa un punto de interés.
- **Clasificador:** Viene de la inteligencia artificial, es una información almacenada que se utiliza como patrón para el reconocimiento de características de un objeto.
- **Fern:** Término utilizado en el estudio *Fast Keypoint Recognition using Random Ferns.* Objetivo divide y vencerás, es una forma de agrupamiento de pruebas.
- **OpenCV:** Librería de visión artificial desarrollada por Intel.
- **Pixel:** Es la menor unidad de representación de una imagen.
- **Punto de interés:** Coordenadas x,y de una imagen sobre la que queremos extraer información de su estructura para su posterior reconocimiento.
- **Visual C++.** Entorno de programación y depuración en lenguaje C.
-

10 Bibliografía.

- Programación en C. ISBN: 84-481-3013-8
- Apuntes de Inteligencia Artificial II. Material didáctico de la UOC.
- Wikipedia. Consulta sobre conceptos varios.
- WWW.BioID.com. Web sobre biométrica de la cara. También ha proporcionado la colección de imágenes para este proyecto.
- Learning OpenCV. ISBN 9780596516130. Libro electrónico, biblioteca UOC.
- Visual C++® 6 Unleashed. ISBN-13: 978-0-672-31241-0 . Libro electrónico, biblioteca de la UOC.
- *Fast Keypoint Recognition using Random Ferns* (http://cvlab.epfl.ch) . Documento de inspiración para la realización del proyecto.
- http://sourceforge.net/projects/opencvlibrary Web para la obtención de recursos sobre visión por computador.
- http://www.microsoft.com/express/Downloads/ Descarga del entorno de desarrollo para este proyecto.
- http://www.cmake.org/cmake/resources/software.html Programa para compilar OpenCV con el compilador nativo del ordenador.

11 Anexos .

11.1 Estructura directorios.

Para poder ejecutar la aplicación de forma adecuada, es necesario crear la siguiente estructura de directorios.

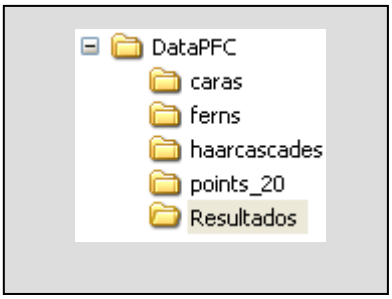


Ilustración 29. Directorios datos.

DataPFC: es un subdirectorio de C:\
Caras: Es donde guardamos los archivos BioID_0000.pgm
Ferns: Quedan almacenados los 20 clasificadores después de un entrenamiento.
Haarcascades: Clasificadores de OpenCV para la detección de objetos.
Points_20: Es donde guardamos los ficheros con la información de los puntos de interés de cada cara.
Resultados: después de un test de aciertos se almacena el resultado en el fichero Resultados.txt.

11.2 Instalación aplicación.

Como se ha podido apreciar a lo largo de este trabajo el objetivo final no solo era disponer de una aplicación que realizara un entrenamiento y el posterior test de aciertos, sino que debíamos realizar una serie de ensayos para obtener información que validara la efectividad del sistema. Este último hecho me ha llevado ha dejar una aplicación un tanto abierta para que las modificaciones para realizar los ensayos fueran posibles.
Si se hubiera tenido que implementar la casuística para realizar los ensayos desde el interface gráfico no habría quedado tiempo para los ensayos.

Por este motivo para poder ejecutar el programa es necesario instalar el entorno de desarrollo Visual C++ 2010 Express, es gratuito y se obtiene la licencia de uso desde la propia web http://www.microsoft.com/express/Downloads/

Para instalar la librería OpenCV utilizada la descargaremos de la dirección http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.1/OpenCV-2.1.0-win.zip y la instalaremos en C:\OpenCV2.1

Aquí no acaba todo para poder utilizar las librerías de forma correcta es necesario compilar todas las fuentes de OpenCV para el compilador nativo instalado en el ordenador. Esto se consigue con la aplicación libre CMake que se puede obtener de

http://www.cmake.org/cmake/resources/software.html
Una captura de este proceso lo podemos ver a continuación.

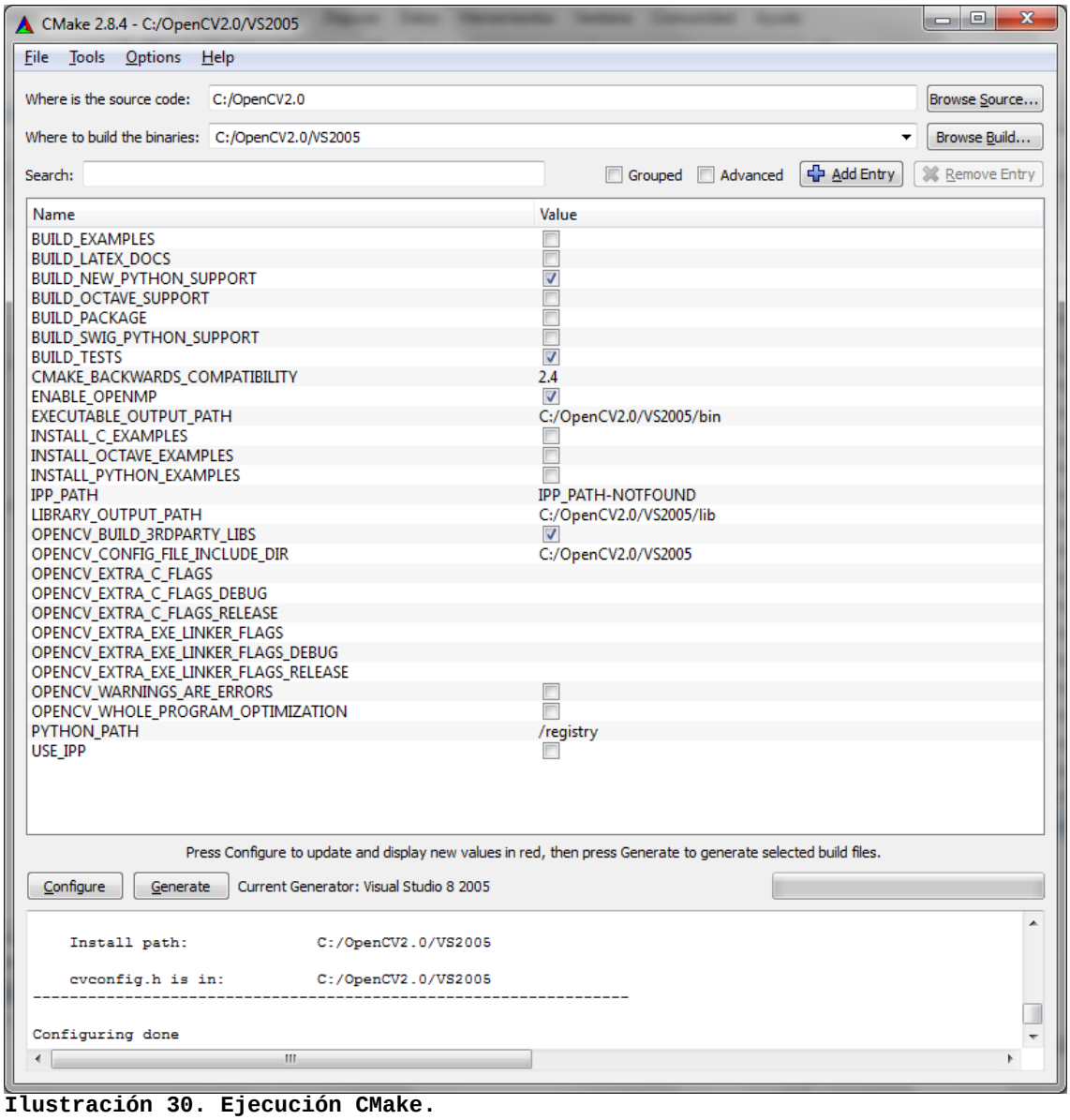


Ilustración 30. Ejecución CMake.

Una vez creado el proyecto OpenCV ejecutamos el proyecto ALL_BUILD y ya tendremos las librerías perfectamente compiladas.
Una vez realizado este paso, ya podemos referenciar desde el entorno de desarrollo las librerías generadas.

11.3 Configuración.

Para poder configurar correctamente el entorno de desarrollo creo que lo más fácil es seguir las siguientes imágenes.
Copiar toda la aplicación en un directorio de nuestro ordenador. Ejecutar el fichero AppPFC.sln en el directorio donde se ha copiado el proyecto.
Una vez abierto el entorno de desarrollo deberemos ver el explorador de soluciones con la siguiente estructura.

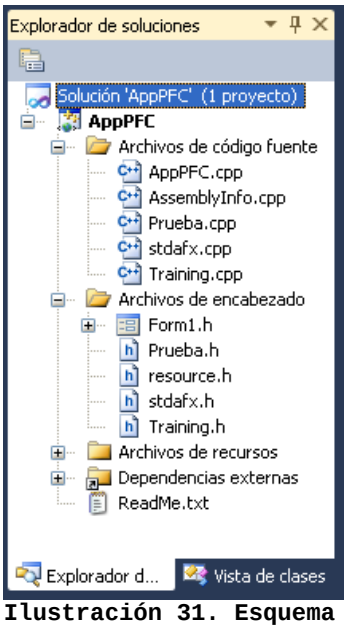


Ilustración 31. Esquema solución.

Para configurar correctamente el entorno indicaremos al sistema donde encontrar las librerías.

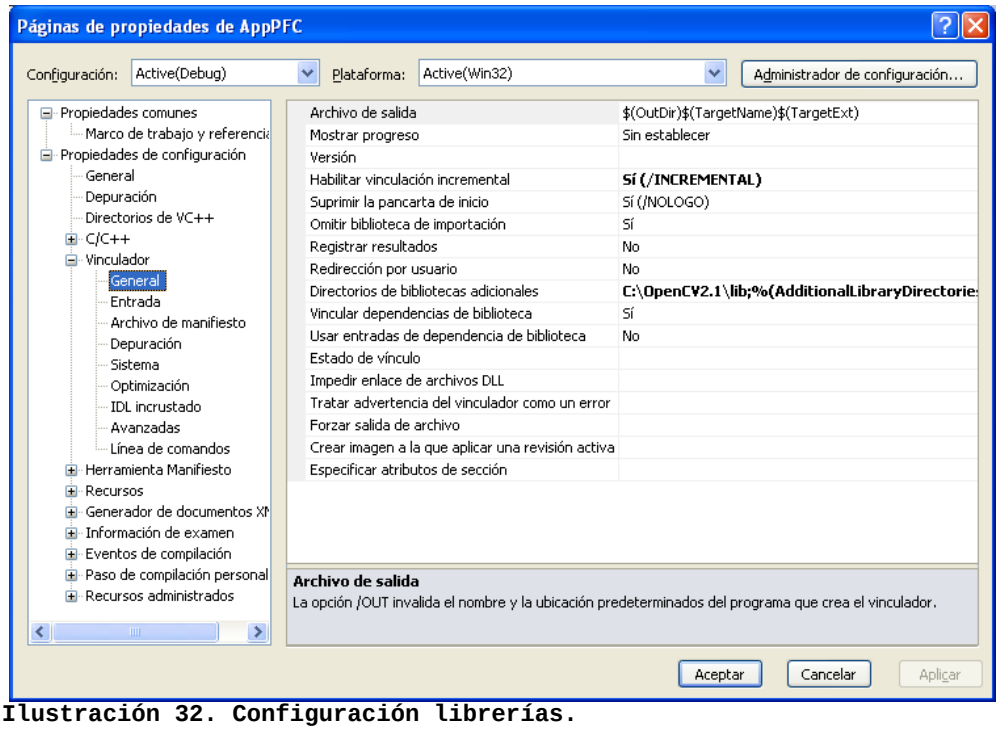


Ilustración 32. Configuración librerías.

Seguidamente indicaremos los directorios de VC++.

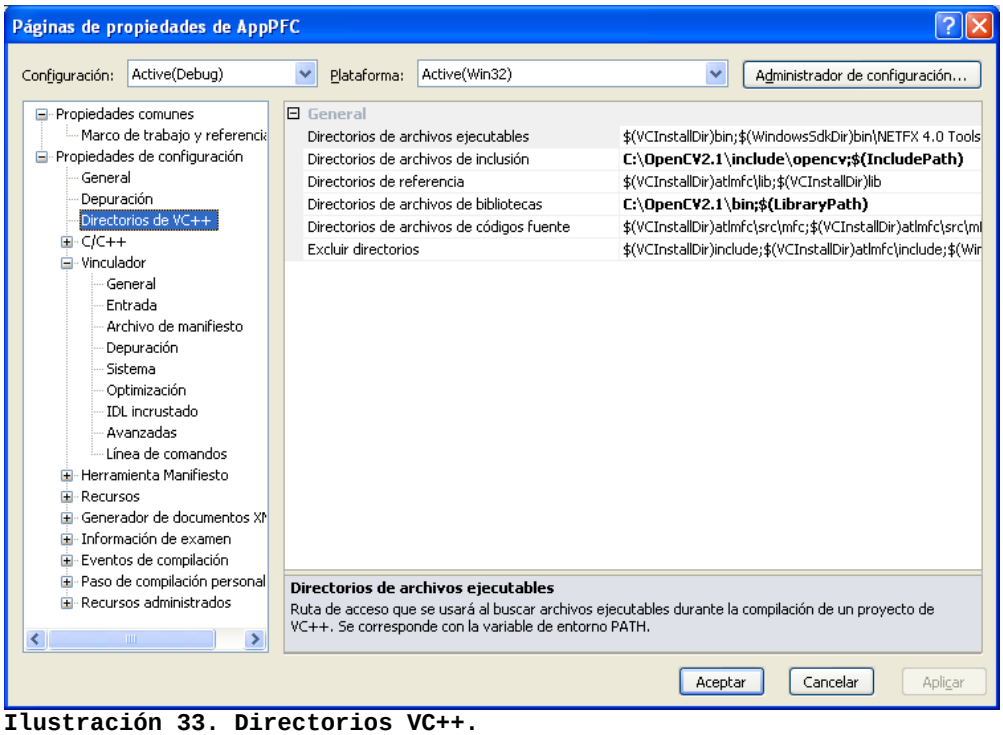


Ilustración 33. Directorios VC++.

Y por último las dependencias adicionales.

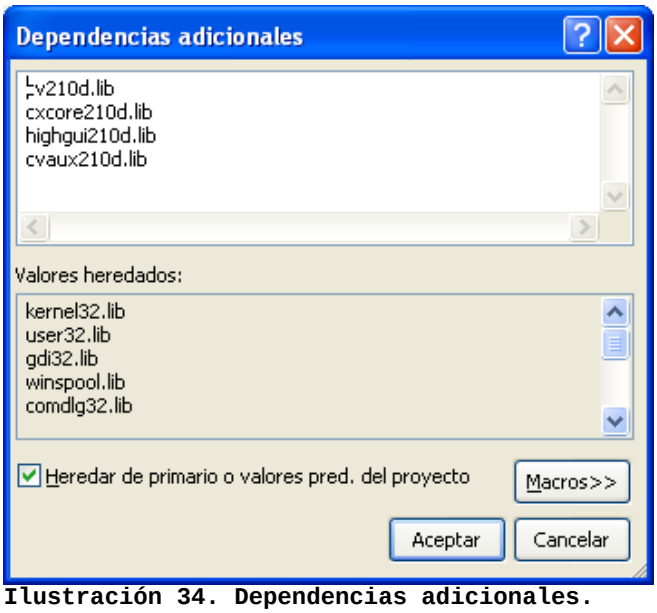


Ilustración 34. Dependencias adicionales.

En estos momentos ya estamos en disposición de compilar el proyecto y ejecutarlo.