FIB Facultat d'Informàtica de Barcelona

Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial
UNIVERSITAT POLITÈCNICA DE CATALUNYA

VISIÓN POR COMPUTADOR

Short Project

Facultad de Informática de Barcelona

Adrian Cristian Crisan

Filip Gedung Dorm

Pablo Vega Gallego

Barcelona, diciembre de 2021

Índice

1. Objetivos del proyecto

El objetivo del proyecto es implementar un sistema automático para detectar la mirada mediante visión por computador. El sistema ha de ser capaz de detectar dónde están los ojos y posteriormente medir el ángulo horizontal de la mirada en base a la posición del iris. Las imágenes con las que se trabajarán serán con un encuadro tipo busto de un usuario sin gafas en una orientación vertical con rotaciones de la cabeza poco apreciables; tal como se muestra a continuación en la imagen de la figura siguiente:

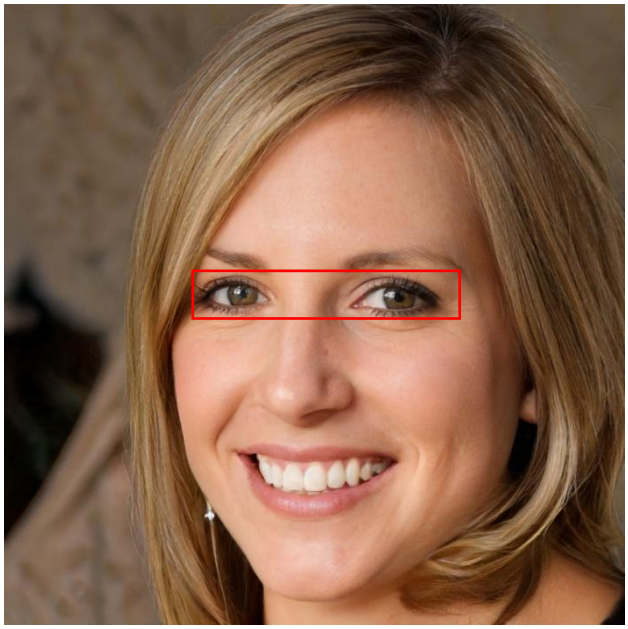


Figura 1 – Imagen de ejemplo de una detección de ojos (imagen extraída de thispersondoesnotexist.com)

2. Recopilación de la base de datos

Para que nuestro programa sepa si la imagen que estamos analizando contiene ojos o no, necesitamos que nuestra base de datos esté compuesta por imágenes con ojos e imágenes sin ojos.

2.1. Imágenes de ojos

Para este paso, hemos recogido las imágenes ofrecidas por la asignatura y hemos eliminado las personas que tenían gafas y añadido nuevas imágenes, teniendo en total de 220 caras.

Dado que todas las imágenes con ojos tendrán estos en aproximadamente la misma posición y tendrán el mismo tamaño siempre, podemos recortar todas las imágenes en la misma posición. Por lo que para recortar las imágenes con ojos, hemos ido ajustando el recorte para que abarque todos los ojos de las personas, dando cierto margen.

Figura 2.1 – Ejemplo de recorte para la base de datos para los ojos

El recorte se realiza en el punto (256, 426) y tiene unas dimensiones de 513 de ancho por 113 de alto, en el anexo podemos ver cómo se han realizado los cálculos y la ejecución de este apartado.

2.2. Imágenes de no ojos

En este caso, para obtener imágenes de no ojos, o de un solo ojo, lo que hemos hecho, han sido un programa que haga recortes de la misma dimensión que en el caso anterior, pero realizando recortes de puntos escogidos de manera aleatorio de las propias caras, por cada cara recogemos 20 recortes de no ojos. Lo hacemos de la siguiente manera:

1. Seleccionamos 2 puntos aleatorios de la imagen entre 1 y 1024.
2. Comprobamos que estos puntos no se encuentren en el área resaltada de color rojo en la figura 2.2.
    a. Esta área toma como punto central el punto (256, 426).
    b. Se extiende 512/2 por la izquierda y por la derecha formando un rectángulo de 512 de ancho.
    c. Para la altura de esta área prohibida hacemos lo mismo que en el caso anterior, pero obteniendo como resultado 112 píxeles de altura.
3. Hacemos el recorte de la imagen.

De esta manera nos aseguramos de que los recortes que hace el algoritmo no se encuentren los dos ojos enteros. A su vez tenemos también partes de ojos que no consideramos ojos enteros y que estarían mal detectarlos como ojos.

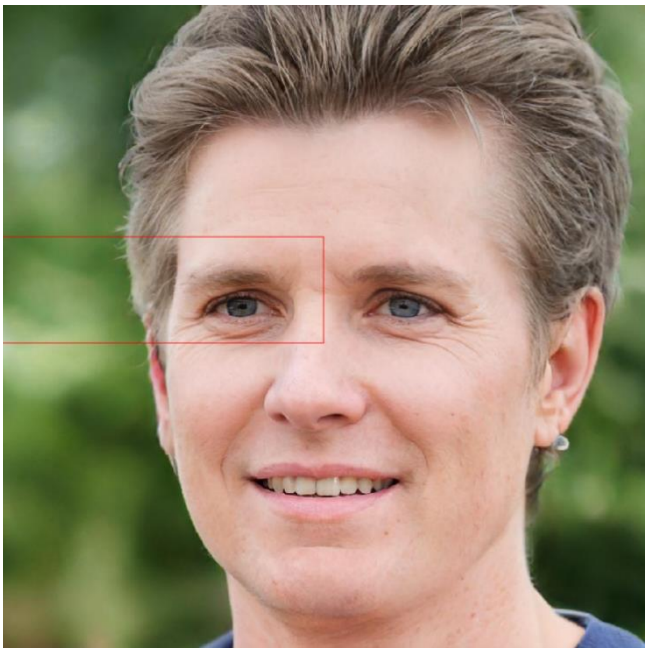


Figura 2.2 – Ejemplo de área prohibida para recorte de trozos de cara sin ojos

Para asegurarnos que el programa si ve un ojo lo detecte como no ojo, hemos creado un algoritmo que realiza el recorte por cada imagen de una persona su ojo izquierdo y derecho y lo hemos añadido a la base de datos de no ojos.

Figura 2.3 – Ejemplo recorte de solo un ojo (izquierdo y derecho)

Por último para tener referencias de espacios comunes que podrían salir en escenas de no ojos hemos seleccionado un conjunto de imágenes extraídas de la siguiente página web: https://search.bwh.harvard.edu/new/CBDatabase.html dónde se recopilan una serie de imágenes de oficinas e interiores.

Este conjunto de fondos contenía repeticiones de la misma imagen con pequeñas variaciones, por lo que creamos un algoritmo en Python que eliminara estas fotos que en nuestro caso hemos considerado repetidas.

Para este último paso, hemos cogido por cada imagen de fondo que teníamos, 18 recortes de esta misma de tamaño 113x513 (alto x ancho). Para evitar repeticiones los puntos de recorte el programa los escoge de manera aleatoria.

Finalmente, obtenemos un total de 11.032 imágenes de no ojos y 220 de ojos, lo que hace una proporción de $50.1\overline{45}$ imágenes de no ojos por cada una de ojos.

A modo de resumen, dejamos especificadas de donde sale cada imagen:

- Imágenes de ojos (220 imágenes)
  - 220 imágenes de caras
- Imágenes de no ojos (11.032 imágenes)
  - $220\ caras\ *\ \frac{20\ trozos}{cara} = 4400\ imágenes\ de\ trozos\ de\ cara$
  - $220\ caras * \frac{2\ ojos}{cara} = 440\ ojos$ (sólo un ojo)
  - $344\ fondos * 18\ recortes = 6192\ imágenes\ de\ fondos$

Dejamos un enlace al Google drive con un .zip para descargar los datasets.

2.3. Train/Test sets

Para seleccionar los conjuntos de imágenes de manera aleatoria hemos creado un script de Python que seleccione las imágenes, las introduzca en una lista y las mezcle de manera aleatoria, posteriormente copia las imágenes que corresponden al train y test de los ojos y no ojos en sus respectivas carpetas para usarlas a continuación.

Finalmente, para que los algoritmos que usaremos vayan más rápido, en concreto para evitar las lecturas por separado de cada imagen que usaremos, creamos una matriz de todas las imágenes para el *train de los ojos*, *test de los ojos*, *train de los no ojos* y *test de los no ojos*.

El porcentaje de fotos que usamos para entrenar los modelos es del 90%, el 10% restante nos lo quedamos para introducir nuevos inputs al probar los modelos.

3. HOGs

Para la entrega final del Short Project hemos decidido utilizar HOGs, para ello, Matlab tiene una función *extractHOGFeatures* que nos permite extraer las características de una imagen y nos la deja el resultado en un vector de características. Una de las opciones que nos deja configurar esta función es el tamaño de las celdas que se usan para recorrer la imagen.

Usamos un algoritmo que extrae estas características de las imágenes que tenemos, en la entrega del Checkpoint probamos dos configuraciones diferentes para observar los resultados del parámetro “CellSize”, primero con la opción por defecto [8 8] y posteriormente la opción [16 16], para esta entrega final, hemos usado sólo la opción de [16 16] dado que nos fue más satisfactoria.

4. LBP

Además de HOGs, para esta entrega final, hemos usado LBP, para ello Matlab tiene una función llamada extractLBPFeatures que nos permite extraer las características, funciona de la misma manera que el caso anterior y hemos seleccionado las opciones del caso anterior.

Hemos seleccionado estas dos features porque al leer documentación sobre diferentes features, hemos visto que estas dos juntas dan buenos resultados.

5. Training

Para este paso, insertamos la tabla creada con las dos features anteriores en la app llamada "Classification Learner" y seleccionamos todos los modelos que hay disponibles en la aplicación. Para que vaya más rápido, nos hemos descargado un Add-on de Matlab, llamada *Parallel Computing Toolbox*, que nos permite paralelizar el entrenamiento de los modelos e ir más rápido en este proceso.

Para todos los modelos usamos el método Cross-Validation. Después de una larga espera, obtenemos los siguientes resultados:

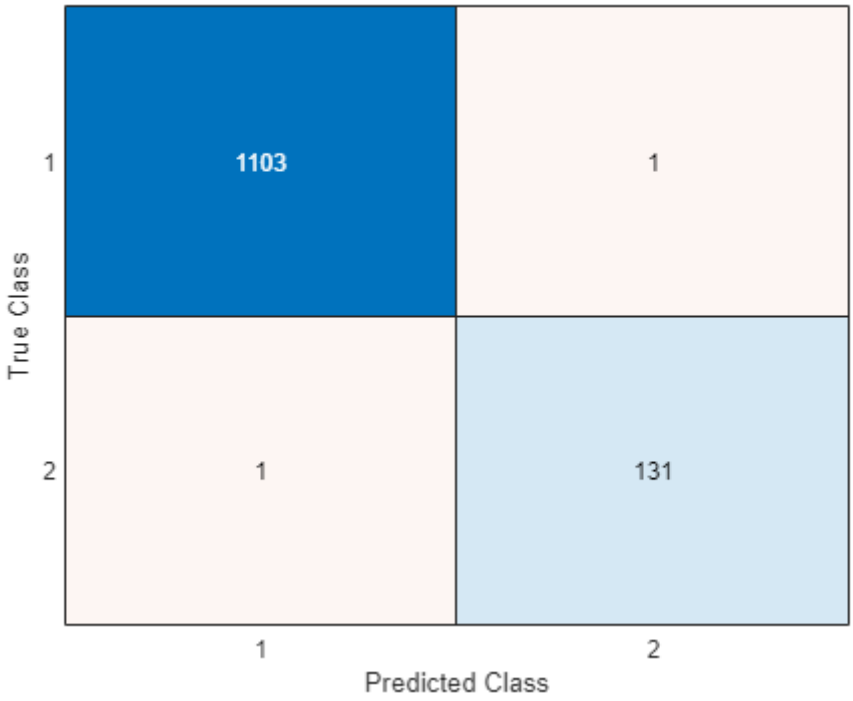
- Fine Tree — Accuracy: 96.2%
- Medium Tree — Accuracy: 96.8%
- Coarse Tree — Accuracy: 95.7%
- Linear Discriminant — Accuracy: 97.3%
- Logistic Regression — Accuracy: 97.0%
- Gaussian Naive Bayes — Accuracy: 94.3%
- Kernel Naive Bayes — Accuracy: 95.6%
- Linear SVM — Accuracy: 99.7%
- Quadratic SVM — Accuracy: 99.8%
- **Cubic SVM** — **Accuracy: 99.9%**
- Fine Gaussian SVM — Accuracy: 89.3%
- Medium Gaussian SVM — Accuracy: 99.9%
- Coarse Gaussian SVM — Accuracy: 99.3%
- Fine KNN — Accuracy: 99.7%
- Medium KNN — Accuracy: 99.1%
- Coarse KNN — Accuracy: 98.0%
- Cosine KNN — Accuracy: 99.6%
- Cubic KNN — Accuracy: 99.0%
- Weighted KNN — Accuracy: 99.3%
- Boosted Trees — Accuracy: 99.1%
- Bagged Trees — Accuracy: 98.3%
- RUSBoosted Trees — Accuracy: 98.6%
- Narrow Neural Network — Accuracy: 99.6%

Obtenemos mejores resultados que en la entrega anterior, el mejor de ellos el Cubic SVM con un 99.9%.

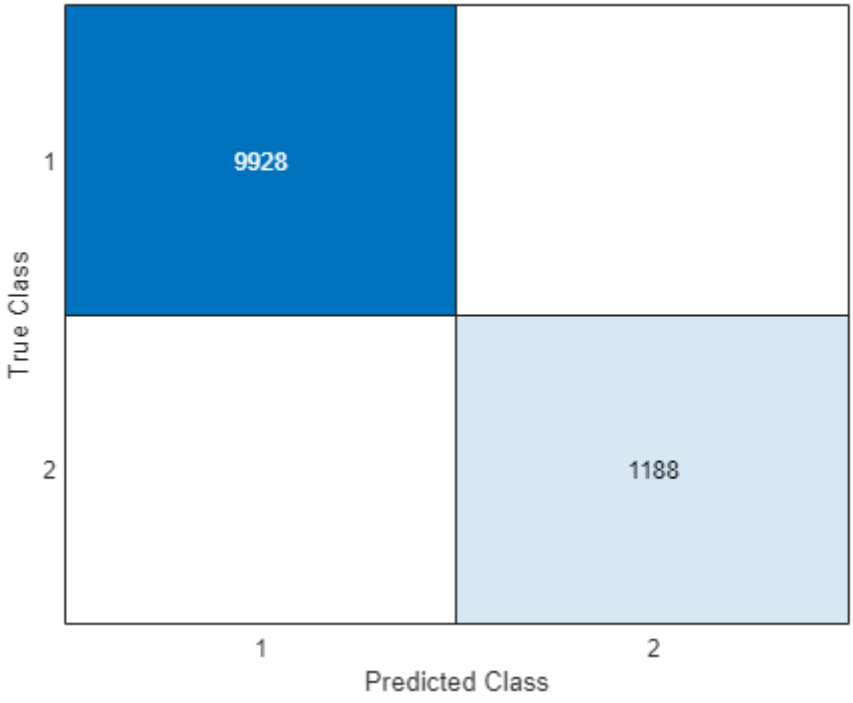
6. Resultados

Para este apartado, exponemos el mejor modelo obtenido, el modelo Cubic SVM, obtenemos los siguientes resultados:

- El resultado que nos devuelve la app usada es:



- Cuando lo usamos con las imágenes de prueba reservadas el resultado es el siguiente:



7. Detección de mirada

Primero, queremos recalcar que intentamos usar la función detectHarrisFeatures, para detectar el lagrimal y la comisura, aunque el lagrimal lo detectaba correctamente, la comisura no, por ello desistimos en usar este método.

De la imagen de ojos obtenida después la detección y recortamos el ojo izquierdo (de la persona).

Figura 7.1 – Recorte de ojo izquierdo

Para detectar el iris usamos la función imfindcircles, obtenemos el siguiente resultado.

Figura 7.2 – Iris detectado

Binarizamos la imagen y hacemos un open para limpiar la imagen.

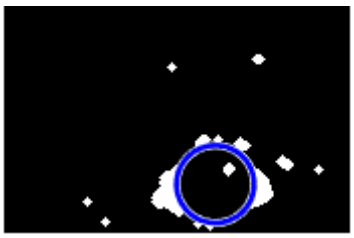


Figura 7.3 – Imagen binarizada

Obtenemos la imagen binarizada del círculo del iris.

Figura 7.4 – Imagen binarizada de sólo el iris

Finalmente fusionamos las dos imágenes anteriores.

Figura 7.5 – Or del iris y la imagen del ojo binarizados

Dilatamos y eliminamos las manchas que no forman parte del ojo.

Figura 7.6 – Resultado de la imagen binarizada.

Finalmente para saber hacia donde esta mirando, buscamos el centro del ojo y el mínimo y el máximo de la imagen binarizada.

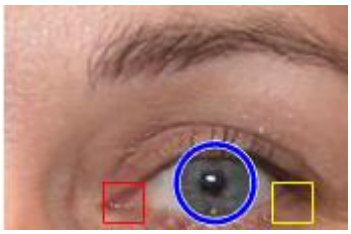


Figura 7.7 – Resultado de la detección final

Y mediante un factor enviamos un mensaje pop-up diciendo hacia donde esta mirando, entre 3 opciones: centro, derecha izquierda.

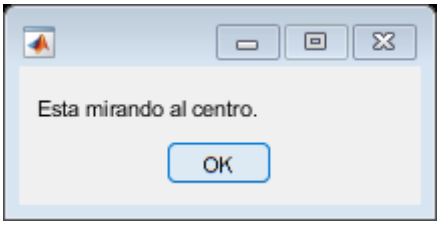


Figura 7.8 – Pop-up

8. Distancia entre cejas

Una vez detectados los ojos, hacemos un recorte usando una proporción.

Figura 8.1 – Recorte para obtener la distancia entre cejas

Binarismos con una condición que sea inferior a 100, ejecutamos un imclose para unir componentes, y encontramos las cejas.

Figura 8.2 – Imagen binarizada

Encontramos un el punto intermedio de la imagen, y empezamos a dilatar el punto hasta encontrar las cejas.

Figura 8.3 – Imdilate hasta encontrar cejas

Reconstruimos la imagen para obtener una imagen binarizada con solo las cejas.

Figura 8.4 – Imagen binarizada de solo cejas

Podemos ver el resultado sobrepuesto a la imagen inicial.

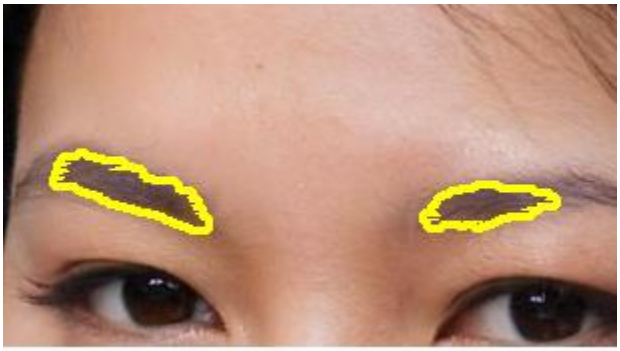


Figura 8.5 – Cejas detectadas

Para encontrar la distancia buscamos el punto máximo de la ceja derecha (ceja derecha de la persona, no según cómo lo vemos), y el punto mínimo de la ceja izquierda para contar el número de pixeles que dista entre ellos. Finalmente nos lanza un pop-up con la distancia.

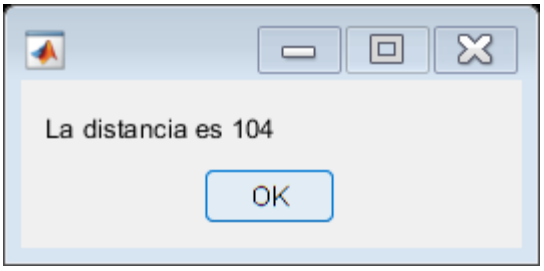


Figura 8.6 – Distancia cejas

9. Cambios entre el CheckPoint y la entrega final

Aquí vamos a hacer un breve resumen de los cambios realizados respecto a la entrega anterior, aunque hemos ido añadiendo los cambios en los apartados correspondientes del documento.

Para la primera entrega sólo implementamos HOG features, para esta segunda hemos añadido también LBP features.

Al hacer el training, obtenemos más modelos satisfactorios y además el resultado es mejor, podemos ver la confusion matrix tanto del train como la de las imágenes de test reservadas en el apartado de resultados.

Para detectar ojos en una imagen, independientemente de dónde estén situados, a diferencia del checkpoint, pasamos un kernel de 513x113 que va deslizándose por toda la imagen viendo donde están situados los ojos.

Hemos añadido la detección de mirada y hemos medido la distancia entre cejas, podemos ver los resultados y el procedimiento en las secciones correspondientes.

Las funciones extras las hemos añadido en el apartado 10 y los códigos usados en la segunda entrega diferentes o añadidos están en el apartado de anexo.

10. Funciones usadas

Funciones que no son nuestras:

- `dir()`
- `length()`
- `height()`
- `width()`
- `size()`
- `floor()`
- `rand()`
- `imwrite()`
- `imread()`
- `imcrop()`
- `rgb2gray()`
- `extractHOGFeatures()`
- `zeros()`
- `ones()`
- `strings()`
- `int2str()`
- `transpose()`
- `array2table()`
- `model.predictFcn()`
- `confusionchart()`

- confusionmat()
- webcam()
- getrect()
- snapshot()
- imshow()
- imresize()
- single()
- tic
- toc

Funciones usadas en la entrega final, a parte de las usadas anteriormente:

- extractLBPFeatures()
- confusionchart()
- confusionmat()
- imdilate()
- strel()
- imoverlay()
- imbinarize()
- bwdist()
- round()
- bwconncomp()
- cellfun()
- max()
- min()
- false()
- sum()
- transpose()
- Imfindcircles()

El resto de las funciones y código es completamente nuestro.

11. Muestra del funcionamiento

Realizamos una fotografía, en nuestro caso, con una webcam.

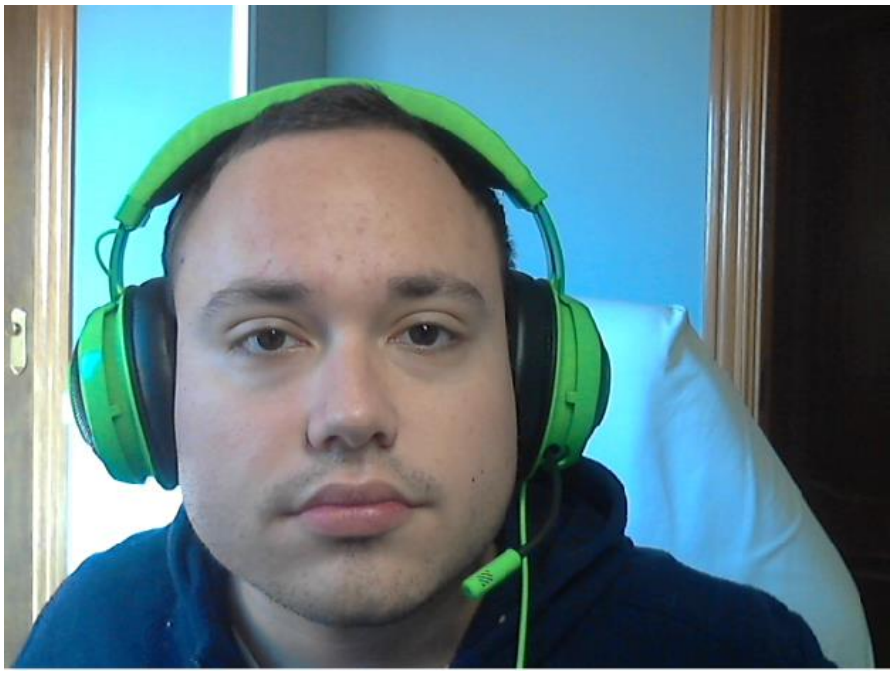


Figura 11.1 – Imagen de prueba

Buscamos la cara, la recortamos y hacemos un resize.

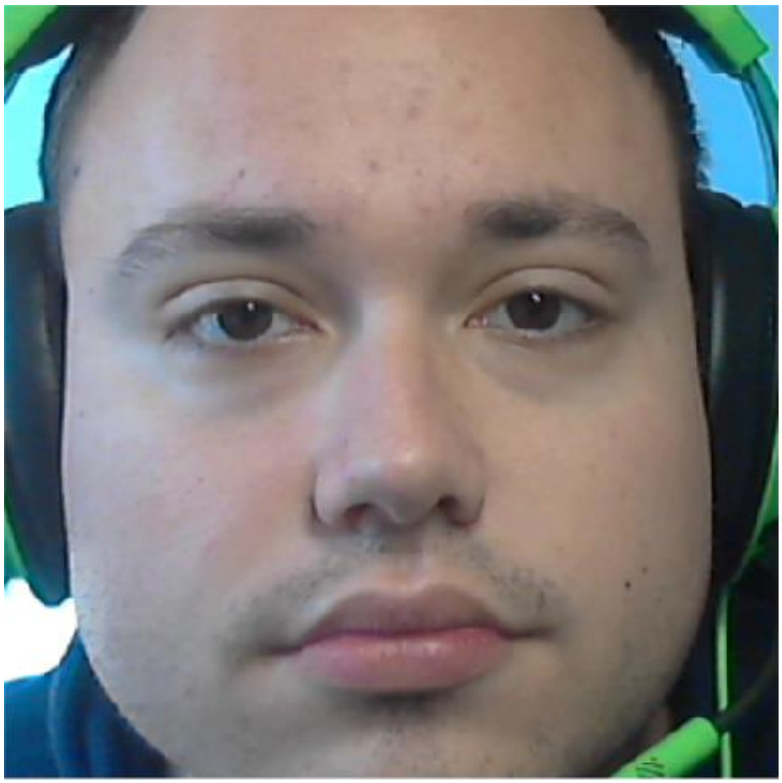


Figura 11.2 – Imagen de la cara

Pasamos la imagen a nivel de grises y ejecutamos nuestro modelo para que encuentre los ojos y hacemos un recorte de ellos.

Figura 11.3 – Recorte de ojos

Cogemos el ojo izquierdo y hacemos un recorte.

Figura 11.4 – Recorte ojo izquierdo

Aplicamos el algoritmo y obtenemos la ubicación del iris, el lagrimal y la comisura.

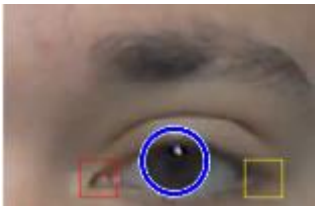


Figura 11.5 – Ubicación del iris, el lagrimal y la comisura

Nos sale una Pop-up de hacia dónde está mirando.

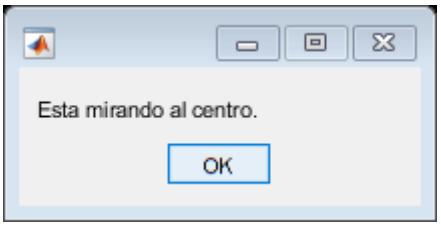


Figura 11.6 – Dirección de la mirada

Hacemos un recorte que englobe las cejas.

Figura 11.7 – Recorte para obtener la distancia entre cejas

Binarizamos la imagen.

Figura 11.8 – Recorte binarizado

Aplicamos el algoritmo de detección de cejas.

Figura 11.9 – Detección de cejas

La distancia entre cejas nos sale como resultado en un pop-up.

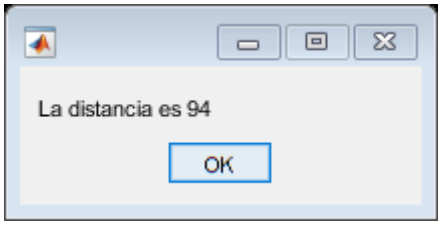


Figura 11.10 – Distancia entre cejas (en píxeles)

12. Enllaç Video Del funcionament

https://drive.google.com/file/d/1FoKJk8EXkVpTVB--8RKg33VVX-7iuPmE/view?usp=sharing

13. Anexo

Programa de recorte de ojos.

```
Imagefiles = dir('VC/Final Project/DataFaces/images_faces/*.jpg');
for I = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    alto = height(image);
    ancho = width(image);
    x = floor(ancho/4);
    y = floor(alto/2.4);
    distance_h = floor(ancho/4 + ancho/2) - x;
    distance_w = floor(alto/1.9) - y;

    imwrite(imcrop(image, [x, y, distance_h, distance_w]), "eye_" + I +
".jpg");
end
```

Programa para recortar diferentes trozos de la cara.

```
% Recogemos los datos de recorte de una imagen de prueba para no hacer los
cálculos en todas las iteraciones.

Image = imread("VC/Final
Project/DataFaces/images_faces/000c3c3ebe3e1c1e.jpg");

alto = height(image);
ancho = width(image);
x = floor(ancho/4);
y = floor(alto/2.4);
distance_x = floor(ancho/4 + ancho/2) - x;
distance_y = floor(alto/1.9) - y;

% Recorremos y recortamos las imágenes
count  = 1;
imagefiles = dir('VC/Final Project/DataFaces/images_faces/*.jpg');
for I = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    prohibited_x_left = 1;
    prohibited_x_right = x + distance_x / 2;
    prohibited_y_top = y - distance_y / 2;
    prohibited_y_bottom = y + distance_y;

    for j = 1:20
        T = false;
        while (~T)
            random_x = rand() * (ancho - 1) + 1;
            random_y = rand() * (alto - 1) + 1;

            cond1 = prohibited_y_top > random_y || random_y >
prohibited_y_bottom;
```

```
            cond2 = prohibited_x_left > random_x || random_x >
prohibited_x_right;
            if (cond1 || cond2)
                h = floor(random_x + distance_x);
                w = floor(random_y + distance_y);
                if (h < alto && w < ancho)
                    T = true;
                end
            end
        end

        imwrite(imcrop(image, [random_x, random_y, distance_x, distance_y]),
"VC/Final Project/DataFaces/not_eyes/not_eye_" + count + ".jpg");
        im = imread("VC/Final Project/DataFaces/not_eyes/not_eye_" + count +
".jpg");
        count = count + 1;
    end
end
```

Algoritmo de recorte de los ojos por separado.

```
Count = 1;
imagefiles = dir('VC/Final Project/DataFaces/images_faces/*.jpg');
for I = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    alto = height(image);
    ancho = width(image);
    x = floor(ancho/4);
    y = floor(alto/2.4);
    distance_x = floor(ancho/4 + ancho/2) - x;
    distance_y = floor(alto/1.9) - y;

    alto = height(image);
    ancho = width(image);
    x = floor(ancho/4);
    y = floor(alto/2.4);
    distance_x = floor(ancho/4 + ancho/2) - x;
    distance_y = floor(alto/1.9) - y;

    imwrite(imcrop(image, [1, y, distance_x, distance_y]), "VC/Final
Project/DataFaces/not_eyes/half_eyes_" + count + ".jpg");
    count = count + 1;
    imwrite(imcrop(image, [distance_x, y, distance_x, distance_y]),
"VC/Final Project/DataFaces/not_eyes/half_eyes_" + count + ".jpg");
    count = count + 1;
end
```

Programa que elimina imágenes repetidas de las imágenes que usamos como fondo.

```
# import required module
import os
import random
```

```
import shutil
# assign directory
# iterate over files in
# that directory
print("hola")
n = int(input("Introduce cada cuantas imágenes quieres borrar\n"))
directory = input("Introduce nombre de la carpeta\n")
donde = input("Donde\n")

lista = os.listdir(directory)
18ítulos18e = len(lista)

lists = list(range(0, n))

print(lista)

print(directory)
print(donde)

for I in range(0, 18ítulos18e, n):
    adder = random.choice(lists)
    indice = I + adder

    original = directory + "\\" + lista[18ítulo]
    target = donde + "\\" + lista[18ítulo]

    print("original: " + original)
    print("copia: " + target)

    shutil.copyfile(original, target)
```

Programa que recorta las imágenes de fondo.

```
Image = imread("VC/Final
Project/DataFaces/images_faces/000c3c3ebe3e1c1e.jpg");

alto = height(image);
ancho = width(image);
x = floor(ancho/4);
y = floor(alto/2.4);
distance_x = floor(ancho/4 + ancho/2) - x;
distance_y = floor(alto/1.9) - y;

count = 1;
imagefiles = dir('VC/Final Project/DataFaces/backgrounds/*.jpg');
for I = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);
    alto = height(image);
    ancho = width(image);
```

```
    for j = 1:18
        T = false;
        while (~T)
            random_x = rand() * (ancho - 1) + 1;
            random_y = rand() * (alto - 1) + 1;
            w = random_x + distance_x;
            h = random_y + distance_y;
            if (w < ancho && h < alto)
                T = true;
            end
        end

        imwrite(imcrop(image, [random_x, random_y, distance_x, distance_y]),
"VC/Final Project/DataFaces/not_eyes/background_" + count + ".jpg");
        count = count + 1;
    end
end
```

Programa que comprueba que todas las imágenes son de tamaño 113x513

```
imagefiles = dir('VC/Final Project/DataFaces/not_eyes/*.jpg');
for I = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    if (height(image) ~= 113 || width(image) ~= 513)
        currentimage
    end
end
```

Programa que mezcla las imágenes para el train y el test de ojos y no ojos aleatoriamente.

```
Import os
import random
import shutil

directory = input("Introduce nombre de la carpeta\n")
donde1 = input("Donde el train\n")
donde2 = input("Donde el test\n")
porcentaje = input("Porcentaje del train (valores de 0 a 1)\n")

lista = os.listdir(directory)
longitud = len(lista)

random.shuffle(lista)
print(lista)
n = 19ítulos19e * float(porcentaje);

for I in range(0, int(n)):
    original = directory + "\\" + lista[i]
    target = donde1 + "\\" + lista[i]
```

```
    print("original: " + original)
    print("copia: " + target)

    shutil.copyfile(original, target)

for I in range(int(n), 20ítulos20e):
    original = directory + "\\" + lista[i]
    target = donde2 + "\\" + lista[i]

    print("original: " + original)
    print("copia: " + target)

    20ítul.copyfile(original, target)
```

Algoritmo que pasa las imágenes a una matriz.

```
Imagefiles = dir('eyes_test/*.jpg');
for I = 1:length(imagefiles)
    currentimage = imagefiles(i).name;
    image = imread(currentimage);

    EyesTestData(:, :, :, i) = image;
end
```

Algoritmo que usamos para extraer el vector de características.

```
Load EyesTrainData.mat
load NotEyesTrainData.mat

image = EyesTrainData(:, :, :, 1);
imageGray = rgb2gray(image);
cell = [16 16];
hog = extractHOGFeatures(imageGray, 'CellSize', cell);

mida_eyes = size(EyesTrainData);
train_data_eyes = zeros(mida_eyes(4), width(hog));
for i=1:mida_eyes(4)
    image = EyesTrainData(:, :, :, i);
    imageGray = rgb2gray(image);
    train_data_eyes(I, 😊 = extractHOGFeatures(imageGray, 'CellSize', cell);
end

mida_not_eyes = size(NotEyesTrainData);
train_data_not_eyes = zeros(mida_not_eyes(4), width(hog));
for i=1:mida_not_eyes(4)
    if (mod(I, 500) == 0)
        i
    end
    image = NotEyesTrainData(:, :, :, i);
    imageGray = rgb2gray(image);
    train_data_not_eyes(I, 😊 = extractHOGFeatures(imageGray, 'CellSize',
cell);
end

train_data = [train_data_eyes, ones(mida_eyes(4), 1); train_data_not_eyes,
zeros(mida_not_eyes(4), 1)];
```

```
21ítulos = strings(width(hog), 1);
for I = 1:width(hog)
    21ítulos(i) = “Atr “ + int2str(i);
end

21ítulos = [21ítulos; “Clase”];
21ítulos = transpose(21ítulos);


T = array2table(train_data, ‘VariableNames’, 21ítulos);
```

Algoritmo para comprobar los modelos.

```
Load EyesTestData.mat
load NotEyesTestData.mat

load CosineKNNModelKernel16.mat

cell = [16 16]

image = EyesTestData(:, :, :, 1);
imageGray = rgb2gray(image);
hog = extractHOGFeatures(imageGray, ‘CellSize’, cell);

mida_eyes = size(EyesTestData);
test_data_eyes = zeros(mida_eyes(4), width(hog));
for i=1:mida_eyes(4)
    image = EyesTestData(:, :, :, i);
    imageGray = rgb2gray(image);
    test_data_eyes(I, 😊 = extractHOGFeatures(imageGray, ‘CellSize’, cell);
end


mida_not_eyes = size(NotEyesTestData);
test_data_not_eyes = zeros(mida_not_eyes(4), width(hog));
for i=1:mida_not_eyes(4)
    if (mod(I, 500) == 0)
        i
    end
    image = NotEyesTestData(:, :, :, i);
    imageGray = rgb2gray(image);
    test_data_not_eyes(I, 😊 = extractHOGFeatures(imageGray, ‘CellSize’,
cell);
end

test_data = [test_data_eyes, ones(mida_eyes(4), 1); test_data_not_eyes,
zeros(mida_not_eyes(4), 1)];
T = array2table(test_data);

21ítulos = strings(width(hog), 1);
for I = 1: width(hog)
    21ítulos(i) = “Atr “ + int2str(i);
end

21ítulos = [21ítulos; “Clase”];
```

```
T.Properties.VariableNames = 22ítulos;

should = T.Clase;
yfit = CosineKNNModelKernel16.predictFcn(T);

confusionchart(confusionmat(should, yfit))
```

Programa de detección de ojos.

```
Load CosineKNNModelKernel16.mat

webcamlist;
cam = webcam(1);


imagenOriginal = snapshot(cam);

imshow(imagenOriginal)
rect = getrect()

imagenSinRecortar = imagenOriginal(rect(2):rect(2)+rect(4),
rect(1):rect(1)+rect(3), 😉;

imshow(imagenSinRecortar)

imagenRecortada = imresize(imagenOriginal(rect(2):rect(2)+rect(4),
rect(1):rect(1)+rect(3), 😉, [113 513]);
imshow(imagenRecortada)

ejecutarPredictor(imagenRecortada, CosineKNNModelKernel16)
```

Función ejecutarPredictor()

```
function[prediction] = ejecutarPredictor(imagen, clasificador)

    imagenGris = rgb2gray(imagen);

    imshow(imagenGris)

    HOGFeatures = extractHOGFeatures(imagenGris, 'CellSize', [16 16]);
    HOGFeatures = [HOGFeatures 1];

    22ítulos = strings(width(HOGFeatures), 1);
    for I = 1: width(HOGFeatures)
        22ítulos(i) = "Atr " + int2str(i);
    end

    22ítulos(width(HOGFeatures)) = "Clase";

    tabla = array2table(HOGFeatures);
    tabla.Properties.VariableNames = 22ítulos;

    prediction = single(clasificador.predictFcn(tabla));
end
```

Códigos extra entrega final

Código para obtener las confusion matrix.

```
load EyesTestData.mat
load NotEyesTestData.mat
load EyesTrainData.mat
load NotEyesTrainData.mat

load cuvicSVM.mat

modelo = cubicSVM;

cell = [16 16];

image = EyesTestData(:, :, :, 1);
imageGray = rgb2gray(image);
hog = extractHOGFeatures(imageGray, 'CellSize', cell);

mida_eyes = size(EyesTestData);
test_data_eyes = zeros(mida_eyes(4), width(hog));
for i=1:mida_eyes(4)
    image = EyesTestData(:, :, :, i);
    imageGray = rgb2gray(image);
    test_data_eyes(i, :) = extractHOGFeatures(imageGray, 'CellSize', cell);
end

mida_not_eyes = size(NotEyesTestData);
test_data_not_eyes = zeros(mida_not_eyes(4), width(hog));
for i=1:mida_not_eyes(4)
    if (mod(i, 500) == 0)
        i
    end
    image = NotEyesTestData(:, :, :, i);
    imageGray = rgb2gray(image);
    test_data_not_eyes(i,  :)  =  extractHOGFeatures(imageGray,  'CellSize',
cell);
end

test_data  =  [test_data_eyes,  ones(mida_eyes(4),  1);  test_data_not_eyes,
zeros(mida_not_eyes(4), 1)];
T = array2table(test_data);

titulos = strings(width(hog), 1);
for i = 1: width(hog)
    titulos(i) = "Atr " + int2str(i);
end

titulos = [titulos; "Clase"];

T.Properties.VariableNames = titulos;

should = T.Clase;
yfit = modelo.predictFcn(T);

cmatrix = confusionchart(confusionmat(should, yfit))

image = EyesTrainData(:, :, :, 1);
imageGray = rgb2gray(image);
```

```
hog = extractHOGFeatures(imageGray, 'CellSize', cell);

mida_eyes = size(EyesTrainData);
train_data_eyes = zeros(mida_eyes(4), width(hog));
for i=1:mida_eyes(4)
    image = EyesTrainData(:, :, :, i);
    imageGray = rgb2gray(image);
    train_data_eyes(i, :) = extractHOGFeatures(imageGray, 'CellSize', cell);
end

mida_not_eyes = size(NotEyesTrainData);
train_data_not_eyes = zeros(mida_not_eyes(4), width(hog));
for i=1:mida_not_eyes(4)
    if (mod(i, 500) == 0)
        i
    end
    image = NotEyesTrainData(:, :, :, i);
    imageGray = rgb2gray(image);
    train_data_not_eyes(i,  :)  =  extractHOGFeatures(imageGray,  'CellSize',
cell);
end
train_data  =  [train_data_eyes,  ones(mida_eyes(4),  1);  train_data_not_eyes,
zeros(mida_not_eyes(4), 1)];
T = array2table(train_data);

titulos = strings(width(hog), 1);
for i = 1: width(hog)
    titulos(i) = "Atr " + int2str(i);
end

titulos = [titulos; "Clase"];

T.Properties.VariableNames = titulos;

should = T.Clase;
yfit = modelo.predictFcn(T);

cmatrix = confusionchart(confusionmat(should, yfit))
```

Código de detección de ojos

```
clear
close all
clf

load cuvicSVM.mat
modelo = cubicSVM;

% webcamlist;
% cam = webcam(1);
% imagen = snapshot(cam);

imagen = imread('00183c1c3c3c3c7f.jpg');
imshow(imagen)
[cara, bbox] = searchFace(imagen);
imshow(cara)
resizear = imresize(cara, 513/width(cara));
imshow(resizear)
```

```
hayOjos = false;
inicio = 0;
soloOjos = [];
masOjos = [];

for i = 1:(round(113/2) - 1):(height(resizear)-113)
    trozo = resizear(i:(i+112), :, :);
    imshow(trozo)

    [score, label] = ejecutarPredictor(trozo, modelo);
    if (score == 1)
        masOjos = resizear((i - 60):(i+112), :, :);
        hayOjos = true;
        inicio = i;
        soloOjos = trozo;
        break;
    end
end
if hayOjos == true
    factorInverso = width(cara)/513;
    rectangulo  =[bbox(1),  bbox(2)  +  round(i*factorInverso),  round(513  *
factorInverso), round(113 * factorInverso)];

    outImg = imcrop(imagen, rectangulo);
    imshow(outImg)

    marca = false([height(imagen), width(imagen)]);
    marca(rectangulo(2):(rectangulo(2)            +            rectangulo(4)),
rectangulo(1):(rectangulo(1) + rectangulo(3)) , :) = true;
    marcaGrande = imdilate(marca, strel('square', 10));
    marcaFinal = marcaGrande - marca;

    imagenMarcada = imoverlay(imagen, marcaFinal, 'red');

    imshow(imagenMarcada)
end
```

Código de detección de mirada

```
I = soloOjos;
mitad = imcrop(I, [width(I)/2, 0, round(width(I)/3), height(I)]);
%mitad = imcrop(I, [0, 0, round(width(I)/2), height(I)]);
imshow(mitad)

Rmin = 12;
Rmax = 50;
[center, radii] = imfindcircles(mitad,[Rmin Rmax],'ObjectPolarity','dark');
center = center(1,:);
radii = radii(1);

binarizada = imbinarize(rgb2gray(mitad), "adaptive");
circulito = false([height(mitad) width(mitad)]);
R = radii - 2.5;
circulito(round(center(2)), round(center(1))) = 1;
soloCirculo = bwdist(circulito) <= R;
imshow(soloCirculo)
temp4 = temp1 | soloCirculo;
imshow(temp4)
temp5 = imdilate(temp4, [1 1 1 1 1 1]);
```

```
imshow(temp5)
componentes = bwconncomp(temp5);
limpieza = imdilate((binarizada | soloCirculo), [1 1 1 1 1 1]);
componentes = bwconncomp(limpieza);
numPixels = cellfun(@numel,componentes.PixelIdxList);
[biggest,idx] = max(numPixels);

canvasVacio = false([height(binarizada) width(binarizada)]);
canvasVacio(componentes.PixelIdxList{idx}) = 1;
% imshow(canvasVacio)
mirar = imoverlay(mitad, canvasVacio);
imshow(mirar)
puntos = componentes.PixelIdxList{idx};
sumita = sum(canvasVacio);
imin = 1;
found = false;
tamano = size(sumita);
while (imin < tamano(2))
    if (sumita(imin) > 0)
        break
    end
    imin = imin + 1;
end

imax = tamano(2);
while (imax > 0)
    if (sumita(imax) > 0)
        break
    end
    imax = imax - 1;
end

sumitaAlReves = sum(transpose(canvasVacio));
tamano = size(sumitaAlReves);
jmin = 1;
while (jmin < tamano(2))
    if (sumitaAlReves(jmin) > 0)
        break
    end
    jmin = jmin + 1;
end

jmax = tamano(2);
while (jmax > 0)
    if (sumitaAlReves(jmax) > 0)
        break
    end
    jmax = jmax - 1;
end

media = fix((jmax + jmin) / 2);
%plot(rightPoint(1), rightPoint(2), 's', 'Color', 'r', MarkerSize=20);
%plot(leftPoint(1), leftPoint(2), 's', 'Color', 'r', MarkerSize=20);
hold on;
imshow(mitad);
imin = imin;
imax = imax - 5;
medio = round(center(2)) + 10;
viscircles(center, radii,'Color','b');
```

```
plot(imin,medio , 's','MarkerSize', 20 ,'Color','r');
plot(imax, medio, 's','MarkerSize', 20 ,'Color','yellow');
hold off;
d1 = center(1) - imin;
d2 = imax - center(1);
factorizamos = d1/d2;

if (factorizamos < 0.8)
    msgbox("Esta mirando a la izquierda");
elseif(factorizamos > 1.2)
    msgbox("Esta mirando a la derecha");
else
    msgbox("Esta mirando al centro.")
end

To have a function that after detecting the center of the eye for right eye.
Detect lagrima first. Lagrima is leftmost point not too far away from iris.

function pointsOfInterest = findPoints(center, radii, corners)
    strongCoord = corners.selectStrongest(50).Location;
    i=1;
    %can add for loop if several centers to test
    radii(i)
    center(i,:)

    reachable  =  strongCoord(abs(center(i:1)-strongCoord(:,1))<3*radii(i),:);
%filter points with x coord further away than x*radii

    [~,leftPointIndex] = min(reachable(:,1));
    leftPoint = reachable(leftPointIndex,:);

    [~,rightPointIndex] = max(reachable(:,1));
    rightPoint = reachable(rightPointIndex,:);

    pointsOfInterest = [leftPoint;
                        center(i,:);
                        rightPoint];

end
```

Código de que calcula la distancia entre cejas:

```
I = masOjos;
iniciox = 1;
inicioy = round(2*width(I)/11);

I = I(iniciox:height(I), inicioy:round(7*width(I)/9), :);
grises = rgb2gray(I);
hold on;
imshow(grises);
hold off;
binarizada = grises <100;
cerrada = imclose(binarizada, [1 1 1 1 1 1 1 1 1]);
imshow(cerrada);
hold on;

iniciox = round(width(I)/2 + center(1));
inicioy = round(60 + center(2) - 30);
plot(iniciox, inicioy, 'o');
```

```
hold off;
% primeraCeja = round(inicioy);
% while (primeraCeja > 0)
%     if (cerrada(primeraCeja, iniciox) == 1)
%         break;
%     end
%     primeraCeja = primeraCeja - 1;
% end
%
% cejaIzquierda = [iniciox primeraCeja];
%
% segundaCeja = cejaIzquierda(1) - 50;
% while (segundaCeja > 0)
%     if (cerrada(primeraCeja, segundaCeja) == 1)
%         break;
%     end
%     segundaCeja = segundaCeja - 1;
% end

canvas = false([height(I) width(I)]);
canvas(round(height(canvas)/2), round(width(canvas)/2)) = 1;
imshow(canvas);
canvas = imdilate(canvas, strel('rectangle', [30 200]));
imshow(imoverlay(I, canvas));
reconstruccion = imopen(canvas & cerrada, strel('square', 4));
imshow(reconstruccion);
i = 0;
while(i < 100)
    temporal = imdilate(reconstruccion, strel('square', 5));
    reconstruccion = temporal & cerrada;
    i = i + 1;
end
imshow(reconstruccion);
marcas = imdilate(reconstruccion, strel('disk', 5)) - reconstruccion;
imshow(imoverlay(I, marcas));
jmin = round(width(I)/2);
while (jmin > 0)
    if (sum(marcas(:, jmin)) > 1)
        break;
    end
    jmin = jmin - 1;
end

jmax = round(width(I)/2);
while (jmax < width(I))
    if (sum(marcas(:, jmax)) > 1)
        break;
    end
    jmax = jmax + 1;
end

distancia = jmax - jmin;

msgbox("La distancia es " + int2str(distancia));
```

Código que ejecuta un video detectando el iris, el lagrimal y la comisura

```
% Create a cascade detector object.
Detector = vision.CascadeObjectDetector('FrontalFaceLBP');

% Read a video frame and run the face detector.
videoReader = VideoReader('Pablo.mp4');

set(gca,'nextplot','replacechildren');
video = VideoWriter('video3.avi');
open(video);
i = 0;
while hasFrame(videoReader)
    % get the next frame
    videoFrame = readFrame(videoReader);
    bbox = step(Detector, videoFrame);

    % Draw the returned bounding box around the detected face.

    %videoFrame = insertShape(videoFrame, 'Rectangle', bbox);

    imagen = videoFrame;
    if mod(i, 5) == 0
        [soloOjos,   ~,   caraMarcada,   inicioHorizontal,   inicioVertical,
factorInverso, ~, ~, ~, encontrado1] = caras(imagen, modelo);
        if encontrado1 == true
    %     imshow(soloOjos);
    %     imshow(masOjos);
    %     imshow(caraMarcada);
    %     hold on;
    %           plot(finalHorizontal  +  inicioHorizontal,  finalVertical  +
inicioVertical, 'o');
    %     hold off;
            [imin,  imax,  center,  radii,  ~,  ~,  medio,  encontrado]  =
lagrimal(soloOjos);
            inicioHorizontal2 = round(width(soloOjos)/2) * factorInverso;
            inicioVertical2 = 0;
            if encontrado == true
                imshow(caraMarcada);
                center(1) = (center(1) * factorInverso) + inicioHorizontal +
inicioHorizontal2;
                center(2) = (center(2) * factorInverso) + inicioVertical +
inicioVertical2;
                hold on;
                radii(1) = radii(1) * 0.5;
                viscircles(center, radii,'Color','b');
                plot(inicioHorizontal   +   inicioHorizontal2   +   (imin   *
factorInverso), (medio * factorInverso) + inicioVertical + inicioVertical2 ,
's','MarkerSize', 10 ,'Color','r');
                plot(inicioHorizontal   +   inicioHorizontal2   +   (imax   *
factorInverso), (medio * factorInverso) + inicioVertical + inicioVertical2,
's','MarkerSize', 10 ,'Color','yellow');
                hold off;
            end
        end
        pause(0.9);

        frame = getframe(gcf);
        writeVideo(video, frame);
```

```
    end
    i = i + 1;
end
```