

Desafío 11

Mikel Dalmau

7 de Enero de 2019

Índice

1	Enunciado	1
2	Clasificador en cascada	2
3	Transformaciones Morfológicas	2
4	Transformada de Hough Circular	5
5	Resultados	5
6	Código completo	6

1 Enunciado

El objetivo de este desafío es realizar una aplicación que detecte la pupila del ojo en vídeo de tiempo real utilizando la API de OpenCV.

El proceso propuesto, asumiendo que la imagen contiene uno de los ojos completamente, aplicar el detector de ojos basado en clasificadores en cascada proporcionado por OpenCV (tutorial-99). Este detector proporciona una región de la imagen conteniendo el ojo. Aplicar la transformada de Hough circular para encontrar la pupila, posiblemente tras procesos de extracción de bordes (tutorial-19) y eliminación de ruido usando procesos morfológicos (tutorial-17).

Resultado del trabajo debería ser un vídeo anotado con las detecciones y la secuencia temporal de los radios de los círculos detectados.

2 Clasificador en cascada

En la aplicación utilizo dos detectores basados en clasificadores en cascada exactamente igual que en el tutorial-99 proporcionado por OpenCV.

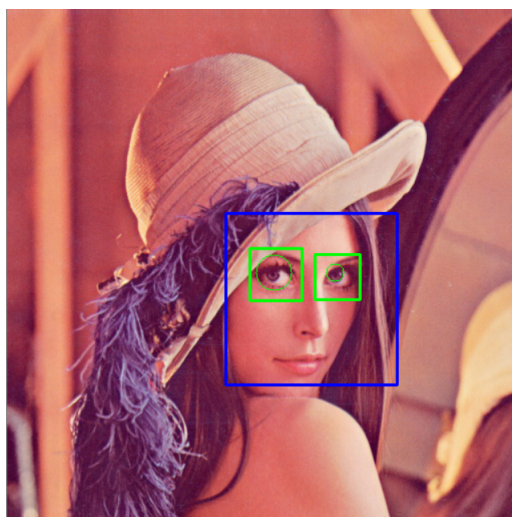


Figura 1: Resultado aplicación de detectores tutorial-99

Con esto, logro las regiones de interés de ambos ojos y así puedo aplicar la transformada de Hough sobre un espacio mucho más localizado, mejorando la eficiencia y resultados.

```
# Para cada cara hallada, extraer la region de interes
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = frame[y:y+h, x:x+w]

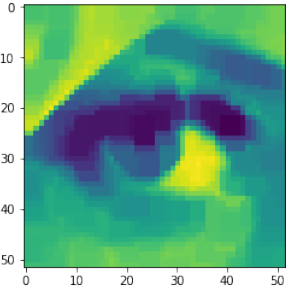
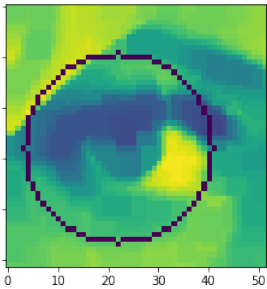
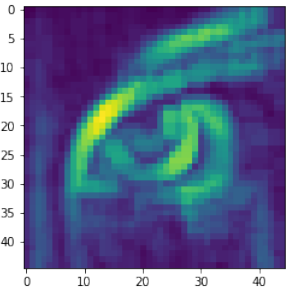
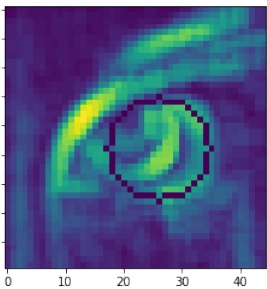
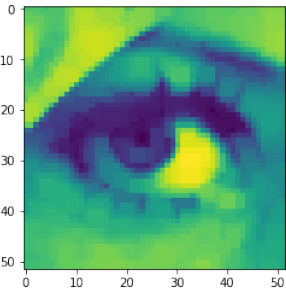
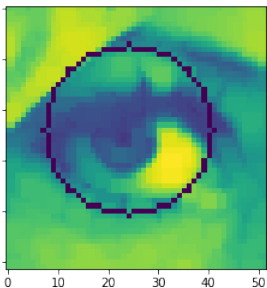
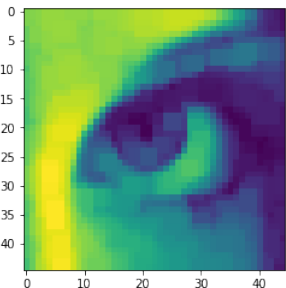
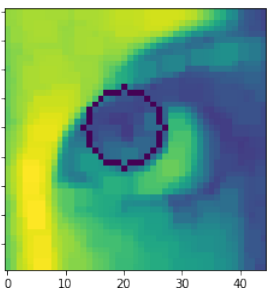
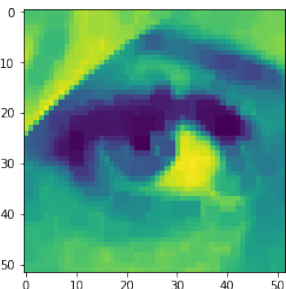
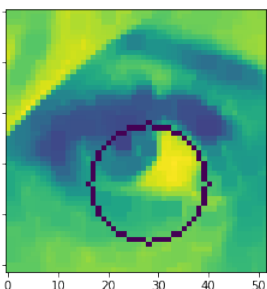
# Para cada ojo, extraer la region de interes
eyes = eye_cascade.detectMultiScale(roi_gray)
for (ex,ey,ew,eh) in eyes:
    roi_eyes = roi_color[ey:ey+eh, ex:ex+ew]
    roi_eyes_gray = roi_gray[ey:ey+eh, ex:ex+ew]
```

3 Transformaciones Morfológicas

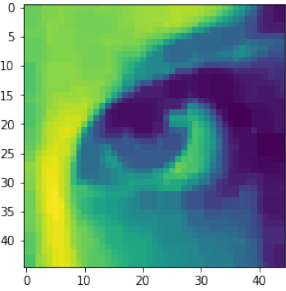
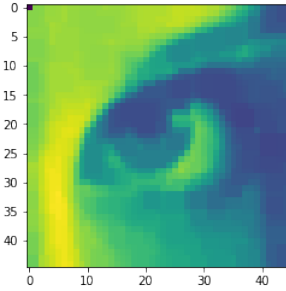
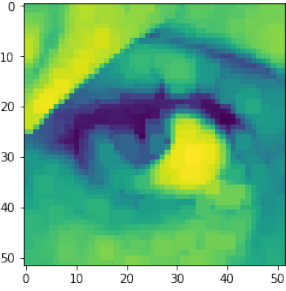
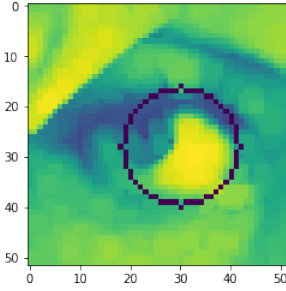
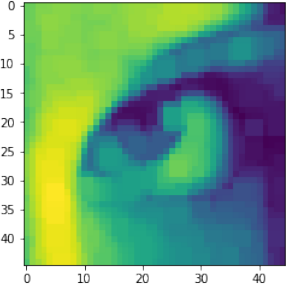
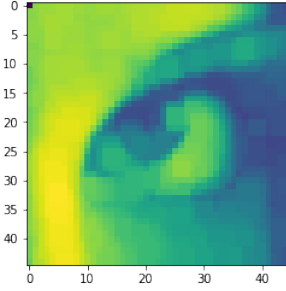
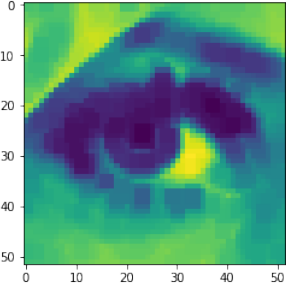
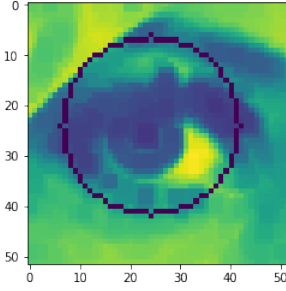
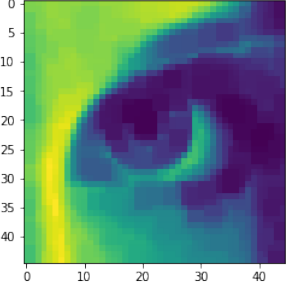
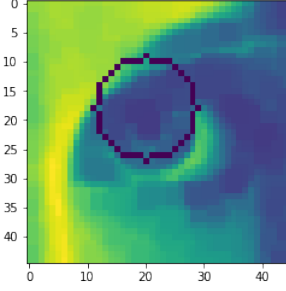
En las siguientes páginas se muestran los resultados de aplicar distintas transformaciones morfológicas a los bordes de dos tipos distintos de ojos.

Aunque no he tenido tanto en cuenta estos resultados, me he decidido por utilizar la transformación Opening o Apertura que es una combinación de una erosión seguida de una dilatación y es útil para eliminar el ruido de las imágenes ampliando también el blanco del ojo en la dilatación y por lo tanto aislando la pupila.



Morfología	Kernel	Resultado	Transformada de Hough
closing	4x4		
gradient	4x4		
open	3x3		
open	3x3		
close	3x3		



Morfología	Kernel	Resultado	Transformada de Hough
close	3x3		
dilate	3x3		
dilate	3x3		
erode	3x3		
erode	3x3		



4 Transformada de Hough Circular

Finalmente, para detectar la pupila utilizo la transformada de Hough circular y para ello me valgo de la siguiente función.

Cabe destacar de esta función los parámetros param1 y param2, ya que internamente HoughCircles llama a la función Canny del tutorial 19 para la extracción de bordes y es con estos parametros que controlamos esta función. Sin entrar más en ello e determinado por prueba y error unos parámetros que resultaban aceptables.

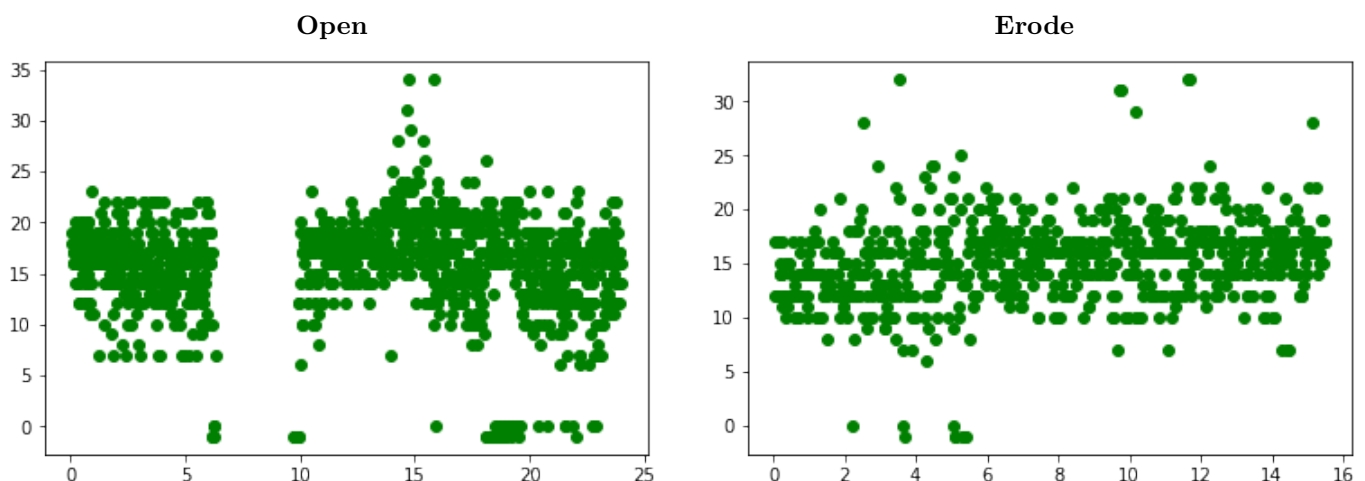
```
# aplicar una transformacion morfologica y la transformada de Hough Circular
roi_eyes_gray = cv2.morphologyEx(roi_eyes_gray, cv2.MORPH_OPEN, kernel)
circles = cv2.HoughCircles(roi_eyes_gray, cv2.HOUGH_GRADIENT, 1, 30, param1=100,
                           param2=20, minRadius=5, maxRadius=50)
```

5 Resultados

Finalmente tras solucionar los problemas relacionados con los códecs de vídeos y otros detalles he creado un par de vídeos con la detección de pupilas en tiempo real, se llaman desafío11-erode y desafío11-open y en cada uno se ha utilizado una de las transformaciones morfológicas.

En la siguiente tabla se muestran los radios obtenidos en cada momento del vídeo, en caso de detectarse un ojo pero no la pupila, el radio es asignado el valor -1, en caso de no detectarse nada no se asigna valor como puede apreciarse por el bloque blanco en una de las imágenes, ya que giro la cabeza en medio del vídeo.

Por otro lado, en el primer gráfico, se trata de un vídeo en el que miro en distintas direcciones lo que explica las variaciones en el gráfico pero la media de los radios es similar en ambos y mayor que el tamaño original de la pupila.





6 Código completo

```
import cv2
import numpy as np
import time
from matplotlib import pyplot as plt

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
kernel = np.ones((2,2),np.uint8)

c = cv2.VideoCapture(0)
_,f = c.read()
out = cv2.VideoWriter('result.avi', cv2.VideoWriter_fourcc(*'XVID'), 20.0, (f.shape[1],f.shape[0]),1)

radios = []
tiempo = []
start = time.time()

# Procesamos cada frame del video
while(c.isOpened()):
    ret, frame = c.read()
    if ret==True:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Para cada cara hallada, extraer la region de interes
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        for (x,y,w,h) in faces:
            roi_gray = gray[y:y+h, x:x+w]
            roi_color = frame[y:y+h, x:x+w]

            # Para cada ojo, extraer la region de interes
            eyes = eye_cascade.detectMultiScale(roi_gray)
            for (ex,ey,ew,eh) in eyes:
                roi_eyes = roi_color[ey:ey+eh, ex:ex+ew]
                roi_eyes_gray = roi_gray[ey:ey+eh, ex:ex+ew]

                # aplicar una transformacion morfologica y la transformada de Hough Circular
                roi_eyes_gray = cv2.morphologyEx(roi_eyes_gray, cv2.MORPH_ERODE, kernel)
                circles = cv2.HoughCircles(roi_eyes_gray,cv2.HOUGH_GRADIENT,1,30,param1=100,
                                                param2=20,minRadius=5,maxRadius=50)

                # Dibujar cada circulo hallado
                if circles is not None:
                    circles = np.uint16(np.around(circles))
                    for i in circles[0,:]:
                        cv2.circle(roi_eyes,(i[0],i[1]),i[2],(0,255,0),1)
                        radios.append(i[2])
                else:
                    radios.append(-1)

            # Tomar el tiempo
            tiempo.append(time.time() - start)

        # escribir el frame
        cv2.imshow('frame',frame)
        try:
            out.write(frame.astype('uint8'))
        except:
            print("Error: video frame did not write")

        if 0xFF & cv2.waitKey(5) == 27:
            break
    else:
```



```
        break

c.release()
out.release()
cv2.destroyAllWindows()

plt.plot(tiempo, radios, 'go')
```

Bibliografía

[Opencv, Hough Circle Transform]

[Opencv, Hough Line Transform]

[StackOverflow] [pencv-video-saving-in-python](#)