

EasYE TRACKING

Giada Colella

Chiara Coletti

15/07/2016

Indice

Sommario

L' Eye tracking è un complesso processo di misurazione del punto di fissazione oculare. I vantaggi che derivano dall'utilizzo di questa tecnica sono molteplici: a partire da indagini statistiche di mercato fino al miglioramento della comunicazione uomo/macchina.

1 Introduzione

Il nostro progetto si pone nella prima fase di realizzazione di un eye tracker. Ha come scopo lo studio del movimento dell'occhio. Partendo da un video, abbiamo cercato di individuare l'occhio e di tracciarne il movimento. Punto focale della nostra ricerca è stato il riconoscimento del centro dell'iride.

Abbiamo sviluppato il nostro progetto usando **"Python"** come linguaggio di programmazione. Per iniziare è necessario aver installato la libreria libera Opencv. Il riconoscimento di volti ed occhi è stato realizzato mediante l'algoritmo "Cascade Classification". Prima di operare, dunque, è stato necessario importare i file classificatori (o "haarcascades") nella cartella di lavoro.

2 Dallo Pseudocodice...

Abbiamo ritenuto utile suddividere il nostro progetto in vari step:

1. Acquisizione video e analisi frame by frame
2. Conversione dei frame acquisiti in scala di grigi
3. Riconoscimento del volto
4. Riconoscimento degli occhi
5. Riconoscimento cerchi nell'area occhi
6. Rappresentazione del movimento oculare
7. Generazione video e chiusura programma

3 ...Al codice

```
1 %matplotlib inline
import matplotlib.pyplot as plt
3 import cv2
import numpy as np
5
sc_fct = 1.1
7 min_neigh = 5
min_size_f = (30,30)
9 min_size_e = (10,10)
mindist_c=30
11 dp_c=1
param1_c=60
13 param2_c=20
minr_c=5
15 maxr_c=15
17
faceCascade = cv2.CascadeClassifier("
    haarcascade_frontalface_default.xml")
19 eyeCascade=cv2.CascadeClassifier("haarcascade_eye.xml")
```

Importiamo le librerie oportune, introduciamo i parametri che ci torneranno utili nella stesura del codice ed importiamo le Haarcascades che userem per il riconoscimento di volto e occhi.

?? **Acquisizione video e analisi frame by frame**

```
1 video_capture = cv2.VideoCapture(0)
cv2.namedWindow("Face_and_eyes")
3
while True:
5     ret, frame = video_capture.read()
```

Abbiamo importato il video dalla webcam, rinominato la finestra video e salvato in "frame" gli elementi su cui poi andremo a lavorare. L'ultima operazione viene inserita all'interno di un ciclo while che verrà interrotto solo al termine dell'esecuzione dell'intero programma.

```
1 x_dim=video_capture.get(3)
  y_dim=video_capture.get(4)
```

Definisco x_dim e y_dim come le dimensioni del video in entrata. Saranno utili in seguito.

?? Conversione dei frame acquisiti in scala di grigi

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Applichiamo la funzione 'cvtColor' specificando il tipo di conversione da eseguire (in questo caso convertiamo da una figura a colori ad una a scala di grigi mediante il comando cv2.COLOR_BGR2GRAY). La funzione viene applicata a 'frame' ovvero la zona di memoria in cui vengono salvati i singoli frame del video.

?? Riconoscimento del volto

```

1     faces = faceCascade.detectMultiScale(
        gray,
3         scaleFactor=sc_fct,
        minNeighbors=min_neigh,
5         minSize=min_size_f
        )

```

Applichiamo la funzione 'detectMultiscale' e utilizziamo le cascaspecificando il tipo di conversione da eseguire (in questo caso convertiamo da una figura a colori ad una a scala di grigi mediante il comando cv2.COLOR_BGR2GRAY). La funzione viene applicata a 'gray' ovvero la zona di memoria in cui abbiamo salvato i risultati della conversione dei vari frame in scala di grigi. Salviamo l'output della funzione (una lista di quattro elementi: coordinata x e coordinata y del vertice superiore sinistro del rettangolo, base e altezza) in "faces".

```

2     for (x, y, w, h) in faces:
        cv2.rectangle(frame,(x, y), (x+w, y+h), (0, 255, 0), 2)

```

Tracciamo i rettangoli attorno ai volti riconosciuti mediante la funzione 'rectangle'.

?? Riconoscimento degli occhi

```

2     roi_gray=gray[y:y+h/2,x:x+w]
        roi_color=frame[y:y+h/2,x:x+w]

```


Definisco nuove aree di lavoro: rappresentano la stessa area (limitata alla metà superiore del volto), la prima è in scala di grigi, la seconda a colori.

```
1     eyes= eyeCascade.detectMultiScale(  
3         roi_gray,  
         scaleFactor=sc_fct,  
         minNeighbors=min_neigh,  
5         minSize=min_size_e  
         )
```

Applichiamo nuovamente la funzione 'detectMultiscale'. Questa volta la funzione viene applicata a 'roi_gray'. Salviamo l'output della funzione in "eyes".

```
2     for (ex, ey, ew, eh) in eyes:  
         cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh),  
            (255, 191, 0), 2)
```

Tracciamo i rettangoli attorno agli occhi riconosciuti mediante la funzione 'rectangle'.

?? Riconoscimento cerchi nell'area occhi

```
2     roi_gray2=roi_gray[ey:ey+eh,ex:ex+ew]  
     roi_color2=roi_color[ey:ey+eh,ex:ex+ew]
```

Definisco nuove aree di lavoro: rappresentano la stessa area (limitata ai rettangoli degli occhi riconosciuti), la prima è in scala di grigi, la seconda a colori.

```
1         circles = cv2.HoughCircles(roi_gray2,cv2.  
            HOUGH_GRADIENT,dp=dp_c,minDist=mindist_c,  
            param1=param1_c,param2=param2_c,minRadius=  
            minr_c,maxRadius=maxr_c)
```

Applichiamo la funzione 'HoughCircles' per il riconoscimento degli occhi nell'area inserita come input (roi_gray2). Salviamo l'output della funzione (una lista di tre elementi: coordinata x, coordinata e raggio del cerchio riconosciuto) in "circles".

```
1         if(circles==None):  
2             continue  
3         else:  
            circles = np.uint16(np.around(circles[0,:]))
```

Introduciamo la condizione che ci permette di "bypassare" i frame in cui non viene riconosciuto alcun cerchio. Nel caso in cui, invece, il cerchio venga riconosciuto, 'circles' viene trasformato in un ato di tipo intero.

```
2         for (x_c,y_c,r) in circles:  
            cv2.circle(roi_color2,(x_c,y_c),r,(0,255,0),2)  
            cv2.circle(roi_color2,(x_c,y_c),2,(0,0,255),3)
```

Tracciamo i cerchi e il loro centro.

?? Rappresentazione del movimento oculare

```
1         plt.scatter(x+ex+x_c,y+ey+y_c, s=100, c='r',alpha  
                =0.3)  
3         plt.axis([0,x_dim,0,y_dim])  
         plt.draw()
```

Introduco uno scatterplot di dimensioni (x_dim e y_dim) pari alle dimensioni del video in cui visualizzo il centro di cerchio trovato. I punti hanno una trasparenza tale da permettere di osservare le zone dello scatterplot più dense di punti: ovvero quelle zone in cui lo il centro dell'iride è stato più presente. Si ottiene un output di questo tipo:

?? Generazione video e chiusura programma

```
1         cv2.imshow('Face_and_eyes', frame)
```

Mostriamo il video. In 'frame' saranno state apportate le modifiche delle varie aree di interesse. Nel video saranno dunque mostrati i rettangoli dei volti e degli occhi riconosciuti e i cerchi trovati all'interno dei rettangoli degli occhi.

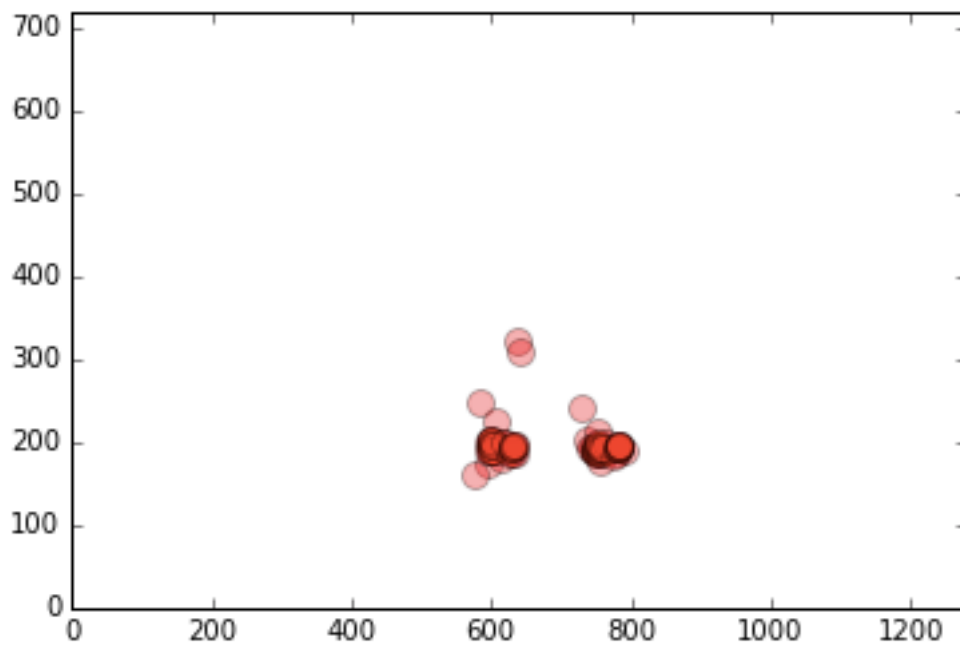


Figura 1: Esempio scatterplot

```
1 if cv2.waitKey(1)==27:  
    break
```

Interruzione del ciclo iniziato al momento della cattura dei frame e chiusura della finestra video quando viene premuto il tasto 'Esc'.

```
1 video_capture.release()  
  cv2.destroyAllWindows()
```

Chiusura di tutte le finestre.

4 Codice completo

```
1 %matplotlib inline
3 import matplotlib.pyplot as plt

5 import cv2
  import numpy as np

7
  #Parameters setting:
9 sc_fct = 1.1 #scaling factor
  min_neigh = 5 #minimum number of neighbours to identify an area as
    the specific object
11 min_size_f = (30,30) #smallest face dimensions accepted
  min_size_e = (10,10) #smallest eye dimension accepted
13 mindist_c=30 #minimum distance between circles' centers
  dp_c=1 #accumulator resolution factor
15 param1_c=60 #specific parameter of the Hough Gradient detection
    method
  param2_c=20 #specific parameter of the Hough Gradient detection
    method
17 minr_c=5 #minimum circle radius accepted
  maxr_c=15 #maximum circle radius accepted
19
21
23 #Loading the cascade classifier files
  faceCascade = cv2.CascadeClassifier("
    haarcascade_frontalface_default.xml")
25 eyeCascade=cv2.CascadeClassifier("haarcascade_eye.xml")

27 #Capturing the video from the webcam and opening a window to
  display it
  video_capture = cv2.VideoCapture(0)
29 cv2.namedWindow("Face_and_eyes")

31 #Saving the window dimensions
  x_dim=video_capture.get(3)
33 y_dim=video_capture.get(4)
```

```

35 while True:
    #Capturing frames from the video
37     ret, frame = video_capture.read()

    #Converting the image from colour to grayscale
39     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
41
    #Using the cascade classifier to identify faces in the video
43     faces = faceCascade.detectMultiScale(
        gray,
45         scaleFactor=sc_fct,
        minNeighbors=min_neigh,
47         minSize=min_size_f
    )

49     #detectMultiscale function returns a list of rectangles
        described by the upper left vertex and the two dimensions

51     #Drawing rectangles recognised by the detectMultiscale function
    for (x, y, w, h) in faces:
53         cv2.rectangle(frame,(x, y), (x+w, y+h), (0, 255, 0), 2)

        #Defining new regions of interest
55         roi_gray=gray[y:y+h/2,x:x+w]
57         roi_color=frame[y:y+h/2,x:x+w]

59         #Using the cascade classifier to identify eyes within the
            new ROI just defined
        eyes= eyeCascade.detectMultiScale(
61             roi_gray,
            scaleFactor=sc_fct,
63             minNeighbors=min_neigh,
            minSize=min_size_e
65             )

67
            #Drawing rectangles returned by the detectMultiscale
            function
69         for (ex, ey, ew, eh) in eyes:

```

```

71         cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh),
           (255, 191, 0), 2)

73         #Defining new regions of interest
roi_gray2=roi_gray[ey:ey+eh,ex:ex+ew]
roi_color2=roi_color[ey:ey+eh,ex:ex+ew]

75         #Applying HoughCircles function to identify iris and
           pupil in the filtered image (HOUGH_GRADIENT is the
           method used)
77         circles = cv2.HoughCircles(roi_gray2,cv2.
           HOUGH_GRADIENT,dp=dp_c,minDist=mindist_c,
           param1=param1_c,param2=param2_c,minRadius=
           minr_c,maxRadius=maxr_c)

79         #The function returns a list of circles described by
           the centre coordinates and the radius

81         #Checking validity of the detected objects
if(circles==None):
83             continue
else:
85         #Covertng floats to integers to make values usable
           by the circle function
           circles = np.uint16(np.around(circles[0,:]))

87         #Drawing identified circles in the video
for (x_c,y_c,r) in circles:
89             cv2.circle(roi_color2,(x_c,y_c),r,(0,255,0),2)
91             cv2.circle(roi_color2,(x_c,y_c),2,(0,0,255),3)

93         #Plotting a scatterplot with the eyes pupils'
           position through time
           plt.scatter(x+ex+x_c,y+ey+y_c, s=100, c='r',alpha
           =0.3)
           plt.axis([0,x_dim,0,y_dim])
           plt.draw()

95         #Showing the filtered image and the webcam video
97         cv2.imshow('Face_and_eyes', frame)

```

```
101         #Breaking the while cicle when 'return' is pressed
103         if cv2.waitKey(1)==27:
104             break
105
106     #Closing al windows and turning webcams off
107 video_capture.release()
cv2.destroyAllWindows()
```


5 Conclusioni