Iris Detector

Funzionamento

IrisDetection.py

Per eseguire il programma è sufficiente, da riga di comando, scrivere il seguente comando:

python IrisDetection.py

Per poter usare questa funzione è necessario inserire nella stessa cartella il file **functions.py**, che contiene alcune funzioni utilizzate dal programma.
Inoltre è possibile passare i seguenti parametri al comando:

- **-o** seguito dal path / in cui vogliamo che venga salvata la foto scattata e le immagini relative all'iride. Se questo parametro non viene passato ne viene impostato uno di default, in particolare /home/pi/Desktop e il nome della foto sarà *foto*, con l'aggiunta di estensioni alfanumeriche a seconda del tipo di immagine
- **-p** seguito dal numero 1 oppure 2, permette di specificare quale funzione per il calcolo del raggio della pupilla utilizzare. Nel caso di 1 viene eseguita **getPupil()**, mentre nel caso di 2 viene eseguita **getPupil2()**, entrambe spiegate successivamente.
- **-e** permette di specificare il tipo di evento che scatena la cattura della foto in alta definizione. Se non è specificato, per eseguire la fotografia, è necessario premere il tasto *"q"*. Se è seguito dalla dicitura "*face"*, oltre alla pressione del tasto *"q"*, la fotografia viene scattata quando la faccia è vicina all'obiettivo da un certo periodo di tempo e il programma sta riconoscendo 2 occhi.

Esempio:
python IrisDetection.py -o /home/pi/Desktop/prova -e face -p 1
Al momento dell'avvio, a seconda della versione di *OpenCV* che si sta utilizzando, potrebbe comparire l'errore seguente **VIDIOC_QUERYMENU: Invalid argument**, ripetute volte.

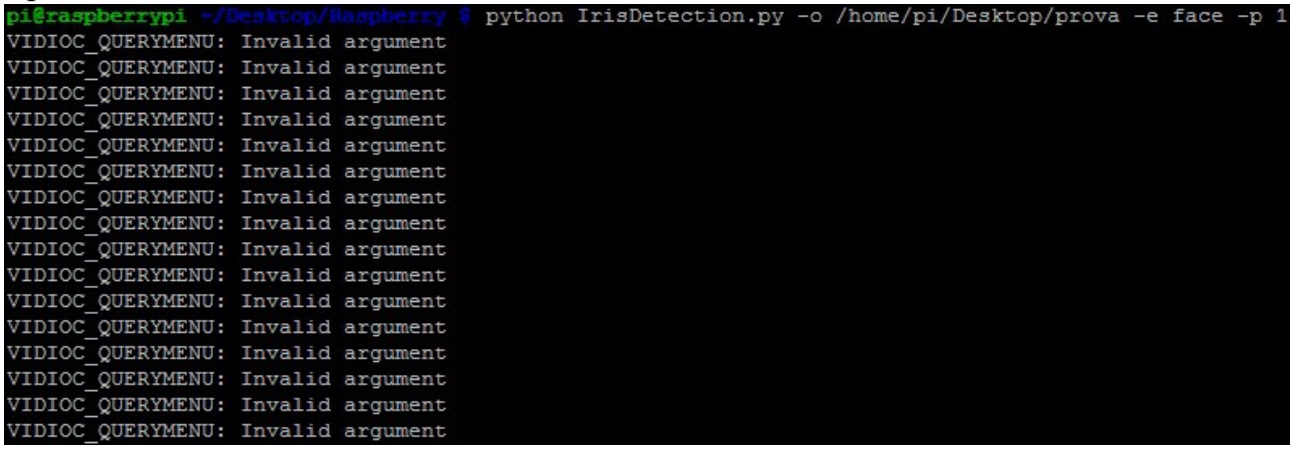

Illustrazione 1: Errore iniziale

Questo errore può essere tranquillamente trascurato ed il programma dovrebbe iniziare a funzionare correttamente subito dopo la visualizzazione di questi messaggi (per la durata di qualche secondo).
L'unica cosa che verrà visualizzata durante l'esecuzione del programma sarà il tempo, in millisecondi, che passa tra la cattura di un frame e l'altro, da parte della fotocamera.

```
time taken for detection = 0.000119654ms
time taken for detection = 0.000102331ms
time taken for detection = 0.000102267ms
time taken for detection = 9.87481e-05ms
time taken for detection = 0.000103463ms
time taken for detection = 0.000101671ms
time taken for detection = 9.70491e-05ms
```

Illustrazione 2: Tempo tra le acquisizioni delle immagini

Nel momento in cui viene scattata la fotografia, il led presente sulla fotocamera si spegnerà per un istante, dopodiché si accenderà di nuovo. Dal momento in cui si riaccende, fino allo spegnimento (definitivo), è necessario mantenere una posizione il più ferma possibile, e gli occhi ben aperti, per permettere al programma di catturare l'intera iride.

Descrizione

IrisDetector è un software scritto in Python, che permette l'estrazione dell'iride, da una foto in alta definizione, scattata durante l'esecuzione di un video che inquadra una faccia, nel momento più ottimale.
Il programma acquisisce, in bassa risoluzione, un video che, allo scatenarsi di un evento, scatta la foto in alta definizione. Lo scatenarsi dell'evento può essere la pressione di un tasto, oppure può avvenire nel momento in cui la faccia è "abbastanza" vicina alla fotocamera, per un tempo sufficientemente lungo, in maniera automatica.
Durante l'acquisizione del video (in bassa risoluzione), avviene un riconoscimento della faccia, grazie alla funzione *detectMultiScale* (della libreria opencv), che fa uso di un classificatore *HaarCascade*, opportunamente caricato all'inizio del programma. Allo stesso modo, solo quando viene trovata una faccia all'interno del frame, all'interno del rettangolo delimitante quest'ultima, avviene il riconoscimento degli occhi presenti. Il riconoscimento degli occhi avviene proprio come nel riconoscimento della faccia, facendo uso di un diverso classificatore.
Nel momento in cui avviene l'evento che scatena la cattura dell'immagine ad alta definizione (spiegato meglio più avanti), avviene un cambio di libreria per la cattura delle immagini, in particolare si carica la fotocamera con il comando "*with picamera.PiCamera() as camera*" della libreria *picamera,* il quale avviene dopo un rilascio della fotocamera da parte della libreria *OpenCV* (con la funzione release()). A questo punto viene fatta una fotografia in una risoluzione di 2592x1944 (4:3) che verrà usata per l'estrazione dell'iride dagli occhi presenti in essa.
Per estrarre gli occhi dalla fotografia, viene fatto un *ROI* su di essa, utilizzando le

coordinate provenienti dal riconoscimento avvenuto in fase di acquisizione a bassa risoluzione. Queste coordinate devono essere opportunamente adattate alle nuove dimensioni della fotografia, essendo questa in una risoluzione più alta della precedente. In fase di riconoscimento, sia della faccia che degli occhi, vengono ricavate le coordinate del punto in alto a sinistra del quadrato contenente tali oggetti, più un scostamento (*offset*) rispetto ad esse in orizzontale e in verticale. Di fatto il risultato è un vettore di quattro componenti **(x, y, w, h)**, le quali verranno moltiplicate per una scala (chiamate **scaleWidth** e **scaleHeight** nel programma), calcolata all'inizio come divisione tra le dimensioni del frame in alta risoluzione con quelle del frame in bassa risoluzione. Essendo il frame in bassa risoluzione in 4:3 (160x120) proprio come l'altra risoluzione, all'ora la moltiplicazione per tale scala non comporta un mal posizionamento dei quadrati sull'immagine.
A questo punto, all'interno dei quadrati contenenti gli occhi, avviene il riconoscimento dell'iride, contenente la pupilla, che verrà estratta successivamente. Nel *ROI* di ogni occhio per prima cosa avviene un equalizzazione dell'immagine con la funzione *equalizeHist()*, dopodiché avviene un processo di thresholing binario su quest'ultima con la funzione *threshold()* con soglia 70 ed infine si trova la posizione del centro dell'iride, e del suo raggio, grazie all'operatore *Canny*, eseguito appunto con la funzione *Canny()* di *OpenCV*.
Fatto questo, nel caso in cui un cerchio relativo all'iride sia stato trovato, si lancia la funzione **getPupil()** (oppure **getPupil2()**), la quale, con in input le coordinate del centro dell'iride e del raggio, cerca di misurare il raggio della pupilla, restituendolo in output.

getPupil()

La funzione **getPupil()**, contenuta nel modulo **functions.py**, riceve in ingresso quattro parametri, ovvero l'immagine dell'occhio, le coordinate *x* ed *y* del centro dell'iride, e il raggio di quest'ultima. Per prima cosa questa funzione crea un *ROI* delle dimensioni del raggio dell'iride al quale viene sottratto un certo *offset*, il che può essere fatto in quanto quasi sicuramente la pupilla sarà di dimensioni molto inferiori a quelle dell'iride. Sull'immagine appena ritagliata viene fatta una normalizzazione, con la funzione **cv2.Normalize()**, per far sì che il colore della pupilla sia più marcato rispetto ai colori dell'iride. Questa funzione rende i colori della pupilla ancora più scuri (quindi tendenti al valore zero della scala di grigi) e gli altri più distanti da quelli della pupilla. Questo è reso necessaria dal fatto che nella fase successiva, verranno creati due istogrammi che indicano il numero di pixel con valore sotto una certa soglia, scandendo il *ROI* prima in orizzontale e poi in verticale. Il risultato, tenuto conto del fatto che ci potrebbero essere dei riflessi sull'immagine, dovrebbe avere la forma di una campana, la cui larghezza dovrebbe rappresentare la larghezza della pupilla.

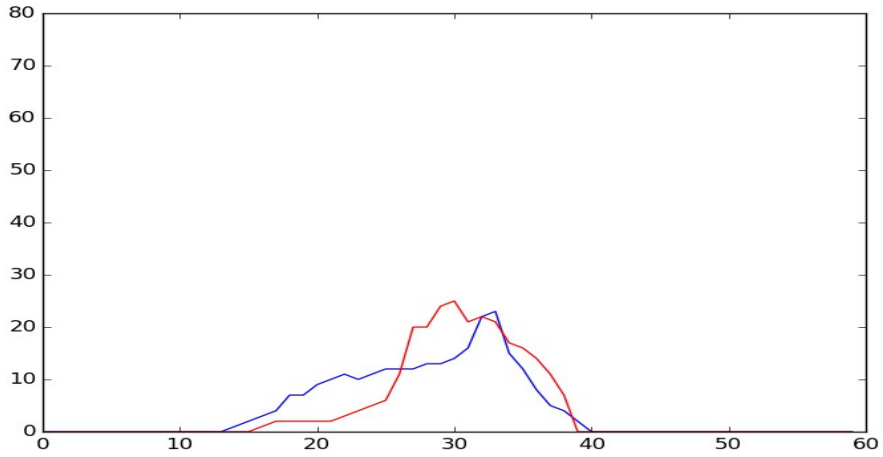

Illustrazione 3: Istogrammi pupilla

Un problema relativo a questo metodo di decisione della larghezza della pupilla, sta nel fatto che la foto potrebbe avere scale di grigio diverse, a causa delle diverse condizioni ambientali, in termini di illuminazione e posizione della faccia rispetto alla fonte di luce. Infatti gli istogrammi potrebbero risultare con una varianza diversa in fotografie diverse, e un diverso spostamento rispetto all'asse delle x. La normalizzazione dell'immagine è utile proprio ad aumentare lo scarto tra i colori della pupilla e quelli circostanti, producendo di volta in volta istogrammi più simili possibile tra di loro, permettendo di definire una soglia unica, sotto la quale un pixel è da considerarsi facente parte della pupilla.
Una volta generati gli istogrammi, il metodo per calcolarne la varianza, consiste nel cercare il valore più alto, che ragionevolmente apparterrà quasi di sicuro alla pupilla, e da lì scorrere a destra e sinistra fino a trovare il punto in cui fa zero. Infine viene fatta la media dei valori risultanti dai due istogrammi per restituire un unico valore del raggio della pupilla.

getPupil2()

La funzione **getPupil2()** è uguale alla precedente, con la differenza che prova a trovare il giusto valore della soglia per la generazione dei due istogrammi. In pratica questa funzione richiama la funzione **getHistograms()**, che genera iterativamente, di volta in volta, gli istogrammi con la soglia che decresce di uno ogni volta. Non appena la percentuale di zeri presenti negli istogrammi è superiore ad un certo valore l'iterazione si blocca, restituendo i due istogrammi, dai quali si ricava il raggio della pupilla come si faceva nella precedente funzione.