3D Eye-Tracking con Kinect v2 e Python

Victor Untila 903147

Sommario—In questo lavoro si è esplorata la possibilità di interfacciare il sensore Kinect v2 con il linguaggio di programmazione Python per effettuare eye-tracking.
Molti dei sistemi attualmente in commercio sfruttano la luce infrarossa per effettuare eye-tracking. Questi sistemi tuttavia, sono sensibili alle condizioni di luminosità oppure richiedono numerose fasi di calibrazione prima di essere utilizzati. L'approccio seguito in questo lavoro si basa sull'utilizzo della camera RGBD del Kinect v2. Il software sviluppato in Python rileva, ad ogni frame ricevuto dal Kinect v2, alcuni landmark facciali, in particolare i landmark che riguardano la regione degli occhi. In seguito vengono calcolati: il centro di rotazione dell'occhio, definito come il punto medio tra i landmark piu esterni della zona dell'occhio; ed il centro della pupilla. Ottenendo questi due dati è possibile determinare il gaze vector che attraversa il centro di rotazione dell'occhio ed il centro della pupilla. Infine, il punto di intersezione tra il gaze vector ed il display è calcolato e ed è visualizzato sullo schermo.

1 Introduzione

La continua evoluzione e innovazione nel campo hardware e software degli ultimi anni ha contribuito estensivamente anche all'espansione e ricerca nel campo dell'interazione naturale. Tramite movimenti del corpo, touch screen, tracking del volto e degli occhi e il riconoscimento vocale abbiamo un nuovo modo di interagire con i computer per svolgere diversi compiti. Tra questi, i metodi di eye-tracking, sono considerati i più intuitivi e confortevoli da usare. L'analisi dell'occhio e dello sguardo infatti, possono mettere in evidenza una serie di informazioni, come ad esempio su cosa sia focalizzata l'attenzione del soggetto e adirittura ottenere una serie di insight riguardanti lo stato d'animo interno. Nel campo dell'interazione naturale, tracciare lo sguardo può servire come rimpiazzo rispetto ai comuni metodi di input come ad esempio mouse o tastiera, inoltre, può fornire un nuovo modo per aggiungere input aggiuntivo in modo da interagire meglio con il computer. Nel campo dell'intrattenimento ad esempio, sempre più videogiochi rendono possibile l'interazione con lo sguardo fornito da un eye tracker per migliorare l'esperienza di gioco. Un altra applicazione possibile è usare i dati provenienti dall'eye tracker per fini di analisi e marketing, in modo da studiare il processo cognitivo umano.
Gli approcci utilizzati per il tracciamento dello sguardo possono essere suddivisi principalmente in due categorie: basati su infrarossi e non. I sistemi in commercio basati su infrarossi spesso usano hardware dedicato, di conseguenza i costi sono più alti. I sistemi che non si basano su infrarossi, per contro, sono meno diffusi e utilizzano delle tecniche che sono relativamente immature e poco precise. Tuttavia, rimangono un tema di ricerca popolare per il fatto che hanno requisiti hardware meno stringenti e per la facilità di integrazione con prodotti come laptop e tablet.
I metodi non IR (infrarossi) possono essere divisi in tre categorie:

Appearance-based [1]: in questo tipo di approcci si cerca di costruire un regressore che mappa l'aspetto dell'occhio a coordinate sullo schermo dove il soggetto sta guardando. L'assunzione implicita di questi metodi è che i cambi nell'aspetto dell'occhio sono influenzati solamente dal movimento della pupilla. Tuttavia l'aspetto dell'occhio può variare in merito ad altri fattori come ad esempio cambi di illuminazione oppure movimento della testa.

Iris-based [2]: si cerca di individuare l'iride tramite il fitting di un elisse. La forma dell'ellisse può essere utilizzata per determinare lo sguardo tramite l'approssimazione del vettore normale. Questo metodo non è molto accurato perché spesso è difficile estrarre la forma esatta dell'iride per via di occlusioni da parte delle ciglia, riflessioni speculari e rumore presente nell'immagine.

Face-model-based: come primo step si individuano una serie di landmark facciali nell'immagine. Le coordinate 3D dei landmark possono essere ottenute tramite l'utilizzo di camere stereo [3] oppure usando un generico modello facciale 3D [4]. Dopo la determinazione della posizione del centro dell'occhio basandosi sui landmark trovati, si definisce l'asse ottico: il vettore che attraversa il centro dell'occhio e la posizione 3D della pupilla. L'uso di un modello facciale generico tuttavia fornisce posizioni 3D dei landmark innacurate. La criticità di questo approcio risiede nel fatto che i dati di profondità forniti dal sensore potrebbero non essere abbastanza precisi per stimare lo sguardo. Anche errori minimi nei landmark 3D possono risultare in grandi errori nella

Victor Untila, Corso di Interazione Naturale, A/A 2018-2019, Universitá degli Studi di Milano, via Celoria 18, Milano, Italy
E-mail: victor.untila@studenti.unimi.it

stima dello sguardo. Tuttavia, questo appproccio risulta, in teoria, robusto rispetto al movimento della testa.

In questo lavoro è stato adottato l'ultimo approccio descritto prendendo come riferimento [5]. I principali contributi forniti sono:

- interfacciamento tra Kinect v2 e Python
- testing di codice fornito dal wrapper Pykinect v2
- implementazione di un face-based-model per gaze tracking;

2 Analisi dello stato dell'arte

Uno dei primi studi che si è occupato di eye-tracking utilizzando il kinect è stato fatto da Xiong, Cai, Liu e Zhang in [6]. Nella soluzione da loro proposta, tuttavia, l'utilizzo del kinect non è esclusivo, si può utilizzare qualsiasi tipo di sensore RGBD per ricavare i dati di profondità. Inoltre, hanno confrontato i risultati nel caso i dati di profondità siano forniti da una camera RGBD, oppure, se la camera non fosse disponbile, minimizzando l'errore di proiezione di un modello 3D generico e i landmark 2D rilevati nell'immagine. Kim e Lee in [5] utilizzano l'HD face model fornito dall'SDK di Kinect v2 per effettuare eye-tracking. Il modello si concentra sull'individuazione di una serie di feature riguardanti l'occhio: il centro di rotazione e la pupilla. Dati che vengono usati per determinare il 3D gaze vector. In un lavoro successivo Kim, Ko, Jang, Han e Lee [7] perfezionano il lavoro precedente di Kim e Lee in [5] combinando il gaze vector con un facial gaze vector tramite somma pesata, dando più peso al facial gaze vector. Wang e Ji [8] effettuano gaze-tracking seguendo il metodo face-model-based. A differenza dei lavori citati in precedenza, in questo lavoro viene effettuata un'analisi approfondita riguardo le caratteristiche anatomiche dell'occhio. Oltre al centro dell'occhio e della pupilla sono considerati anche il centro della cornea e l'angolo kappa, che rappresenta l'offset tra l'asse ottico e l'asse visivo, che varia per ogni soggetto. Vi è anche una fase di calibrazione personale per inferire i parametri dell'occhio dipendenti dal soggetto in questione.

3 Modello teorico

Il modello sviluppato segue le fasi mostrate nel diagramma nella Figura 1.

3.1 3D Eye-tracking

Il viso del soggetto è rilevato tramite la camera RGB del Kinect v2. Per poter rilevare i landmark sul viso, è stato impiegata la libreria dlib ed Opencv per Python che implementa il lavoro pubblicato in [9]. Vengono individuati 68 landmark sul viso come mostrato in Figura 2a. Dei 68 landmark vengono presi in considerazione soltanto quelli posti sul contorno interno ed esterno dell'occhio come mostrato in Figura 2b. Si può procedere dunque con lo step (b) in Figura

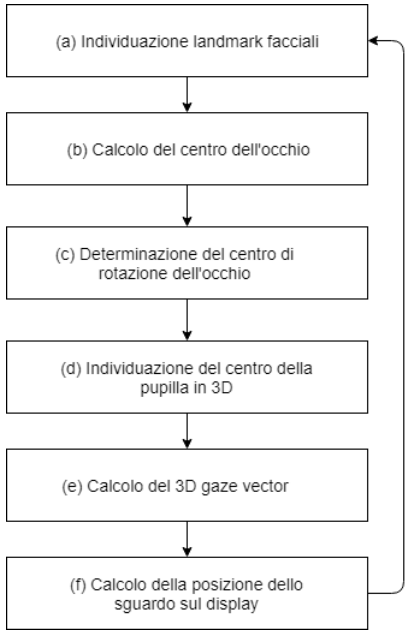

Figura 1: *Passi principali del metodo proposto*

1. Il punto medio tra i due landmark considerati rappresenterà il centro dell'occhio. Generalmente il centro di rotazione dell'occhio è situato 13.5 mm sotto la superficie della cornea come mostrato in Figura 2c. Per trovare il centro di rotazione dell'occhio basta traslare la coordinata del centro dell'occhio, trovato allo step (b) in Figura 1, di 13.5 mm sull'asse Z, come mostrato anche in Figura 2d. In questo modo lo step (c) in Figura 1 è completo. Successivamente si procede allo step (d) in Figura 1. Dall'immagine RGB ottenuta dal Kinect v2 si isola la regione dell'occhio, Figura 3a. Per determinare il centro della pupilla, il primo passaggio è di trasformare il frame isolato dell'occhio in un'immagine in livelli di grigio. Successivamente l'immagine in livelli di grigi viene binarizzata con threshold impostato a 18, Figura 3b. Inoltre, sono applicate delle operazioni morfologiche, dilatazione ed erosione, all'immagine binarizzata in modo da rimuovere eventuale noise. Il risultato è visibile in Figura 3c e 3d. Per trovare il centro della pupilla, si fa uso della libreria Opencv per calcolare il centroide del poligono cosi rilevato. In questo modo troviamo le coordinate (in pixel) del centro della pupilla.

Per poter calcolare il 3D gaze vector, step (e) nella Figura 1, tuttavia, sono richieste le coordinate nello spazio della camera, o coordinate mondo, del centro di rotazione dell'occhio e del centro della pupilla. Le coordinate precedentemente calcolate infatti, sia per il centro di rotazione che per il centro della pupilla, sono coordinate pixel nell'immagine. La coordinata Z (in mm) di entrambi i dati è fornita dal sensore di profondità del Kinect v2. Per quanto riguarda le coordinate X e Y (in mm), invece, fare riferimento alla sezione 4.1. La libreria PyKinect v2 infatti, non espone nessun metodo per ricavare questi dati direttamente.

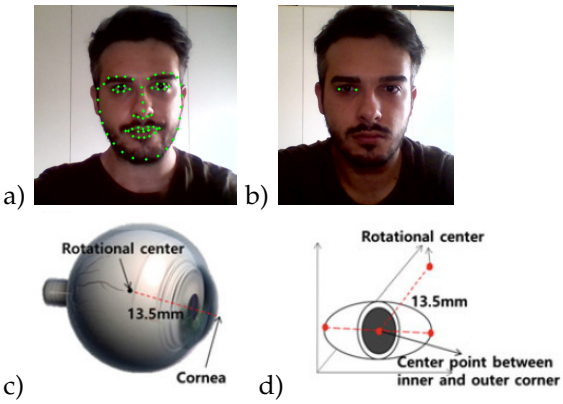

Figura 2: *a) 68 Landmark rilevati, b) 2 landmark dell'occhio, c) Bulbo oculare, d) Traslazione dal centro della cornea al centro di rotazione*

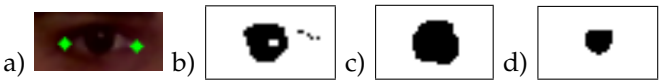

Figura 3: *a) Regione occhio, b) binarizzazione, c) dilazione, d) Erosione*

Ottenute le coordinate mondo del centro di rotazione e del centro della pupilla, definiamo il 3D gaze vector come il vettore che attraversa due punti: il centro di rotazione dell'occhio ed il centro della pupilla. Viene calcolato mediante la seguente equazione:

$$\frac{x - x_i}{x_r - x_i} = \frac{y - y_i}{y_r - y_i} = \frac{z - z_i}{z_r - z_i} \tag{1}$$

dove (x_r, y_r, z_r) rappresentano le coordinate mondo del centro di rotazione dell'occhio e (x_i, y_i, z_i) rapprensentano le coordinate mondo del centro della pupilla.

Infine si procede allo step (f) in Figura 1. Per tracciare lo sguardo, viene calcolato il punto di intersezione tra il gaze vector ed il display tramite le seguenti equazioni:

$$x_c = x_i + \frac{-z_i(x_r - x_i)}{(z_r - z_i)} \qquad y_c = y_i + \frac{-z_i(y_r - y_i)}{(z_r - z_i)} \tag{2}$$

Qui, (x_c, y_c), rappresenta il punto di intersezione nello spazio delle coordinate della camera. Tramite la calibrazione geometrica, (x_c, y_c) è trasformato in un punto fisico sul display.

4 Simulazione ed esperimenti

4.1 Coordinate camera-space

Come menzionato precedentemente, la libreria PyKinect v2 non espone un metodo per ricavare le coordinate camera-space X e Y (in mm). Di seguito è mostrato il processo per ricavare questi dati.
Nell'immagine catturata dal sensore RGB del Kinect v2, le coordinate X e Y si riferiscono a un pixel arbitrario. Per ottenere la coordinata Z tuttavia, viene impiegato il sensore di profondità. Quindi, le coordinate pixel

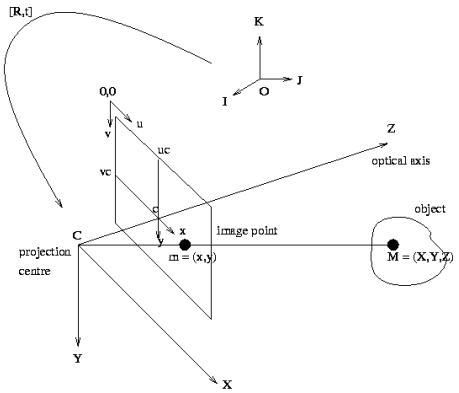

Figura 4: *Proiezione di un punto 3D sull'immagine 2D.*

nell'immagine RGB non corrispondono alle coordinate pixel nell'immagine fornita dal sensore di profondità. La libreria PyKinect v2 mette a disposizone un metodo per sovrappore l'immagine RGB all'immagine Depth, in modo da avere una corrispondenza tra i pixel nelle due immagini. Per ottenere le coordinate X e Y (in mm) nello spazio 3D camera-space, è stato seguito come riferimento il pinhole camera model e la semplice proiezione della prospettiva. In sintesi, in questo modello lo scopo è di riuscire a proiettare un punto 3D definito nello spazio delle coordinate mondo in un punto 2D nell'immagine (Figura 4). Nel modello è cosi definito che:

$$P_{schermo} = I * P_{mondo} \tag{3}$$

In coordinate omogenee:

$$\begin{bmatrix} x_{schermo} \\ y_{schermo} \\ 1 \end{bmatrix} = I * \begin{bmatrix} x_{mondo} \\ y_{mondo} \\ z_{mondo} \\ 1 \end{bmatrix}$$

Dove I rappresenta la matrice 3x4 dei parametri intrinseci della camera:

$$I = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

f_x e f_y rappresentano il punto focale sull'asse orizzontale e verticale. c_x e c_y rappresentano il centro di proiezione. Risolvendo il sistema sopra presentato si ottiene:

$$x_{mondo} = (x_{schermo} - c_x) * \frac{z_{mondo}}{f_x} \tag{4}$$

$$y_{mondo} = (y_{schermo} - c_y) * \frac{z_{mondo}}{f_y} \tag{5}$$

Le coordinate cosi ottenute rappresentano la distanza in millimetri rispetto al centro di proiezione della camera.

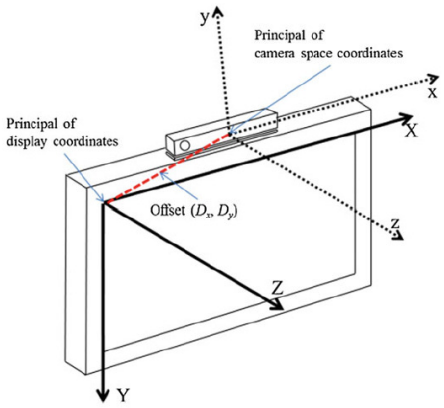

Figura 5: *Relazione geometrica tra le coordinate camera-space e le coordinate display.*

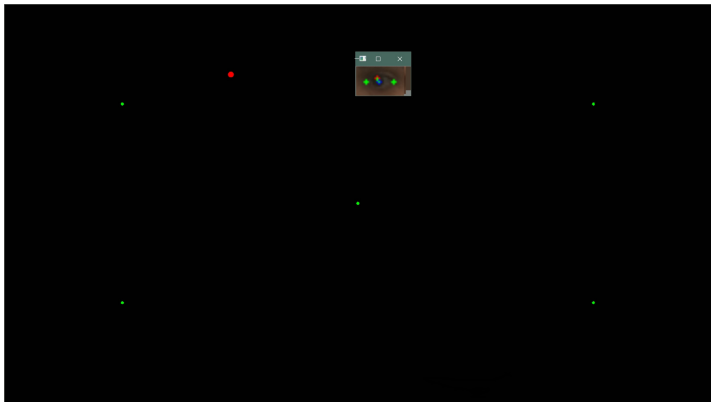

Figura 6: *I punti in verde sono punti di riferimento fissi. Il punto rosso rappresenta il gaze tracciato*

4.2 Calibrazione Geometrica

Per poter tracciare lo sguardo sullo schermo, le coordinate in camera-space devono essere convertite in coordinate sul display. Kim e Lee in [5] propongono una calibrazione che mette in relazione geometrica il Kinect v2 e lo schermo del computer. Come si può notare in Figura 5 il principal point della camera e del display non corrispondono. La trasformazione viene effettuata mediante le seguenti equazioni:

$$x_d = x_c + D_x \tag{6}$$

$$y_d = y_c + D_y \tag{7}$$

Dove x_d e y_d rappresenta il principal point dello schermo, e x_c e y_c rappresenta il principal point del camera-space. D_x e D_y sono gli offset in millimetri tra il principal point della camera e del display. Nella configurazione utilizzata per fare gli esperimenti, D_x e D_y sono rispettivamente 170 e 80 mm.

4.3 Procedura dell'esperimento

L'esperimento consiste nell'avere 5 punti di riferimento predefiniti sullo schermo come mostrato in Figura 6. Il soggetto, posto a una distanza minima di 50 cm dallo schermo, per via del range di funzionamento del sensore di profondità del Kinect v2, può guardare liberamente sullo schermo ed il suo sguardo tracciato è identificato dal punto rosso.
L'esperimento è stato effettuato su una macchina Windows 10 con processore i7-7700HQ (2.80 Ghz), 16 GB di RAM e con una scheda video Nvidia GTX 1070M.

4.4 Dettagli implementativi

Durante gli esperimenti, non è stato applicata l'equazione (6), poiché sull'asse orizzontale lo sguardo tracciato non corrispondeva al punto realmente fissato dal soggetto.
Le librerie principali utilizzate sono:

- **PyKinect v2** *(https://github.com/Kinect/PyKinect2)*, un wrapper che consente di utilizzare alcune delle funzionalità di Kinect v2 in Python.
- **OpenCV**, libreria open source incentrata sulla computer vision e il machine learning.
- **dlib**, libreria open source dedicata al machine learning ed algoritmi di apprendimento. E' stata utilizzata per tracciare i 68 landmark facciali.

5 Risultati ottenuti e conclusioni

In questo paper è stato proposto una soluzione per il gaze tracking applicando un modello basato sulla rilevazione del volto. A differenza del lavoro in [5], è stato utilizzato Python come linguaggio di programmazione. In Figura 7 si possono vedere i risultati ottenuti. Il gaze tracciato non è accurato, come si può notare, la maggior imprecisione è sull'asse verticale. Questo può dipendere da una serie di fattori, come ad esempio, le condizioni di illuminazione, che influenzano negativamente la rilevazione del centro della pupilla e di conseguenza il corretto calcolo del gaze vector. Altri fattori maggiormente critici sono l'indisponibilità dell'HD Face Model in PyKinect v2. Questo metodo nativo, infatti, permette di tracciare 1436 landmark sul viso del soggetto in un immagine in alta definizione. La mancanza, inoltre, del metodo Coordinate Mapper in PyKinect v2, che ha il compito di calcolare le coordinate in camera-space (millimetri) da una coordinata pixel, è un'altra problematica importante. Di conseguenza, non utilizzando i metodi nativi dell'SDK di Kinect v2, si ottengono dati approssimati che influenzano drasticamente i risultati del gaze tracking.

Riferimenti bibliografici

[1] L. Feng, S. Yusuke, O. Takahiro, and S. Yoichi, "Inferring human gaze from appearance via adaptive linear regression," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 153–160.

[2] J.-G. Wang, E. Sung, and R. Venkateswarlu, "Estimating the eye gaze from one eye," *Computer Vision and Image Understanding*, vol. 98, pp. 83–103, 2005.

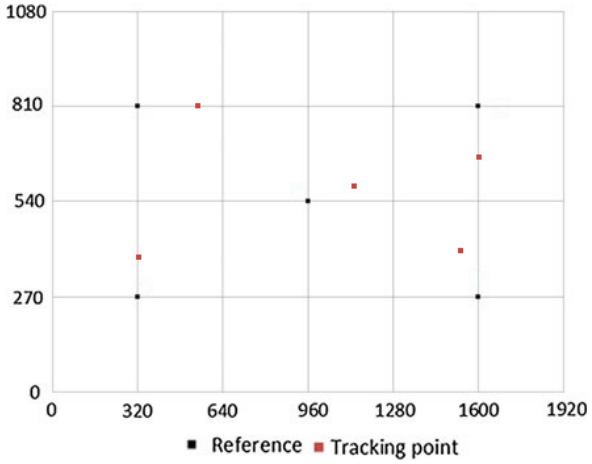

Figura 7: *Risultati ottenuti*

[3] R. Newman, Y. Matsumoto, S. Rougeaux, and A. Zelinsky, "Real-time stereo tracking for head pose and gaze estimation," in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*. IEEE, 2000, pp. 122–128.

[4] J. Chen and Q. Ji, "3d gaze estimation with a single camera without ir illumination," in *2008 19th International Conference on Pattern Recognition*. IEEE, 2008, pp. 1–4.

[5] B. C. Kim and E. C. Lee, "3d eye-tracking method using hd face model of kinect v2," in *Advanced Multimedia and Ubiquitous Engineering*. Springer, 2016, pp. 235–242.

[6] X. Xiong, Z. Liu, Q. Cai, and Z. Zhang, "Eye gaze tracking using an rgbd camera: a comparison with a rgb solution," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, 2014, pp. 1113–1121.

[7] B. C. Kim, D. Ko, U. Jang, H. Han, and E. C. Lee, "3d gaze tracking by combining eye-and facial-gaze vectors," *The Journal of Supercomputing*, vol. 73, no. 7, pp. 3038–3052, 2017.

[8] K. Wang and Q. Ji, "Real time eye gaze tracking with kinect," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2752–2757.

[9] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1867–1874.