POLITECNICO DI TORINO

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

Corso di Image Processing and Computer Vision

Ingegneria Informatica Grafica e Multimedia

Relazione di progetto: tool di supporto alle presentazioni da remoto

Docente:

Bartolomeo Montrucchio

Studenti:

Federica Giorgione, 280128

Carmelo Proetto, 282785

Carlo Vitale, 291128

Esercitatore:

Luigi De Russis

Anno accademico

2021/2022

Contents

1 Introduzione

Questi ultimi anni hanno visto la diffusione di eventi, di varia natura, svolti da remoto. La possibilità di praticare queste attività, quali lezioni e presentazioni, da casa, ha reso evidente la necessità di andare incontro a lavoratori non forniti di ambienti o strumentazione adatti per evitare di mantenere la stessa posizione un quantitativo eccessivo di ore. In questo contesto, il nostro progetto si propone l'obiettivo di creare un programma che fornisca un tool di supporto alle presentazione online attraverso le conoscenze di computer vision acquisite nel corso.
Dopo un'analisi delle principali funzionalità di cui si potrebbe necessitare, abbiamo deciso di implementare le seguenti azioni:

- caricamento della presentazione
- visualizzazione della presentazione
- capacità di prendere annotazioni sulle slide presentate
- zoom della presentazione
- gestione della webcam

Il nostro programma garantisce un utilizzo realtime che va a limitare il più possibile l'utilizzo del mouse, così che l'oratore abbia la possibilità di controllare la presentazione in piedi o ad una distanza più confortevole dal PC. Inoltre, non è necessaria alcuna strumentazione se non una webcam, di cui la maggior parte dei computer odierni è fornita.
Per avviare il programma è necessario attivare i permessi della webcam e far partire da terminale lo script *converter.py*. Nella cartella Presentation abbiamo fornito il PDF *EsempioPresentazione.pdf* per eventuali test.

2 Codice

2.1 Librerie usate

2.1.1 Librerie di computer vision

Di seguito andiamo ad illustrare le principali librerie di computer vision cui abbiamo fatto riferimento.

Per la manipolazione delle immagini abbiamo deciso di utilizzare la libreria mostrata a lezione **OpenCV**. Questa libreria ci è stata di supporto nella cattura della webcam, nella lettura di immagini e icone utili per l'interfaccia del programma, nella gestione delle finestre e nel disegnare alcuni elementi di UI utili, come vedremo in seguito, per rappresentare le azioni che si stanno svolgendo o i vincoli di cattura delle gesture.

MediaPipe è un framework crossplatform sviluppato da Google nato con l'obiettivo di fornire una serie di soluzioni di machine learning real time. In particolare, per il nostro progetto abbiamo utilizzato i moduli *Hands* e *Selfie Segmentation*. Hands, come da nome, si occupa del tracking veloce e ad alta precisione di mani e dita. Un primo modello si occupa del riconoscimento del palmo (precisione media stimata dagli sviluppatori del 95.7%) restituendo un bounding box, mentre un altro modello basato su 21 landmark viene applicato solo su questa regione identificata ottenendo dei keypoint.

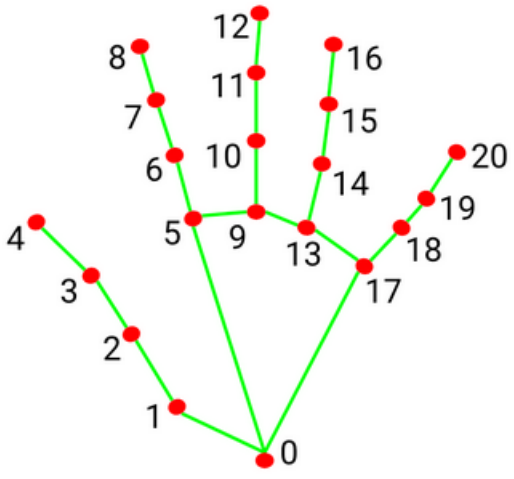

Figure 1: Landmark della mano

Infine, abbiamo utilizzato **CVZone**, una libreria basata a sua volta su OpenCV e MediaPipe che fornisce una serie di moduli e risorse utili per progetti di computer vision. Per il nostro

progetto abbiamo fatto riferimento, in particolare, all'*HandDetector* di *HandTrackingModule* e al *SelfiSegmentationModule*.

2.1.2 Altre librerie usate

Per alcuni elementi di UI delle schermate abbiamo usato la libreria **Tkinter**, adatta per interfacce stile desktop e fornita di molti widget utili, ma anche **ctype**. Per la gestione dei file abbiamo utilizzato la libreria **os**, mentre per la conversione dei pdf **pdfrw**.

2.2 Struttura del programma

2.2.1 converter

converter è uno script che gestisce il caricamento della presentazione come PDF (attualmente è adatto per pagine nel formato 1280x720 o con stesso aspect ratio) per convertire ogni pagina in immagine JPEG. Abbiamo deciso di seguire questa strada per la grande diffusione di questo formato di file e per dare la possibilità, come sarà spiegato quando parleremo del disegno, di poter prendere annotazioni sulle slide ed eventualmente esportare nuovamente la presentazione in un nuovo PDF.

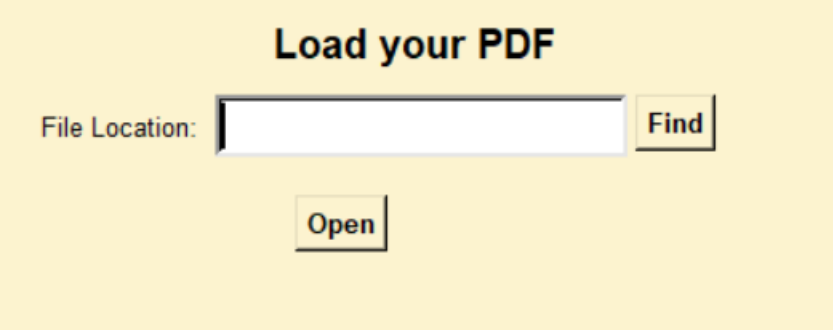

Figure 2: Schermata di caricamento della presentazione

Il bottone Find richiama la funzione *openFile()* che recupera il path della presentazione selezionata dal file system. Il bottone Open richiama invece il cuore dello script, ovvero la funzione *pdf2jpg()*. Tale funzione prende le singole pagine del PDF per salvarle come JPEG nella cartella tempDir. Una volta terminato, viene richiamato il main del *presentationController*. Se invece si dovesse chiudere la finestra, viene chiamata la *deleteAllTmpFile()* che si occupa di eliminare gli eventuali file temporanei (le immagini JPEG) create con la funzione *remove()* di *os*.

2.2.2 presentationController

Questo è lo script principale del progetto, infatti comprende l'acquisizione dei dati da webcam, il riconoscimento delle mani e l'implementazione delle gesture. Il codice permette la gestione di una finestra di supporto e di quella principale.
La finestra di supporto contiene un tutorial (importato come immagine) dove sono spiegate tutte le gesture, affiancato dalla cattura dello stream sovrascritto con le funzioni di OpenCV da indicazioni visuali dei limiti di cattura delle gesture.

```
cv2.line(img, (0, gestureThreshold), (width, gestureThreshold), (0, 255, 0), 5)
cv2.rectangle(img, (width // 2, height - 200), (width-200, 200), (255, 255, 255), 2)  # ROI puntatore con mano destra
cv2.rectangle(img, (200, height - 200), (width // 2, 200), (255, 255, 255), 2)  # ROI puntatore con mano sinistra
```

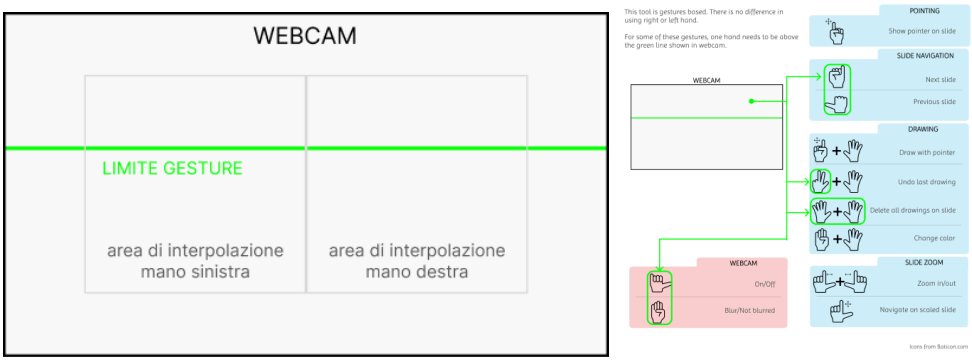

Figure 3: Schermata di tutorial

La schermata principale invece mostra sulla sinistra la cattura della webcam ripulita da ogni indicazione, alcune opzioni di selezione del colore, della webcam e del blur, mentre sulla destra abbiamo inserito la slide con l'indicazione del numero della slide corrente e delle slide totali.

Figure 4: Schermata principale

I vari elementi sono inseriti semplicemente sostituendo pixel per pixel l'immagine di partenza con quella che si vuole sovraimporre, ad esempio:

```
#inserimento switch camera
if camera:
    checkY, checkX, _ = checkedIcon.shape
    backgroundImg[hSmall + 222:hSmall + checkY + 222, 60: 60+checkX] = checkedIcon
else:
    checkY, checkX, _ = uncheckedIcon.shape
    backgroundImg[hSmall + 222: hSmall + checkY + 222, 60: 60 + checkX] = uncheckedIcon
```

Per il riconoscimento delle gesture, prima di tutto, andiamo a creare un *detector* per poter identificare, con la funzione *findHands()*, le mani dell'oratore, identificando inoltre se si tratta di quella destra o sinistra. Si tratta di un controllo importante in quanto, per agevolare i movimenti dell'utente ed evitare gesti troppo ampi che possano far calare la precisione del tracking, abbiamo definito due aree di cattura di alcune gesture a una mano dove si effettua una interpolazione. L'interpolazione prende in considerazione anche il fattore di scala e l'eventuale offset per traslare la coordinata corrispondente all'indice nella parte visibile di schermo.
Una gesture della singola mano è identificata dal numero di dita che sono considerate alzate. Questo controllo è effettuato da *detector* con la funzione *fingersUp()*, la quale confronta la posizione relativa di alcuni landmark delle dita per memorizzare un vettore di zeri (dito abbassato) e uno (dito sollevato).

```
if hands[0]['type']=='Left':  # mano destra (flip)
    rightHand = True
    handR = hands[0]
    fingersR = detector.fingersUp(handR)
    cxR, cyR, = handR['center']  # centro della mano
```

```
lmListR = handR['lmList']
```

In questo momento viene memorizzato anche il centro della mano, informazione che risulterà utile per validare alcune azioni. La validazione consiste semplicemente in un controllo che la coordinata del centro della mano sia al di sopra della linea verde disegnata nella schermata di tutorial. Tale controllo serve a evitare che una gesture venga erroneamente riconosciuta mentre l'oratore gesticola, forzando quindi il riconoscimento solo se la mano si trova nella parte alta dello schermo.
Le azioni possibili possono essere immediate (come il disegno) o possono necessitare il mantenimento della posizione per un certo numero di frame, comportamento gestito con la variabile booleana *buttonPressed*, con *buttonCounter* e con *buttonDelay*.

```
if buttonPressed:
    buttonCounter += 1
    if buttonCounter > buttonDelay:
        buttonCounter = 0
        buttonPressed = False
```

Le gesture ad una mano identificate sono:

- pollice su → mostra la slide precedente decrementando un contatore delle immagini *imgCount*
- mignolo su → mostra la slide successiva incrementando il contatore
- indice su → mostra un puntatore interpolando la posizione rilevata dal landmark corrispondente alla punta dell'indice nell'area della relativa mano
- pollice e mignolo su → permette di attivare e disattivare la webcam con la variabile booleana *camera* che aziona o meno la sovrascrittura della finestra principale con una ROI della webcam
- pollice e indice su → se l'immagine è zoomata, permette di navigare all'interno della slide visualizzandone solo una porzione
- mano aperta tranne pollice → permette di attivare o disattivare il blur con la variabile booleana *blur*

Le gesture a due mani sono invece:

- una mano aperta e l'altra con indice su → disegna nel punto in cui si trova il puntatore
- una mano aperta e l'altra, al di sopra della linea verde, con pollice, indice e medio su → permette di fare l'undo dell'ultimo tratto disegnato sulla slide corrente
- entrambe le mani aperte → permette di cancellare tutte le annotazioni sulla slide corrente
- una mano aperta e l'altra con il solo pollice non sollevato → permette di cambiare colore al tratto
- entrambe le mani con pollice e indice su → permette di effettuare uno zoom fino a 4X proporzionalmente alla distanza tra i due indici

Le annotazioni sulle slide sono memorizzate in *annotations*, una lista (disegno) di liste (coordinata del punto del singolo tratto) con, in parallelo, un'altra lista *colorArray* dei colori usati per ogni disegno, ed un dictionary *dictOfAnnotations* che associa le liste ad ogni pagina della presentazione.

```
if annotationStart is False:
    annotationStart = True
    annotationCounter += 1
    annotations.append([])  # inizio un nuovo disegno
    colorArray.append([])
    dictOfAnnotations[imgCount] = cv2.circle(imgCurrent, indexFingerR, 8, cColor, cv2.FILLED)
annotations[annotationCounter].append(indexFingerR)
colorArray[annotationCounter].append(cColor)
dictOfAnnotations[imgCount] = {'annotations':annotatio'cColor':colorArray}
print(dictOfAnnotations)

...

if len(dictOfAnnotations) != 0:
    if imgCount in dictOfAnnotations:
        note = dictOfAnnotations[imgCount]
        for i in range(len(note['annotations'])):
            for j in range(len(note['annotations'][i])):
                if j != 0:
                    cv2.line(imgCurrent, note['annotations'][i][j - 1], note['annotations'][i][j], note['cColor'][i][j],
                             5)  # disegna una linea tra ogni punto
```

Per eliminare l'ultimo tratto non facciamo altro che richiamare una *pop()*, mentre per cancellare tutti i segni della slide ripuliamo le liste.
Di default, la slide viene ovviamente rappresentata nella sua interezza. Tramite gesture, è possibile non solo fare uno zoom, ma anche navigare nell'immagine scalata. Per calcolare

correttamente l'offset nella matrice ingrandita e permettere la visualizzazione corretta di una porzione di slide, abbiamo immaginato di dividere lo schermo in 9 aree.

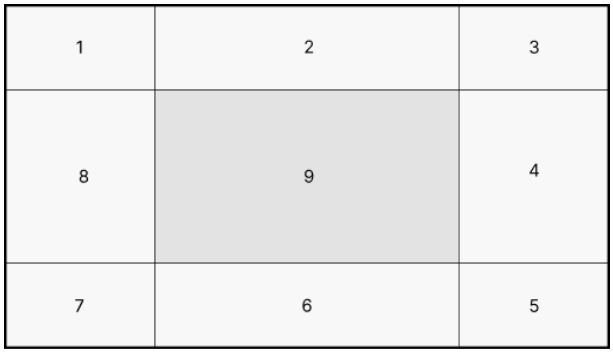

Figure 5: Slide divisa in zone di riconoscimento

In base alla posizione del centro dell'area che si vuole visualizzare, si va a calcolare l'offset del vertice in alto a sinistra lungo la x e la y, utile non solo per sostituire l'immagine che andrà mostrata, ma anche per calcolare nel modo corretto l'interpolazione del puntatore, così che sia sempre nella parte visibile di schermo.

```
if scale==1:
    xValR = int(np.interp(lmListR[8][0], [width // 2, imgCurrent.shape[1]-200], [0, width]))
    yValR = int(np.interp(lmListR[8][1], [200, height - 200], [0, height]))
else:
    xValR = int(np.interp(lmListR[8][0], [width // 2, imgCurrent.shape[1]-200], [int(padX/scale), int((padX-padXneg+width)/scale)]))
    yValR = int(np.interp(lmListR[8][1], [200, height - 200], [int(padY/scale), int((padY-padYneg+height)/scale)]))
indexFingerR = xValR, yValR

...

if scale*cx > waux/4 and scale*cx < 3/4*waux and scale*cy > haux/4 and scale*cy < 3/4*haux:  # zona 9
    print('zona: ', 9)
    padY = int(cy * scale) - 360
    padX = int(cx * scale) - 640
    padYneg = int(cy * scale) - 360
    padXneg = int(cx * scale) - 640

    if padY < 0:
        padY = 0
    else:
        padYneg = 0

    if padX < 0:
        padX = 0
    else:
        padXneg = 0

    zoomedImg = imgCurrent[padY:int(cy * scale) + 360 + padYneg, padX:int(cx * scale) + 640 + padXneg]
    elif scale*cx < waux/4 and scale*cy < haux/4:  # zona 1
        print('zona: ', 1)
        padY = 0
        padX = 0
        zoomedImg = imgCurrent[0:720, 0:1280]
    elif scale*cx < waux/4 and scale*cy > haux*3/4:  # zona 7
        print('zona: ', 7)
```

```
        padY = haux-720
        padX = 0
        zoomedImg = imgCurrent[haux-720:haux, 0:1280]
    elif scale*cx < waux/4:  # zona 8
        print('zona: ', 8)
        padY = int(scale*cy)-360
        padX = 0
        zoomedImg = imgCurrent[int(scale*cy)-360:int(scale*cy)+360, 0:1280]
    elif scale*cx > 3/4*waux and scale*cy < haux/4:  # zona 3
        print('zona: ', 3)
        padY = 0
        padX = waux-1280
        zoomedImg = imgCurrent[0:720, waux-1280:waux]
    elif scale*cx > 3/4*waux and scale*cy > haux*3/4:  # zona 5
        print('zona: ', 5)
        padY = haux-720
        padX = waux-1280
        zoomedImg = imgCurrent[haux-720:haux, waux-1280:waux]
    elif scale*cx > 3/4*waux:  # zona 4
        print('zona: ', 4)
        padY = int(scale*cy)-360
        padX = waux-1280
        zoomedImg = imgCurrent[int(scale*cy)-360: int(scale*cy)+360, waux-1280: waux]
    elif scale*cy < haux/4:  # zona 2
        print('zona: ', 2)
        padY = 0
        padX = int(scale*cx)-640
        zoomedImg = imgCurrent[0:720, int(scale*cx)-640: int(scale*cx)+640]
    elif scale*cy > haux*3/4:  # zona 6
        padY = haux-720
        padX = int(scale * cx) - 640
        print('zona: ', 6)
        zoomedImg = imgCurrent[haux-720:haux, int(scale * cx) - 640: int(scale * cx) + 640]
```

2.2.3 backgroundRemoval

Lo script fornisce la funzione *blurBackground()* che prima identifica il soggetto dell'immagine, ovvero l'oratore, con *SelfiSegmentation()* e poi applica un filtro gaussiano al background.

2.2.4 closureController

Quando si chiude la finestra principale, viene richiamata la funzione *closingApp()* di questo script. La funzione mostra una finestra che permette di scegliere se salvare le slide modificate in un nuovo PDF. Accettando, si fa partire la *savedSlide()* che a sua volta, attraverso la *addNote()*, sovrascrive sull'immagine le annotazioni con la *cv2.line()*, prendendo le coordinate e il colore dal *dictOfAnnotations* dello script *presentationController*. Effettua poi la conversione inversa con *jpg2pdf()* che memorizza il PDF nella cartella *savedSlidesDir*. Infine, al termine della conversione (o rifiutando di salvare) vengono eliminati i file temporanei.