Miniprojet2-3_Final_Junhui LUO

December 14, 2020

Les étapes 2 et 3 utiliseront le tracker dlib (the correlation trackers). Lorsqu'il s'agit de suivre plusieurs corps humains, dlib fonctionne souvent mieux que MIL. Et la détection des visages continue d'utiliser le classificateur en cascade de haar.

Il y a eu quelques problèmes lors de la construction de dlib. Puisque dlib est construit en C ++, quelques suppléments supplémentaires sont nécessaires. Mais l'environnement a été établi pour nous à Anaconda. Par conséquent, j'ai finalement utilisé conda create -n env_dlib python = 3.6 pour établir une nouvelle plate-forme pour python3.6 et terminé avec succès la construction de l'environnement dlib.

Remarque: les algorithmes et les fonctions qui ont été expliqués à l'étape 1 ne seront pas répétés ici.

Commentaires en cours d'exécution: La détection de visage effectuée toutes les quinze images aura des blocages évidents. Et en observant les données imprimées, il peut être confondu avec un nouveau visage lors du déplacement.

[1]:
```
import cv2 as cv
import dlib
```

[2]:
```
face_cascade = cv.CascadeClassifier(cv.data.haarcascades +␣
↪"haarcascade_frontalface_alt.xml")
```

Dans le loop: un loop est effectuée pour chaque frame de la caméra, et un loop de détection de visage est effectuée toutes les quinze frame: 1. Commencer par augmenter le nombre de framecounter.

2. Déterminer la qualité du suivi actuel du tracker et s'il peut continuer à suivre. Sinon, supprimer le tracker.
3. Loop de détection de visage toutes les 15 frame: a, détection de visage. b. Comparer les points de la détection de visage et du traqueur existants pour déterminer si le suivi correspond aux points du visage. S'il ne correspond pas, un nouveau tracker est créé. c. Créer un nouvel outil de suivi et compter le nombre de faces.
4. Créer un cadre rectangulaire pour le tracker qui a été déterminé.

[3]:
```
def main():
    cap = cv.VideoCapture(1)
    #Initialiser deux compteurs, l'un est le nombre de frames en cours␣
↪d'exécution, l'autre est le nombre de faces.
    framecounter = 0
```

```
    faceidcounter = 0
    #Nous avons préparé une collection de trackers pour chaque visage à␣
↪l'avance pour the correlation trackers.
    trackers = {}

    #Start loop principale
    while True:
        ret,frame = cap.read()
        #Chaque loop augmentera le compteur de frame
        framecounter += 1

        #Mettre à jour tous les trackers.
        #Et supprimez le tracker correspondant au visage dont la qualité de␣
↪suivi n'est pas bonne ou a quitté l'écran.
        #roitodelete est une collection de trackers qui doivent être supprimés.
        RoiToDelete = []
        #Pour les mauvais ROI dans la collection de trackers. Nous les␣
↪transférons à roitodelete.
        for roi in trackers:
            #Update tracker en même temps pendant ce processus.
            Quality = trackers[roi].update(frame)
            if Quality < 7:
                #append () signifie ajouter le ROI à roitodelete.
                RoiToDelete.append(roi)
        #Démarrer le traitement du ROI dans roitodelete.
        for roi in RoiToDelete:
            #print est également très utile pour la vérification.
            print("Removing roi " + str(roi) + " from list of trackers")
            #pop() est utilisé pour supprimer et renvoyer des éléments du␣
↪tableau. La valeur par défaut est le dernier élément.
            #Ici, notre paramètre est le ROI.
            trackers.pop(roi)

        #Toutes les quinze images,
        #nous effectuerons à nouveau la reconnaissance faciale et confirmerons␣
↪quels visages sont toujours dans l'appareil photo.
        #if (framecounter % 15) == 0 Cela signifie que le reste après avoir␣
↪divisé un nombre par 15 est égal à 0.
        if (framecounter % 15) == 0:

            gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray, 1.05, 3)

            #Cette étape compare le point central de (x, y, w, h) après la␣
↪reconnaissance faciale avec le point central du tracker.
```

```
            #S'il correspond, le tracker actuel est précis. S'il ne correspond␣
↪pas, un tracker est construit pour le nouveau visage.
            #De plus, si l'ancien tracker n'est plus précis, il sera supprimé␣
↪dans RoiToDelete.
            #Bien sûr, si aucun visage n'est détecté, aucun tracker ne sera␣
↪construit.
            for (x,y,w,h) in faces:

                #Default: ne correspond pas
                matchedroi = None
                #calculer le point central
                x_center = x + 0.5 * w
                y_center = y + 0.5 * h

                #loop sur tous les trackers et vérifie si le centre du visage␣
↪se trouve dans la boîte d'un tracker
                for roi in trackers:
                    #dlib n'est pas basé sur python, donc get_position ()␣
↪apparaîtra pour lire les coordonnées.
                    tracker_position = trackers[roi].get_position()
                    t_x = int(tracker_position.left())
                    t_y = int(tracker_position.top())
                    t_w = int(tracker_position.width())
                    t_h = int(tracker_position.height())

                    #calculer le point central
                    t_x_center = t_x + 0.5 * t_w
                    t_y_center = t_y + 0.5 * t_h

                    #check si le centre du visage est dans le rectangle d'une␣
↪région de suivi.
                    #De plus, le point central de la région de suivi doit se␣
↪trouver dans la région détectée en tant que visage.
                    if ( ( t_x <= x_center   <= (t_x + t_w)) and
                         ( t_y <= y_center   <= (t_y + t_h)) and
                         ( x   <= t_x_center <= (x   + w  )) and
                         ( y   <= t_y_center <= (y   + h  ))):
                        #Obtener un match.
                        matchedroi = roi

                #S'il ne correspond pas, il est considéré comme un nouveau␣
↪visage. Nous allons construire un nouveau tracker.
                if matchedroi is None:
                    print("Creating new tracker " + str(faceidcounter))
                    #Utiliser the correlation tracker
                    tracker = dlib.correlation_tracker()
```

```
                    #la boîte de the correlation tracker
                    tracker.start_track(frame, dlib.rectangle(x, y, x+w, y+h))
                    #Ajoutez le tracker à la collection préparée à l'avance.
                    trackers[faceidcounter] = tracker

                    #Et comptez les visages.
                    faceidcounter += 1

        #Enfin, définissez un cadre rectangulaire pour tous les trackers. Cette␣
↪étape est à la fin, mais pas dans la boucle de 15 images.
        for roi in trackers:
            tracker_position = trackers[roi].get_position()
            t_x = int(tracker_position.left())
            t_y = int(tracker_position.top())
            t_w = int(tracker_position.width())
            t_h = int(tracker_position.height())
            cv.rectangle(frame, (t_x, t_y), (t_x + t_w , t_y + t_h), (0, 0,␣
↪255) ,2)

        cv.imshow("Result", frame)
        if cv.waitKey(1) == ord('q'):
            break

    cap.release()
    cv.destroyAllWindows()
```

[4]:
```
main()
```

```
Creating new tracker 0
Creating new tracker 1
Creating new tracker 2
Creating new tracker 3
Creating new tracker 4
Creating new tracker 5
Removing roi 1 from list of trackers
Removing roi 2 from list of trackers
Removing roi 3 from list of trackers
Removing roi 4 from list of trackers
Removing roi 5 from list of trackers
Creating new tracker 6
Removing roi 6 from list of trackers
Creating new tracker 7
Creating new tracker 8
Removing roi 8 from list of trackers
Removing roi 7 from list of trackers
Removing roi 0 from list of trackers
```