Guillaume Faynot

Vision par ordinateur :
Reconnaissance de mains et application au comptage

Avant-propos

Ce document est un compte-rendu de projet informatique traitant sur la vision par ordinateur appliquée au suivi de main, permettant notamment de compter sur les doigts. Celui-ci a été réalisé sur mon temps libre, motivé par des lectures, MOOC et tutoriels vidéos. Le code source est disponible sur mon profil GitHub à l'adresse suivante : https://github.com/guillfay. D'autres projets liés à l'informatique et aux mathématiques appliqués sont également mis en avant, ainsi que des projets académiques menés en classe préparatoire et lors de mes deux premières années d'étude à l'ENSAM.

Introduction

La vision par ordinateur est une branche de l'intelligence artificielle qui se concentre sur la capacité d'un ordinateur à interpréter et comprendre les images et les vidéos. Cela implique l'utilisation d'algorithmes pour analyser les images numériques et extraire des informations utiles telles que les objets, les couleurs, les textures, les formes et les mouvements. La vision par ordinateur peut être utilisée pour de nombreuses applications telles que la reconnaissance de visages, la reconnaissance de la circulation, la surveillance et la vérification de la qualité.

Au regard des « caractéristiques humaines », les sujets de recherche les plus courants sont le visage et les mains. C'est ce dernier point qui nous intéresse ici. L'identification et le suivi des mains peuvent être utiles dans diverses applications, telles que la mise en œuvre du contrôle gestuel, l'interprétation d'un langage à base de signes ou l'amélioration des solutions pour les applications de réalité augmentée. Une première approche peut être de compter les doigts présentés par un utilisateur et créer une base qui peut par la suite évoluer vers une reconnaissance ou traduction de la langue des signes.

Reconnaissance de la main

La reconnaissance de mains dans la vision par ordinateur est réalisée en utilisant des algorithmes de traitement d'images pour détecter et suivre les mains dans des images ou des vidéos en temps réel. Les étapes générales du processus sont les suivantes :

- Pré-traitement: Le premier pas consiste à convertir les images en format numérique pour les préparer à l'analyse. Cela peut inclure des opérations telles que la correction des couleurs, le redimensionnement et le filtrage du bruit.
- Détection de la main: Utilisation d'algorithmes de détection d'objets pour localiser les mains dans l'image. Cela peut inclure la détection de caractéristiques telles que les contours, les couleurs et les textures pour identifier les mains.
- Segmentation de la main: Une fois que les mains ont été détectées, il est nécessaire de séparer les mains des autres objets présents dans l'image. Cela peut être accompli en utilisant des algorithmes de segmentation tels que les algorithmes basés sur les contours ou les algorithmes basés sur les régions.

- Analyse de la forme de la main: Après la segmentation, il est possible d'analyser la forme et la taille de la main pour en extraire des informations supplémentaires. Cela peut inclure la reconnaissance des doigts, la mesure de la distance entre les doigts et la détection des mouvements de la main.
- Classification: La dernière étape consiste à utiliser des algorithmes de classification pour identifier la main en question et associer les gestes ou les mouvements à des actions spécifiques.

Utilisation de la librairie MediaPipe sous Python

MediaPipe est une librairie qui propose des solutions de Machine Learning (ML) personnalisables (comme la détection du visage et des mains, la segmentation des cheveux, le suivi du mouvement, etc.) de manière continue et en direct depuis un flux vidéo.

Un module spécialisé dans la détection de mains, nommé MediaPipe Hands utilise le ML pour détecter la paume de la main, et associer à chaque articulation de doigts un repère (Landmark). Au total, ce sont pour chaque main 21 marqueurs qui sont placés sur des points morphologiquement stratégiques, comme présentés dans la figure ci-dessous.

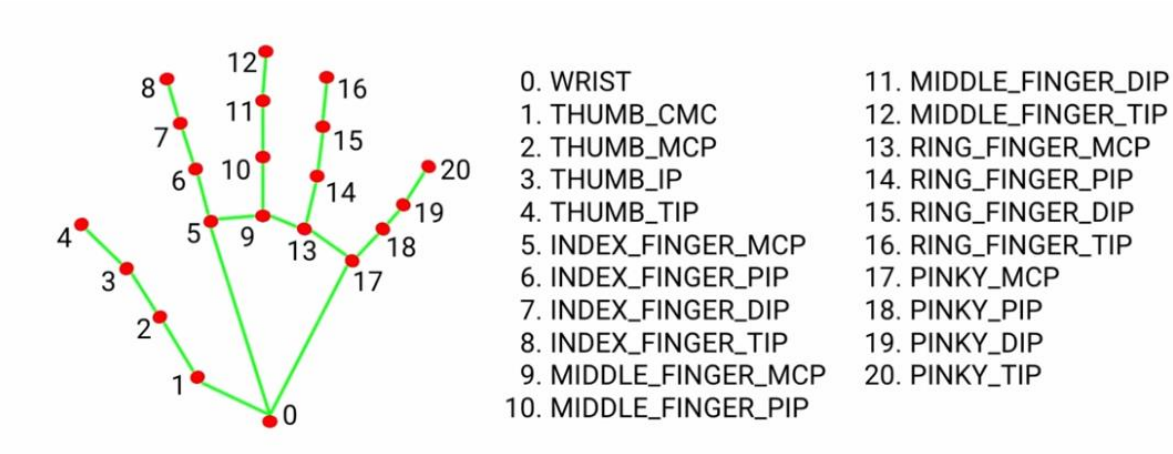


Figure 1 : Landmarks utilisés par MediaPipe Hands

Chaque repère est composé de trois coordonnées (trois dimensions x, y, z) normalisées selon les dimensions du flux vidéo. On ne s'intéresse pas dans ce projet à une quelconque « profondeur » et raisonne selon deux dimensions.

Détails du code

Librairies

Les librairies utilisées dans ce programme sont bien évidemment MediaPipe, cv2.

cv2 est une bibliothèque Python pour la vision par ordinateur et le traitement d'images. Elle est basée sur la bibliothèque OpenCV (Open Source Computer Vision Library), une bibliothèque de traitement d'images en C ++ très populaire et largement utilisée dans le monde de la vision par ordinateur. Cv2 offre alors un accès simplifié, via Python, à la vision par ordinateur.

C'est la fonction « VideoCapture » de cv2 qui est utilisée pour l'acquisition vidéo avec la webcam de l'ordinateur sur lequel est utilisé le programme.

Deux programmes

La solution retenue pour le comptage est d'utiliser deux programmes : l'un nommé « ModuleTrackingMain.py » qui assure la détection des mains avec MediaPipe et renvoie les Landmarks. L'autre nommé « Compteur.py » appelle ce dernier module et interprète ses résultats pour prendre une décision et compter le nombre de doigts, en fonction de la position de chacun des Landmarks.

ModuleTrackingMain.py

Ce programme pour être exécutable contient une fonction main() (pour être exécutable) et une classe detectionMains(), qui contient tous les éléments nécessaire à la reconnaissance des mains

- La fonction __init__() initialise le programme et appelle le module MediaPipe Hands et les fonctions nécessaires à la reconnaissance de mains.

```
def __init__(self,mode=False,max_mains=2,detection_accurracy=0.5,track_ac
curracy=0.5):
        #Valeurs laissées par défaut, deux mains détectables au maximum
        self.mode=mode
        self.max_mains=max_mains
        self.detection_accurracy=detection_accurracy
        self.track_accurracy=track_accurracy

        #Appelle le module MediaPipe Hands
        self.mpMains=mp.solutions.hands
        self.mains=self.mpMains.Hands()
        #Active la fonction qui relie graphiquement les Landmarks
        self.mpTrack=mp.solutions.drawing_utils
```

- La fonction trackMains(), inverse la couleur des images du flux vidéo pour permettre à MediaPipe la reconnaissance de mains, et notamment de déterminer la position des Landmarks. Ceux-ci sont stockés dans la fonction self.resultats.multi_hand_landmarks(). Les Landmarks peuvent alors être tracés en superposition avec le flux vidéo, qui est renvoyé par la fonction.

```
def trackMains(self,img,affichage=True):
        #Inversion des couleurs du flux vidéo
        imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
        #Processs de l'image acquise pour stocker la position des
Landmarks
        self.resultats=self.mains.process(imgRGB)
        #Affichage des Landmarks sur la main
        if self.resultats.multi_hand_landmarks:
            for handlandmarks in self.resultats.multi_hand_landmarks:
                if affichage:
                    self.mpTrack.draw_landmarks(img,handlandmarks,self.m
pMains.HAND_CONNECTIONS)

        #Renvoie l'image du flux vidéo ayant subi le process
        return img
```

- La fonction positionMains renvoie la position normalisée de chacun des Landmarks de chaque main, dans le cas ou il y a une ou deux mains dans la capture vidéo, et trace un cercle en superposition sur les Landmarks : en jaune pour la main 1, en vert pour la main 2.

```
def positionMains(self,img,nb_mains=0,affichage=True):
        # Liste des mains 1 et 2, stockant l'identifiant de chaque
Landmark et ses coordonnées normalisées à la taille du flux vidéo
        lmListe1=[]
        lmListe2=[]

        if self.resultats.multi_hand_landmarks:
            #Cas ou 1 seule main visible
            if len(self.resultats.multi_hand_landmarks)==1:
                maMain=self.resultats.multi_hand_landmarks[nb_mains]
                #Parcours les Landmarks de la main considérée
                for id, lm in enumerate(maMain.landmark):
                    L,l,h=img.shape #Dimensions du cadre vidéo
                    cx,cy=int(lm.x*l),int(lm.y*L)   #Normalisation des
coordonnées du Landmark
                    #print(id,cx,cy)
                    lmListe1.append([id,cx,cy])
                    #Trace un cercle jaune sur les Landmarks
                    if affichage:
                        cv2.circle(img,(cx,cy),7,(0,255,255),cv2.FILLED)

            if len(self.resultats.multi_hand_landmarks)==2:
                maMain1=self.resultats.multi_hand_landmarks[nb_mains]
                maMain2=self.resultats.multi_hand_landmarks[nb_mains+1]
                for id, lm in enumerate(maMain1.landmark):
                    L,l,h=img.shape
                    cx,cy=int(lm.x*l),int(lm.y*L)
```

```
                    #print(id,cx,cy)
                    lmListe1.append([id,cx,cy])
                    if affichage:
                        cv2.circle(img,(cx,cy),7,(0,255,255),cv2.FILLED)
                for id, lm in enumerate(maMain2.landmark):
                    L,l,h=img.shape
                    cx,cy=int(lm.x*l),int(lm.y*L)
                    #print(id,cx,cy)
                    lmListe2.append([id,cx,cy])
                    if affichage:
                        cv2.circle(img,(cx,cy),7,(0,255,0),cv2.FILLED)
            return [lmListe1,lmListe2]
```

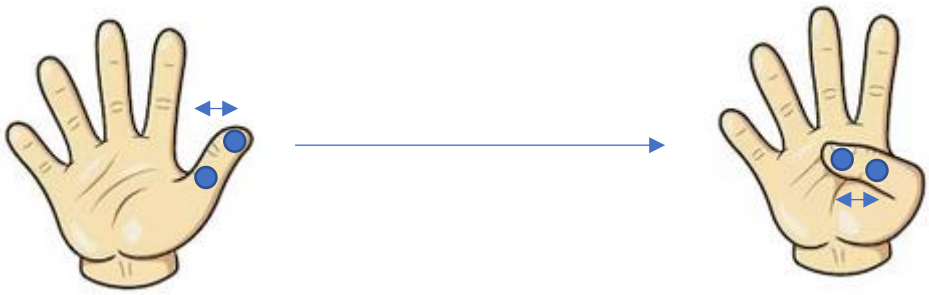
Compteur.py

Ce programme appelle ModuleTrackingMain.py et notamment la classe detectionMain(). Il assure l'acquisition du flux vidéo, le comptage du temps écoulé pour le process d'une image (afin de déterminer la fréquence d'affichage FPS du flux vidéo en sortie), ainsi que le comptage sur les doigts.

La solution retenue pour réaliser le comptage de doigts est de comparer la position du Landmark placé sur le bout de chaque doigts (respectivement 4, 8, 12, 16, 20, cf Figure 1), à la position du Landmarks n-1 s'il s'agit du pouce, ou n-2 pour les autres doigts, c'est-à-dire respectivement 3, 6, 10, 14, 18.

En effet, de manière générale, le comportement le plus fréquent est :

- Pour le pouce, de rabattre celui-ci vers le centre de la paume, ce qui entraine un changement de signe de la différence de position horizontale des Landmarks du doigts considérés



$\Delta x > 0$ *Figure 2 : Décision du comptage pour le pouce* $\Delta x < 0$

- Pour les autres doigts, on les rabat naturellement vers l'intérieur de la paume, ce qui entraine également un changement de signe de la différence de position verticale des Landmarks

$\Delta y > 0$ $\Delta y < 0$

Figure 3 : Décision du comptage des autres doigts

Ainsi, pour chaque acquisition de Landmarks depuis le flux vidéo, on compte pour chaque doigts s'il est levé ou baissé, et la condition de décision est la position du bout du doigt par rapport à la phalange correspondante.

Une boucle while assure la répétition du calcul et on affiche en temps réel sur le flux vidéo le nombre de doigts qui sont comptés :

```
#Définit les identifiants des Landmarks au bout de chaque doigts
boutsDoigtsID=[4,8,12,16,20]
while True:
    success,img=capture.read()
    img=detection.trackMains(img)
    [lmListe1,lmListe2]=detection.positionMains(img)
    # Stocke la liste des doigts comptabilisés pour le comptage
    doigts_tot=[]
    for lmListe in [lmListe1,lmListe2]:
        if len(lmListe)!=0:
            doigts=[]

            #Disjonction de cas entre le pouce et les autres doigts
            # Pouce
            if lmListe[boutsDoigtsID[3]][1]<lmListe[boutsDoigtsID[4]][1]:
                if lmListe[boutsDoigtsID[0]][1]<lmListe[boutsDoigtsID[0]-1][1]:
                    #On comptabilise le doigt s'il est ouvert, ie en ajoutant 1
dans la liste doigts, 0 sinon
                    doigts.append(1)
                else:
                    doigts.append(0)
            else:
                if lmListe[boutsDoigtsID[0]][1]>lmListe[boutsDoigtsID[0]-1][1]:
                    doigts.append(1)
                else:
                    doigts.append(0)

            # Autres doigts
            for i in range(1,5):
                if lmListe[boutsDoigtsID[i]][2]<lmListe[boutsDoigtsID[i]-2][2]:
                    doigts.append(1)
                else:
                    doigts.append(0)
            #Production une liste comptant tous les doigts levés
            doigts_tot+=doigts
    #Compte tous les 1 de la liste doigts_tot, ie les doigts levés
    totalDoigts=doigts_tot.count(1)

    cv2.putText(img,str(totalDoigts),(45,375),cv2.FONT_HERSHEY_PLAIN,10,(255,0,0),
7)
```

Résultats

Ce code, tel que présenté ci-dessus, ne renvoie et ne stocke aucune donnée. Il est toutefois envisageable de stocker des données en prévision d'une acquisition pour du Machine Learning.

Ainsi, une fenêtre vidéo ouverte par le programme se présente comme suit :

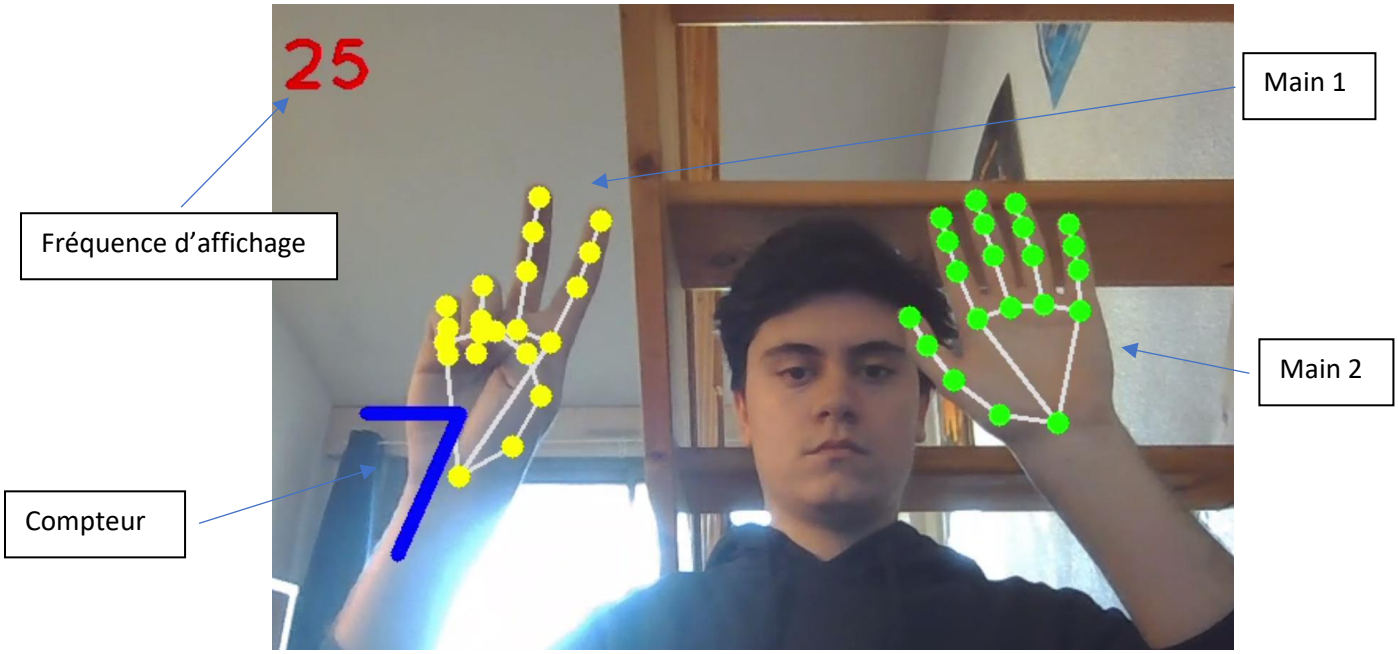


Figure 4 : Affichage vidéo du compteur

On peut voir que la condition de décision est bien respectée, ici pour le pouce :

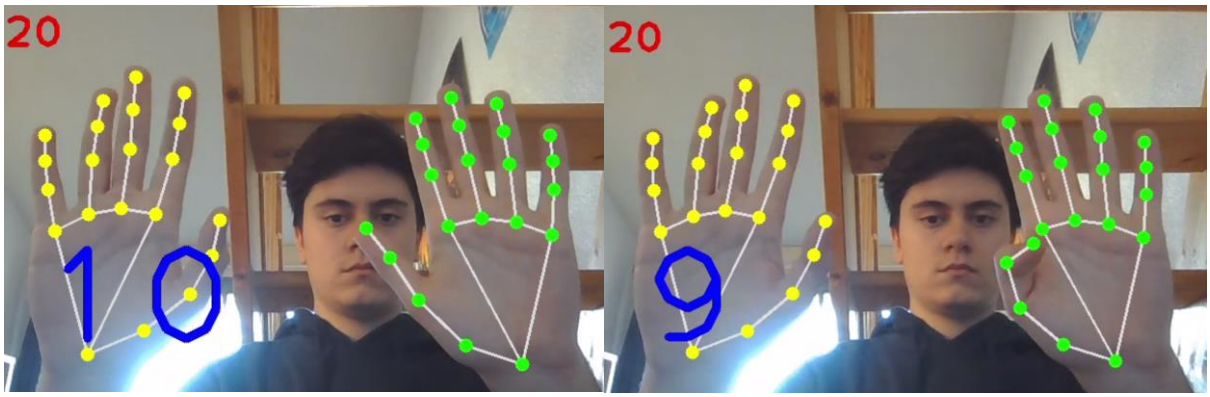


Figure 5 : Comptage du pouce

Le principal avantage de la structure de ce code est la possibilité de compter jusqu'à dix en considérant les deux mains, mais aussi de faire des additions entre les doigts levés avec une grande fiabilité. Par exemple, 4 peut se signer avec 4 doigts d'une même main levée, mais aussi en levant deux doigts de chaque main, comme on peut le voir en figure suivante :

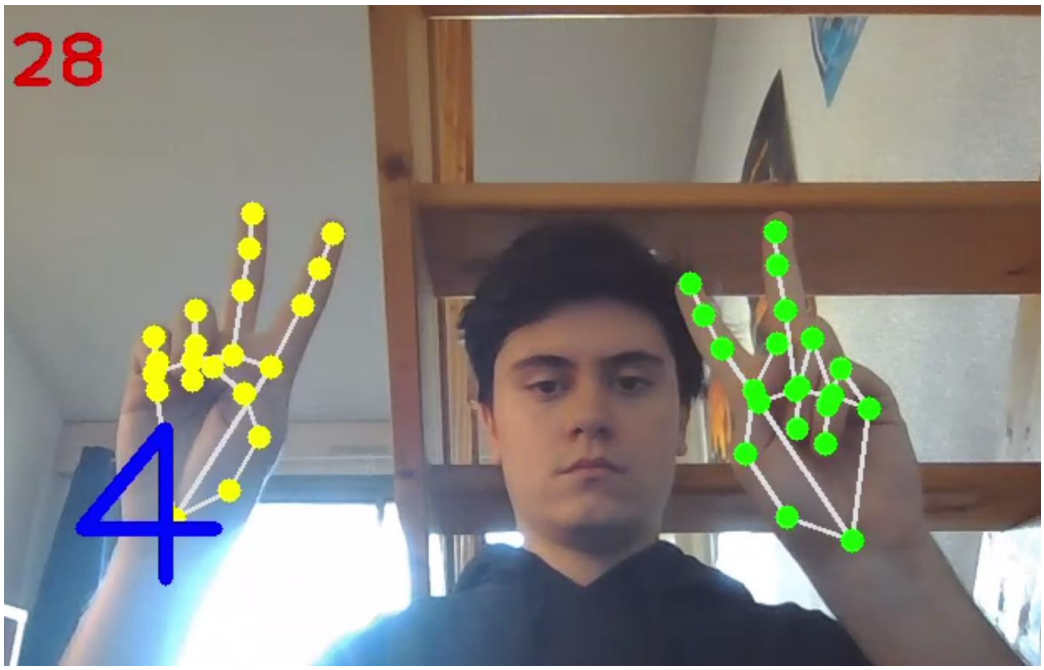


Figure 6 : Le comptage est réalisé avec les deux mains

Notons également que le flux vidéo se stabilise au-delà de 20 images par secondes, ce qui rend cette méthode applicable à la reconnaissance de signes réalisés rapidement, comme pour traduire la langue des signes.

Une démonstration vidéo est disponible sur mon profil GiHub, rubrique Comptage-Vision par ordinateur.

Conclusion et perspectives

Ainsi, ce projet permet de compter de manière fiable et précise les doigts présentés par une personne devant une caméra. Au moyen d'un script Python intégrant initialisation du modèle, acquisition vidéo et prise de décision, un retour vidéo propose un diagnostic du nombre affiché par l'utilisateur, indépendamment de la main montrée.

Ce projet, qui constitue une première approche de Vision par Ordinateur peut trouver de réelles application, dans la traduction assistée par ordinateur de la langue des signes, et ce en temps réel. Au moyen d'une plus grande banque de signes connus, c'est-à-dire de positions de Landmarks, nourrie par des algorithmes de Deep Learning, la création d'un logiciel de traduction est envisageable et accessible avec de faibles de ressources de calcul.

Par ailleurs, MediaPipe et d'autres librairies proposent des modules capables de la reconnaissance du corps humain. Le process post-acquisition vidéo permet de placer des Landmarks, non pas sur les articulation de la main, mais sur les articulations du corps entier. Ceci ouvre la porte de la santé connectée, offrant des moyens de suivis de performance du sportif (évolution, répétition du bon geste) et une solution au suivi de la rééducation : évolution du mouvement lors du réapprentissage post-traumatique par exemple.

Bibliographie

Munir Oudah, Ali Al-Naji, Javaan Chahl, *Hand Gesture Recognition Based on Computer Vision: A Review of Techniques, Journal of Imaging*, 23/07/2020

Marcella Cavalcanti, *Hand Tracking and Finger Counting in Python with MediaPipe, geekering.com*, 21/06/2022, https://www.geekering.com/categories/computer-vision/marcellacavalcanti/hand-tracking-and-finger-counting-in-python-with-mediapipe/

Aman Preet Gulati, *Hand landmarks detection on an image using Mediapipe, analyticsvidhya.com,* 1/03/2022, https://www.analyticsvidhya.com/blog/2022/03/hand-landmarks-detection-on-an-image-using-mediapipe/

Site internet de MediaPipe, https://mediapipe.dev/