Projet ML-IOT

Système de surveillance à distance

Réalisé par:

Katia CHAIBI

Sami KHELLAFI

Omar DJAGHMOUM

Ahmed ABDELHAMID

Année 2020-2021

RECONNAISSANCE FACIAL ET DETECTION DE L'ORIENTATION :

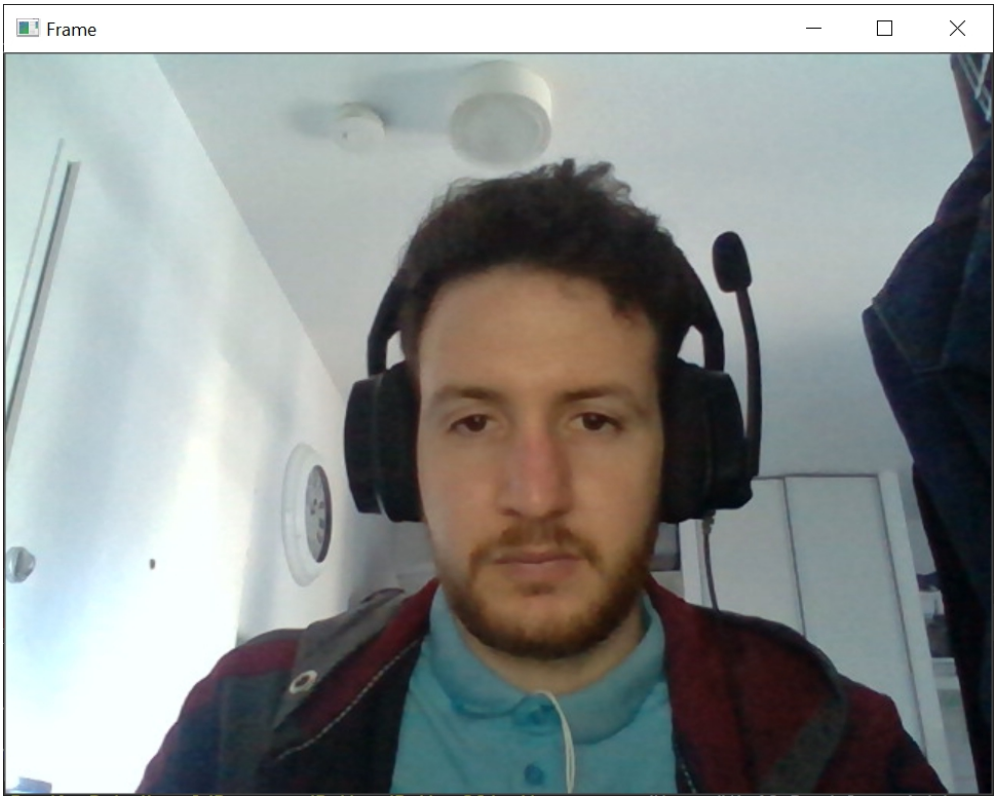
I. Orientation :

1. ***Activation de la caméra :***

   Activation de la caméra à l'aide de la bibliothèque openCV

```
import cv2


cap = cv2.VideoCapture(0)

while True:
     _, frame = cap.read()

     cv2.imshow("Frame", frame)

     key = cv2.waitKey(1)

     if key == 27:
          break
```

Après exécution du code ci-dessous notre caméra se lance comme suit :

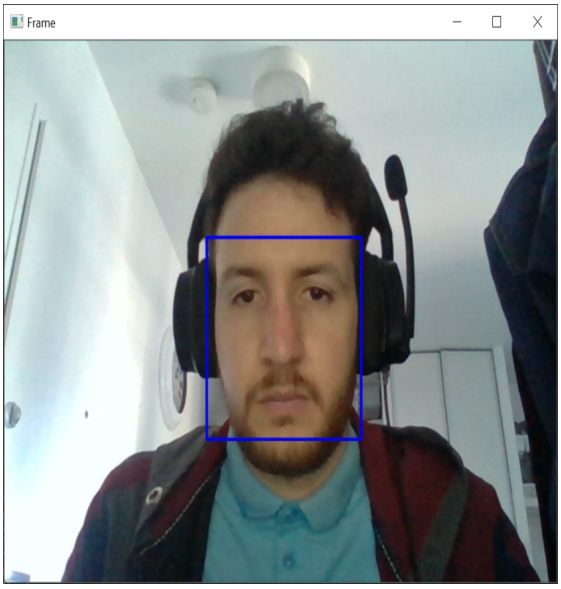


2. Localisation d'un visage dans une image :

Le code suivant va me permettre de localiser des visages en dessinant un rectangle sur chaque visage détecter en utilisant la librairie « dlib » :

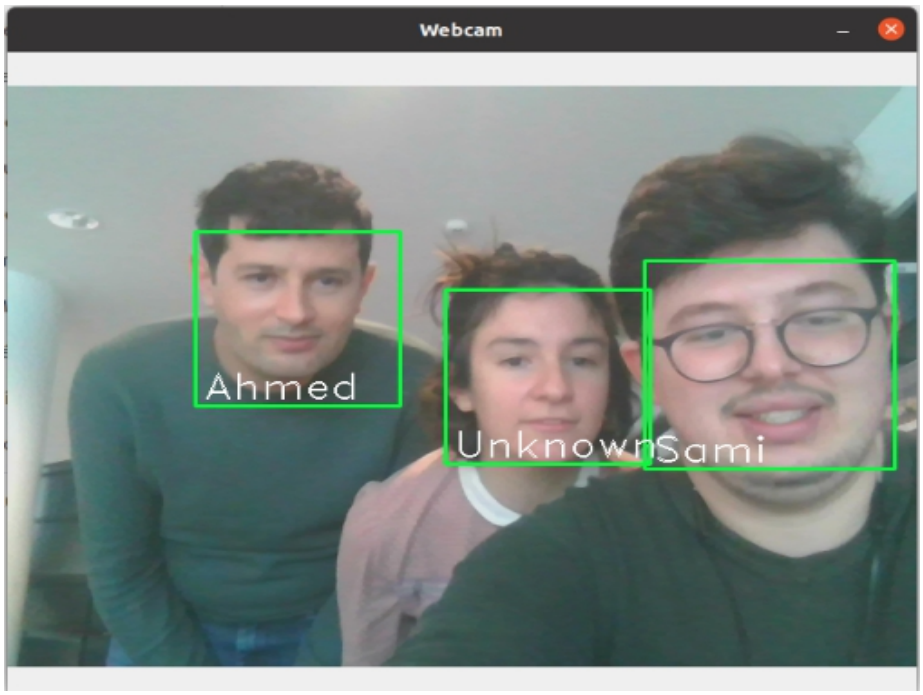
```
import cv2
import numpy as np
import dlib

cap=cv2.VideoCapture(0)
detector=dlib.get_frontal_face_detector()

while True:
    ret, frame=cap.read()
    tickmark=cv2.getTickCount()
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces=detector(gray)
    for face in faces:
        x1=face.left()
        y1=face.top()
        x2=face.right()
        y2=face.bottom()
        cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
    fps=cv2.getTickFrequency()/(cv2.getTickCount()-tickmark)
    cv2.imshow("Frame", frame)
    key=cv2.waitKey(1)&0xFF
    if key==ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

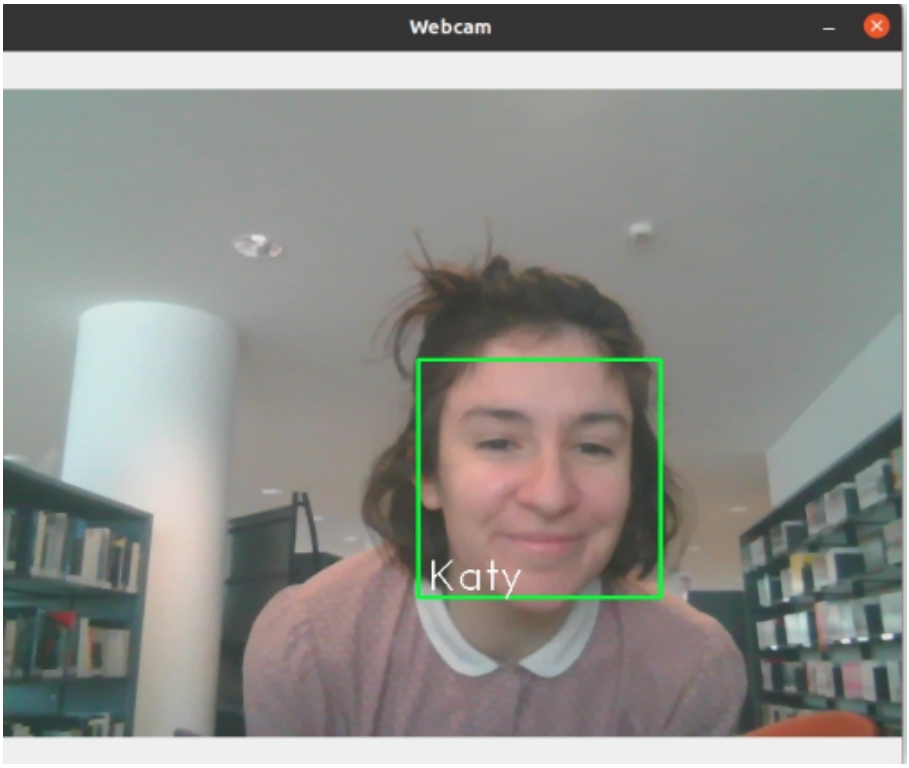
Résultat après exécution :

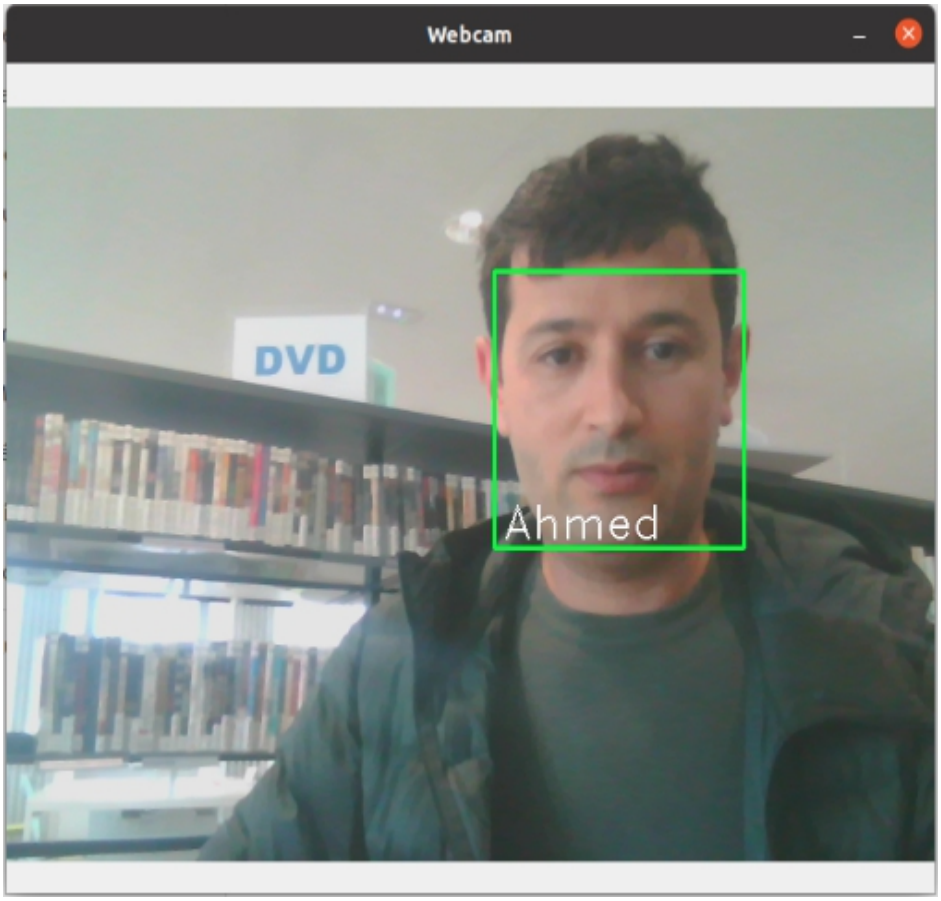


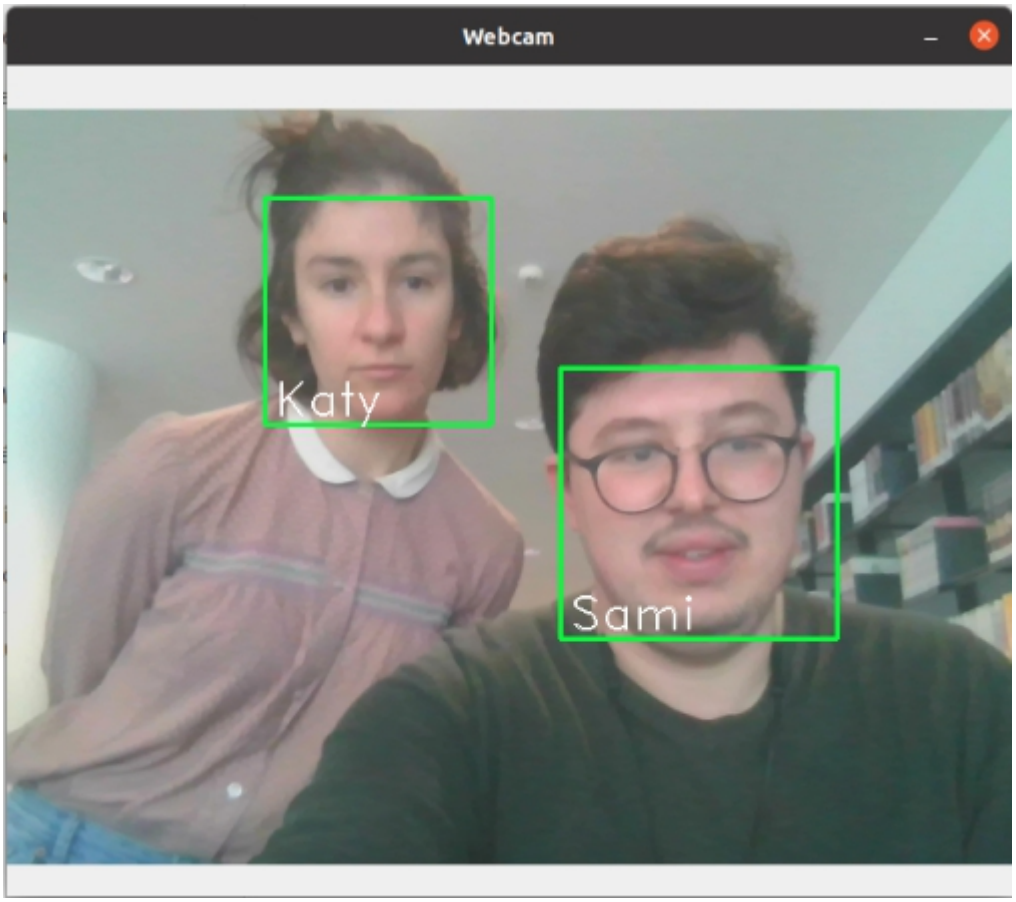
Le code suivant permet d'identifier la personne devant la webcam en récupérant les labels de la base de données.

```
36
37 for nm in my_list:
38     curImg = cv2.imread(f'{path}/{nm}')
39     pictures.append(curImg)
40     names.append(os.path.splitext(nm)[0])
41
```

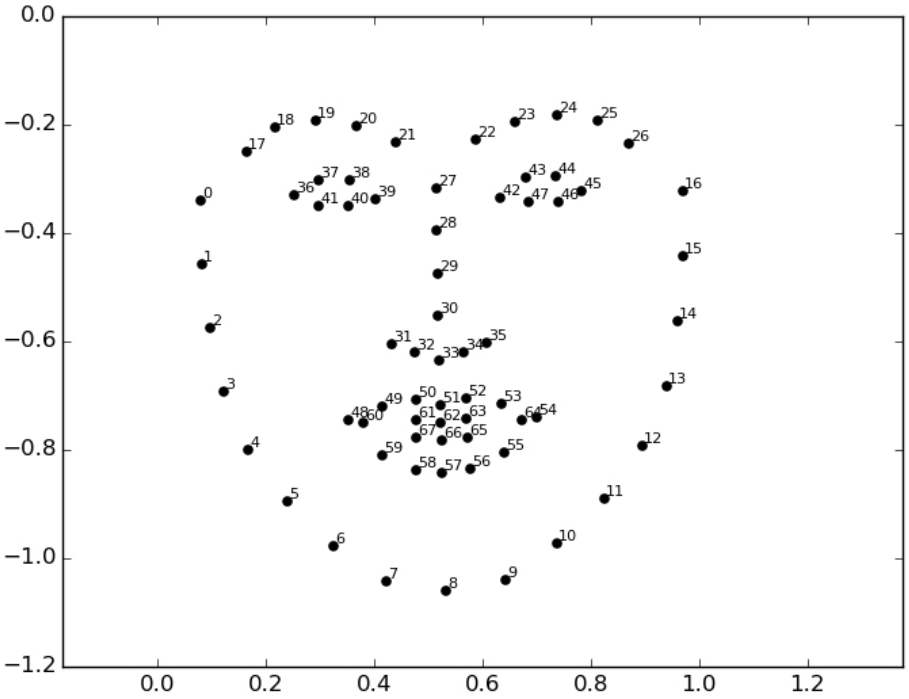



Webcam
Katy


Webcam
DVD
Ahmed


Webcam
Katy
Sami

3. Points qui dessinent le visage :

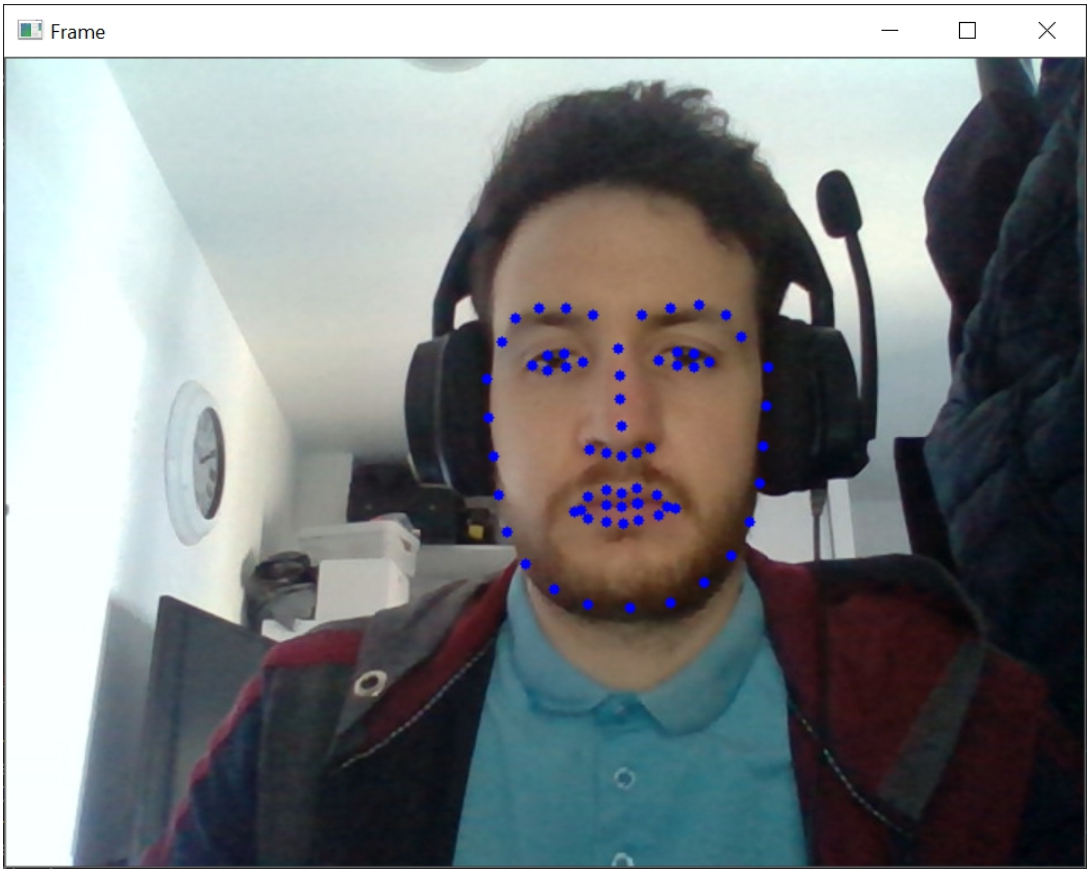


Voici ci-dessous le code qui me permets de dessiner les 69 points sur le visage et pour cela j'ai importé un fichier qui est le résultat d'un apprentissage sur le lien suivant :
https://github.com/davisking/dlib-models
Le code source :

```
import cv2
import numpy as np
import dlib
import math

cap=cv2.VideoCapture(0)
detector=dlib.get_frontal_face_detector()
predictor=dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

while True:
    ret, frame=cap.read()
    tickmark=cv2.getTickCount()
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces=detector(gray)
    for face in faces:
        landmarks=predictor(gray, face)
        i=np.zeros(shape=(frame.shape), dtype=np.uint8)
        for n in range(0, 68):
            x=landmarks.part(n).x
            y=landmarks.part(n).y
            cv2.circle(frame, (x, y), 3, (255, 0, 0), -1)
```

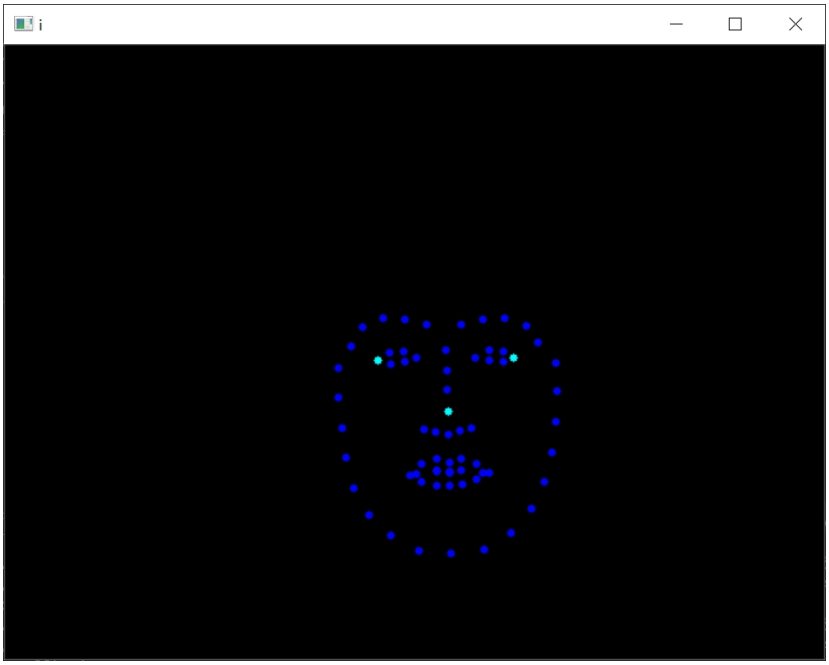
```
            if n==30 or n==36 or n==45:
                cv2.circle(i, (x, y), 3, (255, 255, 0), -1)
            else:
                cv2.circle(i, (x, y), 3, (255, 0, 0), -1)
        #cv2.imshow("i", i)
    fps=cv2.getTickFrequency()/(cv2.getTickCount()-tickmark)
    #cv2.putText(frame, "FPS: {:05.2f}".format(fps), (10, 30), cv2.FONT_HERSHE
Y_PLAIN, 2, (255, 0, 0), 2)
    cv2.imshow("Frame", frame)
    key=cv2.waitKey(1)&0xFF
    if key==ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Après exécution du code j'obtiens le résultat suivant :

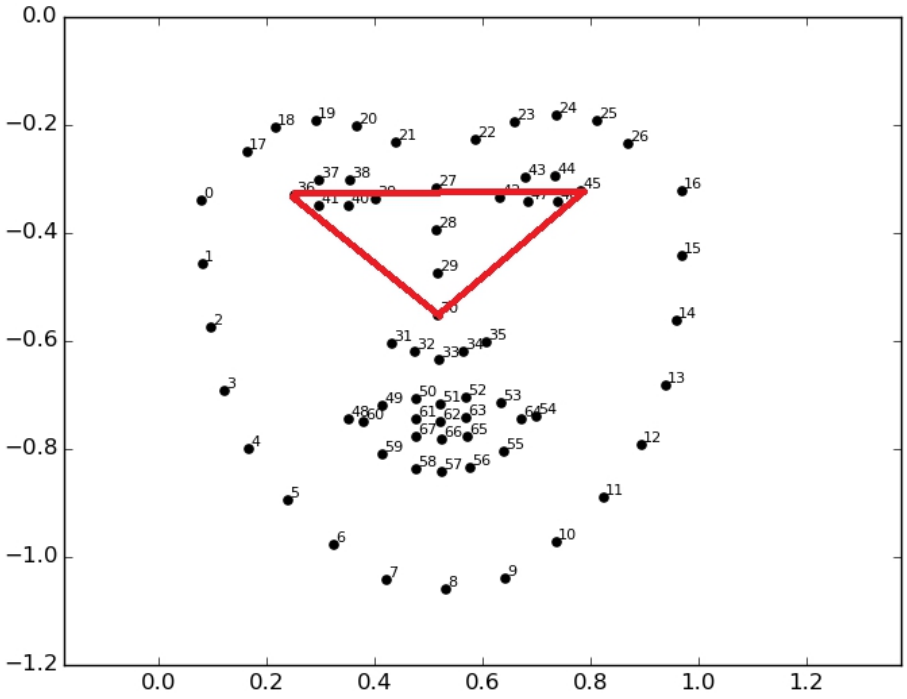


4. *Orientation du visage :*

Pour cette tache je vais me baser sur les trois points « les côtés des yeux ’36 & 45’ et le bout du nez ‘30’ » comme on peut les voir sur la figure suivante en bleue claire :



Ou sur le schéma comme suit :



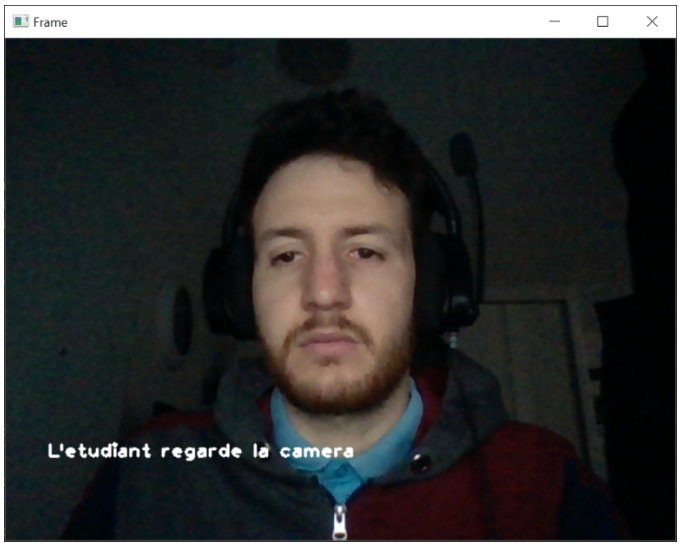
On calcule les différentes distances entre ces trois points dans de différentes positions du visage on pourra alors déterminer l’orientation du visage a chaque fois comme le montre le code ci-dessous :

```
import cv2
import numpy as np
import dlib
import math

cap=cv2.VideoCapture(0)

detector=dlib.get_frontal_face_detector()
predictor=dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

while True:
    ret, frame=cap.read()
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces=detector(gray)
    if faces is not None:
        i=np.zeros(shape=(frame.shape), dtype=np.uint8)
    for face in faces:
        landmarks=predictor(gray, face)

        d_eyes=math.sqrt(math.pow(landmarks.part(36).x-
landmarks.part(45).x, 2)+math.pow(landmarks.part(36).y-
landmarks.part(45).y, 2))
        d1=math.sqrt(math.pow(landmarks.part(36).x-
landmarks.part(30).x, 2)+math.pow(landmarks.part(36).y-
landmarks.part(30).y, 2))
        d2=math.sqrt(math.pow(landmarks.part(45).x-
landmarks.part(30).x, 2)+math.pow(landmarks.part(45).y-
landmarks.part(30).y, 2))
        coeff=d1+d2

        a1=int(250*(landmarks.part(36).y-landmarks.part(45).y)/coeff)
        a2=int(250*(d1-d2)/coeff)
        cosb=min((math.pow(d2, 2)-
math.pow(d1, 2)+math.pow(d_eyes, 2))/(2*d2*d_eyes), 1)
        a3=int(250*(d2*math.sin(math.acos(cosb))-coeff/4)/coeff)

        for n in range(0, 68):
            x=landmarks.part(n).x
            y=landmarks.part(n).y
            if n==30 or n==36 or n==45:
                cv2.circle(i, (x, y), 3, (255, 255, 0), -1)
            else:
                cv2.circle(i, (x, y), 3, (255, 0, 0), -1)
        #print("{:+05d} {:+05d} {:+05d}".format(a1, a2, a3))
        flag=1
        txt="L'etudiant regarde "
        if a2<-40:
            txt+="a droite "
```

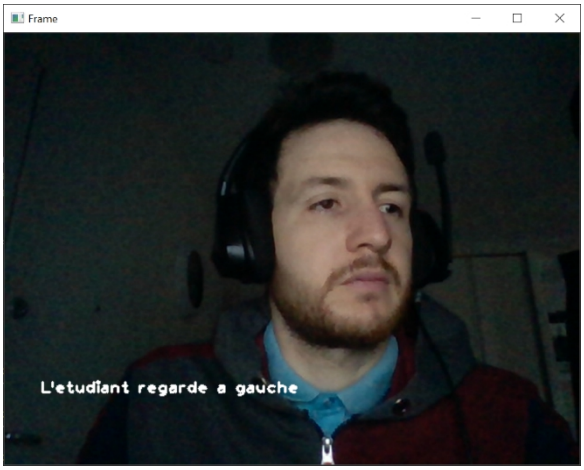
```
            flag=0
        if a2>40:
            txt+="a gauche "
            flag=0
        if a3<-10:
            txt+="en haut "
            flag=0
        if a3>10:
            txt+="en bas "
            flag=0
        if flag:
            txt+="la camera "
        if a1<-40:
            txt+="et incline la tete a gauche "
        if a1>40:
            txt+="et incline la tete a droite "
        cv2.putText(frame, txt, (40, 400), cv2.FONT_HERSHEY_PLAIN, 1.2, (255,
255, 255), 2)
    cv2.imshow("Frame", frame)
    key=cv2.waitKey(1)&0xFF
    if key==ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

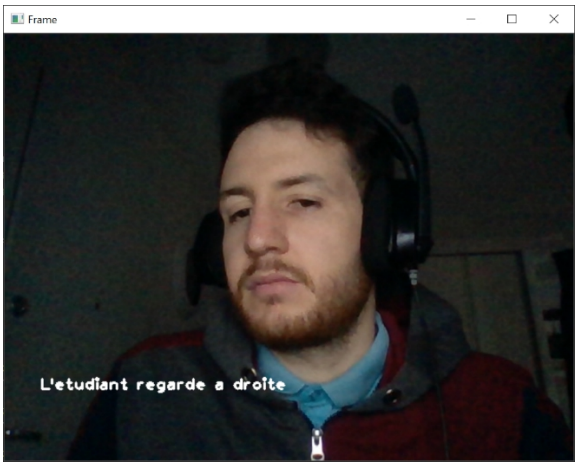
Après exécution :

- Quand on regarde droit vers la caméra :

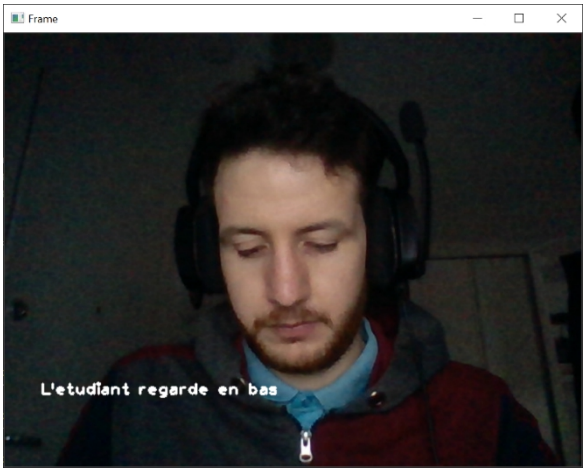


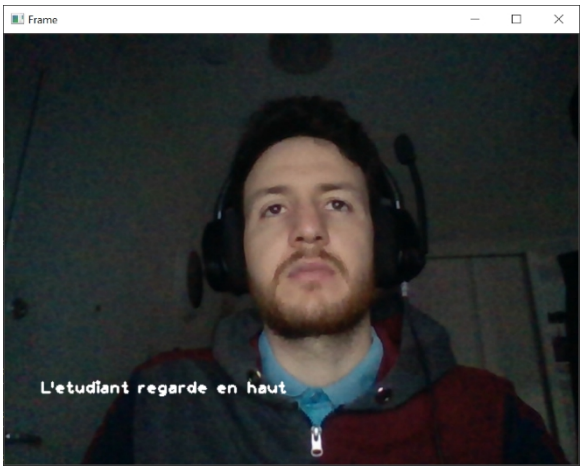
- Quand on regarde à gauche ou à droite :



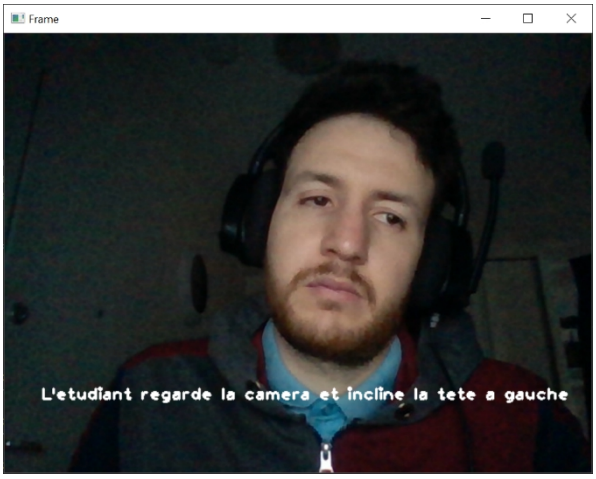


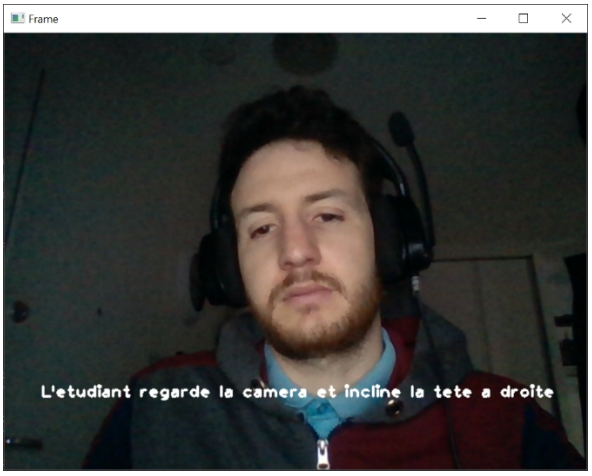
- Quand on regarde en haut ou en bas :



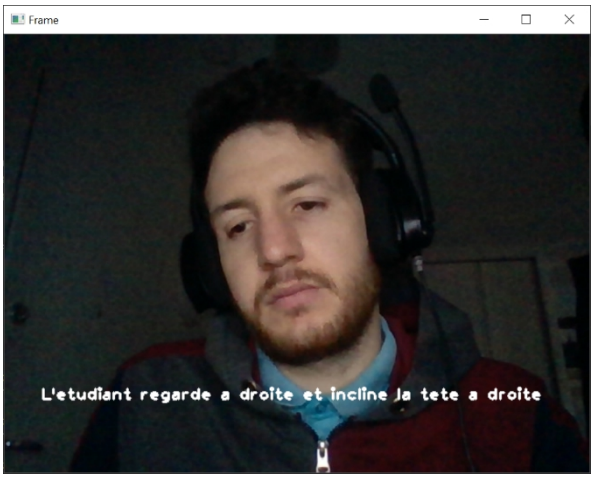


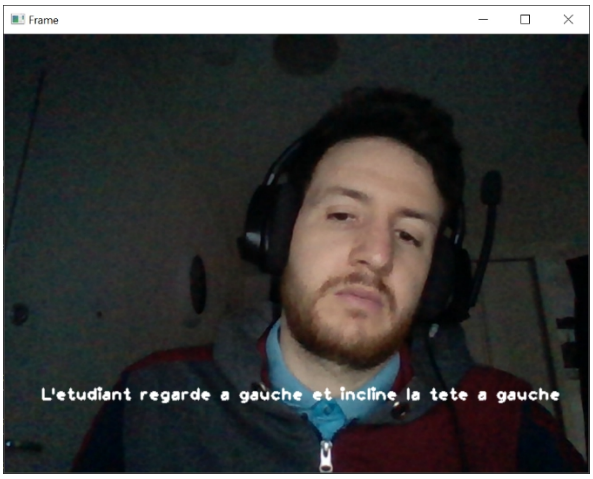
- Quand on regarde la camera & on incline la tête :

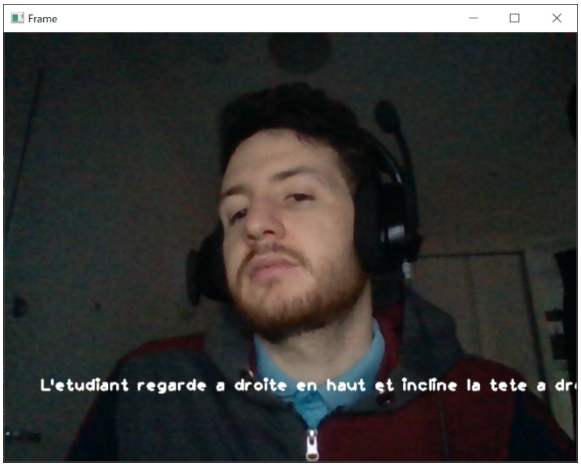


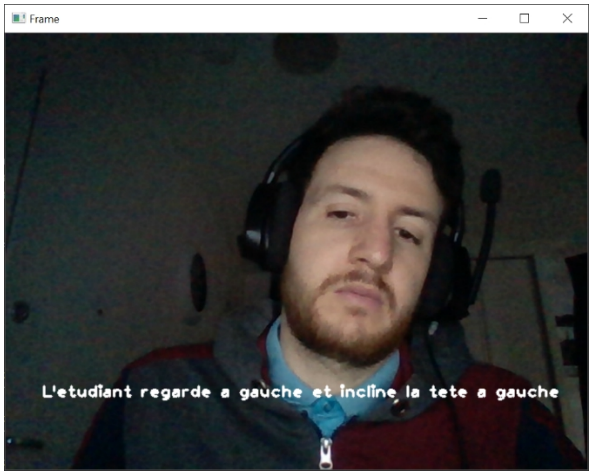


- Quand on regarde a droite ou a gauche & on incline la tête :

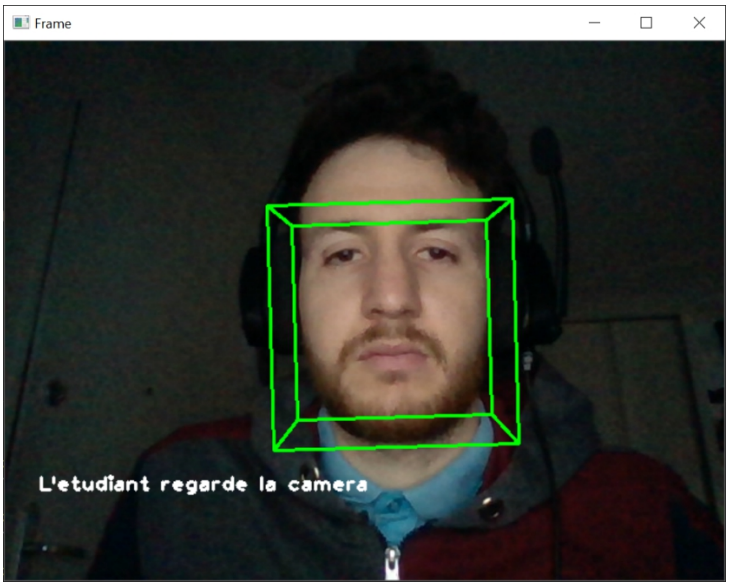








Et pour avoir plus de précision j'ai rajouté un cube qui suit exactement l'orientation du visage comme le montre la figure ci-dessous :



Et voici pour le code qui me permet de faire ça :

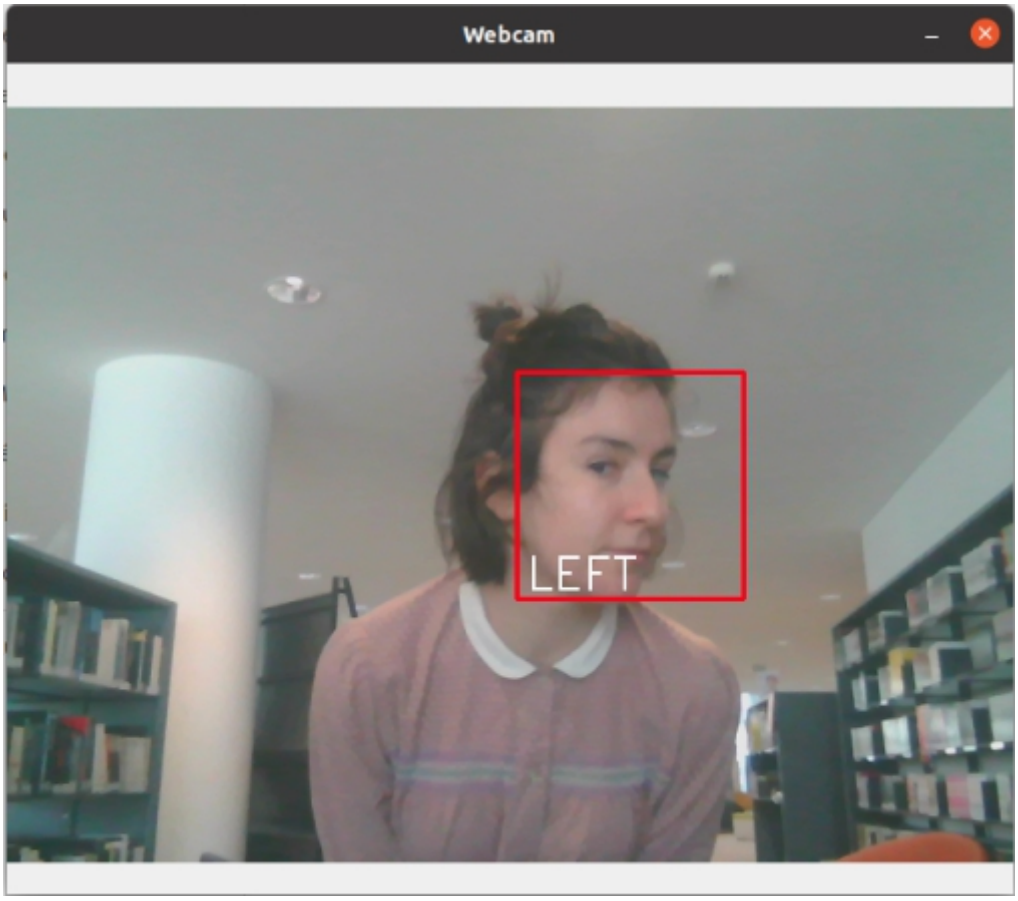
```
import cv2
import numpy as np
import dlib
import math

cap=cv2.VideoCapture(0)
#cap=cv2.VideoCapture("debat.webm")

detector=dlib.get_frontal_face_detector()
predictor=dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

def tr(c, o, coeff):
    return(int((c-o)*coeff)+o)

def cube(image, pt1, pt2, a1, a2, a3):
    color=(0, 255, 0)
    epaisseur=2
    offset=1.6
    offset2=2

    d_eyes=math.sqrt(math.pow(landmarks.part(36).x-
landmarks.part(45).x, 2)+math.pow(landmarks.part(36).y-
landmarks.part(45).y, 2))

    ox1=int((-(pt2.y-pt1.y)+pt2.x-pt1.x)/2)+pt1.x
    oy1=int(((pt2.x-pt1.x+pt2.y)-pt1.y)/2)+pt1.y

    cv2.line(image,
        (tr(pt1.x, ox1, offset), tr(pt1.y, oy1, offset)),
        (tr(pt2.x, ox1, offset), tr(pt2.y, oy1, offset)),
        color, epaisseur)
    cv2.line(image,
        (tr(pt2.x, ox1, offset), tr(pt2.y, oy1, offset)),
        (tr(-(pt2.y-pt1.y)+pt2.x, ox1, offset), tr(pt2.x-
pt1.x+pt2.y, oy1, offset)),
        color, epaisseur)
    cv2.line(image,
        (tr(-(pt2.y-pt1.y)+pt2.x, ox1, offset), tr(pt2.x-
pt1.x+pt2.y, oy1, offset)),
        (tr(-(pt2.y-pt1.y)+pt1.x, ox1, offset), tr(pt2.x-
pt1.x+pt1.y, oy1, offset)),
        color, epaisseur)
    cv2.line(image,
        (tr(-(pt2.y-pt1.y)+pt1.x, ox1, offset), tr(pt2.x-
pt1.x+pt1.y, oy1, offset)),
        (tr(pt1.x, ox1, offset), tr(pt1.y, oy1, offset)),
        color, epaisseur)
```

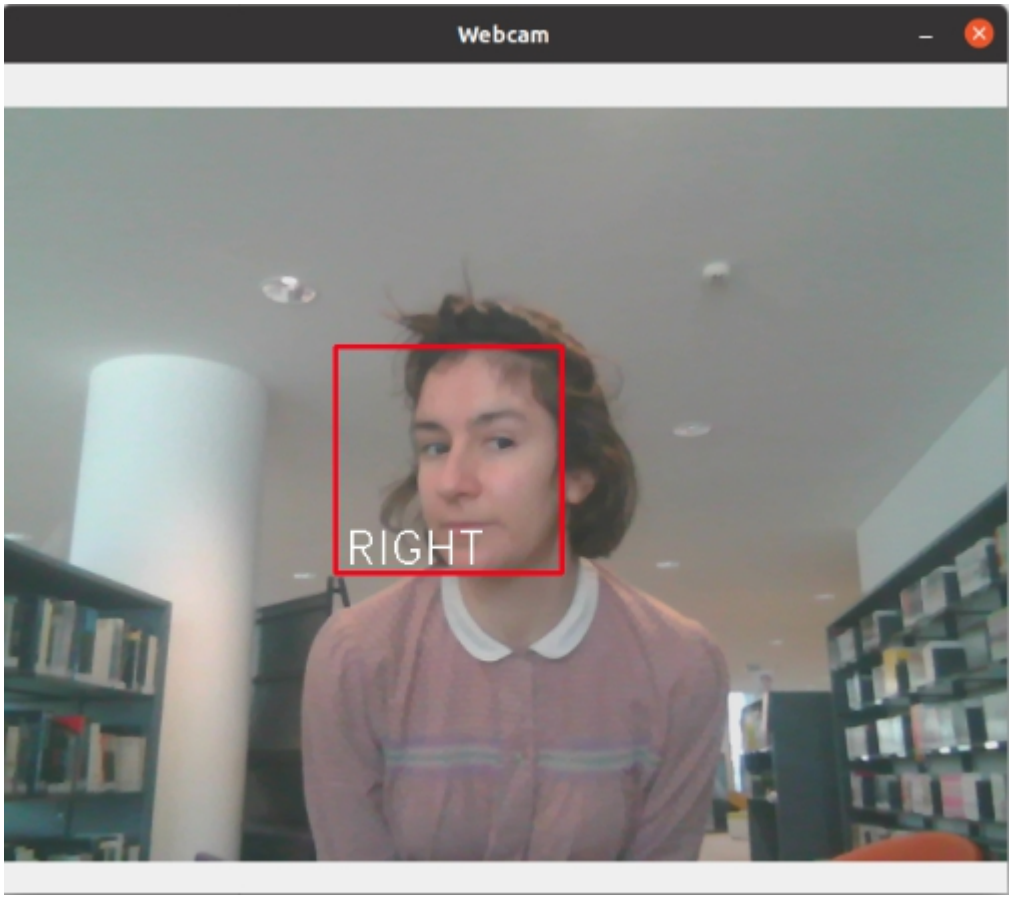
```
    ox2=int((-(pt2.y-pt1.y)+pt2.x-pt1.x)/2)+pt1.x+int(a2)
    oy2=int(((pt2.x-pt1.x+pt2.y)-pt1.y)/2)+pt1.y+int(a3)

    cv2.line(image,
        (tr(pt1.x+a2, ox2, offset2), tr(pt1.y+a3, oy2, offset2)),
        (tr(pt2.x+a2, ox2, offset2), tr(pt2.y+a3, oy2, offset2)),
        color, epaisseur)
    cv2.line(image,
        (tr(pt2.x+a2, ox2, offset2), tr(pt2.y+a3, oy2, offset2)),
        (tr(-(pt2.y-pt1.y)+pt2.x+a2, ox2, offset2), tr(pt2.x-
pt1.x+pt2.y+a3, oy2, offset2)),
        color, epaisseur)
    cv2.line(image,
        (tr(-(pt2.y-pt1.y)+pt2.x+a2, ox2, offset2), tr(pt2.x-
pt1.x+pt2.y+a3, oy2, offset2)),
        (tr(-(pt2.y-pt1.y)+pt1.x+a2, ox2, offset2), tr(pt2.x-
pt1.x+pt1.y+a3, oy2, offset2)),
        color, epaisseur)
    cv2.line(image,
        (tr(-(pt2.y-pt1.y)+pt1.x+a2, ox2, offset2), tr(pt2.x-
pt1.x+pt1.y+a3, oy2, offset2)),
        (tr(pt1.x+a2, ox2, offset2), tr(pt1.y+a3, oy2, offset2)),
        color, epaisseur)

    cv2.line(image,
        (tr(pt1.x, ox1, offset), tr(pt1.y, oy1, offset)),
        (tr(pt1.x+a2, ox2, offset2), tr(pt1.y+a3, oy2, offset2)),
        color, epaisseur)
    cv2.line(image,
        (tr(pt2.x, ox1, offset), tr(pt2.y, oy1, offset)),
        (tr(pt2.x+a2, ox2, offset2), tr(pt2.y+a3, oy2, offset2)),
        color, epaisseur)
    cv2.line(image,
        (tr(-(pt2.y-pt1.y)+pt2.x, ox1, offset), tr(pt2.x-
pt1.x+pt2.y, oy1, offset)),
        (tr(-(pt2.y-pt1.y)+pt2.x+a2, ox2, offset2), tr(pt2.x-
pt1.x+pt2.y+a3, oy2, offset2)),
        color, epaisseur)
    cv2.line(image,
        (tr(-(pt2.y-pt1.y)+pt1.x, ox1, offset), tr(pt2.x-
pt1.x+pt1.y, oy1, offset)),
        (tr(-(pt2.y-pt1.y)+pt1.x+a2, ox2, offset2), tr(pt2.x-
pt1.x+pt1.y+a3, oy2, offset2)),
        color, epaisseur)

while True:
    ret, frame=cap.read()
    tickmark=cv2.getTickCount()
```

```
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces=detector(gray)
    for face in faces:
        x1=face.left()
        y1=face.top()
        x2=face.right()
        y2=face.bottom()

        landmarks=predictor(gray, face)

        d_eyes=math.sqrt(math.pow(landmarks.part(36).x-
landmarks.part(45).x, 2)+math.pow(landmarks.part(36).y-
landmarks.part(45).y, 2))
        d1=math.sqrt(math.pow(landmarks.part(36).x-
landmarks.part(30).x, 2)+math.pow(landmarks.part(36).y-
landmarks.part(30).y, 2))
        d2=math.sqrt(math.pow(landmarks.part(45).x-
landmarks.part(30).x, 2)+math.pow(landmarks.part(45).y-
landmarks.part(30).y, 2))
        coeff=d1+d2

        a1=int(250*(landmarks.part(36).y-landmarks.part(45).y)/coeff)
        a2=int(250*(d1-d2)/coeff)
        cosb=min((math.pow(d2, 2)-
math.pow(d1, 2)+math.pow(d_eyes, 2))/(2*d2*d_eyes), 1)
        a3=int(250*(d2*math.sin(math.acos(cosb))-coeff/4)/coeff)

        cube(frame, landmarks.part(36), landmarks.part(45), a1, a2, a3)
        flag=1
        txt="L'etudiant regarde "
        if a2<-40:
            txt+="a droite "
            flag=0
        if a2>40:
            txt+="a gauche "
            flag=0
        if a3<-10:
            txt+="en haut "
            flag=0
        if a3>10:
            txt+="en bas "
            flag=0
        if flag:
            txt+="la camera "
        if a1<-40:
            txt+="et incline la tete a gauche "
        if a1>40:
            txt+="et incline la tete a droite "
```

```
        cv2.putText(frame, txt, (30, 400), cv2.FONT_HERSHEY_PLAIN, 1.2, (255,
255, 255), 2)
    cv2.imshow("Frame", frame)

    key=cv2.waitKey(1)&0xFF
    if key==ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Voici un autre exemple de détection de mouvement du visage d'une personne:




Webcam
RIGHT

II. Reconnaissance faciale :

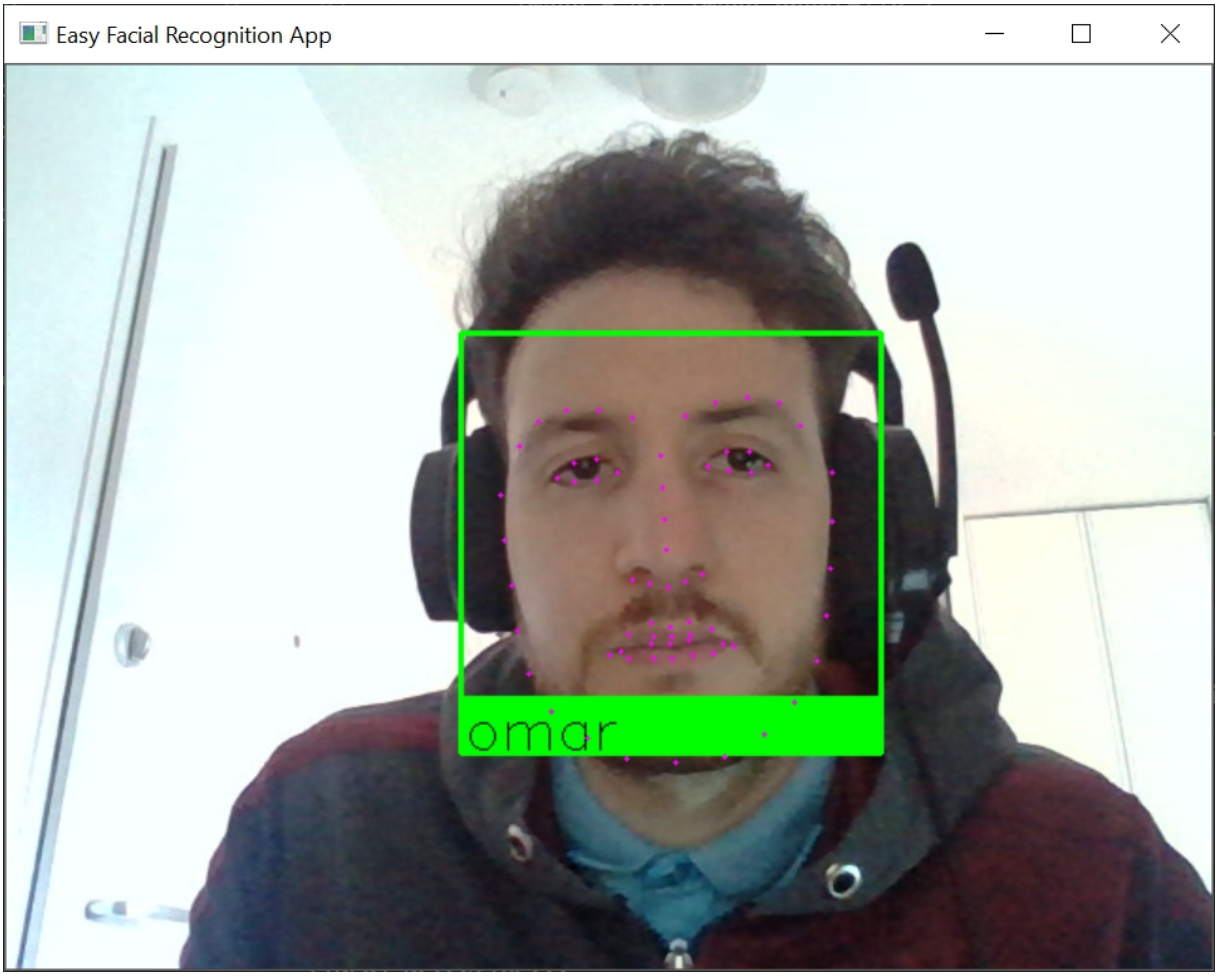
Pour cette tache j'ai utilisé des modèles prés entrainer que j'ai récupéré sur ce lien https://github.com/davisking/dlib-models et voilà mon code :

```
import cv2
import dlib
import PIL.Image
import numpy as np
from imutils import face_utils
import argparse
from pathlib import Path
import os
import ntpath

parser = argparse.ArgumentParser(description='Easy Facial Recognition App')
parser.add_argument('-i', '--
input', type=str, required=True, help='directory of input known faces')

print('[INFO] Starting System...')
print('[INFO] Importing pretrained model..')
pose_predictor_68_point = dlib.shape_predictor("pretrained_model/shape_predict
or_68_face_landmarks.dat")
pose_predictor_5_point = dlib.shape_predictor("pretrained_model/shape_predicto
r_5_face_landmarks.dat")
face_encoder = dlib.face_recognition_model_v1("pretrained_model/dlib_face_reco
gnition_resnet_model_v1.dat")
face_detector = dlib.get_frontal_face_detector()
print('[INFO] Importing pretrained model..')


def transform(image, face_locations):
    coord_faces = []
    for face in face_locations:
        rect = face.top(), face.right(), face.bottom(), face.left()
        coord_face = max(rect[0], 0), min(rect[1], image.shape[1]), min(rect[2]
, image.shape[0]), max(rect[3], 0)
        coord_faces.append(coord_face)
    return coord_faces


def encode_face(image):
    face_locations = face_detector(image, 1)
    face_encodings_list = []
    landmarks_list = []
```

```
    for face_location in face_locations:
        # DETECT FACES
        shape = pose_predictor_68_point(image, face_location)
        face_encodings_list.append(np.array(face_encoder.compute_face_descript
or(image, shape, num_jitters=1)))
        # GET LANDMARKS
        shape = face_utils.shape_to_np(shape)
        landmarks_list.append(shape)
    face_locations = transform(image, face_locations)
    return face_encodings_list, face_locations, landmarks_list


def easy_face_reco(frame, known_face_encodings, known_face_names):
    rgb_small_frame = frame[:, :, ::-1]
    # ENCODING FACE
    face_encodings_list, face_locations_list, landmarks_list = encode_face(rgb
_small_frame)
    face_names = []
    for face_encoding in face_encodings_list:
        if len(face_encoding) == 0:
            return np.empty((0))
        # CHECK DISTANCE BETWEEN KNOWN FACES AND FACES DETECTED
        vectors = np.linalg.norm(known_face_encodings - face_encoding, axis=1)
        tolerance = 0.6
        result = []
        for vector in vectors:
            if vector <= tolerance:
                result.append(True)
            else:
                result.append(False)
        if True in result:
            first_match_index = result.index(True)
            name = known_face_names[first_match_index]
        else:
            name = "Unknown"
        face_names.append(name)

    for (top, right, bottom, left), name in zip(face_locations_list, face_name
s):
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
        cv2.rectangle(frame, (left, bottom -
 30), (right, bottom), (0, 255, 0), cv2.FILLED)
        cv2.putText(frame, name, (left + 2, bottom -
 2), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 1)

    for shape in landmarks_list:
        for (x, y) in shape:
            cv2.circle(frame, (x, y), 1, (255, 0, 255), -1)
```

```
if __name__ == '__main__':
    args = parser.parse_args()

    print('[INFO] Importing faces...')
    face_to_encode_path = Path(args.input)
    files = [file_ for file_ in face_to_encode_path.rglob('*.jpg')]

    for file_ in face_to_encode_path.rglob('*.png'):
        files.append(file_)
    if len(files)==0:
        raise ValueError('No faces detect in the directory: {}'.format(face_to
_encode_path))
    known_face_names = [os.path.splitext(ntpath.basename(file_))[0] for file_
in files]

    known_face_encodings = []
    for file_ in files:
        image = PIL.Image.open(file_)
        image = np.array(image)
        face_encoded = encode_face(image)[0][0]
        known_face_encodings.append(face_encoded)

    print('[INFO] Faces well imported')
    print('[INFO] Starting Webcam...')
    video_capture = cv2.VideoCapture(0)
    print('[INFO] Webcam well started')
    print('[INFO] Detecting...')
    while True:
        ret, frame = video_capture.read()
        easy_face_reco(frame, known_face_encodings, known_face_names)
        cv2.imshow('Easy Facial Recognition App', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    print('[INFO] Stopping System')
    video_capture.release()
    cv2.destroyAllWindows()
```

Et voici le résultat d’exécution dans la figure ci-dessous :


Easy Facial Recognition App
omar

Reconnaissance des objets:

Notre système est capable de détecter des objets autour de la personne, dans notre cas ça sert à détecter la présence d'objets de triche ou de non-triche autour d'un étudiant pendant l'examen.

La liste d'objets reconnaissables sont stockés dans la base de données, la détection dobjets se fait via YOLOv3.

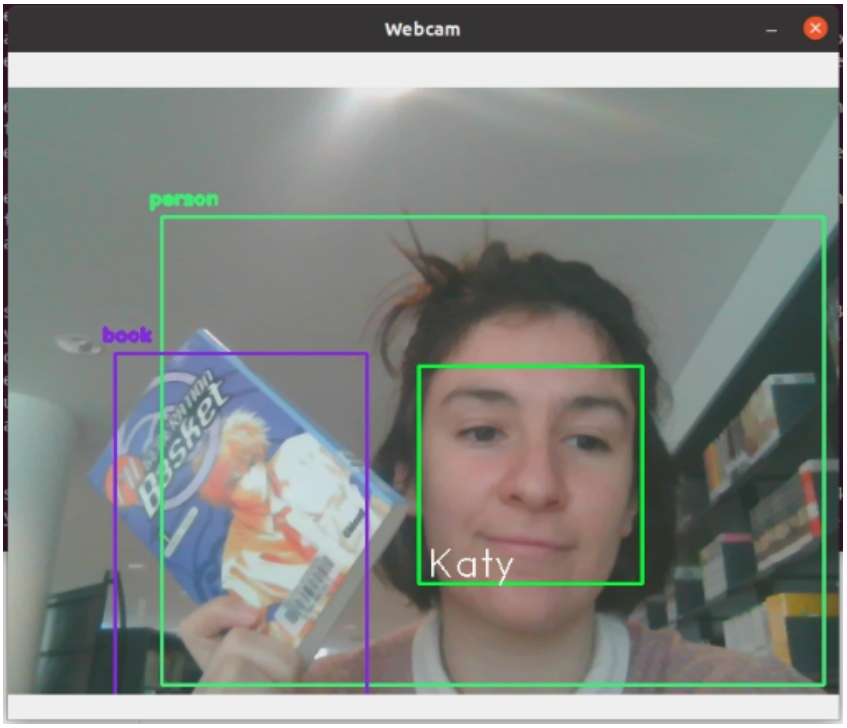
```
58
59 for out in outs:
60     for detection in out:
61         scores = detection[5:]
62         class_id = np.argmax(scores)
63         confidence = scores[class_id]
64         if confidence > 0.5:
65             center_x = int(detection[0] * Width)
66             center_y = int(detection[1] * Height)
67             w = int(detection[2] * Width)
68             h = int(detection[3] * Height)
69             x = center_x - w / 2
70             y = center_y - h / 2
71             class_ids.append(class_id)
72             confidences.append(float(confidence))
73             boxes.append([x, y, w, h])
74
75
76 indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
77
78 for i in indices:
79     i = i[0]
80     box = boxes[i]
81     x = box[0]
82     y = box[1]
83     w = box[2]
84     h = box[3]
85     draw_prediction(image, class_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))
86
87 cv2.imshow("object detection", image)
88 cv2.waitKey()
89
90 cv2.imwrite("object-detection.jpg", image)
91 cv2.destroyAllWindows()
```

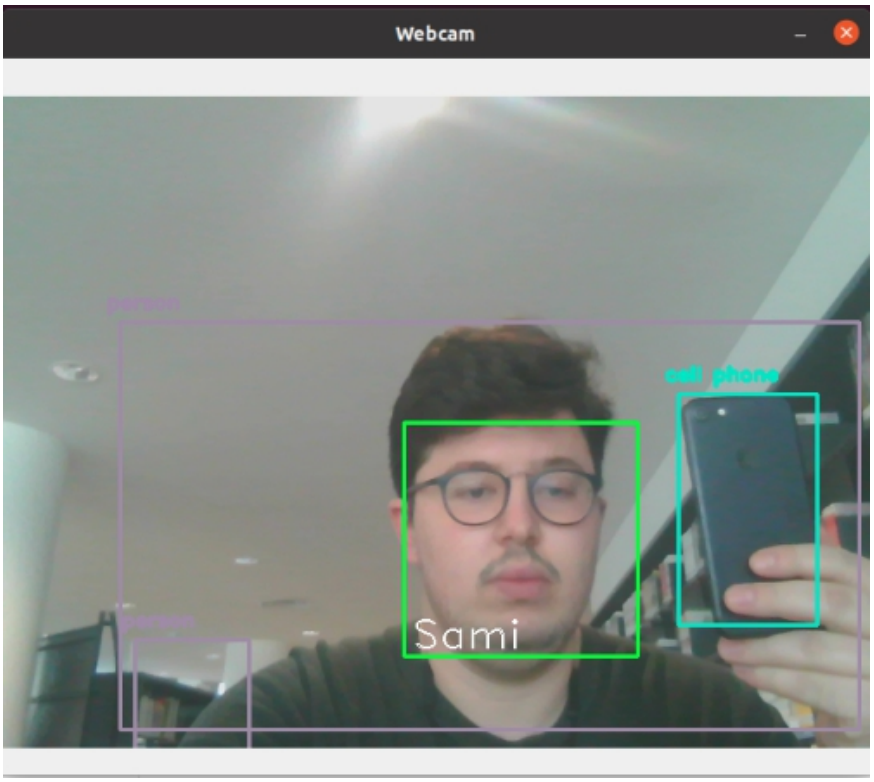
```
30
31 image = cv2.imread(args.image)
32
33 Width = image.shape[1]
34 Height = image.shape[0]
35 scale = 0.00392
36
37 classes = None
38
39 with open(args.classes, 'r') as f:
40     classes = [line.strip() for line in f.readlines()]
41
42 COLORS = np.random.uniform(0, 255, size=(len(classes), 3))
43
44 net = cv2.dnn.readNet(args.weights, args.config)
45
46 blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)
47
48 net.setInput(blob)
49
50 outs = net.forward(get_output_layers(net))
51
52 class_ids = []
53 confidences = []
54 boxes = []
55 conf_threshold = 0.5
56 nms_threshold = 0.4
57
```

Base de données:

```
 1 person
 2 bicycle
 3 car
 4 motorcycle
 5 airplane
 6 bus
 7 train
 8 truck
 9 boat
10 traffic light
11 fire hydrant
12 stop sign
13 parking meter
14 bench
15 bird
16 cat
17 dog
18 horse
19 sheep
20 cow
21 elephant
22 bear
23 zebra
24 giraffe
```

Le code ci-dessus permet de détecter les objets de fraude (livre, téléphone,...):





Webographie:

https://github.com/mdwade/reconaissance_faciale

https://owdin.live/2018/03/28/yolo-v3-performance-de-la-reconnaissance-visuelle-en-temps-reel/

https://www.youtube.com/watch?v=JmfIILAL55Q

https://www.youtube.com/watch?v=imrlsWKK9NU

https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/?fbclid=IwAR1oxqQjpMCAjRFqWPaPeWOPRelmZEl4gNsNdjBcyUcH-GL3MEiXYugZkBU

https://pypi.org/project/facerecognition/?fbclid=IwAR2vgCf3NxuU4BGYTfYVVmnOXKRerEIakeUFEx5js34NY08enLa3iLqeZ4E