تمرین شماره ۳ سامانه‌های چند رسانه‌ای

علی بهمنیار - ۹۸۲۳۰۱۸

خرداد ماه ۱۴۰۲

۱ سوال اول

۱.۱ تشخیص چهره‌های موجود در تصویر

جهت تشخیص تعداد چهره‌های موجود در تصویر از مدل Haar Frontal Face استفاده شده است، این مدل سه گونه‌ی alt، alt2 و default دارد. برای بهینه‌سازی تشخیص چهره‌ها می‌توان نوع گونه‌ی مدل استفاده شده را تغییر داد یا پارامتر minNeighbours را بهینه کرد. در نهایت با استفاده از مدل haarcascade_frontalface_alt2 و minNeighbours=7 مطابق با کد ۱ نتیجه‌ی بهینه حاصل شد و تعداد ۲۹ چهره در تصویر تشخیص داده شد (شکل ۱).

شکل ۱: تشخیص چهره‌های موجود در تصویر

```python
import cv2
from matplotlib import pyplot

detector = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_alt2.xml")
image = cv2.imread("faces.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
rects = detector.detectMultiScale(gray, scaleFactor=1.05,
  minNeighbors=7, minSize=(20, 20), flags=cv2.CASCADE_SCALE_IMAGE)

c = 0
for (x, y, w, h) in rects:
  cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
  c += 1
pyplot.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB), cmap='gray')
pyplot.show()
print("Number of detected faces:", c)
```

کد ۱: تشخیص چهره‌های موجود در تصویر

۲.۱ تشخیص چهره در استریم دریافتی از وب‌کم

در این بخش از سوال استریم دریافتی از وب‌کم پردازش شده و چهره‌ی موجود در آن تشخیص داده می‌شود، برای این کار مجدداً از مدل Haar استفاده شده است. برای سنجش عملکرد کد نیز زمان مورد نیاز برای پردازش 100 فریم محاسبه‌شده و از روی آن میانگین زمان پردازش هر فریم و نیز FPS به دست می‌آید. مقدار FPS طبیعتاً به میزان پردازش مورد نیاز برای هر فریم بستگی دارد. یکی از عوامل اصلی در این مورد اندازه‌ی تصویر است، هر چه اندازه‌ی تصویر بزرگ‌تر باشد، میزان پردازش مورد نیاز بیش‌تر و FPS کمتر است، برای مثال اگر اندازه‌ی تصویر 480×640 باشد مقدار FPS میانگین حدود 30 است. اما با کاهش اندازه‌ی تصویر به 75×100 مقدار FPS به 155 افزایش می‌یابد.

عامل موثر دیگر در FPS فاکتورهای مربوط به scale در متد detectMultiScale می‌باشد، این متد به تشخیص چهره‌های موجود در تصویر در ابعادهای مختلف می‌پردازد با دو پارامتر minSize و scaleFactor می‌توان تعداد این ابعادهای مورد مقایسه را تعیین کرد، هر چه تعداد مقایسه‌ها بیش‌تر باشد طبیعتاً FPS حاصل کمتر خواهد بود؛ برای مثال در همان سایز 480×640 و scaleFactor=1.05 مقدار 30 FPS بود اما با افزایش scaleFactor به 1.5 مقدار FPS حاصل 136 می‌شود. البته تغییر هر دوی پارامترهای مذکور در کیفیت تصویر و تشخیص حاصل تأثیر گذارند و با افزایش FPS کیفیت تصویر یا کیفیت تشخیص چهره کاهش می‌یابد.

```python
import cv2
import time
import imutils
from imutils.video import VideoStream
from matplotlib import pyplot

detector = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_alt2.xml")

vs = VideoStream(src=0).start()
time.sleep(2.0)

c = 0
start = time.time()

while True:
  frame = vs.read()
  frame = imutils.resize(frame, width=480)
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

  rects = detector.detectMultiScale(
              gray, scaleFactor=1.5, minNeighbors=5, minSize=(30, 30),
              flags=cv2.CASCADE_SCALE_IMAGE)

  for (x, y, w, h) in rects:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

  cv2.imshow("Frame", frame)
  key = cv2.waitKey(1) & 0xFF

  if key == ord("q"):
    break

  if c == 100: # Print average fps for the last 100 frames
    end = time.time()
    print(frame.shape)
    print("FPS: ", 100/(end-start))
    c = 0
    start = time.time()

  c += 1

cv2.destroyAllWindows()
vs.stop()
```

کد ۲: تشخیص چهره در استریم دریافتی از وب‌کم و محاسبه‌ی FPS

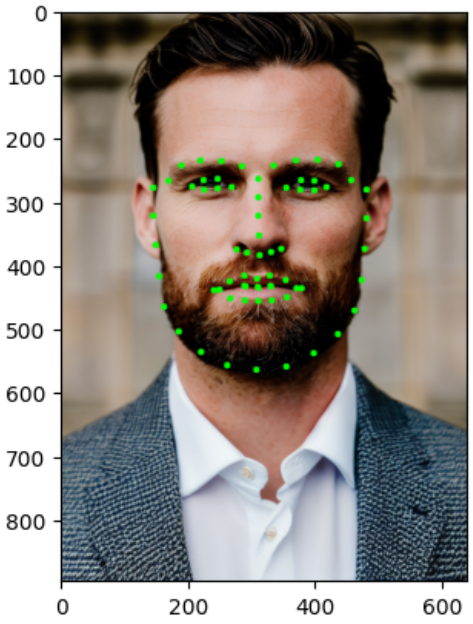
۲ سوال دوم

۱.۲ تشخیص نقاط کلیدی چهره

ابتدا توسط مدل Dreamlike Photoreal 2.0 یک تصویر چهره ایجاد شد (شکل ۲).

شکل ۲: تصویر چهره‌ی ایجاد شده

حال برای تشخیص Landmarkهای چهره‌ی ایجاد شده از کتابخانه‌ی Dlib و مدل shape_predictor_68_face_landmarks استفاده می‌کنیم، این مدل محل ۶۸ نقطه از نقاط کلیدی صورت را به ما می‌دهد:



شکل ۳: Landmarkهای تشخیص داده شده در چهره‌ی ایجاد شده

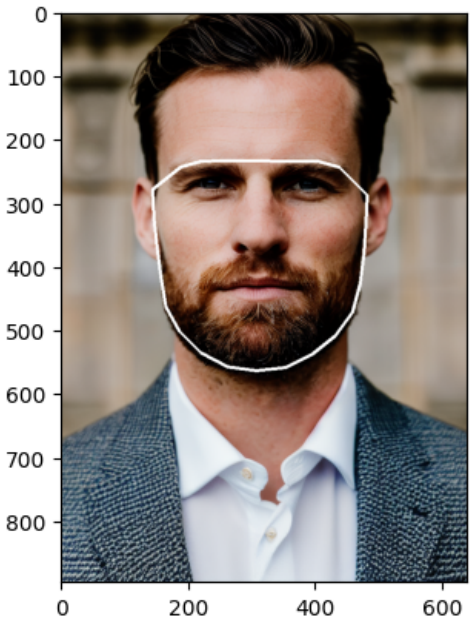
نحوه‌ی استفاده از کتابخانه‌ی Dlib جهت تشخیص ویژگی‌های مذکور در کد ۳ مشخص شده است:

```python
import dlib
import cv2
import numpy as np
from matplotlib import pyplot as plt

face_detector = dlib.get_frontal_face_detector()
landmark_detector = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")


img_path = "img4.png"

img = dlib.load_rgb_image(img_path)
faces = face_detector(img, 1)

landmark_tuple = []
for k, d in enumerate(faces):
   landmarks = landmark_detector(img, d)
   for n in range(0, 68):
      x = landmarks.part(n).x
      y = landmarks.part(n).y
      landmark_tuple.append((x, y))
      cv2.circle(img, (x, y), 5, (0, 255, 0), -1)

plt.imshow(img)
```

کد ۳: تشخیص نقاط کلیدی صورت ایجاد شده

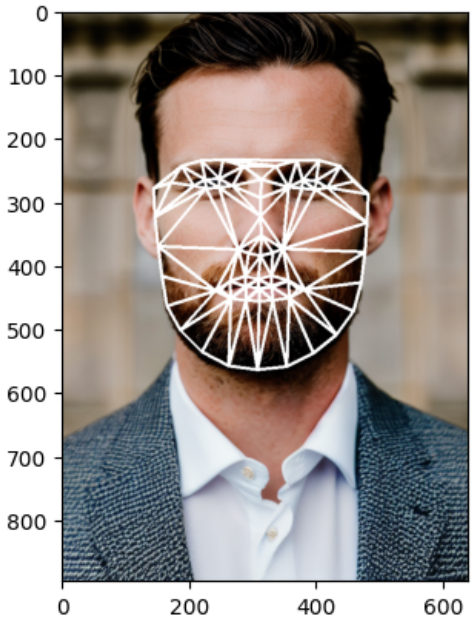
۲.۲ جابه‌جا کردن دو چهره با یکدیگر

برای تعویض دو چهره به طور کلی مراحل زیر را طی می‌کنیم:
ابتدا نقاط کلیدی چهره (به دست آمده از بخش قبل) را در یک Convex Hull محصور می‌کنیم:



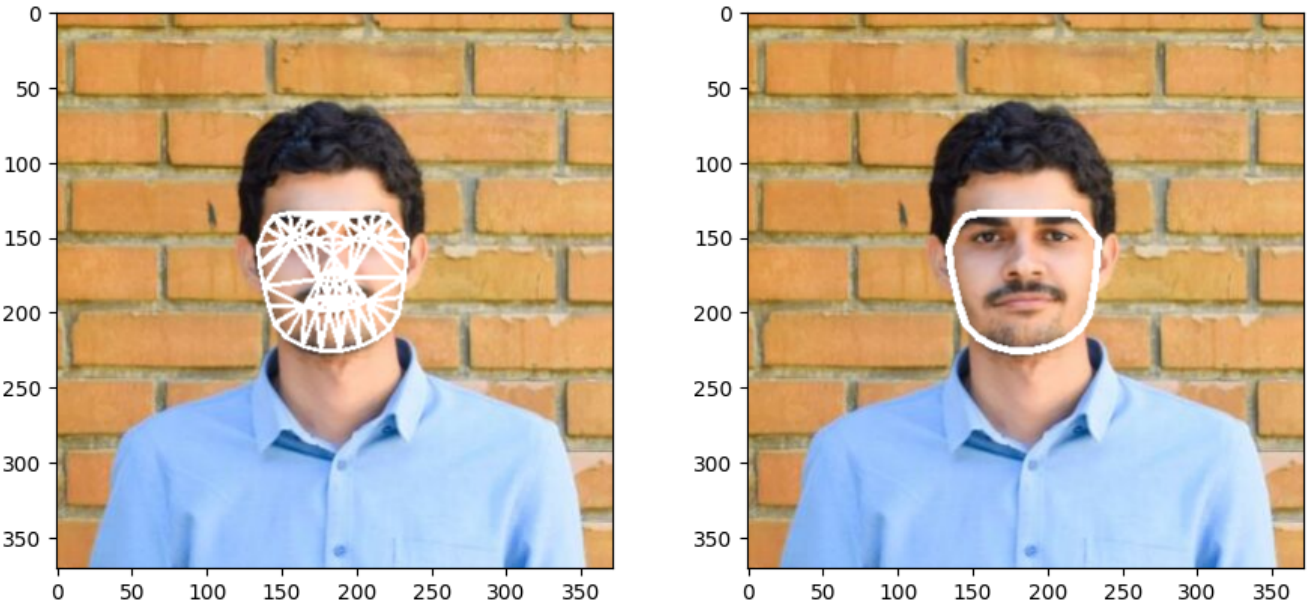
شکل ۴: چند ضلعی محدب محصور کننده‌ی تمام چهره

سپس ناحیه‌ی ایجاد شده را با استفاده از نقاط کلیدی استخراج شده و کلاس cv2.Subdic2D به تعدادی ناحیه‌ی مثلثی تقسیم بندی می‌کنیم:



شکل ۵: تقسیم‌بندی ناحیه‌ی چهره به نواحی مثلثی

این مراحل را برای چهره‌ی دیگر نیز انجام می‌دهیم:



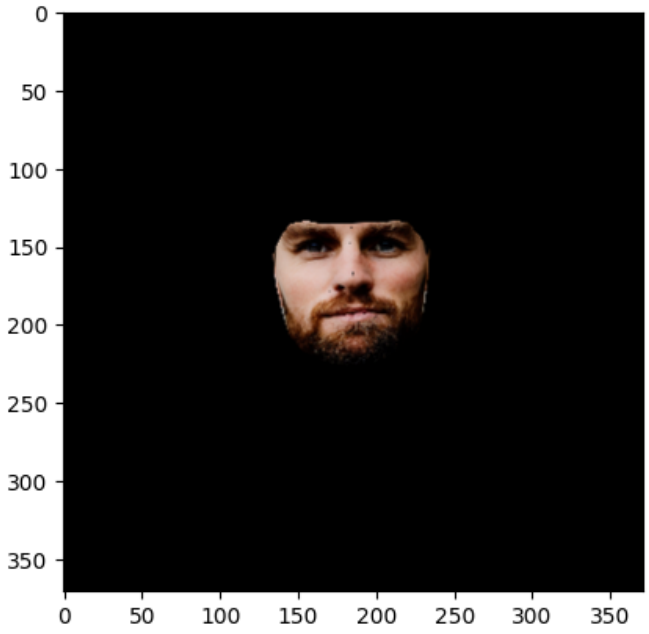
(آ) چند ضلعی محدب محصور کننده‌ی تمام چهره

(ب) تقسیم‌بندی ناحیه‌ی چهره به نواحی مثلثی

شکل ۶: محصور کردن و مثلث‌بندی چهره‌ی دوم

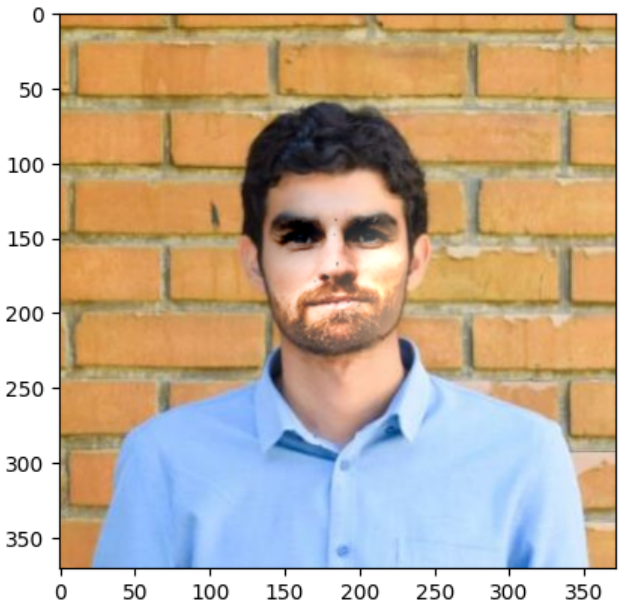
با توجه به شکل‌های ۵ و ۶ب می‌توانیم مرحله‌ی بعدی و اصلی کار را حدس بزنیم؛ کافی است هر یک از مثلث های موجود در چهره‌ی اول را انتخاب کرده و آن را بر روی مثلث متناظر از چهره‌ی دوم منطبق کنیم و این کار را برای تمام مثلث‌ها انجام دهیم. برای این کار از متدهای cv2.getAffineTransform و cv2.warpAffine استفاده می‌شود؛ متد اول با در اختیار داشتن دو مثلث تبدیلی را به ما می‌دهد که با اعمال آن بر یک تصویر، مثلث اول به مثلث دوم تبدیل می‌شود، متد دوم نیز این تبدیل به دست آمده را بر روی تصویر اعمال می‌کند.

پس از اعمال تمام تبدیل‌های مذکور بر روی تصویر اول به نتیجه شکل ۷ می‌رسیم:

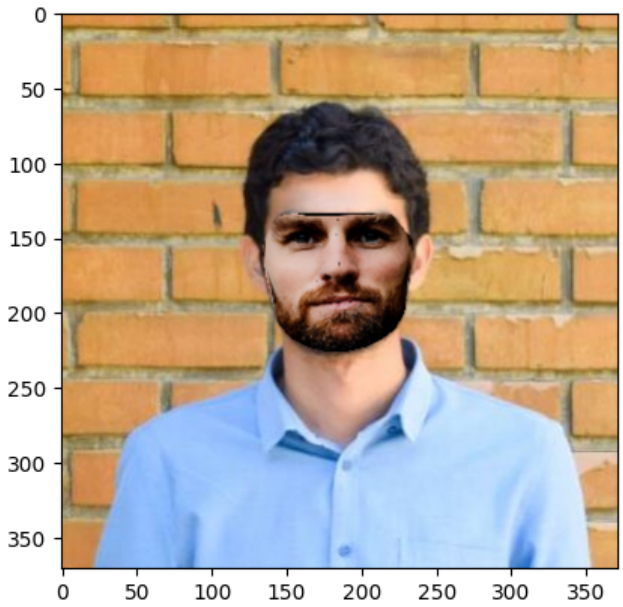


شکل ۷: نتیجه‌ی به دست آمده پس از منطبق کردن تمام مثلث‌های چهره‌ی اول بر روی چهره‌ی دوم

حال در بخش نهایی تنها کافی است این نتیجه‌ی به دست آمده را بر روی تصویر دوم قرار دهیم. همچنین با استفاده از متد cv2.seamlessClone می‌توانیم مرزهای حاصل از انداختن تصویر اول بر روی تصویر دوم را محو و یکنواخت کنیم:



(ب) محو کردن مرزهای تصویر ایجاد شده



(آ) انداختن تصویر حاصل شده از فرآیند بالا روی تصویر دوم

شکل ۸: تصاویر نهایی