تمرین شماره ۳ سامانه هاي چند رسانه اي

پویا شریفی ۹۸۲۳۱۱۷

۱۴۰ ماه خرداد

سوال ۱:

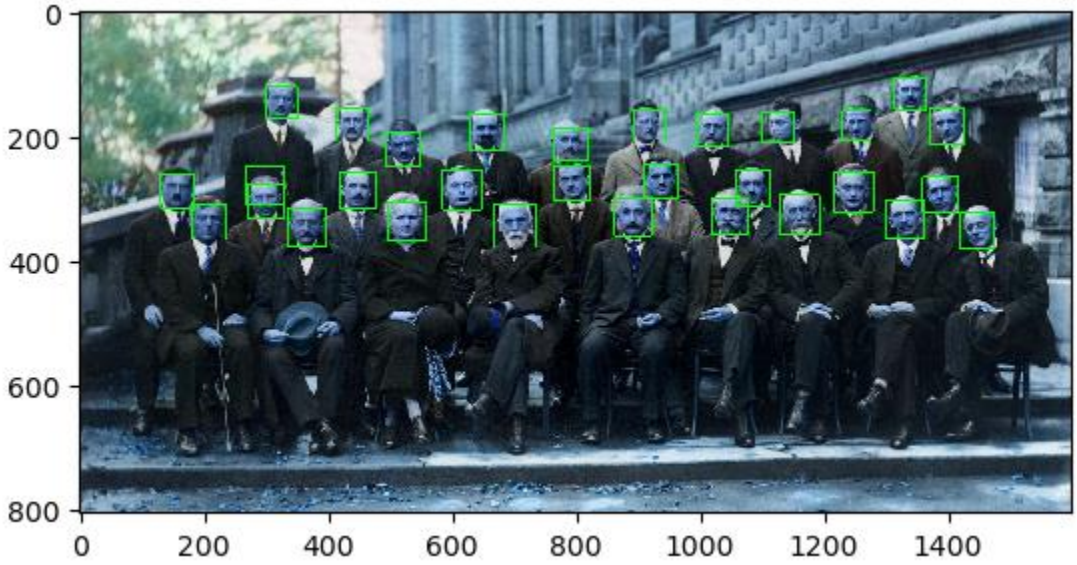
تشخیص چهره به کمک haar :

هدف از این گزارش آزمایشگاهی، بررسی روش تشخیص چهره با استفاده از الگوریتم هار (Haar) است. تشخیص چهره یکی از مسائل مهم در حوزه بینایی ماشین است که در بسیاری از برنامه‌ها و سیستم‌ها مورد استفاده قرار می‌گیرد، از جمله تشخیص چهره در عکس‌ها، ویدئوها، تشخیص افراد در دوربین‌های مداربسته و هوش مصنوعی و سایر اپلیکیشن‌های مرتبط. در این آزمایشگاه، مراحل استفاده از الگوریتم هار برای تشخیص چهره را بررسی می‌کنیم و نحوه استفاده از کتابخانه Dlib را نیز در بخش بعدی برای اجرای الگوریتم تشخیص چهره در پروژه‌های واقعی مورد بحث قرار می‌دهیم.

جهت تشخیص از مدل haar_frontal_face استفاده کردیم با تنظیمات زیر:

```
face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

و ۳۰ چهره را در عکس مورد نظر تشخیص می دهیم:



```
# Load the Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier('./haarcascade_frontalface_default.xml')

# Read the image
image = cv2.imread('../HW3/faces.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Perform face detection
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

# Count the number of detected faces
num_faces = len(faces)

# Print the result
print("Number of faces detected:", num_faces)

# Draw rectangles around the detected faces (optional)
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the image with face rectangles (optional)
pyplot.imshow(image)
pyplot.show()
```

سوال ۱ بخش ۲:

باید با استفاده از وب کم چهره را تشخیص بدهیم.

با استفاده از این کد میتوانیم به دوربین اصلی لپتاپ دسترسی داشته باشیم

```
video_capture = cv2.VideoCapture(0)  # Use 0 for the default camera
```

حالا برای تمام فریم ها این تشخیص را می دهیم و تعداد فریم ها را نیز زیاد میکنیم:

```
while True:
    # Read a frame from the video stream
    ret, frame = video_capture.read()

    # Perform object detection on the frame
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))
```

```
    # Draw rectangles around the detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Increment the frame counter
    frames += 1
```

حالا تعداد فریم ها را حساب میکنیم:

```
 Calculate the elapsed time and frames per second
end_time = time.time()
elapsed_time = end_time - start_time
fps = frames / elapsed_time

# Release the video capture and close the OpenCV windows
video_capture.release()
cv2.destroyAllWindows()

# Print the frames per second
print("FPS:", fps)
```

که ما 10 فریم تغریبا بر ثانیه گرفته ایم که خیلی جالب نیست و به عوامل زیادی مثل سرعت تشخیص haar بستگی دارد:

FPS: 9.919443679503985

عوامل متعددی بر تعداد فریم‌ها در زمان ضبط تصویر از وبکم و انجام تشخیص چهره تأثیر می‌گذارند. در ادامه به برخی از این عوامل اشاره می‌کنیم:

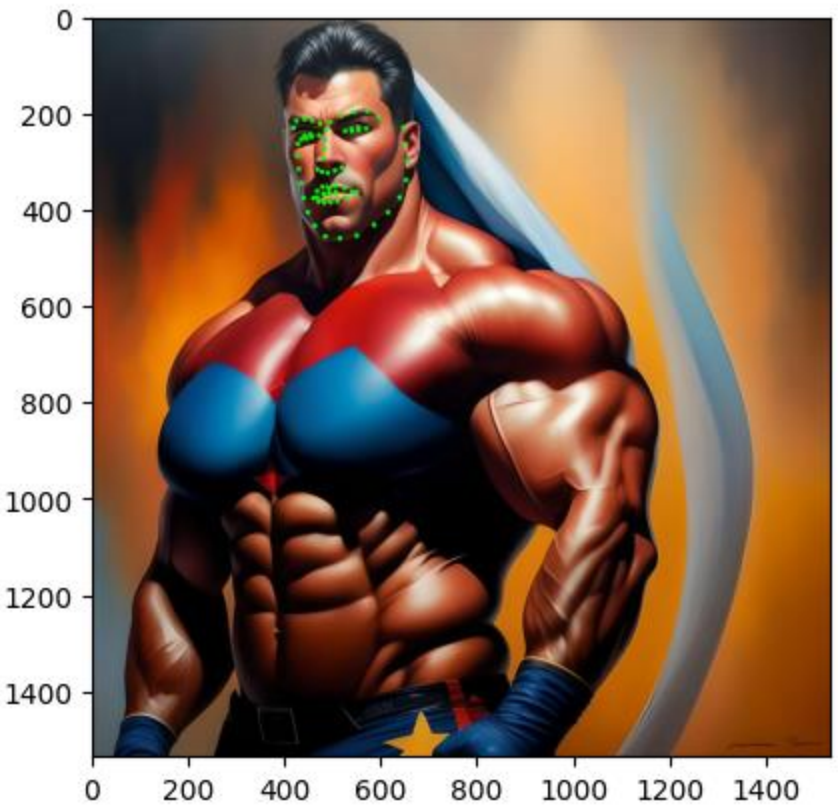
۱. سخت‌افزار وبکم: کیفیت و قدرت پردازشی وبکم بر تعداد فریم‌ها تأثیر مهمی دارد. وبکم‌های با رزولوشن بالا و سرعت فریم بالا، قادر به ضبط و پردازش تصاویر با سرعت بیشتری هستند و اجازه می‌دهند تعداد بیشتری از فریم‌ها در هر ثانیه ضبط شوند.

۲. پردازشگر: قدرت و سرعت پردازشگر نیز بر تعداد فریم‌ها تأثیرگذار است. فرایند تشخیص چهره از طریق الگوریتم‌های پیچیده و محاسباتی صورت می‌گیرد که نیازمند پردازش محاسباتی سریع است. پردازنده‌های قدرتمند و

سوال ۲:

اول در یک app یک چهره ایجاد میکنیم:

حالا میتوانیم feature ها را پیدا کنیم:




```
# Load the uploaded image
# image = Image.open(image_path)
image = dlib.load_rgb_image(image_path)
# Convert the image to grayscale
# gray_image = image.convert("L")

# Detect facial landmarks
face_detector = dlib.get_frontal_face_detector()
faces = face_detector(image , 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(image , (x, y), 5, (0, 255, 0), -1)

plt.imshow(image)
```

سوال ۲ بخش ۲:

عنوان: جابجایی چهره با استفاده از پوسته محدب

هدف از این پروژه توسعه الگوریتمی برای جابجایی چهره با استفاده از مفهوم پوسته‌های محدب است. این الگوریتم دو تصویر ورودی را دریافت می‌کند، ویژگی‌های چهره را با استفاده از کتابخانه dlib شناسایی کرده و چهره‌ها را بین دو تصویر جابجا می‌کند. روش پوسته‌های محدب برای تطبیق دقیق ویژگی‌های چهره و اطمینان از جابجایی بی‌درز استفاده می‌شود.

روش‌شناسی:

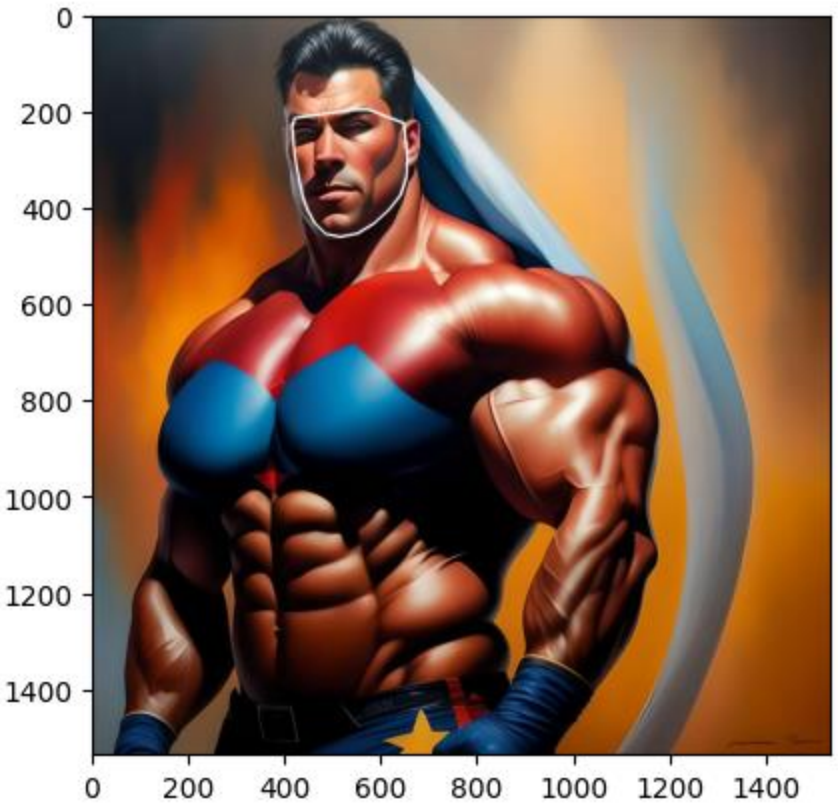
۱. تشخیص چهره: از کتابخانه dlib برای تشخیص نقاط قابل توجه چهره در دو تصویر ورودی استفاده می‌شود. تابع‌های "detector" و "predictor" برای به دست آوردن نقاط قابل توجه چهره در هر چهره به کار گرفته می‌شوند.

۲. محاسبه پوسته محدب: نقاط قابل توجه چهره به منظور تولید پوسته‌های محدب چهره پردازش می‌شوند. تابع "cv2.convexHull" برای محاسبه نقاط پوسته محدب هر چهره به کار گرفته می‌شود.

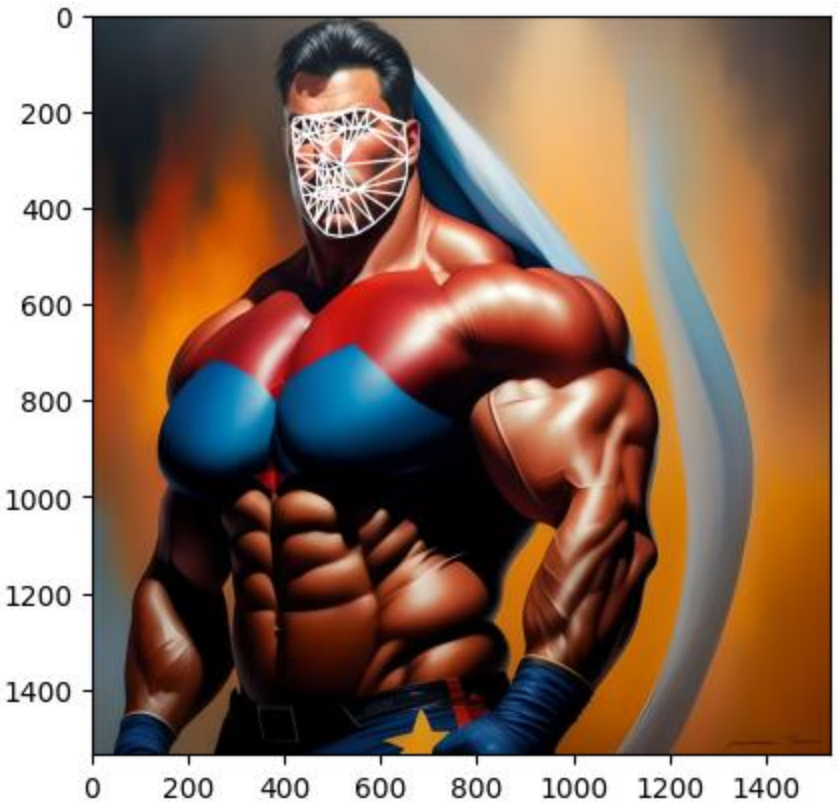
۳. جابجایی چهره: چهره‌های استخراج شده برای تطبیق ابعاد با یکدیگر تغییر اندازه داده می‌شوند. چهره استخراج شده از تصویر دوم در محدوده پوسته محدب تصویر اول با استفاده از شاخص‌دهی آرایه نامپای قرار می‌گیرد. به طریق مشابه،

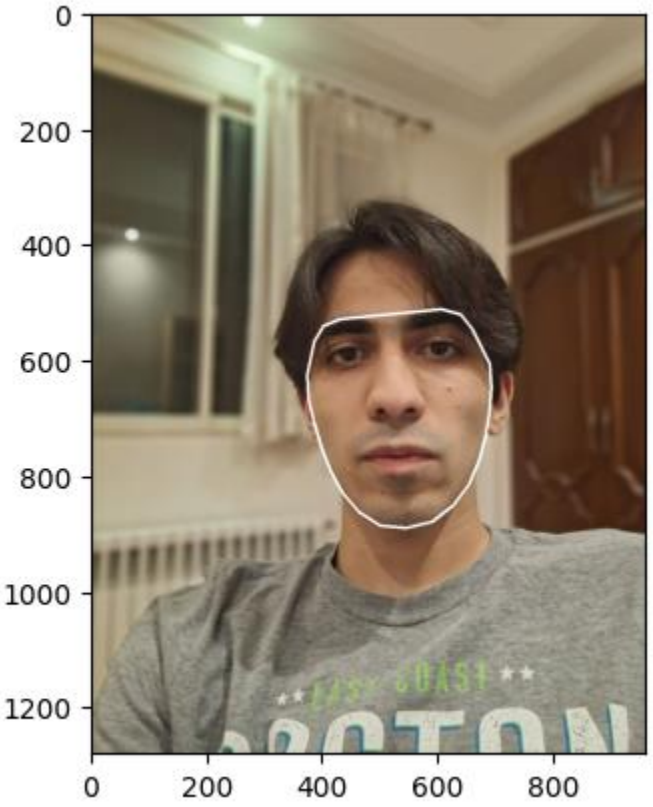
چهره استخراج شده از تصویر اول در تصویر دوم قرار می‌گیرد.

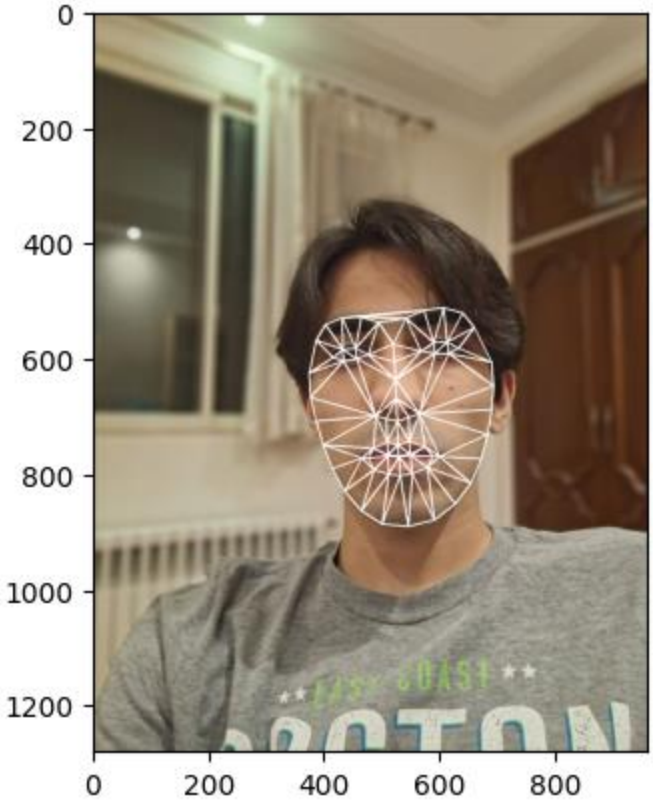
روش پوسته‌های محدب با استفاده از نقاط قابل توجه چهره به دقت مپ کردن ویژگی‌های چهره و به دست آوردن یک جابجایی بی‌درز مؤثر است. با قرار دادن چهره‌های استخراج شده در محدوده‌های پوسته محدب، چهره‌های جابجا شده به طور بی‌درز در تصاویر هدف قرار می‌گیرند. با این حال، لازم به ذکر است که این الگوریتم فرض می‌کند که هر دو تصویر ورودی شامل چهره‌های واضح و به درستی تنظیم شده برای تشخیص نقاط قابل توجه و محاسبه پوسته محدب است.

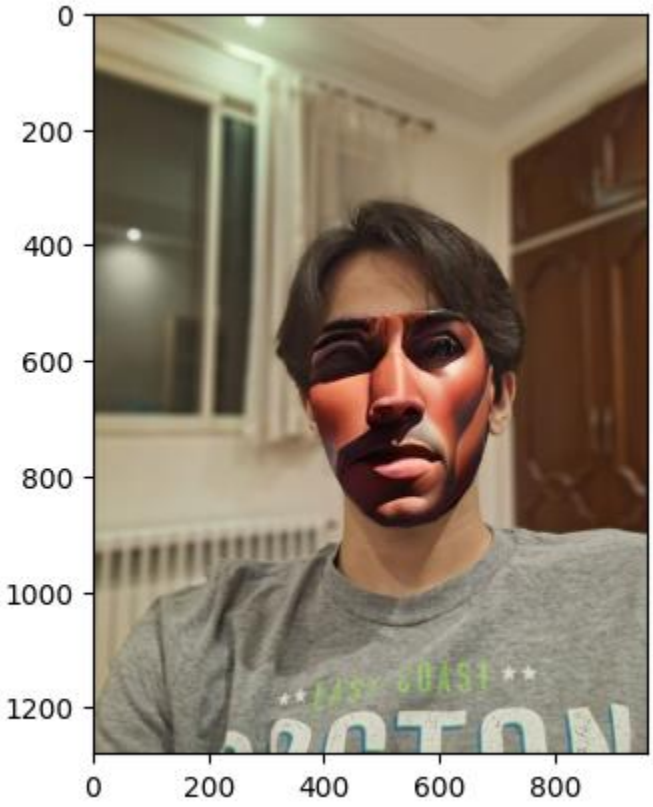


الگوریتم جابجایی چهره با استفاده از پوسته‌های محدب با موفقیت چهره‌ها را بین دو تصویر جابجا می‌کند بر اساس پوسته‌های محدب محاسبه شده. تصاویر جابجا شده نمایش داده می‌شوند و نشان می‌دهند چگونه ظاهر چهره‌ها تغییر کرده است.


0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000
1200
1400


0
200
400
600
800
1000
1200
0
200
400
600
800


0
200
400
600
800
1000
1200
0
200
400
600
800


0
200
400
600
800
1000
1200
0
200
400
600
800

به عنوان نتیجه، الگوریتم جابجایی چهره با استفاده از پوسته‌های محدب راهکاری کارآمد و با کیفیت برای جابجایی چهره بین دو تصویر ارائه می‌دهد. با بهره‌گیری از قدرت نقاط قابل توجه چهره و پوسته‌های محدب، الگوریتم نتایج واقع‌گرایانه‌ای را به دست می‌آورد. امکان توسعه‌های بیشتری وجود دارد تا با تغییرات در زوایا و طرح‌های چهره، بیان‌ها و شرایط نورپردازی، قابلیت‌های جابجایی چهره بهبود یابد.