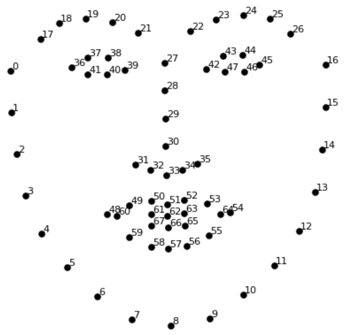
# OpenCV 105 Computer Vision Applications II Python course
# Project 1: Virtual Makeup — Apply glasses

In this document I will describe the steps followed to apply glasses to a face identifying the right location and size to place them based on landmark detection.

We use the provided landmark detector and calculated points as well as the girl's image as starting point.

## Step 1: Detect landmarks

We just need to run the provided cells in the notebook to get an array of points for the landmark calculation, we will get points as per the image below.



### Step 2: Detect the distance between points

Identify the distance between Points[0] and Points[16] to get the length of the face.

```
import math
dist = points[16][0] - points[0][0]
```

### Step 3: read and resize the glasses

Resize the sunglasses image based on the distance identified so it fits right between Points[0] and Points[16]

Read the sunglasses image file with alpha channel and resize them. We are just deciding to scale by a factor of 0.5 the vertical glasses' dimension

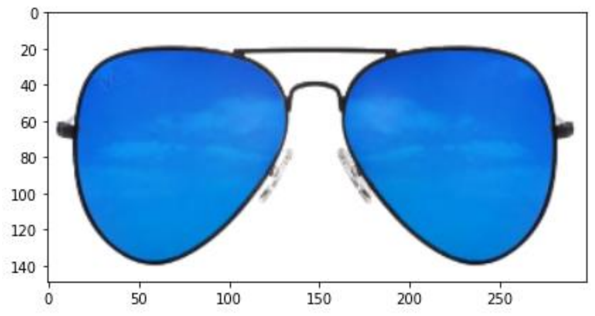
```
glasses = cv2.imread("sunglass.png",-1)

dw = dist
```

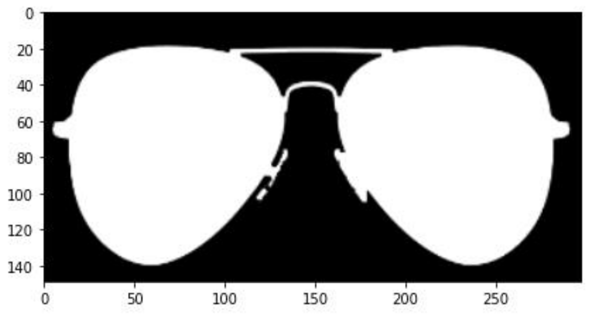
```
dh = int(dw * 0.50)
glassesResize = cv2.resize(glasses,(dw,dh))
```

### Step 4: Detect the distance between points

Extract the RGB channels and the alpha channel to separate variables

```
glassesRGB = glasses[:,:,0:3]
glassesRGB = cv2.cvtColor(glassesRGB,cv2.COLOR_BGR2RGB)
glassesMASK = glasses[:,:,3]
```





### Step 5: Identify the right position on the face

From the original image, the bridge between the glasses is roughly present at 0.4 of the entire height, so the bridge placement should coincide with Points[27] landmark of the face. So Y1 value can be determined using this.

Y1 = Y value of Points[27] minus the 0.4 times of the height of the sunglasses image
Y2 = Y1 plus the height of the sunglasses image.
X1 and X2 is determined by the x value of points[27]
X1 = x value of points[27] — ( half of width of the sunglasses image)
X2 = x value of points[27]+( half of width of the sunglasses image)

```
midpointX, midpointY = points[27]
print("{},{}".format(midpointX,midpointY))
y1 = midpointY - int(dh * 0.4)
y2 = y1 + dh
x1 = midpointX - int(dw / 2)
x2 = midpointX + int(dw / 2)
```

### Step 6: Create the different masks needed to merge glasses with the original image

Create the 3 channel image from the alpha channel

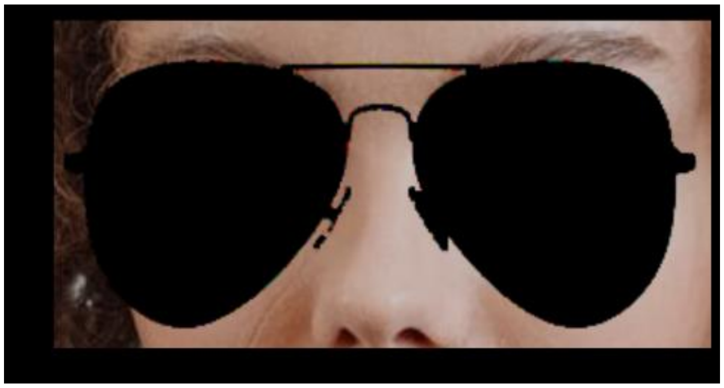
```
maskedGlassesImage = cv2.merge((glassesMASK,glassesMASK,glassesMASK))
```

Do a bitwise_and between glassesRGB and maskedGlassesImage

```
augGlassesMasked = cv2.bitwise_and(glassesRGB,maskedGlassesImage)
```

Do a bitwise_and with the region of interest and the bitwise_not of maskedGlassesImage

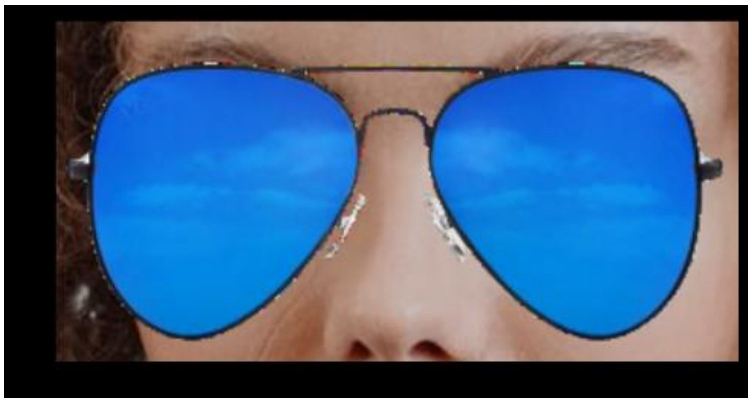
```
glassesGirlROI = imDlib.copy()
glassesGirlROI = glassesGirlROI[y1:y2,x1:x2]
glassesGirlROIImage = cv2.bitwise_and(glassesGirlROI,cv2.bitwise_not(maskedGlassesImage))
plt.imshow(glassesGirlROIImage)
```



### Step 7: Incorporate the modified ROI to the full image

Finally, we modify the region of interest with the bitwise_or of glassesGirlROIImage and augGlassesMasked

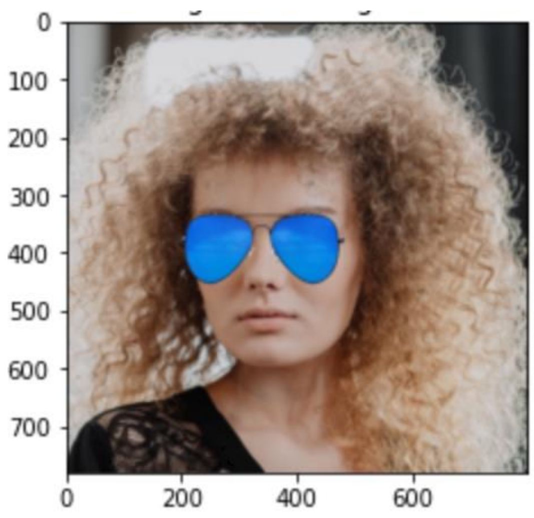
```
glassesGirlROIFinal = cv2.bitwise_or(glassesGirlROIImage , augGlassesMasked)
plt.imshow(glassesGirlROIFinal)
```



And now we replace the pixels in the original image with the pixels of modified region of interest

```
glassesGirlfinal = imDlib.copy()
```

```
glassesGirlfinal[y1:y2,x1:x2] = glassesGirlROIFinal
plt.imshow(glassesGirlfinal)
```



### Step 8: Adding transparency

As an extra step, I used the cv2.addWeighted() method to create a semitransparency effect.

```
# Create a semitransparent sunglasses
faceWithGlassesArithmetic=cv2.addWeighted(glassesGirlROIFinal, 0.7, glassesGirlROI, 0.3, 0)
plt.imshow(faceWithGlassesArithmetic)
```