Accessories application

October 14, 2021

[1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.colors as mpl_colors
import matplotlib.patches as patches
import dlib # using for face shape models

#You can download a trained facial shape predictor from:
#    http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2
```

1 Adding accessories application

(images have been changed from the original submission for privacy reasons) Date : 15.08.2021

Names: Anusha Bhat, Wenwen Bai, Rulla Gabarine, Bryony Bar

1.1 Introduction

Since the year 2020, the world has been experiencing a pandemic which has changed our lifestyles drastically. We have however learnt to adapt to these times and shifted towards online platforms to continue with our everyday life activities. Online shopping has also become very preferential during these times and has even saved many businesses. To give customers a shopping experience which can be as close to real as possible, our project presents an experience which allows users to try on facial accessories from the comfort of their home.

1.2 Table of content

1.2.1 Loading models

[2]:
```
face_formula = cv2.CascadeClassifier()
eyes_formula = cv2.CascadeClassifier()
nose_formula = cv2.CascadeClassifier()

face_formula.load(cv2.samples.findFile('haarcascade_frontalface_alt.xml'))
eyes_formula.load(cv2.samples.findFile('haarcascade_eye.xml'))
nose_formula.load(cv2.samples.findFile('haarcascade_mcs_nose.xml'))

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

print('Models are loaded.')
```

```
Models are loaded.
```

2 Take a snapshot from webcam

Run this code section until you are satifsied with the photo

[30]:
```
# Turn on camera
cap = cv2.VideoCapture(0)
```

[27]:
```
# Take a snapshot
if cap.isOpened():
    ret, snapshot = cap.read()
    snapshot= snapshot[:,:,::-1]
# Plot snapshot
plt.figure(figsize=(10,5))
plt.imshow(snapshot)
print(snapshot.shape)
```

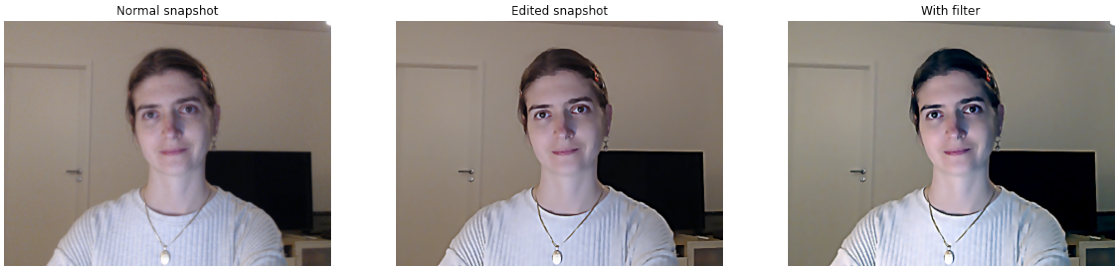
```
(480, 640, 3)
```

[28]:
```
# Close camera when you're done
cap.release()
```

2.1 Filters

[29]:
```
# Edited snapshot
img = snapshot.copy()
for idx in range(4):
    blur_img = cv2.GaussianBlur(img, (0, 0), 2.0)
    unsharp_mask = img.astype(int) - blur_img.astype(int)

    unsharp_image = (img + 0.5*unsharp_mask)
    unsharp_image[unsharp_image<0] = 0
    unsharp_image[unsharp_image>255] = 255
    img = unsharp_image.astype(np.uint8)

    img = cv2.bilateralFilter(img,7,35,35)


# Filtered snapshot
snapshot_rgb = img.copy() / 255.0
```

```
v_light = 0.5

mean_r = np.mean(snapshot_rgb[:,:,0])
mean_g = np.mean(snapshot_rgb[:,:,1])
mean_b = np.mean(snapshot_rgb[:,:,2])

snapshot_rgb[:,:,0] = 1.4*(snapshot_rgb[:,:,0] - mean_r)
snapshot_rgb[:,:,1] = 1.2*(snapshot_rgb[:,:,1] - mean_g)
snapshot_rgb[:,:,2] = 1.1*(snapshot_rgb[:,:,2] - mean_b)

snapshot_rgb += v_light
snapshot_rgb[snapshot_rgb<0] = 0
snapshot_rgb[snapshot_rgb>1] = 1
snapshot_rgb = (255*snapshot_rgb).astype(np.uint8)

plt.figure(figsize=(20,7))

# plotting images
plt.subplot(131)
plt.imshow(snapshot)
plt.axis(False)
plt.title('Normal snapshot')

plt.subplot(132)
plt.imshow(img)
plt.axis(False)
plt.title('Edited snapshot')

plt.subplot(133)
plt.imshow(snapshot_rgb)
plt.axis(False)
plt.title('With filter')

#img = (255*snapshot_rgb).astype(np.uint8)
plt.show()
```



3 Detecting facial features

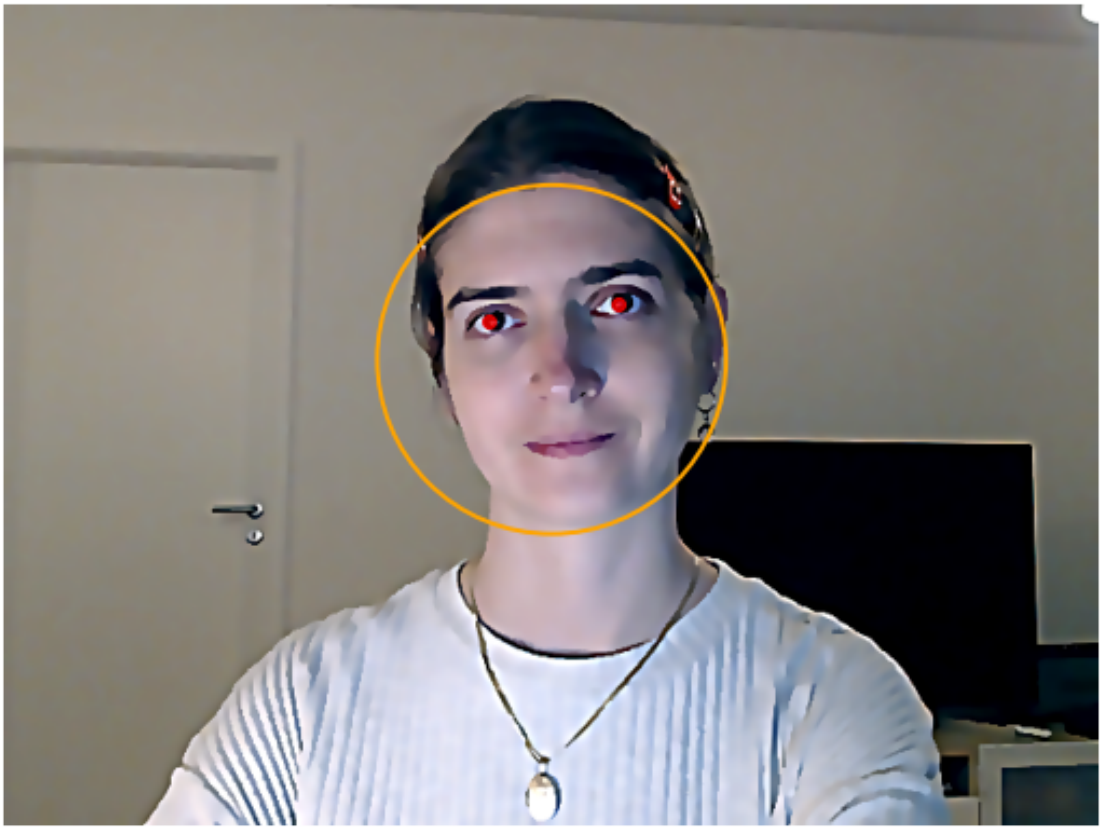
Check if face/eyes/nose have been detected

3.0.1 Detecting features on webcam snapshot

[31]:
```
true_eyes = []
true_face = []
true_nose = []

# plotting the webcam snapshot
faces = face_formula.detectMultiScale(snapshot_rgb)
if len(faces) == 0:
    print('No face has been found - Please, try to adjust parameters.')

plt.figure(figsize=(10,7))
plt.imshow(snapshot_rgb)
plt.axis(False)

# If a face has been detected
for (x,y,w,h) in faces:
    # Calculating the face center
    center = (x + w/2, y + h/2)

    # Add ellipse over the face
    axes = plt.gca()
    mpl_ellipse = patches.Ellipse(center, w, h, linewidth=2,␣
↪edgecolor='orange', facecolor='none')
    axes.add_patch(mpl_ellipse)

    # creating a region of interest for eyes
    faceROI = snapshot_rgb[y:y+2*h//3,x:x+w, :]
    # I am checking just on the upper 2/3 of the face

    #-- In each face, detect eyes
    eyes = eyes_formula.detectMultiScale(faceROI)
    if len(faces) == 0:
        print('No eyes have been found - Please, try to adjust parameters.')
    if len(eyes) == 2: # only if only two eyes have been located, continue␣
↪calculation
        true_eyes.append(eyes)
        true_face.append((x,y,w,h))

        for (x2,y2,w2,h2) in eyes:
            eye_center = (x + x2 + w2//2, y + y2 + h2//2)
            plt.scatter(eye_center[0], eye_center[1],  c='r', alpha=0.8)
    # nose
```

```
    nose = nose_formula.detectMultiScale(snapshot, 1.5, 5)
    if len(nose) == 0:
        print('No nose has been found - Please, try to adjust parameters.')

    for (x,y,w,h) in nose:
            center = (x + w/2, y + h/2)

        # Add ellipse over the nose
            axes = plt.gca()
            mpl_ellipse = patches.Ellipse(center, w, h, linewidth=1,␣
↪edgecolor='black', facecolor='none')
            axes.add_patch(mpl_ellipse)
```

```
No nose has been found - Please, try to adjust parameters.
```



4 Upload a picture (optional instead of webcam photo)

4.1 Filters

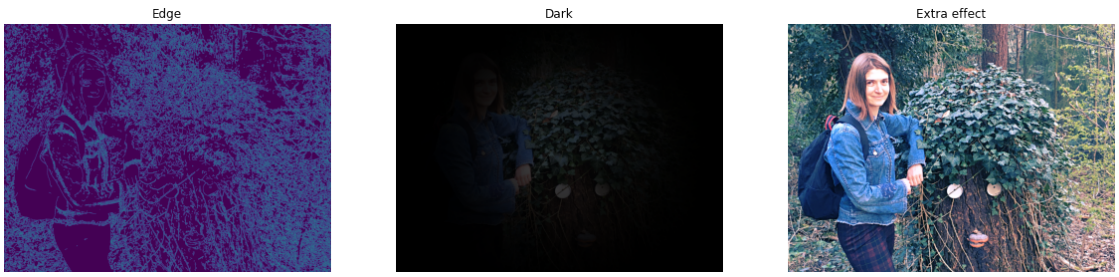
[32]:
```
# Read and plot image
img_bgr = cv2.imread('image.jpg')
img = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB);
plt.figure(figsize=(15,7))

# Convert to grayscale
img_gray = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Blur filter
blur = cv2.GaussianBlur(img,(15,15),cv2.BORDER_DEFAULT)

plt.figure(figsize=(20,5))
plt.subplot(131)
plt.imshow(img)
plt.axis(False)
plt.title('Normal')

plt.subplot(132)
plt.imshow(img_gray, cmap='gray')
plt.axis(False)
plt.title('Black and white')

plt.subplot(133)
plt.imshow(blur)
plt.axis(False)
plt.title('Blurred')

plt.show()
```

```
<Figure size 1080x504 with 0 Axes>
```

[33]:
```
## Detect edges
edges = cv2.Canny(img,100,200)

## Creating dark frame filter
```

```
rows, cols = img.shape[:2]
kernel_x = cv2.getGaussianKernel(cols,500)
kernel_y = cv2.getGaussianKernel(rows,500)
kernel = kernel_y * kernel_x.T
filter = 255 * kernel / np.linalg.norm(kernel)

#applying the filter to input image
dark_img = np.copy(img)
for i in range(3):
    dark_img[:,:,i] = dark_img[:,:,i] * filter

##another colour effect filter
#rgb values can be adjusted to create different colour effects

import skimage
from skimage import io, filters

def channel_adjust(channel, values):
    og_size = channel.shape
    flat_channel = channel.flatten()
    adjusted = np.interp(flat_channel, np.linspace(0, 1, len(values)), values)
    return adjusted.reshape(og_size)

og_image = skimage.img_as_float(io.imread("image.jpg"))
r = og_image[:, :, 0]
b = og_image[:, :, 2]
r_boost_lower = channel_adjust(r, [
    0, 0.05, 0.1, 0.2, 0.3,
    0.5, 0.7, 0.8, 0.9,
    0.95, 1.0])
b_more = np.clip(b + 0.03, 0, 1.0)
merged = np.stack([r_boost_lower, og_image[:, :, 1], b_more], axis=2)
blurred = filters.gaussian(merged, sigma=10, multichannel=True)
final = np.clip(merged * 1.3 - blurred * 0.3, 0, 1.0)
b = final[:, :, 2]
b_adjusted = channel_adjust(b, [
    0, 0.047, 0.118, 0.251, 0.318,
    0.392, 0.42, 0.439, 0.475,
    0.561, 0.58, 0.627, 0.671,
    0.733, 0.847, 0.925, 1])
final[:, :, 2] = b_adjusted

# Plotting the image with different filters
plt.figure(figsize=(20,5))
plt.subplot(131)
plt.imshow(edges)
plt.axis(False)
```

```
plt.title('Edge')

plt.subplot(132)
plt.imshow(dark_img)
plt.axis(False)
plt.title('Dark')

plt.subplot(133)
plt.imshow(final)
plt.axis(False)
plt.title('Extra effect')

plt.show()
```



4.1.1 Detecting features on uploaded image

[34]:
```
# Plotting uploaded image
faces = face_formula.detectMultiScale(img)
if len(faces) == 0:
    print('No face has been found - Please, try to adjust parameters.')

plt.figure(figsize=(10,7))
plt.imshow(img, cmap='gray')
plt.axis(False)

# If a face has been detected
for (x,y,w,h) in faces:
    # Calculate the face center
    center = (x + w/2, y + h/2)

    # Add rectangle over the face
    axes = plt.gca()
    draw_rect = patches.Rectangle((x, y), w, h, linewidth=2,␣
 ↪edgecolor='orange', facecolor='none')
    axes.add_patch(draw_rect)
```

```
   # create a region of interest for eyes
   faceROI = img[y:y+2*h//3,x:x+w, :]
   # I am checking just on the upper 2/3 of the face

   #-- In each face, detect eyes
   eyes = eyes_formula.detectMultiScale(faceROI)

   if len(eyes) == 2: # only if only two eyes have been located, continue␣
↪ccalculation
       true_eyes.append(eyes)
       true_face.append((x,y,w,h))
   for (x2,y2,w2,h2) in eyes:
           eye_center = (x + x2 + w2//2, y + y2 + h2//2)
           plt.scatter(eye_center[0], eye_center[1],  c='r', alpha=0.8)

   nose = nose_formula.detectMultiScale(img)
   if len(nose) == 0:
       print('No identified nose')
       # If a nose has been detected
   for (x,y,w,h) in nose:
           # Calculate the face center
           center = (x + w/2, y + h/2)

           # Add ellipse over the nose
           axes = plt.gca()
           mpl_ellipse = patches.Ellipse(center, w, h, linewidth=1,␣
↪edgecolor='black', facecolor='none')
           axes.add_patch(mpl_ellipse)
```

5 Add glasses

Current availability of glasses >Yellow glasses (*'glasses1.png'*) >Pink glasses (*'glasses2.png'*) >Beach glasses (*'cool_glasses.png'*)

[35]:
```
glasses_img = cv2.imread('glasses3.png', cv2.IMREAD_UNCHANGED)
glasses_img_rgb = glasses_img.copy()
plt.figure(figsize=(10,5))

glasses_rgb = glasses_img[:,:,:3]
glasses_rgb = glasses_rgb[:,:,::-1]
glasses_img_rgb[:,:,:3] = glasses_rgb
plt.imshow(glasses_rgb)

plt.axis(False)
plt.show()
```

[36]:
```
eyes_val = true_eyes[0] # to iterate over all found eyes
face_val = true_face[0]

#(x2,y2,w2,h2)
# the equations is (((x1+w1/2) - (x0+w0/2))^2 + ((y1+h1/2) - (y0+h0/2))^2)^0.5
dist_eyes = ((eyes_val[1][0] - eyes_val[0][0] + (eyes_val[1][2] -␣
 ↪eyes_val[0][2])/2)**2 +\
             (eyes_val[1][1] - eyes_val[0][1] + (eyes_val[1][3] -␣
 ↪eyes_val[0][3])/2)**2)**0.5

#(x2,y2,w2,h2)
#the equations is (((x1+w1/2 + x0+w0/2)/2, (y1+h1/2 + y0+h0/2)/2) + (x_face,␣
 ↪y_face)
eyes_center = np.array([eyes_val[1][0] + eyes_val[0][0] + (eyes_val[1][2] +␣
 ↪eyes_val[0][2])/2,\
                         eyes_val[1][1] + eyes_val[0][1] + (eyes_val[1][3] +␣
 ↪eyes_val[0][3])/2])/2 +\
                (face_val[0], face_val[1])


print('Distance between eyes is: {}'.format(dist_eyes))
print('Center of eyes is: {}'.format(eyes_center))

# how to resize glasses
k_zoom = 2.3
glasses_zoom = k_zoom*dist_eyes/glasses_img.shape[1]
```

```
Distance between eyes is: 76.095334942426
Center of eyes is: [321.25 179.75]
```

[37]:
```
glasses_size = np.array([glasses_zoom*glasses_img.shape[1],␣
 ↪glasses_zoom*glasses_img.shape[0]]).astype(int)
```

```
glasses_cut = cv2.resize(glasses_img_rgb, tuple(glasses_size))

plt.imshow(glasses_cut[:,:,:3])
plt.axis(False)
plt.show()
```

[38]:
```
## rotate glasses
#(x2,y2,w2,h2)
# equation is [(x1 + w1/2)- (x0 + w0/2), (y0 + h0/2)- (y1 + h1/2)]
# invert y as pixels are read from top to bottom (normal y direction is␣
 ↪opposite)

vec_eyes = np.array([(eyes_val[1][0] + eyes_val[1][2]/2) - (eyes_val[0][0] +␣
 ↪eyes_val[0][2]/2), \
             (eyes_val[0][1] + eyes_val[0][3]/2) - (eyes_val[1][1] +␣
 ↪eyes_val[1][3]/2)])

# calculating angle
angle_eyes = np.arctan2(vec_eyes[1], vec_eyes[0])*180/np.pi
# normalizing angle
angle_eyes = angle_eyes - 90*(np.round(angle_eyes/90))

# rotating the image
# calculate the diagonal as the largest dimension
diag_glass = np.ceil((glasses_cut.shape[0]**2 + glasses_cut.shape[1]**2)**0.5).
 ↪astype(int)
img_dim_new = np.array([diag_glass, diag_glass]) # width, height
rot_img = np.zeros((diag_glass, diag_glass, glasses_cut.shape[2])).astype(np.
 ↪uint8)
print(img_dim_new)
rot_img[np.round((img_dim_new[0]-glasses_cut.shape[0])/2).astype(int):\
        np.round((img_dim_new[0]-glasses_cut.shape[0])/2).astype(int) +␣
 ↪glasses_cut.shape[0], \
```

```
        # x direction
        np.round((img_dim_new[1]-glasses_cut.shape[1])/2).astype(int):\
        np.round((img_dim_new[1]-glasses_cut.shape[1])/2).astype(int) +␣
↪glasses_cut.shape[1],:] = glasses_cut


#img_dim = np.array([glasses_cut.shape[0], glasses_cut.shape[1]]) # width,␣
↪height
rotM = cv2.getRotationMatrix2D(tuple(img_dim_new/2),angle_eyes, 1)

# calculate the diagonal as the largest dimension

glasses_rot = cv2.warpAffine(rot_img, rotM, dsize=tuple(img_dim_new),␣
↪borderMode=cv2.BORDER_CONSTANT,\
                             borderValue=0)

# creating a selection mask
sel_mask = glasses_rot[:,:,3].astype(float)/255
sel_mask_arr = np.stack([sel_mask, sel_mask, sel_mask], axis=2)

# displaying the glasses
plt.subplot(121)
plt.imshow(glasses_rot[:,:,:3])
plt.axis(False)

plt.subplot(122)
plt.imshow(sel_mask, cmap='gray')
plt.axis(False)
plt.show()
```

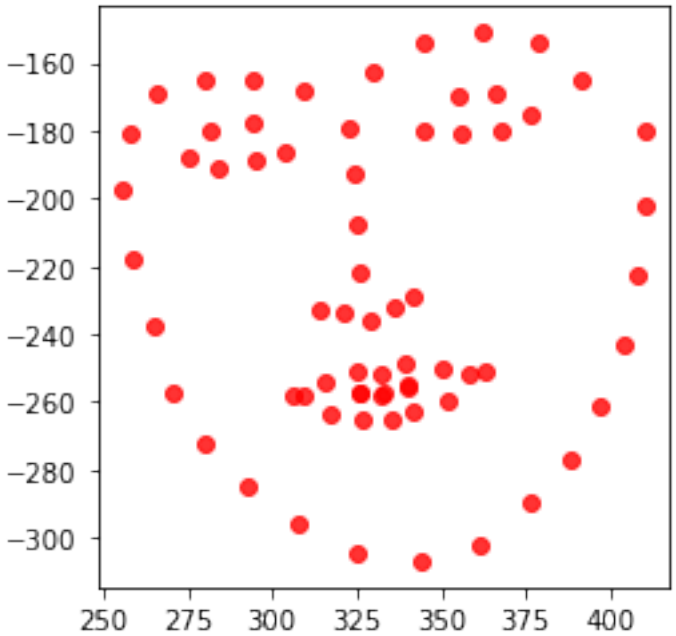
```
[186 186]
```

[39]:
```
## ROI
roi_xl = np.round(eyes_center[0] - (glasses_rot.shape[1]/2)).astype(int)
roi_yt = np.round(eyes_center[1] - (glasses_rot.shape[0]/2)).astype(int)
roi_xr = roi_xl + glasses_rot.shape[1]
roi_yb = roi_yt + glasses_rot.shape[0]

# overlaying the image
resImg = snapshot_rgb.copy()
resImg[roi_yt:roi_yb, roi_xl: roi_xr, :] = glasses_rot[:,:,:3]*sel_mask_arr + \
                                (snapshot_rgb[roi_yt:roi_yb, roi_xl: roi_xr, :
 ↪]*(1-sel_mask_arr))
resImg = resImg.astype(np.uint8)

# displaying the resulting image
plt.figure(figsize=(15,5))
plt.imshow(resImg)
plt.axis(False)
plt.show()
```

6 Add hat

Current availability of hats >Cowbow hat (*'hat1.png'*) >Pink hat (*'hat2.png'*) >Black hat with a red ribbon (*'hat3.png'*)

[40]:
```
# Detecting facial landmarks
true_eyes = []
true_face = []
snapshot_gray = cv2.cvtColor(snapshot_rgb, cv2.COLOR_RGB2GRAY)
# start with plotting the image
faces = detector(snapshot_gray)
if len(faces) == 0:
    print('No face has been found - Please, try to adjust parameters.')
    plt.figure(figsize=(10,7))
    plt.imshow(snapshot_gray, cmap='gray')
    plt.axis(False)
# If a face has been detected
for rect in faces:
# Calculate the face center
    shape = predictor(snapshot_gray, rect)
    coords = np.zeros((shape.num_parts, 2), dtype="int")
    for i in range(shape.num_parts):
        px, py = shape.part(i).x, shape.part(i).y
        coords[i] = (px, py)
        plt.scatter(px, -py, c='r', alpha=0.8)
    plt.axis('scaled')
    plt.show()
```



[41]:
```
hat_img = cv2.imread('hat1.png', cv2.IMREAD_UNCHANGED) #Change hats here
hat_img_rgb = hat_img.copy()
plt.figure(figsize=(10,5))

hat_rgb = hat_img[:,:,:3]
hat_rgb = hat_rgb[:,:,::-1]
hat_img_rgb[:,:,:3] = hat_rgb

plt.imshow(hat_rgb)
plt.axis(False)
plt.show()
```

[42]:
```
eyes_val = (coords[0], coords[16]) # later this can iterate over all found eyes
dist_eyes = ((eyes_val[0][0] - eyes_val[1][0])**2 + \
             (eyes_val[0][1] - eyes_val[1][1])**2) ** 0.5
eyes_center = np.array([(eyes_val[0][0] + eyes_val[1][0])/2,\
                        (eyes_val[0][1] + eyes_val[1][1])/2])
print('Distance between eyes is: {}'.format(dist_eyes))
print('Center of eyes is: {}'.format(eyes_center))

# resize hat
k_zoom = 2
hat_zoom = k_zoom*dist_eyes/hat_img.shape[1]
```

```
Distance between eyes is: 154.93547043850222
```

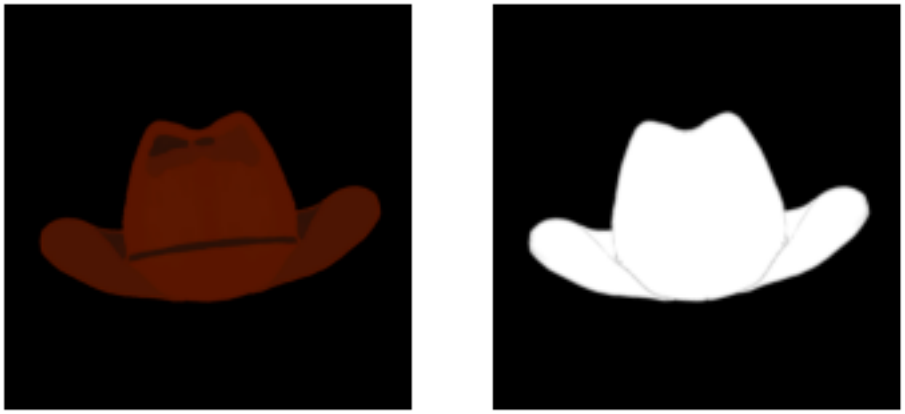
```
Center of eyes is: [333.  188.5]
```

[43]:
```
hat_size = np.array([hat_zoom*hat_img.shape[1], hat_zoom*hat_img.shape[0]]).
 ↪astype(int)
hat_cut = cv2.resize(hat_img_rgb, tuple(hat_size))

plt.imshow(hat_cut[:,:,:3])
plt.axis(False)
plt.show()
```

[44]:
```
## rotate hat
# equation is [(x1 - x0), (y1 - y0)]
vec_eyes = np.array([eyes_val[1][0] - eyes_val[0][0], \
                     eyes_val[1][1] - eyes_val[0][1]])
# calculating angle
angle_eyes = -np.arctan2(vec_eyes[1], vec_eyes[0])*180/np.pi
# normalizing angle
angle_eyes = angle_eyes - 90*(np.round(angle_eyes/90))

# rotating the image
# calculate the diagonal as the largest dimension
diag_hat = np.ceil((hat_cut.shape[0]**2 + hat_cut.shape[1]**2)**0.5).astype(int)
img_dim_new = np.array([diag_hat, diag_hat]) # width, height
rot_img = np.zeros((diag_hat, diag_hat, hat_cut.shape[2])).astype(np.uint8)
print(img_dim_new)
rot_img[np.round((img_dim_new[0]-hat_cut.shape[0])/2).astype(int):\
    np.round((img_dim_new[0]-hat_cut.shape[0])/2).astype(int) + hat_cut.
 ↪shape[0], \
    # x direction
```

```
    np.round((img_dim_new[1]-hat_cut.shape[1])/2).astype(int):\
    np.round((img_dim_new[1]-hat_cut.shape[1])/2).astype(int) + hat_cut.
 ↪shape[1],:] = hat_cut

#img_dim = np.array([hat_cut.shape[0], hat_cut.shape[1]]) # width, height
rotM = cv2.getRotationMatrix2D(tuple(img_dim_new/2),angle_eyes, 1)

# calculate the diagonal as the largest dimension
hat_rot = cv2.warpAffine(rot_img, rotM, dsize=tuple(img_dim_new),␣
 ↪borderMode=cv2.BORDER_CONSTANT,\
borderValue=0)

# creating a selection mask
sel_mask = hat_rot[:,:,3].astype(float)/255
sel_mask_arr = np.stack([sel_mask, sel_mask, sel_mask], axis=2)

# displaying the hat
plt.subplot(121)
plt.imshow(hat_rot[:,:,:3])
plt.axis(False)
plt.subplot(122)
plt.imshow(sel_mask, cmap='gray')
plt.axis(False)
plt.show()
```

```
[355 355]
```



[45]:
```
## ROI
roi_xl = np.round(eyes_center[0] - (hat_rot.shape[1]*0.5)).astype(int)
roi_yt = np.round(eyes_center[1] - (hat_rot.shape[0]*0.8)).astype(int)
roi_xr = roi_xl + hat_rot.shape[1]
roi_yb = roi_yt + hat_rot.shape[0]
```

```
# crop
cxlt, cxrt = max(roi_xl, 0), min(roi_xr, snapshot.shape[1])
cytt, cybt = max(roi_yt, 0), min(roi_yb, snapshot.shape[0])
cxls, cxrs = cxlt - roi_xl, hat_rot.shape[1] + cxrt - roi_xr
cyts, cybs = cytt - roi_yt, hat_rot.shape[0] + cybt - roi_yb

# overlaying the image
#resImg = snapshot.copy()
resImg[cytt:cybt, cxlt:cxrt, :] = \
  resImg[cytt:cybt, cxlt:cxrt, :] * (1-sel_mask_arr[cyts:cybs, cxls:cxrs]) + \
  hat_rot[cyts:cybs, cxls:cxrs, :3]
resImg = resImg.astype(np.uint8)

# displaying the resulting image
plt.figure(figsize=(15,5))
plt.imshow(resImg)
plt.axis(False)
plt.show()
```

7 Add lipstick

[46]:
```
def createBox(snapshot, points, scale=5, masked=False,cropped=True):
    if masked:
        mask = np.zeros_like(snapshot)
        mask = cv2.fillPoly(mask, [points], (255,255,255))
        snapshot = cv2.bitwise_and(snapshot,mask)
    if cropped:
        bbox = cv2.boundingRect(points)
        x,y,w,h = bbox
        imgCrop = snapshot[y:y+h, x:x+w]
        return imgCrop
    else:
        return mask
```

[47]:
```
for face in faces:
    x1,y1 = face.left(),face.top()
    x2,y2 = face.right(),face.bottom()
    #imgOriginal = cv2.rectangle(snapshot, (x1,y1),(x2,y2),(0,255,0),2)
    landmarks = predictor(snapshot_gray, face) # find all landmarks on image
    myPoints = []
    for n in range(68):
        x = landmarks.part(n).x
        y = landmarks.part(n).y
        myPoints.append([x,y])
        # center point, radius, color
        #cv2.circle(imgOriginal,(x,y), 2, (50,50,255),cv2.FILLED)
    myPoints = np.array(myPoints)
    #print(myPoints)
    imgLips= createBox(snapshot_rgb,myPoints[48:61],3, masked=True,␣
↪cropped=False)
    plt.figure(figsize=(10,7))
    plt.axis(False)
    plt.imshow(imgLips)
```

[48]:
```
imgColorLips = np.zeros_like(imgLips)
imgColorLips[:] = 0, 0, 255
imgColorLips = cv2.bitwise_and(imgLips, imgColorLips)
imgColorLips = cv2.GaussianBlur(imgColorLips, (7,7),10)
imgColorLips = cv2.addWeighted(snapshot_rgb,1, imgColorLips,0.2,0)

plt.figure(figsize=(15,7))
plt.axis(False)
plt.imshow(imgColorLips)
```

[48]: <matplotlib.image.AxesImage at 0x20514fc6520>

7.1 Literature

Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In P roceedings of the IEEE conference on computer vision and pattern recognition (pp. 1867-1874).

Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I -I). IEEE.