Vivek Onkarnath Rathi

200256752

ECE 558 Project 03

CODE –

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 12 04:40:55 2019

This code works only for square images
This contains code for P1 a,b,c

@author: Vivek Rathi
"""

import cv2
import numpy as np

#global variables
drawing = False # true if mouse is pressed
mode = True # if True, draw rectangle. Press 'e' to toggle to ellipse
ix,iy = -1,-1
sflag = True # if size of f_img & b_img is not same, s_flag sets to False


# draw function for GUI
def draw(event,x,y,flags,param):
    global ix,iy,lx,ly,drawing,m_shape

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix,iy = x,y

#    elif event == cv2.EVENT_MOUSEMOVE:
#        if drawing == True:
#            cv2.ellipse(fore_img,((x-ix)//2,(y-iy)//2),((y-iy)//2,(x-ix)//2),0,0,360,(0,255,0),1)
```

```
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        if mode == False:
            cv2.ellipse(f_imgcp,((ix+(x-ix)//2),(iy+(y-iy)//2)),(x-ix,y-iy),0,0,360,(0,0,0),1)
        else:
            cv2.rectangle(f_imgcp,(ix,iy),(x,y),(0,0,0),1)

        lx = x;
        ly = y;

#padding function- reflect type only
def padr(img,t,b,l,r):
    ir = img.shape[0]
    ic = img.shape[1]
    if len(img.shape) == 3:
        imgpad = np.zeros((ir+t+b,ic+l+r,3), dtype = 'float32')
    elif len(img.shape) == 2:
        imgpad = np.zeros((ir+t+b,ic+l+r), dtype = 'float32')
    imgpad[t:ir+b,l:ic+r] = img
    rows = imgpad.shape[0]
    cols = imgpad.shape[1]
    for i,j in zip(range(t-1,-1,-1),range(t,t+t,1)):
        imgpad[i,:] = imgpad[j,:]

    for i,j in zip(range(l-1,-1,-1),range(l,l+l,1)):
        imgpad[:,i] = imgpad[:,j]

    for i,j in zip(range(rows-b,rows,1),range(rows-b-1,rows-b-b-1,-1)):
        imgpad[i,:] = imgpad[j,:]

    for i,j in zip(range(cols-r,cols,1),range(cols-r-1,cols-l-l-1,-1)):
        imgpad[:,i] = imgpad[:,j]
```

```
    return imgpad

# Convolution Function- odd kernel only
def conv(img,k):
    imgcopy = np.copy(img).astype('float32')
    t=b=l=r = k.shape[0]//2
    padimg = padr(img,t,b,l,r)

    if len(img.shape) == 3:
        B = np.zeros((k.shape[0],k.shape[1],3))
        k_shift = k.shape[0]
        r = padimg.shape[0]
        c = padimg.shape[1]

        for i in range(r-k_shift-1):
            for j in range(c-k_shift-1):
                B[:,:,0] = np.multiply(k,padimg[i:i+k_shift,j:j+k_shift,0])
                B[:,:,1] = np.multiply(k,padimg[i:i+k_shift,j:j+k_shift,1])
                B[:,:,2] = np.multiply(k,padimg[i:i+k_shift,j:j+k_shift,2])
                imgcopy[i,j,0] = np.sum(B[:,:,0])
                imgcopy[i,j,1] = np.sum(B[:,:,1])
                imgcopy[i,j,2] = np.sum(B[:,:,2])
        return imgcopy

    elif len(img.shape) == 2:
        B = np.zeros((k.shape[0],k.shape[1]))
        k_shift = k.shape[0]
        r = padimg.shape[0]
        c = padimg.shape[1]
        for i in range(r-k_shift):
            for j in range(c-k_shift):
                B = np.multiply(k,padimg[i:i+k_shift,j:j+k_shift])
                imgcopy[i,j] = np.sum(B)
```

```
        return imgcopy

# spread intensities to 0,255 for display
def spread_linear(img,uplimit):
    s = np.copy(img)
    ma = img.max()
    mi = img.min()
    if len(img.shape) == 3:
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                s[i,j,:] = ((uplimit - 1)/(ma - mi)) * (img[i,j,:] - mi)
        return s.astype('uint8')
    else:
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                s[i,j] = ((uplimit - 1)/(ma - mi)) * (img[i,j] - mi)
        return s.astype('uint8')

# Downsamples the image using nearest neighbor interpolation
def downscale(img):
    old_r = img.shape[0]
    old_c = img.shape[1]
    new_r = (np.floor(img.shape[0]/ 2)).astype('int16')
    new_c = (np.floor(img.shape[1]/ 2)).astype('int16')

    r_r = new_r/old_r
    r_c = new_c/old_c

    R = (np.floor((np.arange(0,new_r,1))/r_r)).astype('int16')
    C = (np.floor((np.arange(0,new_c,1))/r_c)).astype('int16')

    dimg = img[R,:]
    dimg = dimg[:,C]
```

```
    return dimg

# Upsamples the image using nearest neighbor interpolation
def upscale(img,n_img):
    old_r = img.shape[0]
    old_c = img.shape[1]

    new_r = n_img.shape[0]
    new_c = n_img.shape[1]

    r_r = new_r/old_r
    r_c = new_c/old_c

    R = (np.floor((np.arange(0,new_r,1))/r_r)).astype('int16')
    C = (np.floor((np.arange(0,new_c,1))/r_c)).astype('int16')

    uimg = img[R,:]
    uimg = uimg[:,C]

    return uimg

# creates pyramid with images of decreasing size
def pyrdown(img):
    c_gaus = conv(img,k)
    return downscale(c_gaus)

# creates pyramid with images of increasing size
def pyrup(img,n_img):
    uimg = upscale(img,n_img)
    c_lap = conv(uimg,k)
    return c_lap
```

```
# creates Gaussian and Laplacian Pyramids
def ComputePyr(img,layers):
    global num_l, k
    num_l = 1
    shape = img.shape[0]
    for i in range(layers-1):
        if (shape//2 < 5):
            print("Sorry! We can only have maximum layers of %d " %num_l)
            layers = num_l
            break
        else:
            shape = shape //2
            num_l += 1

    #Gaussian Kernel
    x = cv2.getGaussianKernel(5,2)
    k = x*x.T

    #Gaussian Pyramid
    G = img.copy().astype('float32')
    gpA = [G]
    for i in range(layers-1):
        G = pyrdown(G)
        gpA.append(G)

    #Laplacian Pyramid
    lpA = [gpA[layers-1]]
    for i in range(layers-1,0,-1):
        GE = pyrup(gpA[i],gpA[i-1])
        L = np.subtract(gpA[i-1],GE)
        lpA.append(L)

    return gpA,lpA
```

```
# creates gaussian pyramid for mask
def mask_gpyr(mask,layers):
    G_M = np.copy(mask).astype('float32')
    gpM = [G_M]
    for i in range(layers-1):
        G_M = pyrdown(G_M)
        gpM.append(G_M)
    gpM.reverse()

    return gpM

# blends images to return blended image
def laplacian_blend(LA,LB,GM,layers):
    LS = []
    for la,lb,mask in zip(LA,LB,GM):
        ls = la * mask + lb * (1 - mask)
        LS.append(np.float32(ls))

    lap_bl = LS[0]
    for i in range(1,layers):
        lap_bl = pyrup(lap_bl,LS[i])
        lap_bl = cv2.add(lap_bl,LS[i])
    return np.clip(lap_bl,0,255).astype('uint8')



# creates mask of foreground image
def create_mask(img,x1,y1,x2,y2):
    mask = np.zeros((img.shape)).astype('float32')
    if mode == False:
        mask=cv2.ellipse(mask,((ix+(lx-ix)//2),(iy+(ly-iy)//2)),(lx-ix,ly-iy),0,0,360,(1,1,1),-1)
    else:
#        mask[y1+1:y2,x1+1:x2] = 1
```

```
        mask=cv2.rectangle(mask,(ix,iy),(lx,ly),(1,1,1),-1)
    return mask

# aligns f_img to b_img
def align(b_img,f_img,r,c):
#    global s_flag
#    s_flag = False
    a_img = np.zeros(b_img.shape)
    a_img[r:f_img.shape[0]+r,c:f_img.shape[1]+c] = f_img
    return a_img.astype('uint8')

#Start Code here

#read image-
b_img = cv2.imread('P3_BG_2.png',1)
f_img = cv2.imread('P3_FG_2.png',1)

# validation for aligning
if (b_img.shape[0] > f_img.shape[0]):
#    global s_flag
#    s_flag = False
    n_f_img = align(b_img,f_img,50,35) # this rows & columns need to be done manually
    f_img = np.copy(n_f_img)
    f_imgcp = np.copy(n_f_img)
else:
    f_imgcp = np.copy(f_img)

# display for GUI
cv2.namedWindow('FOREGROUND IMAGE',cv2.WINDOW_NORMAL)
cv2.setMouseCallback('FOREGROUND IMAGE',draw)

while(1):
    cv2.imshow('FOREGROUND IMAGE',f_imgcp)
```

```
    k = cv2.waitKey(1) & 0xFF
    if k == ord('e'): # ellipse
        mode = not mode
    elif k == 27:
        cv2.destroyAllWindows()
        break

# mask creation
mask = create_mask(f_imgcp,ix,iy,lx,ly)



# display images
cv2.namedWindow('FOREGROUND IMAGE', cv2.WINDOW_NORMAL)
cv2.imshow('FOREGROUND IMAGE',f_img)
cv2.namedWindow('MASK', cv2.WINDOW_NORMAL)
cv2.imshow('MASK',mask)
cv2.namedWindow('BACKGROUND IMAGE', cv2.WINDOW_NORMAL)
cv2.imshow('BACKGROUND IMAGE',b_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# compute pyramids- here you can enter layers as per requirement
GPA,LPA = ComputePyr(f_img,10)
GPB,LPB = ComputePyr(b_img,10)
GPM = mask_gpyr(mask,len(GPA))
final = laplacian_blend(LPA,LPB,GPM,len(GPA))

# display images
cv2.namedWindow('FOREGROUND IMAGE', cv2.WINDOW_NORMAL)
cv2.imshow('FOREGROUND IMAGE',f_imgcp)
cv2.namedWindow('MASK', cv2.WINDOW_NORMAL)
cv2.imshow('MASK',mask)
cv2.namedWindow('BACKGROUND IMAGE', cv2.WINDOW_NORMAL)
```

```
cv2.imshow('BACKGROUND IMAGE',b_img)
cv2.namedWindow('BLENDED IMAGE', cv2.WINDOW_NORMAL)
cv2.imshow('BLENDED IMAGE',final)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUTS

LAYERS VALIDATION

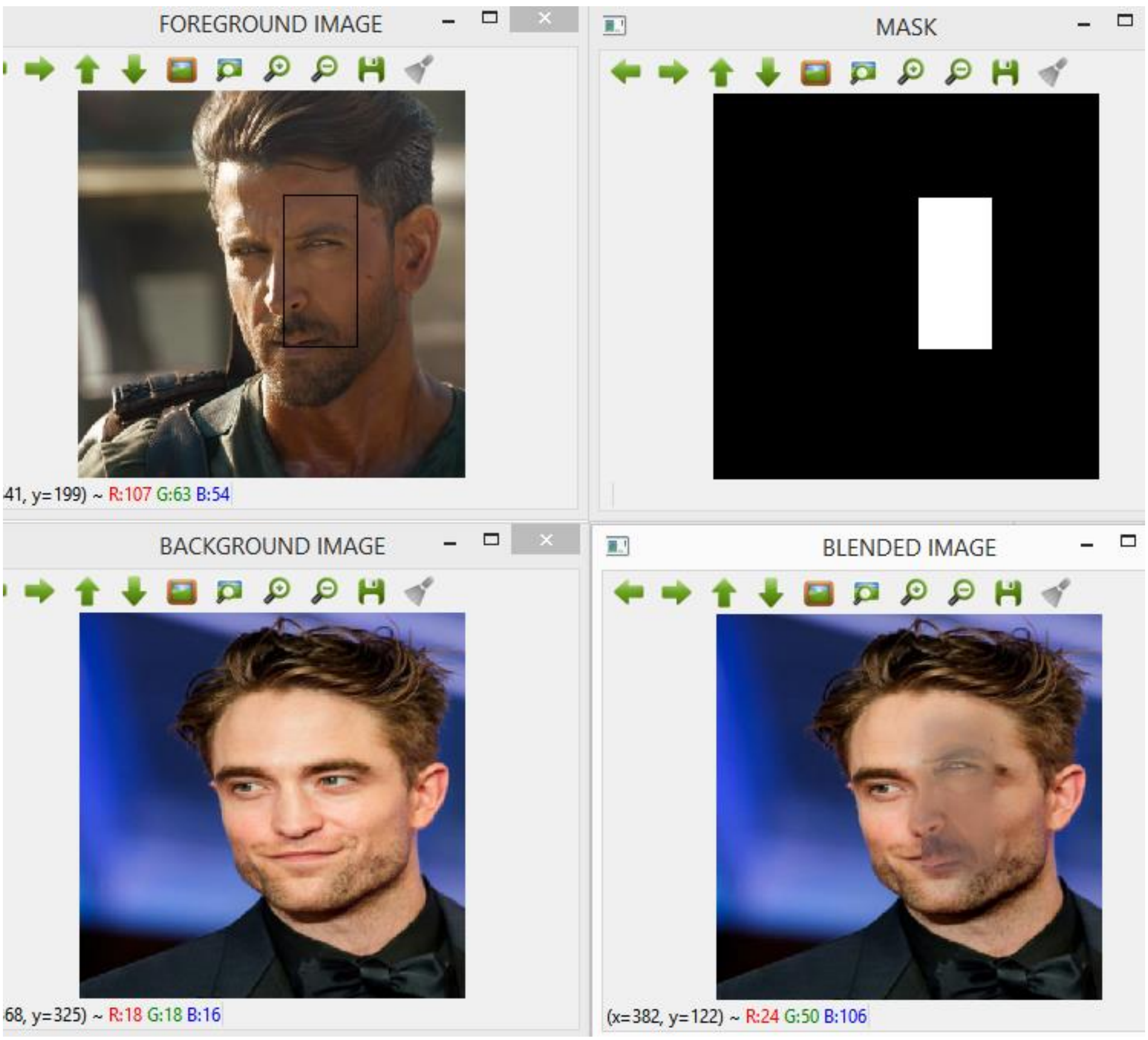
```
# compute pyramids- here you can enter layers as per requirement
GPA,LPA = ComputePyr(f_imgcp,10)
GPB,LPB = ComputePyr(b_img,10)
GPM = mask_gpyr(mask,len(GPA))
final = laplacian_blend(LPA,LPB,GPM,len(GPA))

# display images
cv2.namedWindow('FOREGROUND IMAGE', cv2.WINDOW_NORMAL)
```

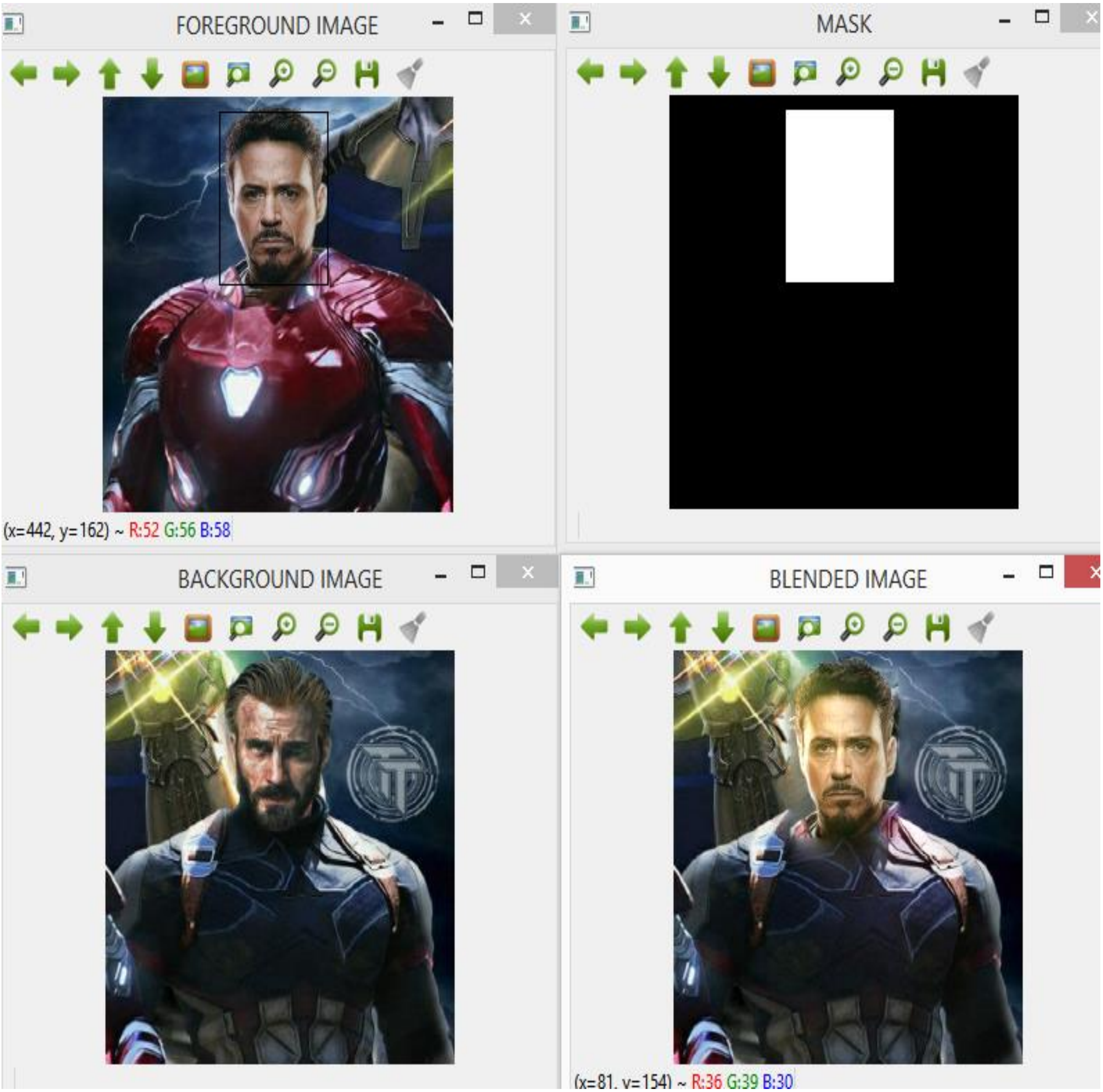
```
In [3]:

In [3]: runfile('D:/NCSTATE/ECE558/OpenCV/Project03/untitled1.py', wdir='D:/NCSTATE/ECE558/OpenCV/Project03')
Sorry! We can only have maximum layers of 7
```

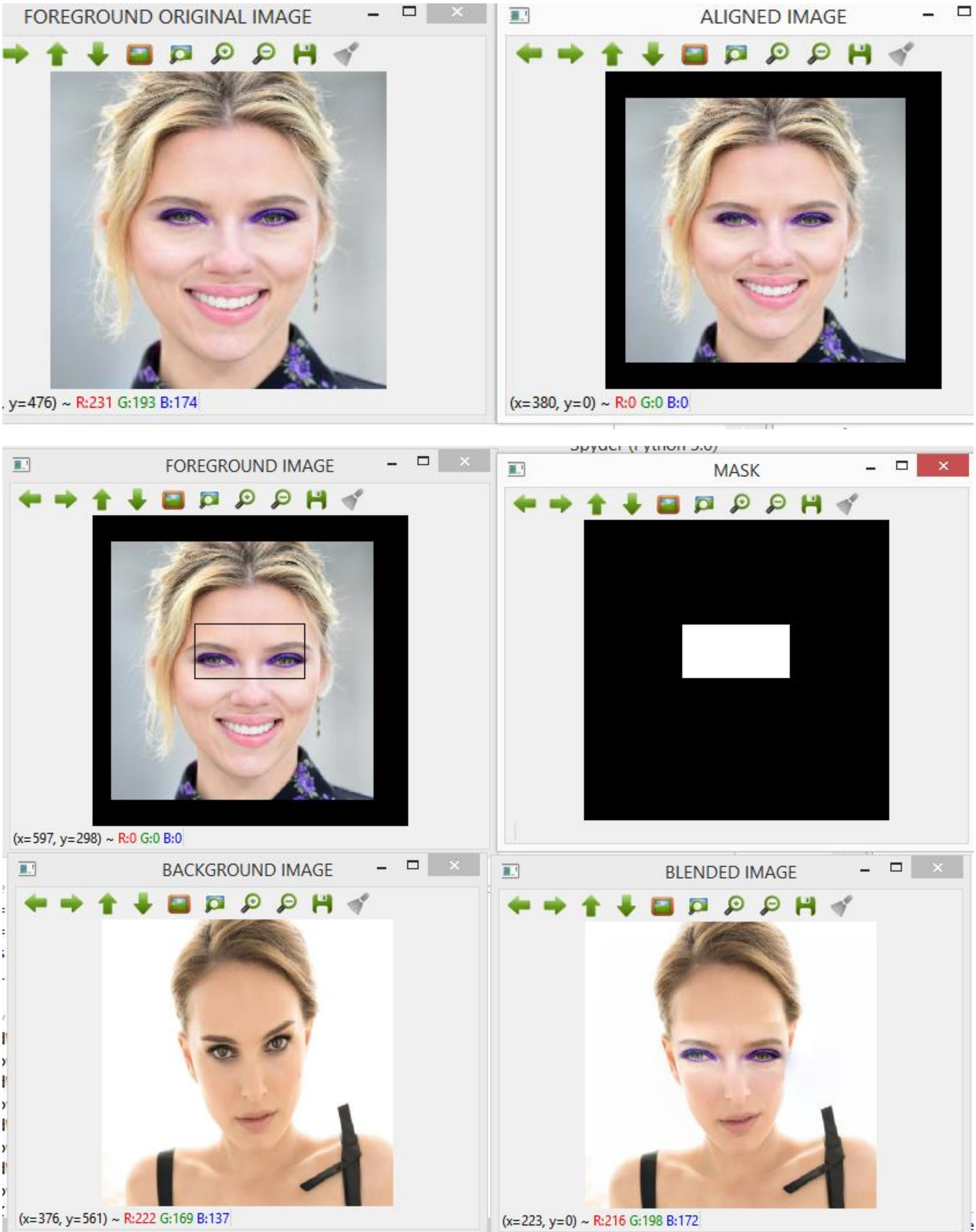
1.



2.


FOREGROUND IMAGE
MASK
(x=442, y=162) ~ R:52 G:56 B:58
BACKGROUND IMAGE
BLENDED IMAGE
(x=81, y=154) ~ R:36 G:39 B:30

3.


FOREGROUND ORIGINAL IMAGE
, y=476) ~ R:231 G:193 B:174
ALIGNED IMAGE
(x=380, y=0) ~ R:0 G:0 B:0
FOREGROUND IMAGE
(x=597, y=298) ~ R:0 G:0 B:0
MASK
BACKGROUND IMAGE
(x=376, y=561) ~ R:222 G:169 B:137
BLENDED IMAGE
(x=223, y=0) ~ R:216 G:198 B:172