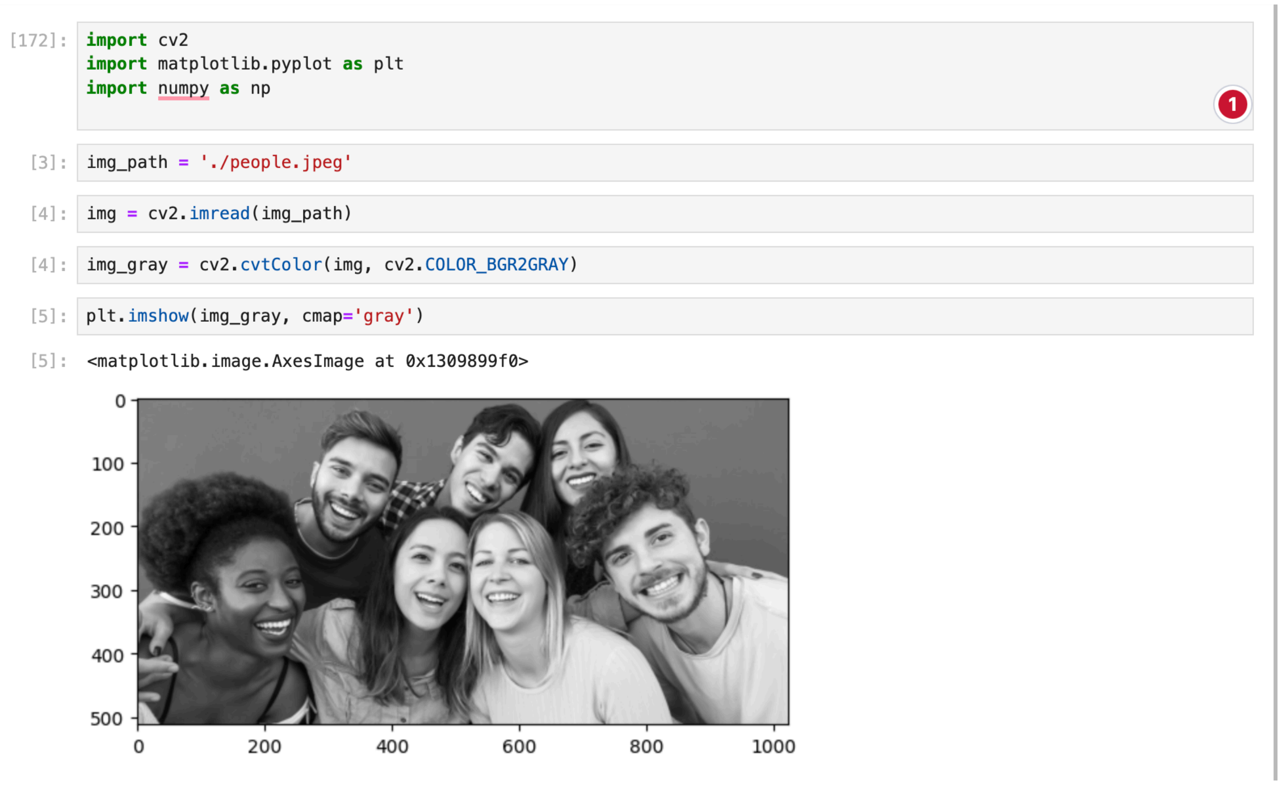
Task 1: Edge Detection and Contour Analysis

Write a Python program to read an image from the file system using OpenCV.



```
[172]: import cv2
       import matplotlib.pyplot as plt
       import numpy as np
```

```
[3]: img_path = './people.jpeg'
```

```
[4]: img = cv2.imread(img_path)
```

```
[4]: img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
[5]: plt.imshow(img_gray, cmap='gray')
```

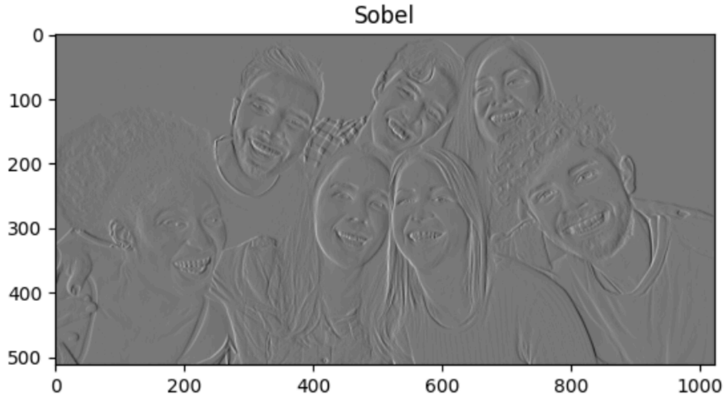
```
[5]: <matplotlib.image.AxesImage at 0x1309899f0>
```

Implement edge detection techniques, such as Sobel, Canny, and Laplacian filters, to highlight edges in the image.

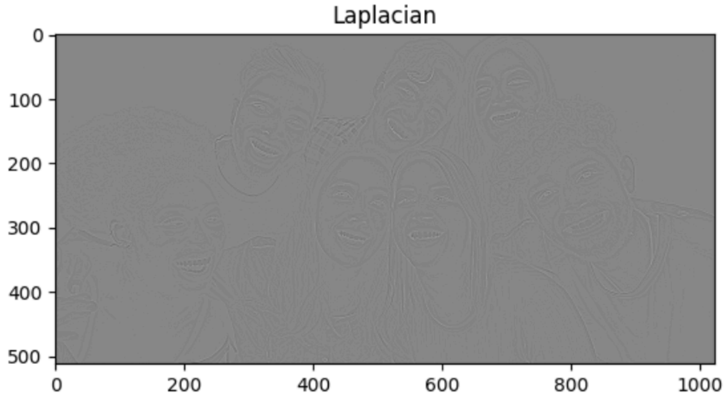
```
[ ]: def edge_detection_and_contours(imgpath):
         img = cv2.imread(imgpath)
         img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
         sobel_x = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, ksize=3)
         sobel_y = cv2.Sobel(img_gray, cv2.CV_64F, 0,   1, ksize=3)
         canny = cv2.Canny(img_gray,100,300)
         laplacian = cv2.Laplacian(img_gray, cv2.CV_64F)
         # Find cuotours in edge detected images
         contours, _ = cv2.findContours(canny.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
         for contour in contours:
             x, y, w, h = cv2.boundingRect(contour)
             cv2.rectangle(img, (x, y), (x+h, y+h), (255, 25, 0), 2)
         plt.imshow(img_gray, cmap='gray')
         plt.title('Gray')
         plt.show()
         plt.imshow(sobel_x)
         plt.title('Sobel')
         plt.show()
         plt.imshow(laplacian)
         plt.title('Laplacian')
         plt.show()
         plt.imshow(canny)
         plt.title('Canny')
         plt.show()
         plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), cmap='gray')
         plt.title('Contour')
```

```
[ ]: edge_detection_and_contours(img_path)
```

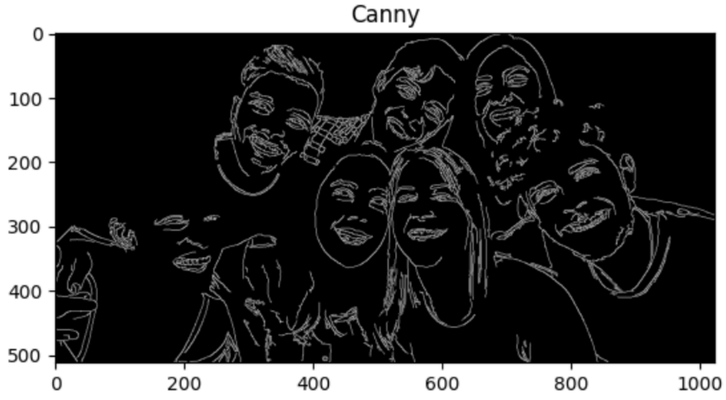
Sobel



Laplacian

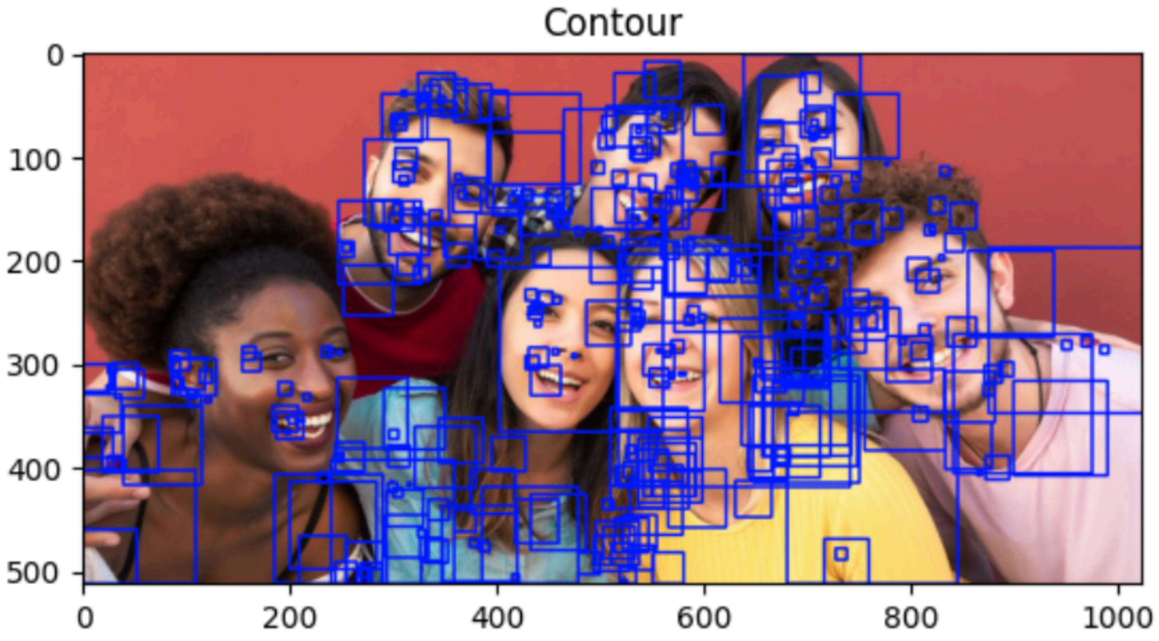


Canny



Apply contour detection on the edge-detected image to find and analyze contours of objects present in the image.

Draw bounding rectangles or circles around the detected contours and display the results.



Task 2: Perspective Transformation and Image Warping

Create a Python script that reads an image from the file system using OpenCV.

Select an object or a region of interest in the image and define four corner points to create a mask around the object.

```
[144]: box_img = cv2.imread('./perspective-img.webp')
```

```
[151]: box_mask = np.zeros_like(box_img)
```

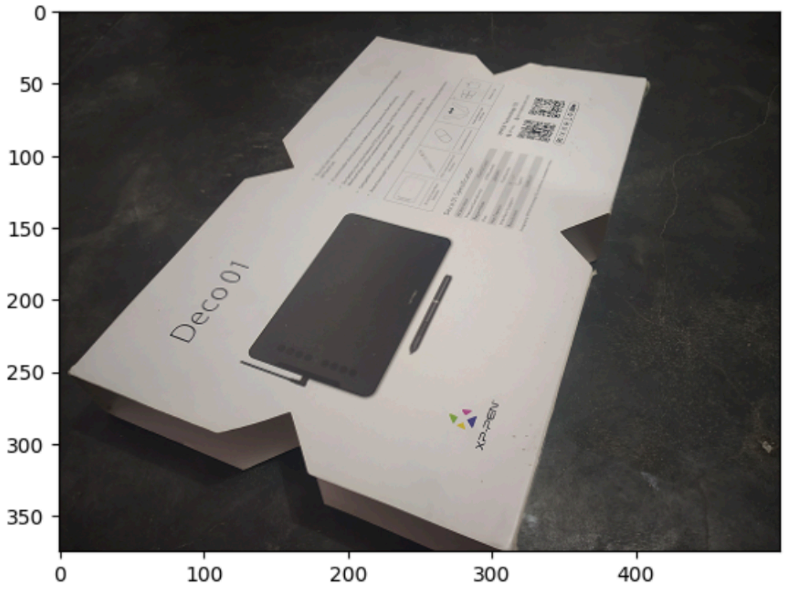
```
[146]: pt_A = [8, 247]
       pt_B = [308, 367]
       pt_C = [415, 56]
       pt_D = [222, 18]
```

```
[147]: pts = np.array([pt_A, pt_B, pt_C, pt_D],np.int32)
```

```
[156]: cv2.fillConvexPoly(box_mask, pts.astype(int), (0,128,255))
       print()
```

```
[224]: plt.imshow(cv2.cvtColor(box_img,cv2.COLOR_BGR2RGB))
```

```
[224]: <matplotlib.image.AxesImage at 0x12fdea830>
```



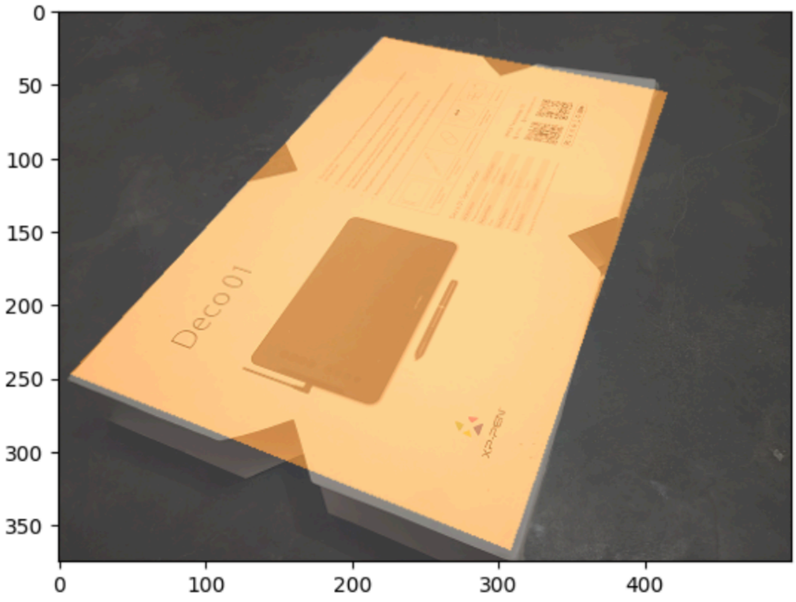
Implement perspective transformation to warp the selected region to a rectangular shape.

Display the original image, the selected region with the mask, and the result of the perspective transformation.

```
[159]: box_masked = cv2.addWeighted(box_img, 0.5, box_mask, 0.5, 50)
```

```
[160]: plt.imshow(cv2.cvtColor(box_masked,cv2.COLOR_BGR2RGB))
```

```
[160]: <matplotlib.image.AxesImage at 0x12dc760b0>
```



```
[219]: # Define the dimensions of the output rectangular shape
       width, height = 300, 300
       destination_pts = np.array([[0,0],
                                   [width-1,0],
                                   [width-1, height-1],
                                   [0, height-1],
                                 ], dtype='float32')

       pts = np.array([pt_A, pt_D, pt_C, pt_B], dtype='float32')
```

```
[220]: matrix = cv2.getPerspectiveTransform(pts, destination_pts)

       warped = cv2.warpPerspective(box_img, matrix, (width, height))
```

```
[221]: plt.imshow(cv2.cvtColor(warped,cv2.COLOR_BGR2RGB))
```

```
[221]: <matplotlib.image.AxesImage at 0x12fc3c1f0>
```