18BCS6201-CV Practical-4 (Gauri Prabhakar) (AI-ML-2)(B)

Aim: To detect contours and shape drawn within a given image using python and OpenCV.

In [18]:
```
# Importing necessary modules.
import cv2
import numpy as np
# Importing custom module.
```

Defining a function to get contours

In [21]:
```
# Defining a function to detect contours.
def getContours(img):
    # Finding contours in a binary image using the function 'findContours()'.
    # It takes the image, a flag which returns only the parents contours (contour retrieval mode) and
    # 'CHAIN_APPROX_NONE' to store all the contour points.
    # Sometimes there are shapes within shapes (parent and children) so to depict the relationship we use hierarchy.
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    # Iterating through the contours.
    for cnt in contours:
        # Storing the contour area.
        area = cv2.contourArea(cnt)
        # Returning the contour area.
        print(area)
        # If area is greater than 500 we enter the loop to draw contour.
        if area > 500:
            # Drawing contour using the 'drawContours()' function.
            # It takes the image, list of contour points, index of contours, color of the contour and thickness of the
            # contour as arguments.
            cv2.drawContours(imgContour, cnt, -1, (70, 0, 233), 3)
            # Creating a variable to store the arclength of the contours using the function 'arcLength()'.
            peri = cv2.arcLength(cnt, True)
            # Returning the arc length.
            print(peri)
            # Approximating the shape of the contours.
            approx = cv2.approxPolyDP(cnt, 0.02 * peri, True)
            # Returning the length of approx.
            print(len(approx))
            # Creating a variable to store the number of items in 'approx'.
            objCor = len(approx)
            # Drawing an approximate rectangle around the binary image.
            x, y, w, h = cv2.boundingRect(approx)

            # Checking the number of items in the 'approx' to classify the contour into different shapes.
            if objCor == 3:
                objectType = "Triangle"
            elif objCor == 4:
                aspRatio = w / float(h)
                if aspRatio > 0.98 and aspRatio < 1.03:
                    objectType = "Square"
                else:
                    objectType = "Rectangle"
            elif objCor > 4:
                objectType = "Circles"
            else:
                objectType = "None"

            # Setting up the properties of the rectangle drawn above.
            cv2.rectangle(imgContour, (x, y), (x + w, y + h), (255, 247, 0), 2)
            # Inserting text to label contours.
            cv2.putText(imgContour, objectType,
                        (x + (w // 2) - 10, y + (h // 2) - 10), cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.7,
                        (0, 0, 0), 2)
```

stackImages

In [22]:
```
# Defining a function 'stackImages()' to stack input images.
def stackImages(scale,imgArray):
    # Using 'len()' to return the number of items in the 'imgArray' object which is used to store 1-D and 2-D images as an array.
    rows = len(imgArray)
    cols = len(imgArray[0])
    # Returning the number of rows.
    print(rows)
    # Returning the number of columns.
    print(cols)
    # Returning the image array in literal format.
    print(imgArray)
    # Checking if we have a multilayer image.
    # The 'isinstance()' function returns true or false.
    # It takes the the columns and the list as an argument.
    rowsAvailable = isinstance(imgArray[0], list)

    # Storing the width and height of the image array.
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    # Returning the width and height of the image array.
    print (width)
    print (height)
    # If 'rowsAvailable' evaluates to True:
    if rowsAvailable:
        for x in range ( 0, rows):
            for y in range(0, cols):
                if imgArray[x][y].shape[:2] == imgArray[0][0].shape [:2]:
                    imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)
                else:
                    imgArray[x][y] = cv2.resize(imgArray[x][y], (imgArray[0][0].shape[1], imgArray[0][0].shape[0]), None, scale, scale)
                if len(imgArray[x][y].shape) == 2: imgArray[x][y]= cv2.cvtColor( imgArray[x][y], cv2.COLOR_GRAY2BGR)
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows

        # Horizontally stacking the image.
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
        # Vertically stacking the image.
        ver = np.vstack(hor)

    # If 'rowsAvailable' evaluates to False:
    else:
        for x in range(0, rows):
            if imgArray[x].shape[:2] == imgArray[0].shape[:2]:
                imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)
            else:
                imgArray[x] = cv2.resize(imgArray[x], (imgArray[0].shape[1], imgArray[0].shape[0]), None,scale, scale)
            if len(imgArray[x].shape) == 2: imgArray[x] = cv2.cvtColor(imgArray[x], cv2.COLOR_GRAY2BGR)

        # Horizontally stacking the image.
        hor = np.hstack(imgArray)
        # Vertically stacking the image.
        ver = hor
    return ver
```

Getting Contours

In [23]:
```
# Writing driver code to trigger the stacks and perform contouring.
# Creating a variable path which stores the path of the target image.
path = (r'C:\Users\gauri\Desktop\OpenCV Media\ice bear.jpg')
# Creating a variable to store the image using the '.imread()' function.
img = cv2.imread(path)
imgContour = img.copy()
# Creating a variable to store the grayscale image using the function '.cvtColor()'.
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Creating a variable to store the gaussian blurred image using the function '.GaussianBlur()'.
imgBlur = cv2.GaussianBlur(imgGray, (7, 7), 1)
# Creating a variable to store the edge detected image using the function '.Canny()'.
imgCanny = cv2.Canny(imgBlur, 50, 50)
# Retrieving contours from the edge detected image.
getContours(imgCanny)
# Creating a blank image.
imgBlank = np.zeros_like(img)
# Stacking the images using 'stackImages.py'.
imgStack = stackImages(0.8, ([img, imgGray, imgBlur],
                             [imgCanny, imgContour, imgBlank]))
# Returning the stacked and contoured images.
cv2.imshow("Stacked and contoured Images", imgStack)
# Setting up '.waitkey()' to wait for a specific time until any key is pressed.
cv2.waitKey(0)
# Destroying all windows.
cv2.destroyAllWindows()
```

```
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
5.5
7.5
0.0
102938.0
1579.047894001007
11
2
3
([array([[[226, 237, 241],
          [226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241],
          [226, 237, 241]],

         [[226, 237, 241],
          [226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241],
          [226, 237, 241]],

         [[226, 237, 241],
          [226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241],
          [226, 237, 241]],

         ...,

         [[226, 237, 241],
          [226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241],
          [226, 237, 241]],

         [[226, 237, 241],
          [226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241],
          [226, 237, 241]],

         [[226, 237, 241],
          [226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241],
          [226, 237, 241]]], dtype=uint8), array([[237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237],
         ...,
         [237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237]], dtype=uint8), array([[237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237],
         ...,
         [237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237],
         [237, 237, 237, ..., 237, 237, 237]], dtype=uint8)], [array([[0, 0, 0, ..., 0, 0, 0],
         [0, 0, 0, ..., 0, 0, 0],
         [0, 0, 0, ..., 0, 0, 0],
         ...,
         [0, 0, 0, ..., 0, 0, 0],
         [0, 0, 0, ..., 0, 0, 0],
         [0, 0, 0, ..., 0, 0, 0]], dtype=uint8), array([[[226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241]],

         [[226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241]],

         ...,

         [[226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241]],

         [[226, 237, 241],
          [226, 237, 241],
          ...,
          [226, 237, 241],
          [226, 237, 241]]], dtype=uint8), array([[[0, 0, 0],
          [0, 0, 0],
          ...,
          [0, 0, 0],
          [0, 0, 0]],

         [[0, 0, 0],
          [0, 0, 0],
          ...,
          [0, 0, 0],
          [0, 0, 0]],

         ...,

         [[0, 0, 0],
          [0, 0, 0],
          ...,
          [0, 0, 0],
          [0, 0, 0]],

         [[0, 0, 0],
          [0, 0, 0],
          ...,
          [0, 0, 0],
          [0, 0, 0]]], dtype=uint8)])
414
400
```

In [ ]:
```
```