18BCS6201-CV Practical-3 (Gauri Prabhakar) (AI-ML-2)(B)

Aim: To extract a warp perspective of a sub image from a larger image and then stack input and output images using python and OpenCV.

In [1]:
```
# Importing necessary modules.
import cv2
import numpy as np
```

Warp Perspective

In [2]:
```
# Creating a variable to store the image using the '.imread()' function.
img = cv2.imread(r'C:\Users\gauri\Desktop\OpenCV Media\cards.jpg')
# Storing the height and width of the image.
width, height = 250, 350
# The co-ordinates of the 3-D sub-image.
# Using '.float32' to convert to decimal values.
# Defining Old position of the sub-image.
pts1 = np.float32([[239, 103], [356, 146], [179, 277], [296, 320]])
# Defining New position of the sub-image.
pts2 = np.float32([[0, 0], [width, 0], [0, height], [width, height]])
# Transforming the perspective using the 'getPerspectiveTransform()' function.
matrix = cv2.getPerspectiveTransform(pts1, pts2)
# Getting the Warp Perspective of the image using the 'warpPerspective()' function.
imgOutput = cv2.warpPerspective(img, matrix, (width, height))
# Returning the original image.
cv2.imshow("Image", img)
# Returning the sub-image.
cv2.imshow("Sub-Image", imgOutput)
# Setting up '.waitKey()' to wait for a specific time until any key is pressed.
cv2.waitKey(0)
# Destroying all windows.
cv2.destroyAllWindows()
```

stackImages

In [3]:
```
# 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
```

Stacking the Images

In [5]:
```
# Writing driver code to trigger the stacks.
# Creating a variable to store the image using the '.imread()' function.
img = cv2.imread(r'C:\Users\gauri\Desktop\OpenCV Media\ice bear.jpg')
# Converting a variable to store the grayscale image using the function '.cvtColor()'.
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Getting the 'stackImages()' function.
imgStack = stackImages(0.5, ([img, imgGray, img], [img, img, img]))
# Horizontally stacking the image.
imgHor = np.hstack((img,img))
# Vertically stacking the image.
imgVer = np.vstack((img,img))

# Returning the horizontally, vertically and stacked images.
cv2.imshow("Horizontally Stacked Images",imgHor)
cv2.imshow("Vertically Stacked Images",imgVer)
cv2.imshow("Image Stack", imgStack)
# Setting up '.waitKey()' to wait for a specific time until any key is pressed.
cv2.waitKey(0)
# Destroying all windows.
cv2.destroyAllWindows()
```

```
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([[[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([[[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)])
414
400
```

In [ ]:
```
```