# Geometric Transformations of Images

## Goals

- Learn to apply different geometric transformations to images, like translation, rotation, affine transformation etc.
- You will see these functions: **cv.getPerspectiveTransform**

## Transformations

OpenCV provides two transformation functions, **cv.warpAffine** and **cv.warpPerspective**, with which you can perform all kinds of transformations. **cv.warpAffine** takes a 2x3 transformation matrix while **cv.warpPerspective** takes a 3x3 transformation matrix as input.

### Scaling

Scaling is just resizing of the image. OpenCV comes with a function **cv.resize()** for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are **cv.INTER_AREA** for shrinking and **cv.INTER_CUBIC** (slow) & **cv.INTER_LINEAR** for zooming. By default, the interpolation method **cv.INTER_LINEAR** is used for all resizing purposes. You can resize an input image with either of following methods:

```
import numpy as np
import cv2 as cv

img = cv.imread('messi5.jpg')

res = cv.resize(img,None,fx=2, fy=2, interpolation = cv.INTER_CUBIC)

#OR

height, width = img.shape[:2]
res = cv.resize(img,(2*width, 2*height), interpolation = cv.INTER_CUBIC)
```

### Translation

Translation is the shifting of an object's location. If you know the shift in the (x,y) direction and let it be $(t_x, t_y)$, you can create the transformation matrix $\mathbf{M}$ as follows:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

You can take make it into a Numpy array of type np.float32 and pass it into the **cv.warpAffine()** function. See the below example for a shift of (100,50):

```
import numpy as np
import cv2 as cv

img = cv.imread('messi5.jpg',0)
rows,cols = img.shape

M = np.float32([[1,0,100],[0,1,50]])
dst = cv.warpAffine(img,M,(cols,rows))

cv.imshow('img',dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

**warning**

The third argument of the **cv.warpAffine()** function is the size of the output image, which should be in the form of **(width, height)**. Remember width = number of columns, and height = number of rows.

See the result below:

**image**

### Rotation

Rotation of an image for an angle $\theta$ is achieved by the transformation matrix of the form

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

But OpenCV provides scaled rotation with adjustable center of rotation so that you can rotate at any location you prefer. The modified transformation matrix is given by

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha) \cdot center.y \end{bmatrix}$$

where:

$$\alpha = scale \cdot \cos\theta,$$
$$\beta = scale \cdot \sin\theta$$

To find this transformation matrix, OpenCV provides a function, **cv.getRotationMatrix2D**. Check out the below example which rotates the image by 90 degree with respect to center without any scaling.

```
img = cv.imread('messi5.jpg',0)
rows,cols = img.shape

# cols-1 and rows-1 are the coordinate limits.
M = cv.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),90,1)
dst = cv.warpAffine(img,M,(cols,rows))
```

See the result:

**image**

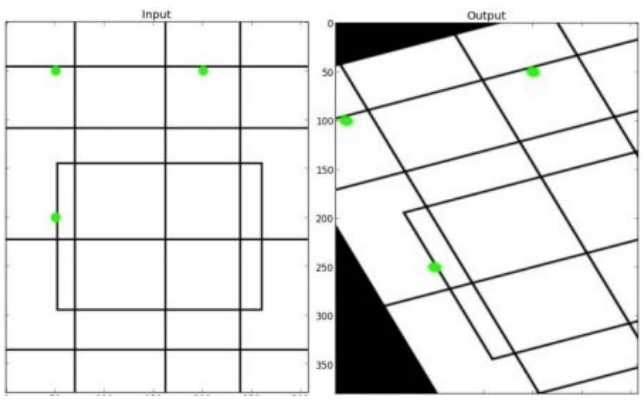
### Affine Transformation

In affine transformation, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from the input image and their corresponding locations in the output image. Then **cv.getAffineTransform** will create a 2x3 matrix which is to be passed to **cv.warpAffine**.

Check the below example, and also look at the points I selected (which are marked in green color):

```
img = cv.imread('drawing.png')
rows,cols,ch = img.shape

pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])

M = cv.getAffineTransform(pts1,pts2)

dst = cv.warpAffine(img,M,(cols,rows))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

See the result:




**image**

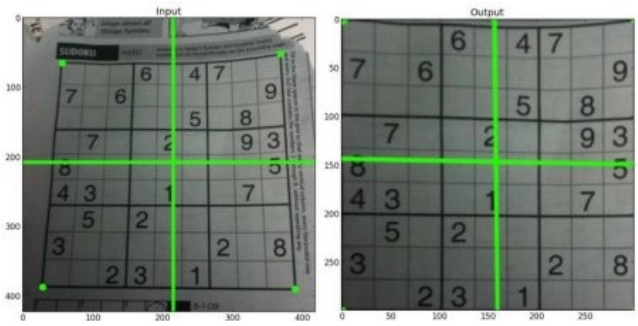
### Perspective Transformation

For perspective transformation, you need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, you need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then the transformation matrix can be found by the function **cv.getPerspectiveTransform**. Then apply **cv.warpPerspective** with this 3x3 transformation matrix.

See the code below:

```
img = cv.imread('sudoku.png')
rows,cols,ch = img.shape

pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])

M = cv.getPerspectiveTransform(pts1,pts2)

dst = cv.warpPerspective(img,M,(300,300))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

Result:




**image**

## Additional Resources

1. "Computer Vision: Algorithms and Applications", Richard Szeliski

## Exercises