Image Processing - Geometric transformations

Geometric Transformations

OpenCV Library provides two transformation functions : `cv.warpAffine` and `cv.warpPerspective` for all sorts of transformations.

- `cv.warpAffine` : takes $2x3$ transformation matrix as an input.
- `cv.warpPerspective` : takes $3x3$ transformation matrix as an input.

> **ELI5: Transformation Matrix**
> It is like a tool that helps us change how we see pictures. Image having a picture in your hand, you want to do different things with it, like move it around, turn it or even make it bigger or smaller. Transformation matrix instructs the computer to do these things in a language computer can process.

Some of the transformation functions we can do using OpenCV Library are following :-

- Scaling
- Translation
- Rotation
- Affine Transformation
- Perspective Transformation

1. Scaling

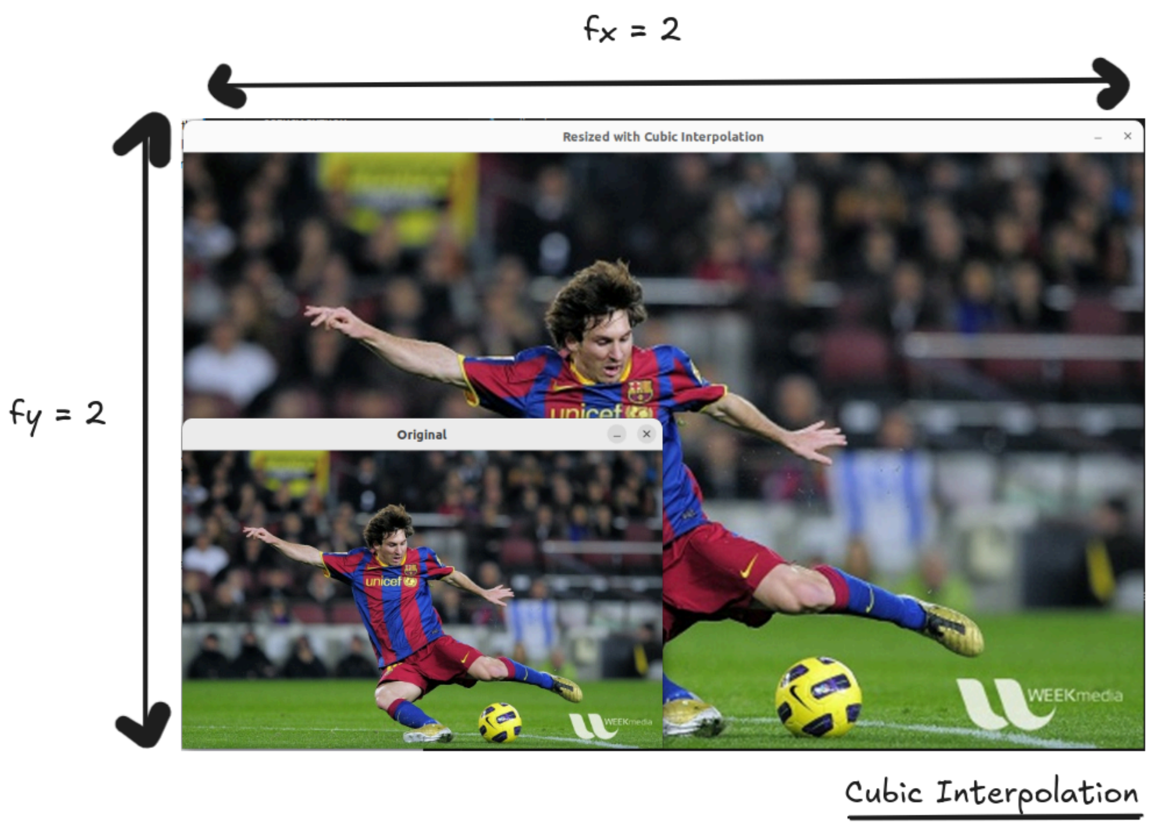
Scaling is just resizing of the image. Making it smaller or larger etc. OpenCV provides a function to scale/resize a source image : `cv.resize()`. The exact size of the image can be specified manually or can be specified using *scaling factor.*

Interpolation : A mathematical technique used to estimate unknown values from a known set of data. In *Computer Vision*, it is used to estimate pixel values at non-integer coordinates when transforming images. Essentially used in operations such as resizing, rotating and translating images.

> ELI5: Interpolation
> When a computer changes a picture (like making it bigger or turning it), it needs to "color in" new parts of the image. Interpolation is how it decides what colors to use for these new parts.

Different interpolation methods are used. Following are the most frequently used :-

- `cv.INTER_AREA` : Preferred for *Shrinking*
- `cv.INTER_CUBIC`: Preferred for *Shrinking* but slow
- `cv.INTER_LINEAR` : Preferred for Zooming. (Default)



Source Code :-

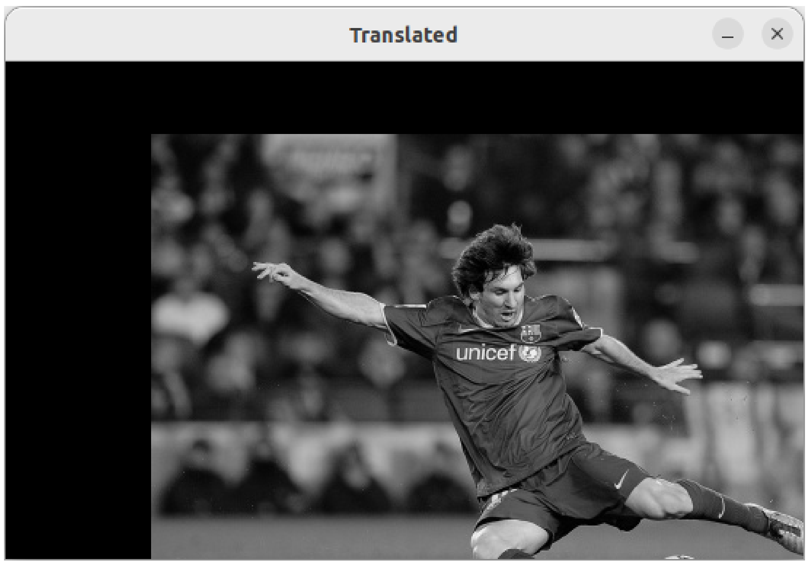
```
import numpy as np
import cv2 as cv

img = cv.imread("samples/messi5.jpg")
res = cv.resize(img, None, fx=2, fy=2, interpolation = cv.INTER_CUBIC)

cv.imshow("Original", img)
cv.imshow("Resized with Cubic Interpolation", res)
cv.waitKey(0)

cv.destroyAllWindows()
```

Code Analysis :-

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

To mention the x and y axis, scaling factor.

2. Translation



Translation is the shifting of an object's location. We shift the image by generating and using a transformation matrix :

$$M = \begin{matrix} 1 & 0 & x \\ 0 & 1 & y \end{matrix}$$

``

, where x, y are the shift in coordinates you want to do.

Approach :-

- Make a numpy array of type *np.float32* of the transformation matrix.
- Use warpAffine function to do translation.

Source Code :-

```
import numpy as np
import cv2 as cv

img = cv.imread("samples/messi5.jpg", cv.IMREAD_GRAYSCALE)
rows, cols = img.shape

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

cv.imshow("Translated", translated)
cv.waitKey(0)
cv.destroyAllWindows()
```

Code Analysis :-

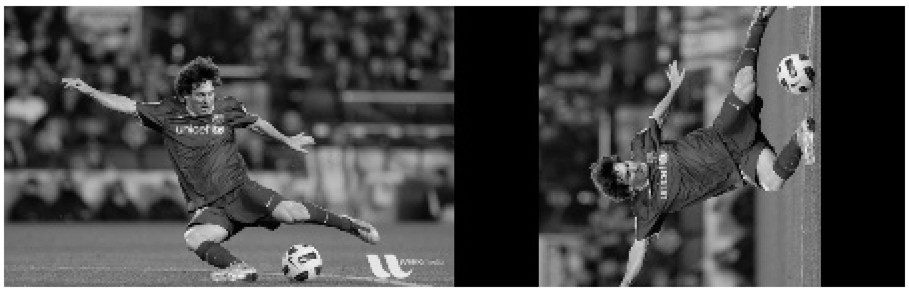
```
M = np.float32([[1, 0, 100], [0, 1, 50]])
```

To create a 2-D array along with the x, y coordinates to translate.

```
translated = cv.warpAffine(img, M, (cols, rows))
```

Notice the order of (cols, rows) in the above statement. Remember width = number of columns and height = number of rows.

3. Rotation



Rotation of an image for an angle θ , it is achieved by using a transformation matrix of the form :-

$$M = \begin{matrix} \alpha & \beta & (1-\alpha) * center * x - \beta * center * y \\ -\beta & \alpha & \beta * center * x + (1-\alpha) * center * y \end{matrix}$$

where α , β = scale.cosθ and scale.sinθ respectively, center being the point of rotation origin.

Source Code :-

```
img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "Unable to find the file!"

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)

rotated = cv.warpAffine(img,M,(cols,rows))
```

Code Analysis :-

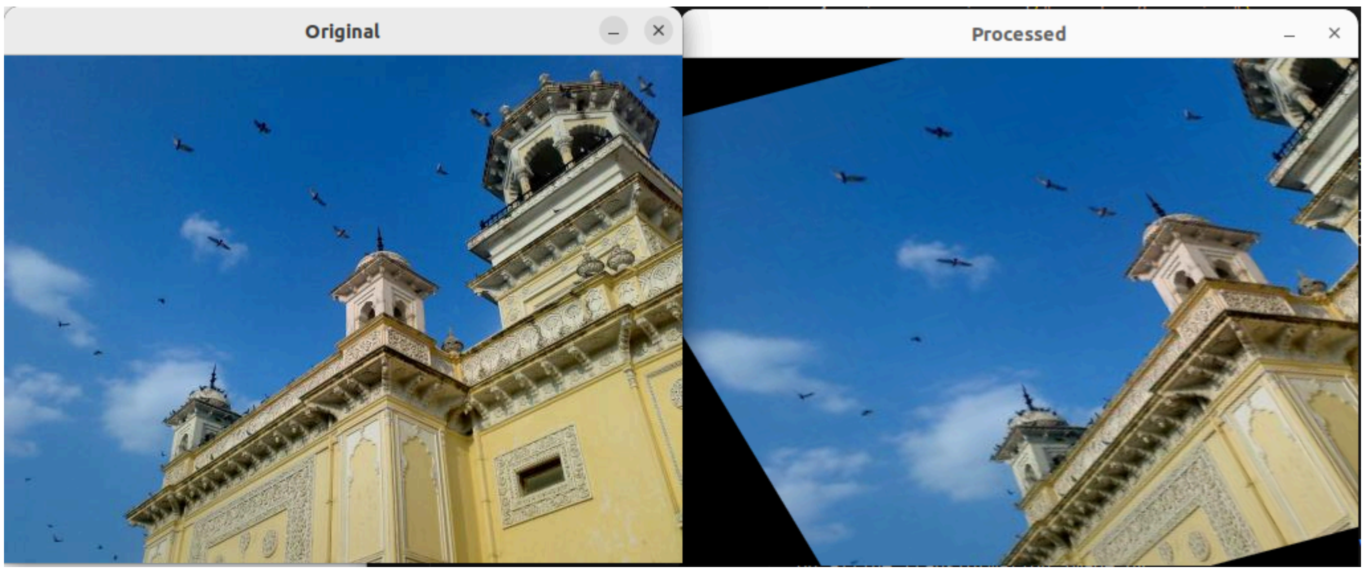
```
M = cv.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),90,1)
```

cols-1 and rows-1 are the limits of coordinates available to process for that particular image. In the above statements, these limit/2 are used to calculate the rotation center point. 90 ° is the clockwise angle to rotate by. 1 is the scale factor to be used.

4. Affine Transformation

Affine : word relates to any sort of transformations that preserve the parallelness if it exist. Parallel lines will still be parallel in the processed image. Though it may modify the distance between points on the lines.

In an affine transformation, all *parallel lines* in the original image will still be *parallel* in the processed image.



To generate a transformation matrix, we need *three points from the input image* and *points where it should be in the processed image.* We don't need to manually create the transformation matrix, OpenCV provides `cv.getAffineTransform` function, that will generate this 2x3 transformation matrix for us.

Source Code :-

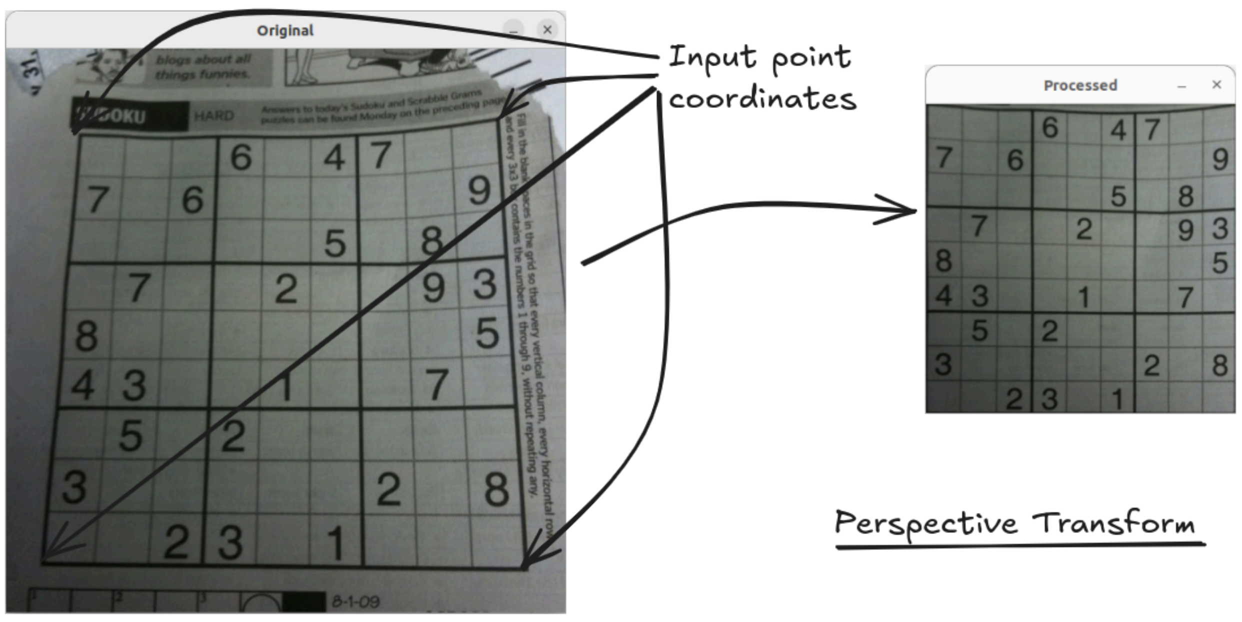
```
img = cv.imread("samples/home.jpg")
assert img is not None, "Unable to find this file in the samples directory
provided!"

rows, cols, ch = img.shape
```

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

affine_matrix = cv.getAffineTransform(pts1, pts2)
output = cv.warpAffine(img, affine_matrix, (cols, rows))

cv.imshow("Original", img)
cv.imshow("Processed", output)
cv.waitKey(0)
```

In the above code, I mentioned point's coordinates for all 3 points in the original and as well as where they should be in the processed image. Then used `cv.warpAffine` function to conduct affine transform.

5. Perspective Transformation

In perspective transformation, straight lines will remain straight even after the transformation. For this particular transformation, we will need a $3 * 3$ transformation matrix



Important points to remember :-

- We need a $3x3$ dimension transformation matrix.
- We need 4 points on the input image and as well as for the output image.
- Among 4 points chosen, 3 points must not be collinear.
- Use `cv.getPerspectiveTransform` function to generate a transformation matrix.

Source Code :-

```
img = cv.imread("samples/sudoku.png")

rows, cols, ch = img.shape
pts1 = np.float32([[73, 86], [489, 70], [37, 514], [519, 519]])
pts2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])

perspective_matrix = cv.getPerspectiveTransform(pts1, pts2)
output = cv.warpPerspective(img, perspective_matrix, (300, 300))

cv.imshow("Original", img)
cv.imshow("Processed", output)
cv.waitKey(0)
cv.destroyAllWindows()
```

Code Analysis :-

```
pts1 = np.float32([[73, 86], [489, 70], [37, 514], [519, 519]])
pts2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
```

Points coordinates of the 4-points to transform relative to. Input and as well as output coordinates (0 - 300) in my case.

```
output = cv.warpPerspective(img, perspective_matrix, (300, 300))
```

`warpPerspective` first parameter is the source image, second is generated matrix, third is size of the processed image.

References :-

Python OpenCV Docs : Geometric transformations in OpenCV