TheAILearner

Mastering Artificial Intelligence

TAG ARCHIVES: PERSPECTIVE TRANSFORMATION

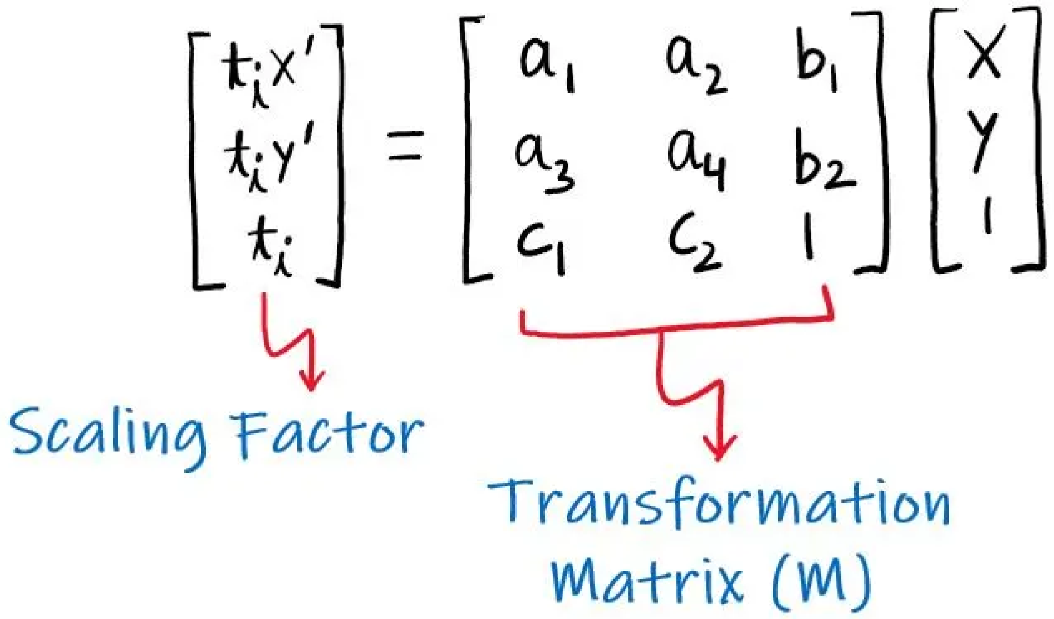
Perspective Transformation

In this blog, we will discuss what is perspective transformation and how to perform this transformation using OpenCV-Python. So, let's get started.

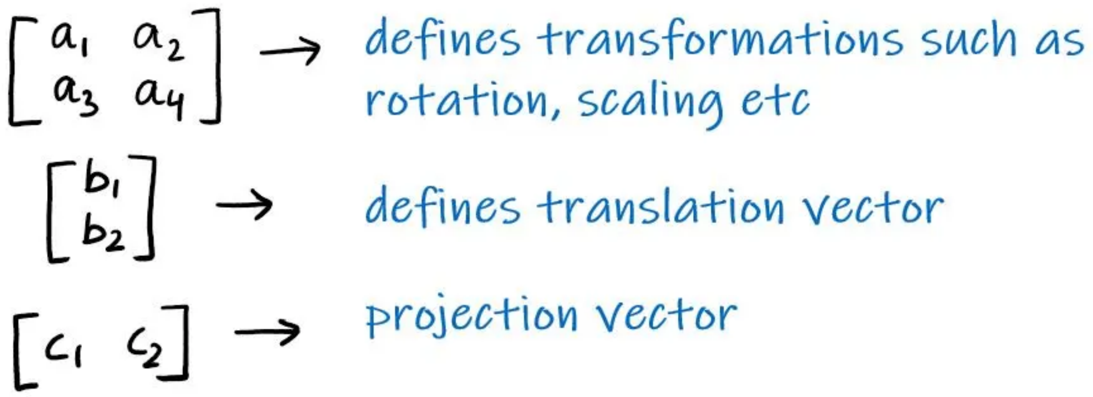
What is Perspective Transformation?

As clear from the name, the perspective transformation is associated with the change in the viewpoint. This type of transformation does not preserve parallelism, length, and angle. But they do preserve collinearity and incidence. This means that the straight lines will remain straight even after the transformation.

In general, the perspective transformation can be expressed as


$\begin{bmatrix} t_i x' \\ t_i y' \\ t_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
Scaling Factor
Transformation Matrix (M)

Here, (x', y') are the transformed points while (x, y) are the input points. The transformation matrix (M) can be seen as a combination of


$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \rightarrow$ defines transformations such as rotation, scaling etc
$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \rightarrow$ defines translation vector
$\begin{bmatrix} c_1 & c_2 \end{bmatrix} \rightarrow$ projection vector

For affine transformation, the projection vector is equal to 0. Thus, affine transformation can be considered as a particular case of perspective transformation.

Since the transformation matrix (M) is defined by 8 constants(degree of freedom), thus to find this matrix we first select 4 points in the input image and map these 4 points to the desired locations in the unknown output image according to the use-case (This way we will have 8 equations and 8 unknowns and that can be easily solved).

Once the transformation matrix is calculated, then we apply the perspective transformation to the entire input image to get the final transformed image. Let's see how to do this using OpenCV-Python.

OpenCV

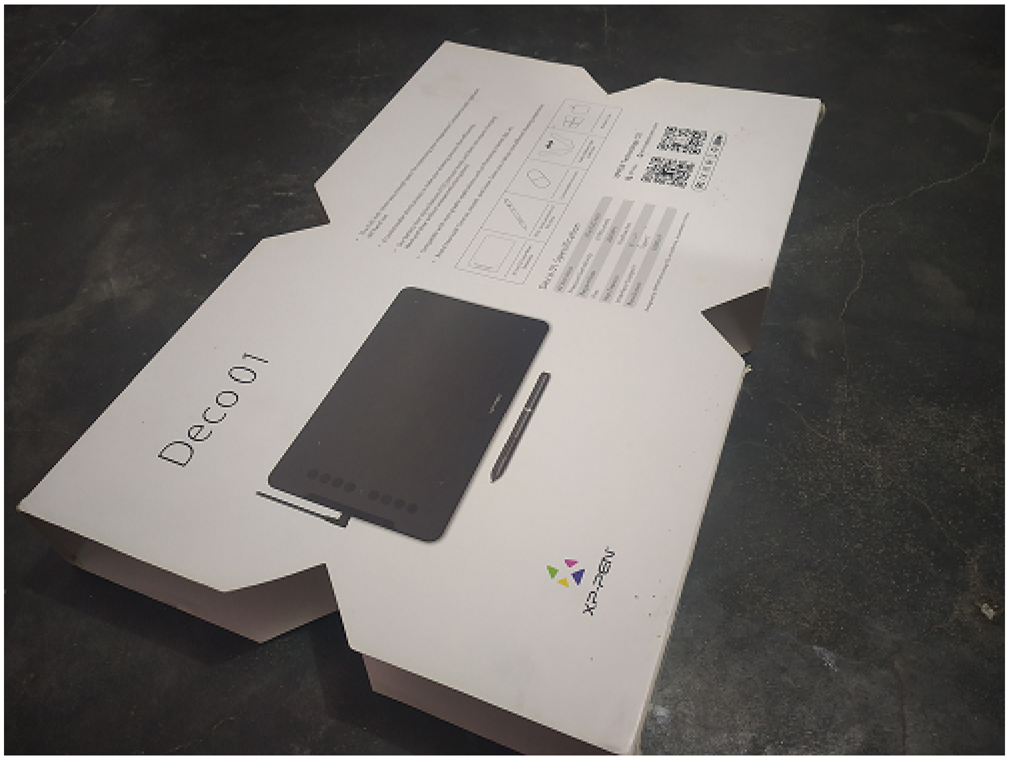
OpenCV provides a function cv2.getPerspectiveTransform() that takes as input the 4 pairs of corresponding points and outputs the transformation matrix. The basic syntax is shown below.

```
transform_mat = cv2.getPerspectiveTransform(src, dst)

# src: coordinates in the source image
# dst: coordinates in the output image
```

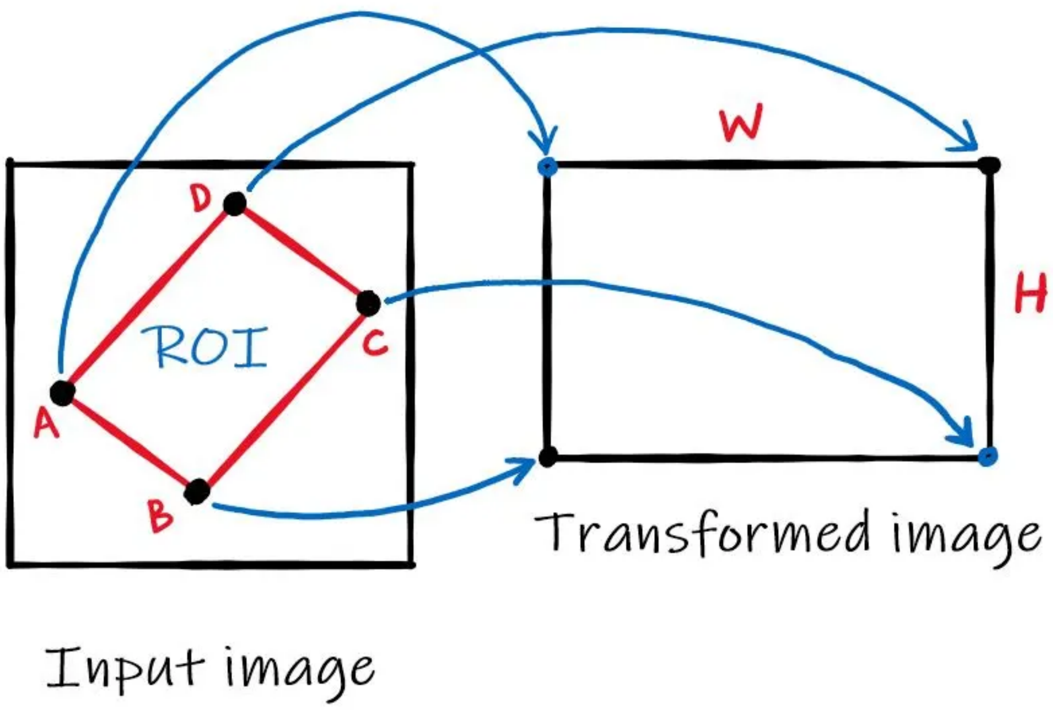
Once the transformation matrix (M) is calculated, pass it to the cv2.warpPerspective() function that applies the perspective transformation to an image. The syntax of this function is given below.

```
dst = cv.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue

# src: input image
# M: Transformation matrix
# dsize: size of the output image
# flags: interpolation method to be used
```

Now, let's take a very classic example of perspective transform to obtain a top-down, "birds-eye view" of an image. Let's understand step by step how to perform perspective transform using the below image.



Below is the image showing the basic idea which we will perform.



First, we will select the representative points (usually the corner points) for our region of interest (ROI). This can be done manually using matplotlib as shown below.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# To open matplotlib in interactive mode
%matplotlib qt5


# Load the image
img = cv2.imread('D:/downloads/deco.jpg')

# Create a copy of the image
img_copy = np.copy(img)

# Convert to RGB so as to display via matplotlib
# Using Matplotlib we can easily find the coordinates
# of the 4 points that is essential for finding the
# transformation matrix
img_copy = cv2.cvtColor(img_copy,cv2.COLOR_BGR2RGB)

plt.imshow(img_copy)
```

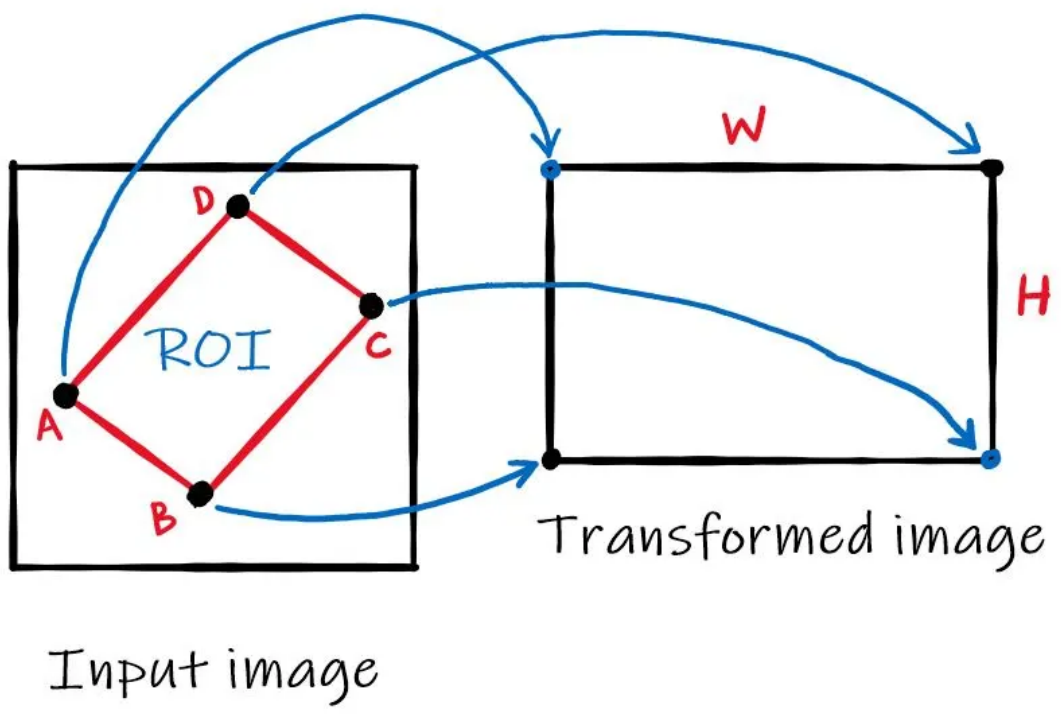
The above code opens up an interactive window. Use the mouse pointer to obtain the 4 corner coordinates which are shown below. Points are ordered anti-clockwise as shown in the above figure.

```
# All points are in format [cols, rows]
pt_A = [41, 2001]
pt_B = [2438, 2986]
pt_C = [3266, 371]
pt_D = [1772, 136]
```

Then we will map these 4 points to the desired locations in the unknown output image according to the use-case. Here, since we used the corner points so we will derive the width and height of the output image from these 4 points as shown below (You can also manually specify the mapping). Below I have used the L2 norm. You can use L1 also.

```
# Here, I have used L2 norm. You can use L1 also.
width_AD = np.sqrt(((pt_A[0] - pt_D[0]) ** 2) + ((pt_A[1] - pt_D[1]) ** 2))
width_BC = np.sqrt(((pt_B[0] - pt_C[0]) ** 2) + ((pt_B[1] - pt_C[1]) ** 2))
maxWidth = max(int(width_AD), int(width_BC))


height_AB = np.sqrt(((pt_A[0] - pt_B[0]) ** 2) + ((pt_A[1] - pt_B[1]) ** 2))
height_CD = np.sqrt(((pt_C[0] - pt_D[0]) ** 2) + ((pt_C[1] - pt_D[1]) ** 2))
maxHeight = max(int(height_AB), int(height_CD))
```

Now specify the mapping as shown in the image below (same image as above).



```
input_pts = np.float32([pt_A, pt_B, pt_C, pt_D])
output_pts = np.float32([[0, 0],
                        [0, maxHeight - 1],
                        [maxWidth - 1, maxHeight - 1],
                        [maxWidth - 1, 0]])
```

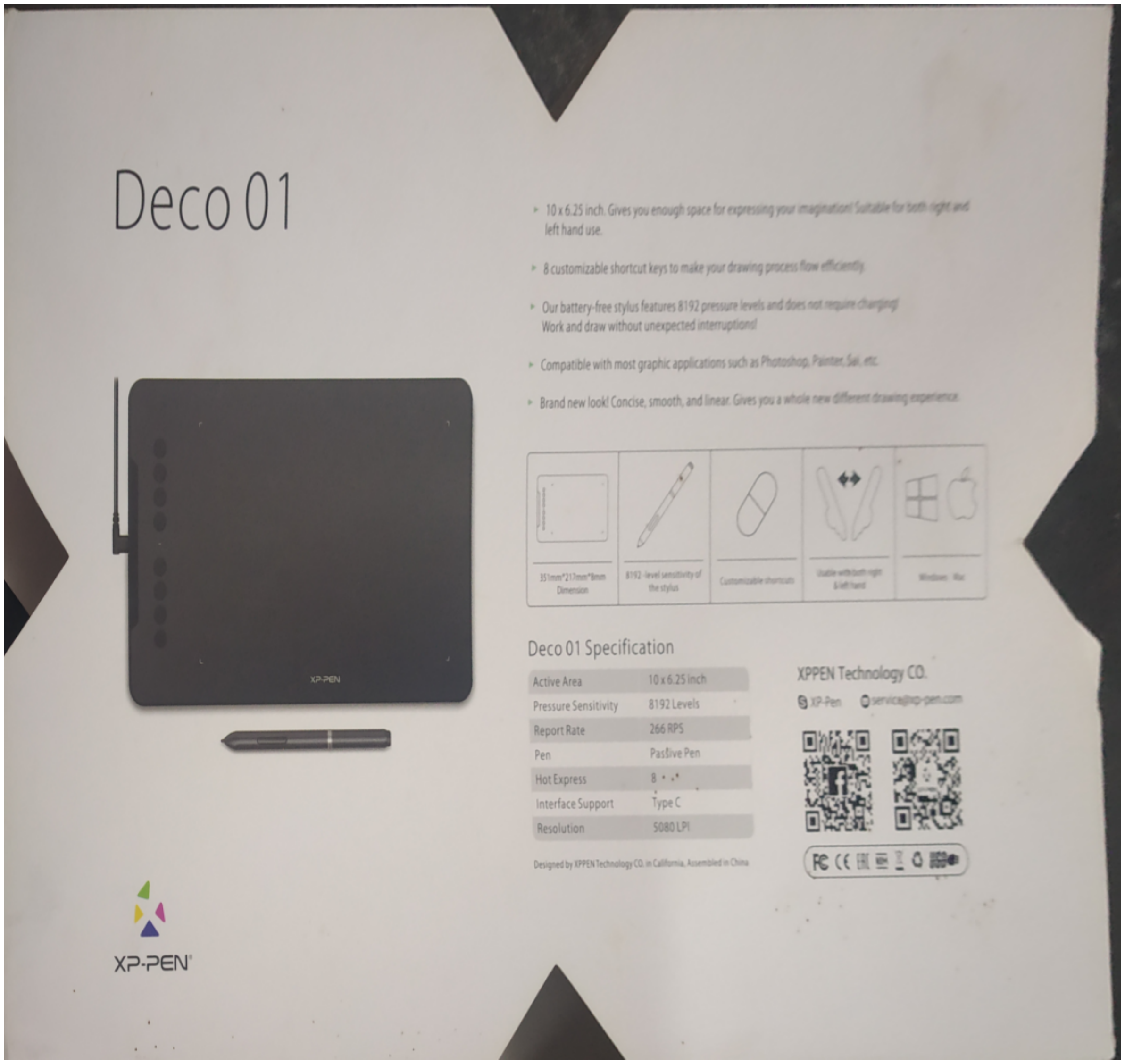
Now, compute the transformation matrix (M) using the cv2.getPerspectiveTransform() function as shown below

```
# Compute the perspective transform M
M = cv2.getPerspectiveTransform(input_pts,output_pts)
```

After calculating the transformation matrix, apply the perspective transformation to the entire input image to get the final transformed image.

```
out = cv2.warpPerspective(img,M,(maxWidth, maxHeight),flags=cv2.INTER_LINEAR)
```

Below is the transformed image.


Deco 01
10 x 6.25 inch. Gives you enough space for expressing your imagination! Suitable for both right and left hand use.
8 customizable shortcut keys to make your drawing process flow efficiently.
Our battery-free stylus features 8192 pressure levels and does not require charging! Work and draw without unexpected interruptions!
Compatible with most graphic applications such as Photoshop, Painter, Sai, etc.
Brand new look! Concise, smooth, and linear. Gives you a whole new different drawing experience.
Deco 01 Specification
Active Area 10 x 6.25 inch
Pressure Sensitivity 8192 Levels
Report Rate 266 RPS
Pen Passive Pen
Interface Support Type C
Resolution 5080 LPI
XPPEN Technology CO.
XP-PEN

That’s all for this blog. In the next blog, we will discuss how to automatically choose the 4 points using the contours. Hope you enjoy reading.

If you have any doubts/suggestions please feel free to ask and I will do my best to help or improve myself. Good-bye until next time.

This entry was posted in Image Processing and tagged cv2.getPerspectiveTransform(), cv2.warpPerspective(), image processing, opencv python, perspective transformation, perspective transformation opencv on 6 Nov 2020 [https://theailearner.com/2020/11/06/perspective-transformation/] by kang & atul.

Geometric Transformation of images using OpenCV-Python

Before reading, please refer to this blog for better understanding.

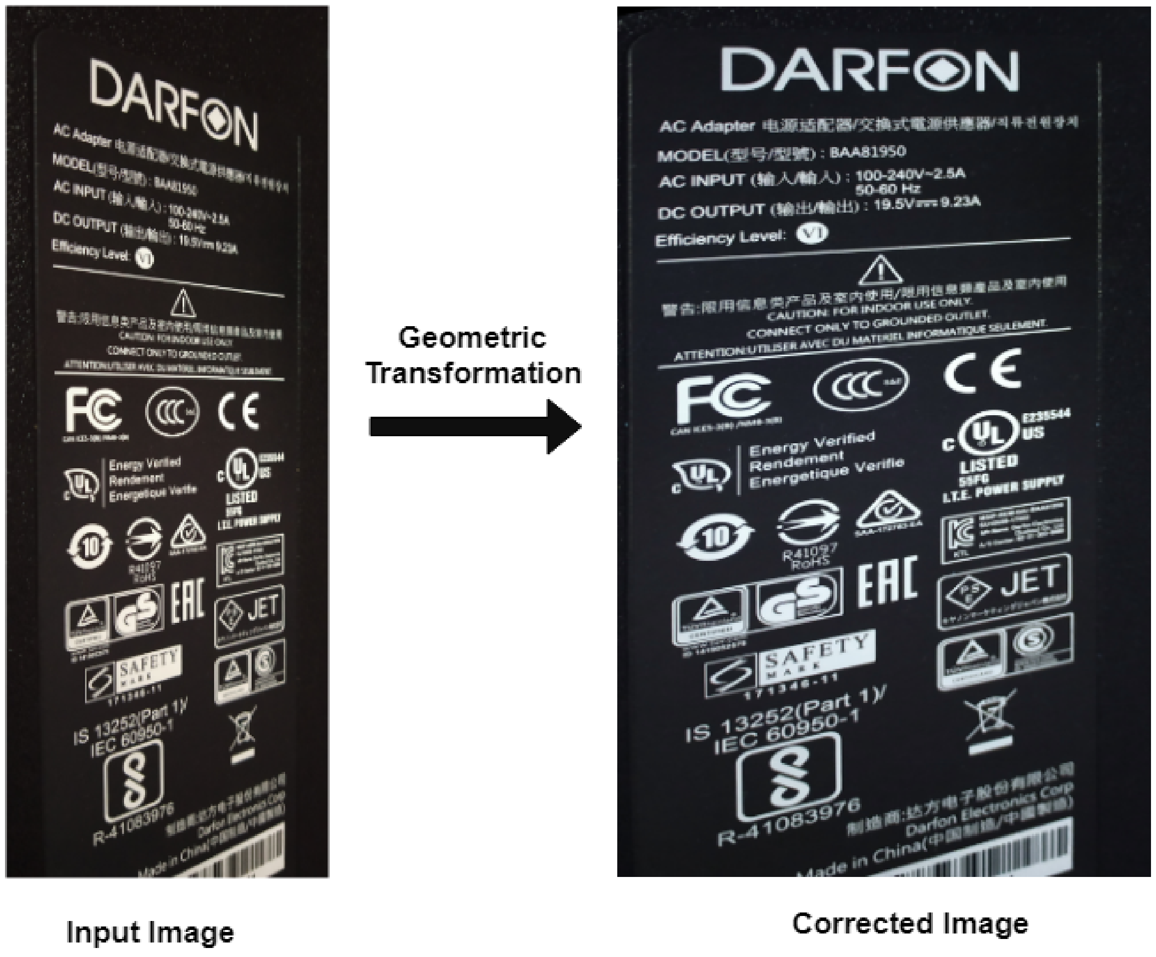
In this blog, we will discuss how to perform a geometric transformation using OpenCV-Python. In geometric transformation, we move the pixels of an image based on some mathematical formulae. This involves translation, rotation, scaling, and distortion (or undistortion!) of images. This is frequently used as a pre-processing step in many applications where the input is distorted while capturing like document scanning, matching temporal images in remote sensing and many more.

There are two basic steps in geometric transformation

- **Spatial Transformation:** Calculating the spatial position of pixels in the transformed image.
- **Intensity Interpolation:** Finding the intensity values at the newly calculated positions.

OpenCV has built-in functions to apply the different geometric transformations to images like translation, rotation, affine transformation, etc. You can find all the functions here: Geometric Transformations of Images

In this blog, we will learn how to change the apparent perspective of an image. This will make the image look more clear and easy to read. Below image summarizes what we want to do. See how easily we can read the words in the corrected image.



For perspective transformation, we need 4 points on the input image and corresponding points on the output image. The points should be selected counterclockwise. From these points, we will calculate the transformation matrix which when applied to the input image yields the corrected image. Let's see the steps using OpenCV-Python

Steps:

- Load the image
- Convert the image to RGB so as to display via matplotlib
- Select 4 points in the input image (counterclockwise, starting from the top left) by using matplotlib interactive window.
- Specify the corresponding output coordinates.
- Compute the perspective transform M using **cv2.getPerspectiveTransform()**
- Apply the perspective transformation to the input image using **cv2.warpPerspective()** to obtain the corrected image.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```

# To open matplotlib in interactive mode
%matplotlib qt

# Load the image
img = cv2.imread('D:/downloads/geometric.jpg')

# Create a copy of the image
img_copy = np.copy(img)

# Convert to RGB so as to display via matplotlib
# Using Matplotlib we can easily find the coordinates
# of the 4 points that is essential for finding the
# transformation matrix
img_copy = cv2.cvtColor(img_copy,cv2.COLOR_BGR2RGB)

plt.imshow(img_copy)

```

By running the above code you will get an interactive matplotlib window popup. Now select any four points(better to select corner points) for the inputs. Then specify the corresponding output points.

```
# Specify input and output coordinates that is used
# to calculate the transformation matrix
input_pts = np.float32([[80,1286],[3890,1253],[3890,122],[450,115]])
output_pts = np.float32([[100,100],[100,3900],[2200,3900],[2200,100]])

# Compute the perspective transform M
M = cv2.getPerspectiveTransform(input_pts,output_pts)

# Apply the perspective transformation to the image
out = cv2.warpPerspective(img,M,(img.shape[1], img.shape[0]),flags=cv2.INTER_L

# Display the transformed image
plt.imshow(out)
```

Hope you enjoy reading.

If you have any doubt/suggestion please feel free to ask and I will do my best to help or improve myself. Good-bye until next time.

This entry was posted in Image Processing and tagged cv2.getPerspectiveTransform(), cv2.warpPerspective(), geometric transformation, image processing, opencv python, perspective transformation on 21 Apr 2019 [https://theailearner.com/2019/04/21/geometric-transformation-of-images-using-opencv-python/] by kang & atul.