How to perform homography using openCV

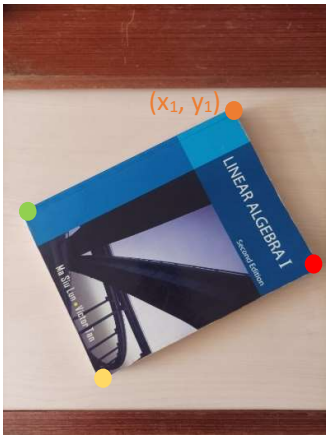
Introduction

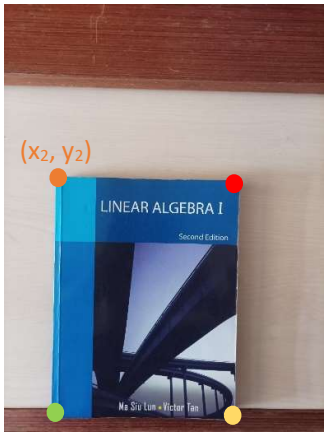
Many of times we find ourselves taking images of our object of interest either from different viewpoints or our object of interest are just simply placed in a position that is not we desired and we are unable to change it physically. Wouldn't we wish to able to transform these objects to our desired position?

The answer is 'yes' we can, by using homography.

What is homography?

Homography is a geometric (3 x 3 matrix) transformation that relates the perspective projection of corresponding points in one image to another image.





Consider a pair of corresponding points in the first image (x_1, y_1) and the second image (x_2, y_2), we can perform the transformation using the following:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}}_{\text{3 x 3 homography matrix}} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

Note: The minimum number of pairs of corresponding points required for homography is 4.

Finding homography matrix in openCV

Homography matrix can be calculated easily using **cv2.findHomography()**

Uses of homography

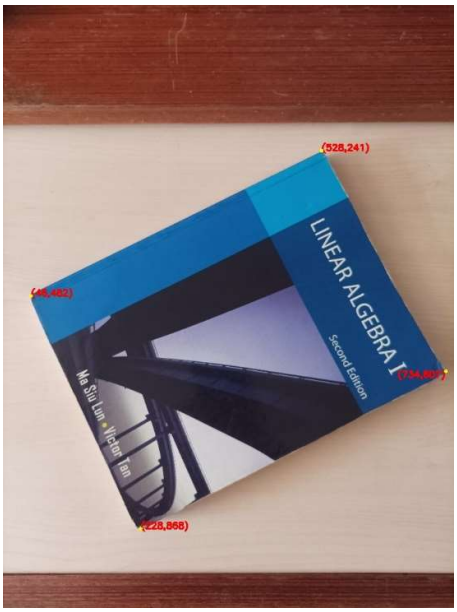
- Image Stitching: Used to align and stitch multiple images together to create a panoramic or wide-angle image. By estimating the homography between overlapping image regions, the images can be seamlessly blended.
- Augmented Reality (AR): Used in AR applications to overlay virtual objects on a real-world scene. By estimating the homography between the camera view and a reference image or model, virtual objects can be rendered accurately in the scene.
- Image Rectification: Used to rectify distorted or skewed images. It can correct perspective distortions caused by camera tilt or rotation, resulting in a planar view of the scene.

Sample implementation of homography using OpenCV

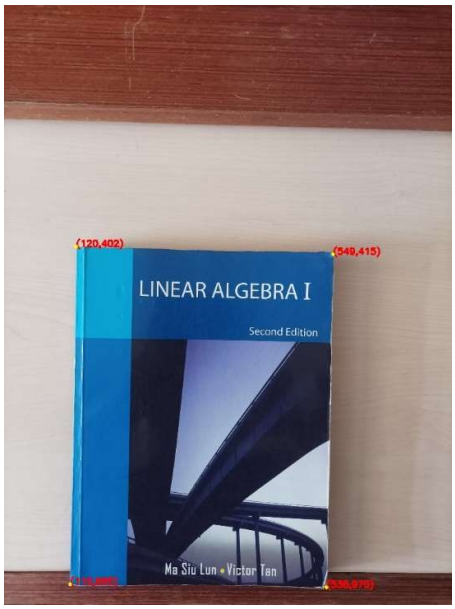
In this section we'll perform homography to warp a source image (book1.jpg) to the destination (book2.jpg)

The source code used in this section can be found in https://github.com/tgt87/Homography

1) Obtain the corresponding coordinates of the 2 images.
   Running "find_coordinates.py" provides a UI for users to click on a point on the image to display its coordinates.



Source image (book1.jpg)



Destination image (book2.jpg)

2) Calculate homography using **cv2.findHomography()** follow by **cv2.warpPerspective()** to wrap the source image to the desired destination.
The code below can be found in “homography.py”.

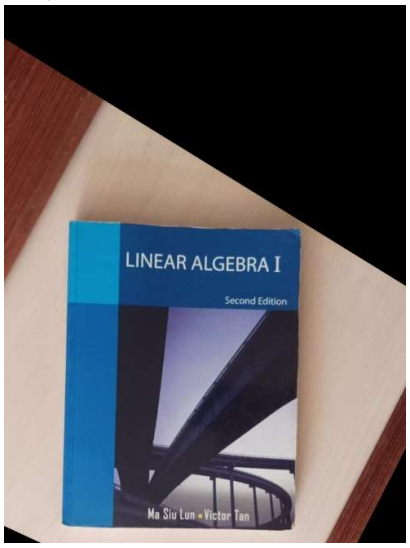
```
# Read source image.
im_src = cv2.imread('images/book1.jpg')
# Four corners of the book in source image
pts_src = np.array([[528, 241], [734, 607], [228, 868], [46, 482]])

# Read destination image.
im_dst = cv2.imread('images/book2.jpg')
# Four corners of the book in destination image.
pts_dst = np.array([[120, 402], [549, 415], [538, 970],[110, 965]])

# Calculate Homography
h, status = cv2.findHomography(pts_src, pts_dst)

# Warp source image to destination based on homography
im_out = cv2.warpPerspective(im_src, h, (im_dst.shape[1],im_dst.shape[0]))
```

Output:



References:

https://learnopencv.com/homography-examples-using-opencv-python-c/#download

https://www.tutorialspoint.com/opencv-python-how-to-display-the-coordinates-of-points-clicked-on-an-image

https://chat.openai.com/

Done By: Tan Gek Teng (A0030151E)