

Introduction to Computer Vision

Assignment #2

Due: Nov.-17 (Thur.) (before 11:59pm)

Instruction

- Submit your source codes in a single compressed file “CV_A2_StudentID.zip” to iCampus.
- Python 3.7 or higher / OpenCV 3.4 or higher will be used to execute your submitted codes.
- In this assignment, you will take grayscale images as input. In order to open an image as grayscale, you can use the following statement:

```
img = cv2.imread( IMAGE_FILE_PATH , cv2.IMREAD_GRAYSCALE )
```

- You can submit at most 4 python files. In other words, you can add 2 additional python files to implement functions commonly utilized across different parts.
- Any work that you turn in should be your own.

Part #1. 2D Transformations [40 pts]

The requirements of Part #1 will be evaluated by running ‘A2_2d_transformations.py’ file.

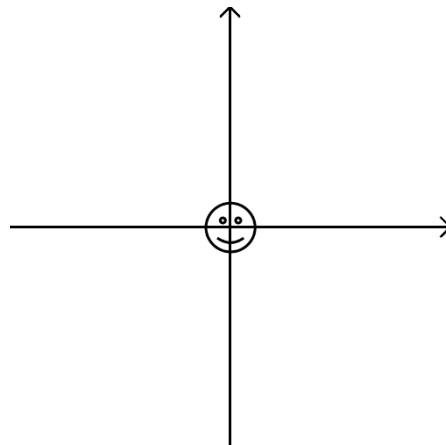
1-1. Visualization of a transformed image on a 2D plane.

- In Part #1, we will use ‘smile.png’ as the input image. The image can be changed for evaluation, however, you can assume that the input image is odd sized along both dimensions.
- Implement a function that returns a plane where the transformed image is displayed. The function gets two parameters, an image `img` and a 3×3 affine transformation matrix `M`. The vertical and horizontal sizes of the plane is fixed to 801×801 and the origin $(0,0)$ is corresponding to the pixel at $(400,400)$. You also need to draw two arrows to visualize x and y axes.

```
function plane = get_transformed_image ( img , M )
```

- We initially place the image centered at the origin. Thus, the function should return the following plane if the matrix `M` describes the identity mapping:

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



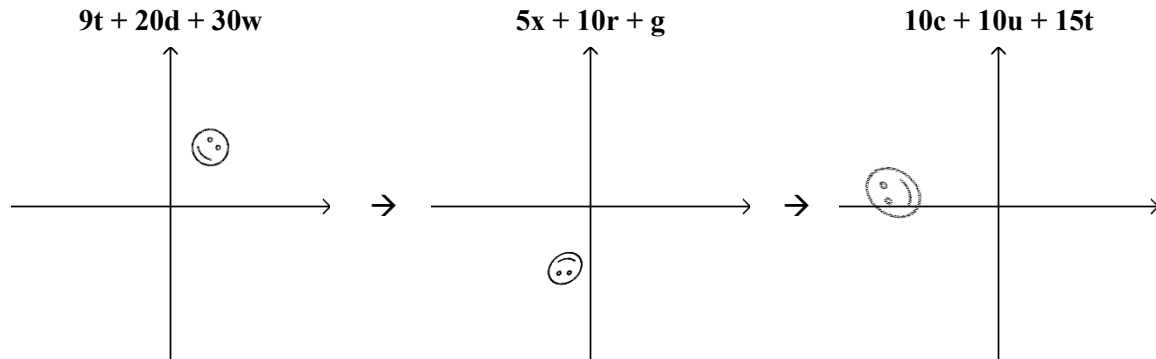
- d) To add arrows to the plane, you can use the built-in function `cv2.arrowedLine(...)`.
- e) You cannot use any built-in function that directly performs image transformations.

1-2. Interactive 2D transformations

- a) Implement your script to support interactions by a keyboard for various transformations of the image. The required actions are described as follows:

Key	Action
'a'	Move to the left by 5 pixels
'd'	Move to the right by 5 pixels
'w'	Move to the upward by 5 pixels
's'	Move to the downward by 5 pixels
'r'	Rotate counter-clockwise by 5 degrees
't'	Rotate clockwise by 5 degrees
'f'	Flip across y axis
'g'	Flip across x axis
'x'	Shrink the size by 5% along to x direction
'c'	Enlarge the size by 5% along to x direction
'y'	Shrink the size by 5% along to y direction
'u'	Enlarge the size by 5% along to y direction
'h'	Restore to the initial state
'q'	Quit

- b) Refer the following examples:



- c) You have to use the function implemented in 1-1.
- d) **Extra credit:** We have some artifacts when we enlarge or rotate the image as shown in the above examples. Reducing the artifacts gets extra credits.

Part #2. Homography [60 pts]

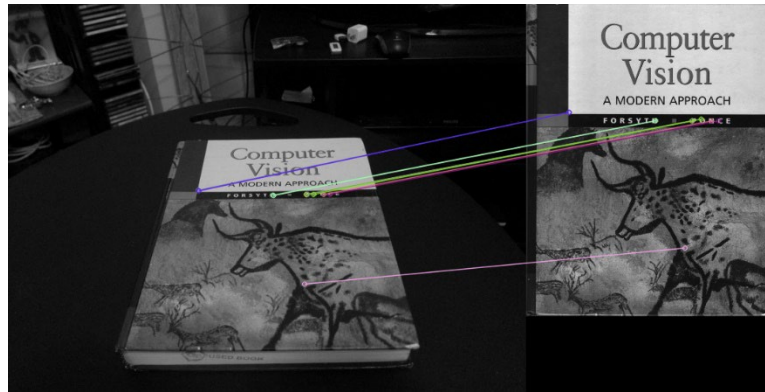
The requirements of Part #2 will be evaluated by running '`A2_homography.py`' file.

2-1. Feature detection, description, and matching

- a) Read a pair of images.
- b) Use the built-in functions to extract ORB (Oriented FAST and Rotated BRIEF) features from two images by referring the following code example:

```
orb = cv2.ORB_create()
kp = orb.detect( img , None )
kp, des = orb.compute( img , kp )
```

- c) Perform feature matching between two images ('cv_desk.png' and 'cv_cover.jpg'), and display top-10 matched pairs according to feature similarities. Note that, the similarities among ORB features should be computed by the Hamming distance.



- d) You cannot use any built-in function that directly performs feature matching such as BFMacher. However, you can use `cv2.drawMatches(...)` to visualize the top-10 matches.

2-2. Computing homography with normalization

- a) Implement a function that returns a homography from a source image to a destination image. The function gets two $N \times 2$ matrices, `srcP` and `destP`, where N is the number of matched feature points and each row is a location in the image, and returns a 3×3 transformation matrix. Note that, the number of matches N should be equal or greater than 15, $N \geq 15$.

```
function H = compute_homography ( srcP , destP )
```

- b) We first normalize feature points $x_S \in \text{srcP}$ based on the following steps:
- 1) Mean subtraction: translate the mean of the points to the origin $(0, 0)$
 - 2) Scaling: scale the points so that the longest distance to the origin is $\sqrt{2}$
- c) Perform 2-2-b) to $x_D \in \text{destP}$. Since aforementioned normalization is a linear transformation, we can denote the transformations as two 3×3 matrices T_S and T_D for the `srcP` and `destP`, respectively. Then, we can write the normalized points as follows:

$$\tilde{x}_S = T_S x_S \text{ and } \tilde{x}_D = T_D x_D \text{ ---- Eq (1)}$$

- d) Once the points are normalized, compute the homography H_N from \tilde{x}_S to \tilde{x}_D by referring the lecture slide (page #27 of *CV_06_Image_Homographies.pdf*). Note that, you are allowed to use the built-in function, `numpy.linalg.svd(...)` to compute SVD.
- e) Since the computed H_N satisfied $\tilde{x}_D = H_N \tilde{x}_S$, we have $T_D x_D = H_N T_S x_S$ according to Eq (1). The transformation from `srcP` to `destP` is finally expressed as follows:

$$x_D = T_D^{-1} H_N T_S x_S$$

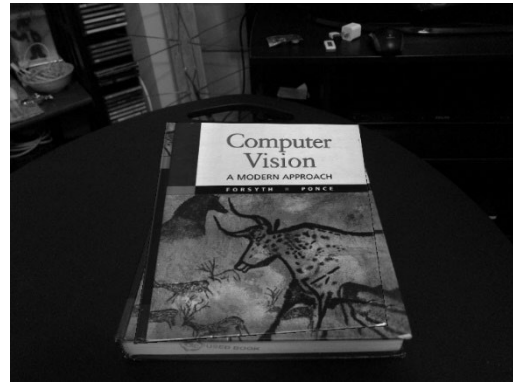
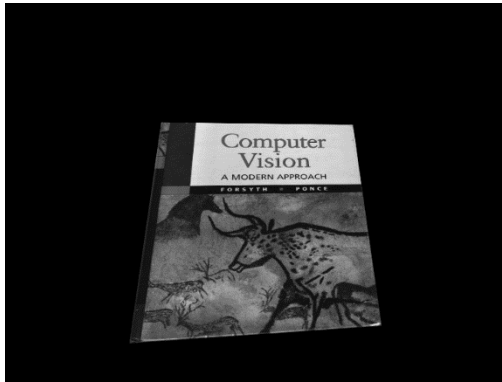
2-3. Computing homography with RANSAC

```
function H = compute_homography_ransac ( srcP , destP , th )
```

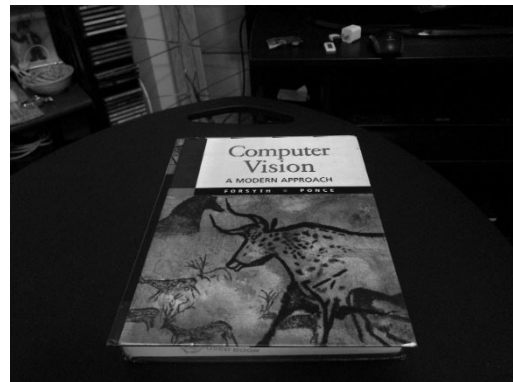
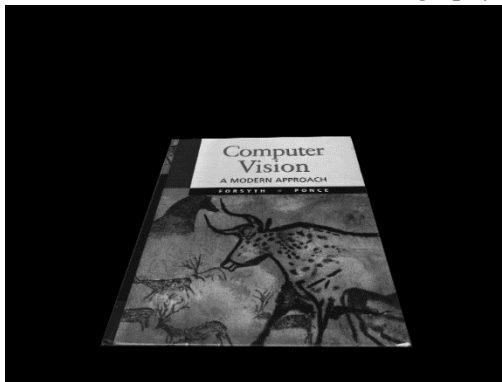
- a) Implement a function that returns a homography with RANSAC by referring the lecture slide (page #39 of *CV_06_Image_Homographies.pdf*). The parameter `th` is used to determine whether a point is inlier or outlier. You have to use the function implemented in 2-2. Note that, your function should produce the homography within 3 seconds.

2-4. Image warping

- a) Read `'cv_desk.png'` and `'cv_cover.jpg'`, and compute homography from the cover image to the desk image.
- b) Wraps `'cv_cover.jpg'` to the dimensions of `'cv_desk.png'`. Display the warped image of `'cv_cover.jpg'` and the composed image. You can use `cv2.warpPerspective(...)` for the wrapping. Compare the results of the homography with normalization and RANSAC.

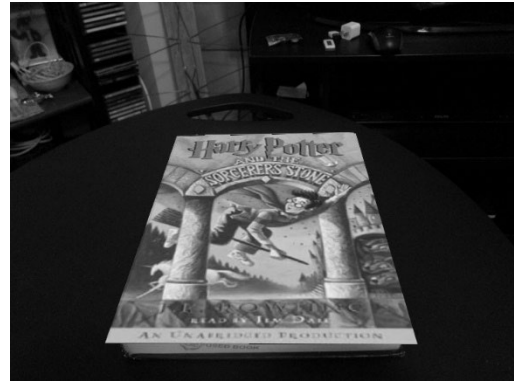
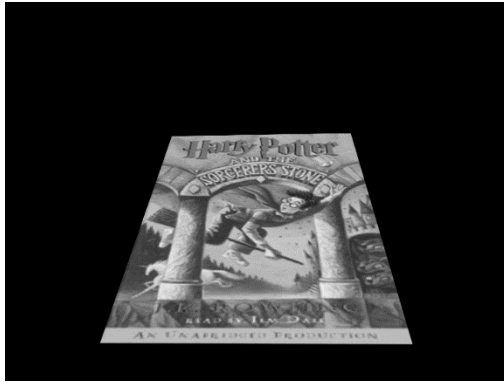


Homography with normalization



Homography with RANSAC

- c) Wrap and compose `'hp_cover.jpg'` to `'cv_desk.png'` based on the homography with RANSAC computed in b), and display them.



2-5. Image stitching

- a) Read '*diamondhead-10.png*' and '*diamondhead-11.png*', and stitch them based on the homography computed with RANSAC. Display the result.



- b) In order to reduce the artifacts on the boundary of two images, perform a simple gradation based blending:

