CSC420H1 L0101, Fall 2018
Assignment 3

Name: JingXi, Hao
Student Number: 1000654188

Due Date: 25 October, 2018 11:59pm

1. Take a A4 paper which is 210x297 (unit: millimeter). Attach the paper on a door. Take a picture of the door such that all four corners of the door are visible on the photo. Take this picture in an oblique view, ie, the door is not a perfect rectangle but rather a quadrilateral in the photo. Using homography theory, estimate the width and height of the door from the picture. Show your derivation, captured image and final result.

   Solution:

   Please see the detailed code implementation for this question in the file, **a3.py**.

   Note that a letter size paper with the dimension of 215.9 by 279.4 in millimeters is used in this question instead of a A4 paper. The picture of the door with a paper attached taken in an oblique view is shown below.

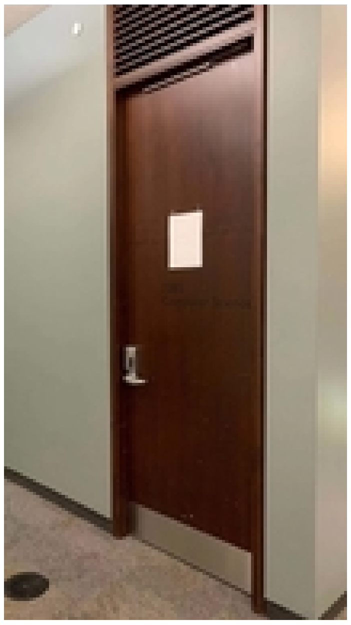


Figure 1: Picture of Door with Paper Attached

Then, in order to compute homogrphy matrix, we first need to find the coordinate for every corner of the letter size paper in $Figure$ 1. These coordinates found are the points on the reference image. Since we know the actual dimension for the letter size paper, then the matching coordinates on the test image are also given in this case. Therefore, based on the formula shown in $Figure$ 2,

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Figure 2: Formula to Compute Homography

we are able to compute the homography matrix in the shape of 9 by 1. Similarly, we find the coordinate for every corner of the door in $Figure$ 1. Then, we follow the formula shown in $Figure$ 3 and do a dot product of each coordinate of the door and the homography matrix which is reshaped in the shape of 3 by 3.

$$\begin{bmatrix} ax'_i \\ ay'_i \\ a \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Figure 3: Formula to Computed Transformed Coordinate

Again, based on $Figure$ 3, we drop the constant coefficient a by divided a from every entry in resulted coordinate vector and take the first two values to form a matching point. Repeat this procedure for each corner coordinate of the door. Consequently, we are able to obtain four matching points and compute the width and the height of the door. Based on my computation, the width of the door is $991.1340759756661mm \approx 1m$ and the height of the door is $2491.5244007752676mm \approx 2.5m$.

2. Take 3 images of yourself holding a hardcover book (which we call them im1.jpg, im2.jpg,im3.jpg). If you use a mirror, remember to flip the images horizontally. Try to make one of the images easy (little out of plane rotation; but do include in-plane rotations); one of the images somewhat more difficult (e.g. a bit further away from the camera and also include 30-40 degrees out of plane rotation; change the lighting a bit); and one of the images difficult (further away, or 40-60 degrees out of plan rotation, or drastically change the lighting). Also, find a picture of the book cover on Amazon or another website (which we call it bookCover.jpg). Finally, download a cover of another book from Amazon or another website (which we call it anotherBookCover.jpg). Reduce the image sizes so none is larger than 640x480 and save them in a compressed (jpg) format so your assignment file is not too large (MarkUs does not allow very large submissions).

   NOTE: Here we first show all images required to solve for this question.

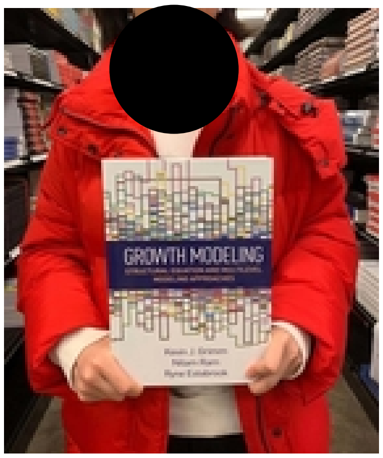


Figure 4: im1

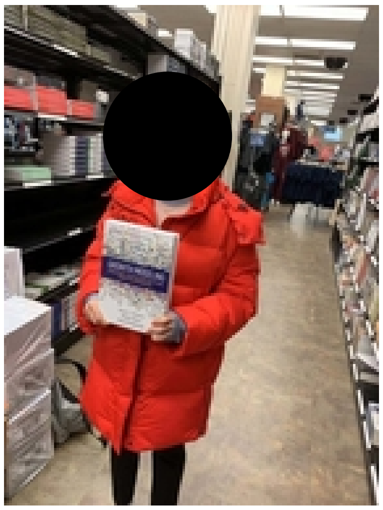


Figure 5: im2

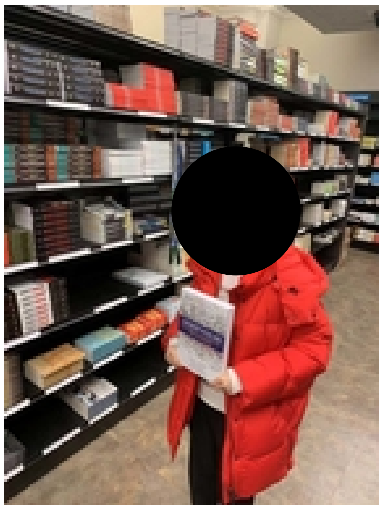


Figure 6: im3

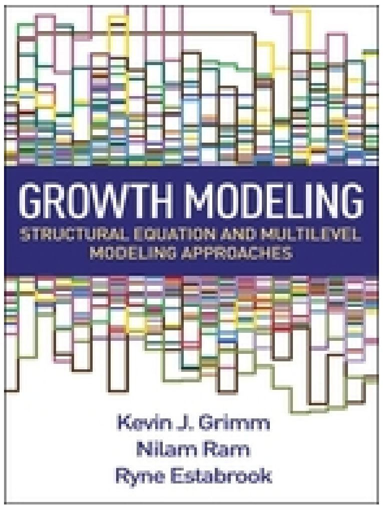


Figure 7: bookCover

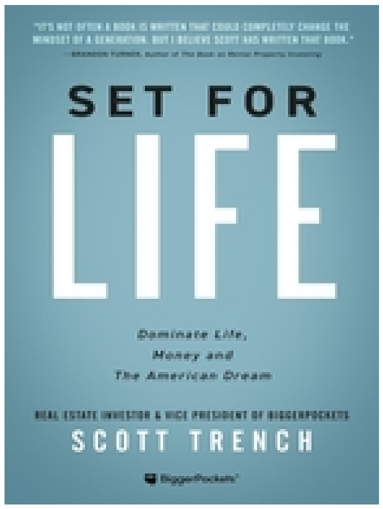


Figure 8: anotherBookCover

(a) Use SIFT (or any other descriptor you like, e.g. SURF) to find point matches between each image (im1, im2, im3) and the book cover (bookCover). Visualize the matches between bookCover and each of the 3 images in a manner similar to slide #10 on lecture8-B. You can use any implementation of SIFT (or SURF or . . .) in Python (OpenCV) or Matlab.

Solution:

Please see the detailed code implementation for this question in the file, **a3.py**.

For this question, $SIFT$ descriptor is used to find matches between two images. Then, we make a ratio test in order to get good matches from all matches. We show the output images below where each indicates the matches between bookCover and one of the 3 images (im1, im2, and im3).

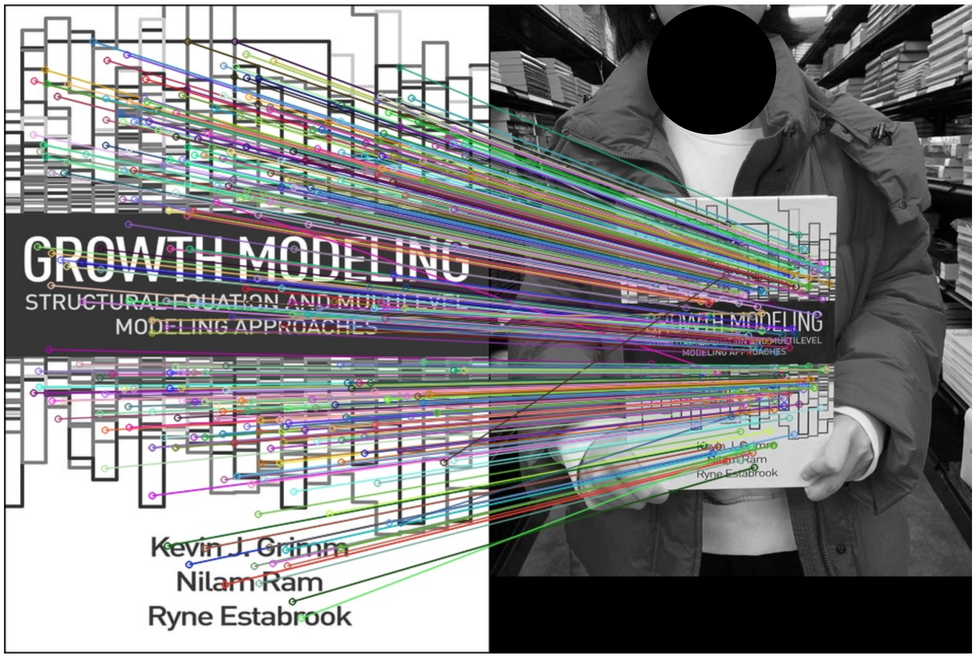

[redacted]

Figure 9: Matches Between bookCover and im1

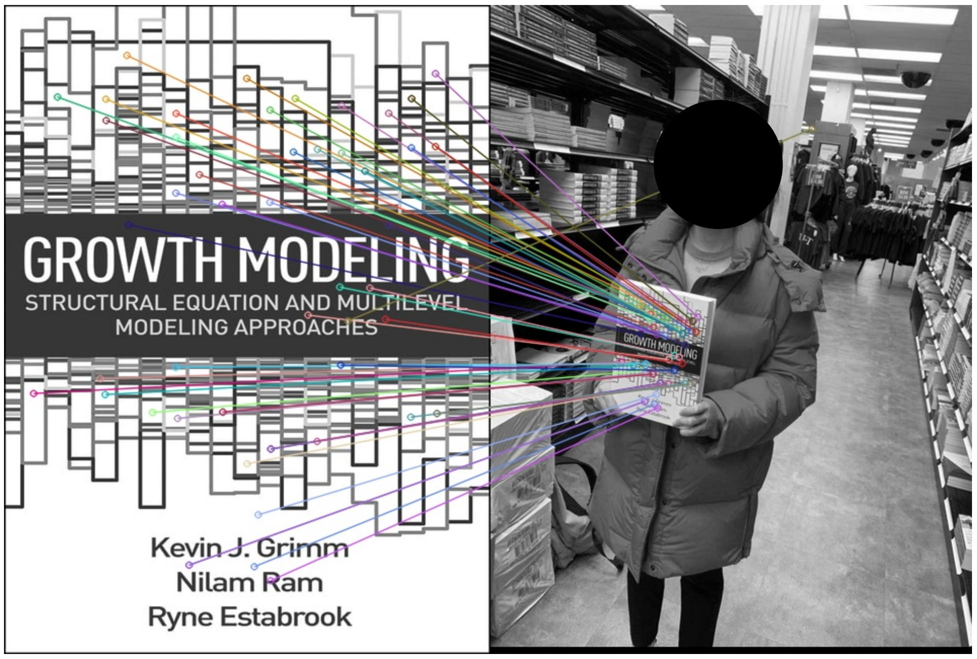

[redacted]

Figure 10: Matches Between bookCover and im2

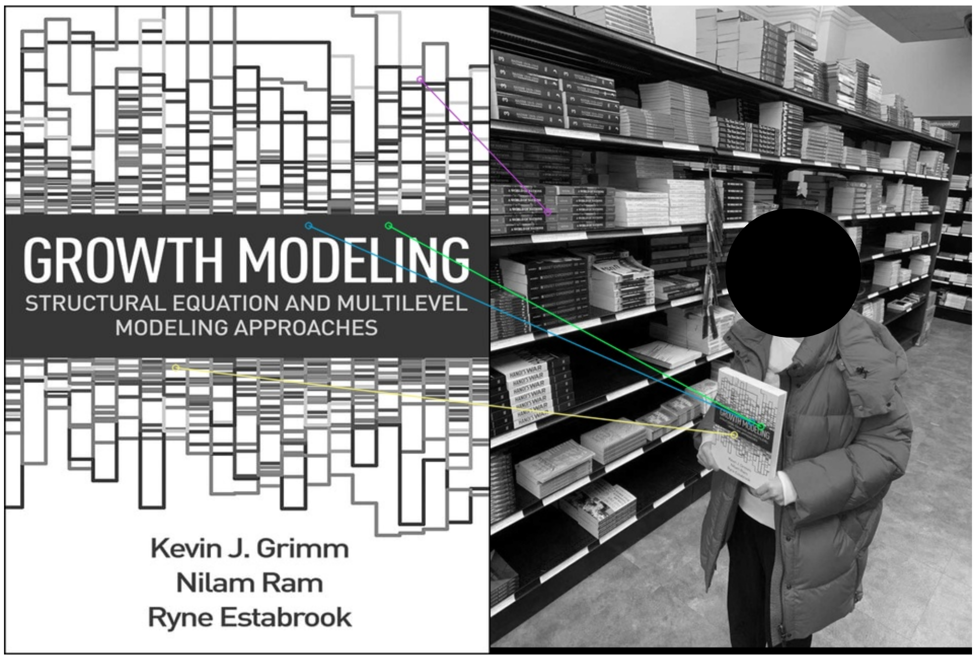


Figure 11: Matches Between bookCover and im3

(b) Visually estimate the percentage of outliers in each case and compute the number of RANSAC iterations to recover the an affine transformation between bookCover and each of the images with ¿ $\geq$ 99% chance. Similarly, estimate the number of iterations required to recover the projective transformation (homography).

Solution:

- Based on $Figure$ 9 shown in $2(a)$, we are able to see that there are three outliers since there are three matches matched wrongly. In this case, we have 366 good matches in total obtained by applying a ratio test. Thus, the percentage of outliers, denoted by q, is $q = \frac{3}{366} = \frac{1}{122} \approx 0.8197\%$, which implies that the percentage of inliers is $p = 1 - q = \frac{121}{122} \approx 99.18\%$. Also, we have that $P = 0.99$ because we want to achieve 99% chance. Since we employ RANSAC algorithm here, so $k = 3$ when recover affine transformation and $k = 4$ when recover projective tranformation (homography).
  - Let $k = 3$. Based on the information we have, we can compute the number of RANSAC iterations by using $S = \frac{log(1-P)}{log(1-p^k)}$. Then, we have that
$$\begin{aligned} S &= \tfrac{log(1-P)}{log(1-p^k)} \\ &= \tfrac{log(1-0.99)}{log(1-(\frac{121}{122})^3)} \\ &\approx 1.2401 \\ &\approx 2 \end{aligned}$$
    Therefore, the number of iterations required to recover the affine transformation is 2.
  - Let $k = 4$. Based on the information we have, we can compute the number of RANSAC iterations by using $S = \frac{log(1-P)}{log(1-p^k)}$. Then, we have that
$$S = \tfrac{log(1-P)}{log(1-p^k)}$$

$$\begin{aligned} &= \frac{log(1-0.99)}{log(1-(\frac{121}{122})^4)} \\ &\approx 1.3426 \\ &\approx 2 \end{aligned}$$

Therefore, the number of iterations required to recover the projective transformation is 2.

- Based on $Figure$ 10 shown in $2(a)$, we are able to see that there is only one outlier. In this case, we have 54 matches in total obtained by applying a ratio test. Thus, the percentage of outliers, denoted by q, is $q = \frac{1}{54} =\approx 1.852\%$, which implies that the percentage of inliers is $p = 1-q = \frac{53}{54} \approx 98.15\%$. Also, we have that $P = 0.99$ because we want to achieve 99% chance. Since we employ RANSAC algorithm here, so $k = 3$ when recover affine transformation and $k = 4$ when recover projective tranformation.
  - Let $k = 3$. Based on the information we have, we can compute the number of RANSAC iterations by using $S = \frac{log(1-P)}{log(1-p^k)}$. Then, we have that
  $$\begin{aligned} S &= \frac{log(1-P)}{log(1-p^k)} \\ &= \frac{log(1-0.99)}{log(1-(\frac{53}{54})^3)} \\ &\approx 1.5831 \\ &\approx 2 \end{aligned}$$
  Therefore, the number of iterations required to recover the affine transformation is 2.
  - Let $k = 4$. Based on the information we have, we can compute the number of RANSAC iterations by using $S = \frac{log(1-P)}{log(1-p^k)}$. Then, we have that
  $$\begin{aligned} S &= \frac{log(1-P)}{log(1-p^k)} \\ &= \frac{log(1-0.99)}{log(1-(\frac{53}{54})^4)} \\ &\approx 1.7507 \\ &\approx 2 \end{aligned}$$
  Therefore, the number of iterations required to recover the projective transformation is 2.
- Based on $Figure$ 11 shown in $2(a)$, we are able to see that there is only one outlier out of 4 matches obtained by applying a ratio test. Thus, the percentage of outliers, denoted by q, is $q = \frac{1}{4} =\approx 25\%$, which implies that the percentage of inliers is $p = 1 - q = \frac{3}{4} \approx 75\%$. Also, we have that $P = 0.99$ because we want to achieve 99% chance. Since we employ RANSAC algorithm here, so $k = 3$ when recover affine transformation and $k = 4$ when recover projective tranformation.
  - Let $k = 3$. Based on the information we have, we can compute the number of RANSAC iterations by using $S = \frac{log(1-P)}{log(1-p^k)}$. Then, we have that
  $$\begin{aligned} S &= \frac{log(1-P)}{log(1-p^k)} \\ &= \frac{log(1-0.99)}{log(1-(0.75)^3)} \\ &\approx 8.4041 \\ &\approx 9 \end{aligned}$$
  Therefore, the number of iterations required to recover the affine transformation is 9.
  - Let $k = 4$. Based on the information we have, we can compute the number of RANSAC iterations by using $S = \frac{log(1-P)}{log(1-p^k)}$. Then, we have that
  $$\begin{aligned} S &= \frac{log(1-P)}{log(1-p^k)} \\ &= \frac{log(1-0.99)}{log(1-(0.75)^4)} \\ &\approx 12.1064 \\ &\approx 13 \end{aligned}$$

Therefore, the number of iterations required to recover the projective transformation is 13.

(c) Using RANSAC (any open source implementation, or your own), find the affine transformation between bookCover and im1, im2, and im3. Use this transformation to paste bookCover onto each of the images. Explain when the method is successful and when it might fail.

Solution:

Please see the detailed code implementation for this question in the file, **a3.py**.

In this question, open source implementation (*ransac*) has been employed for finding affine transformation by using RANSAC. We show the output images below obtained by pasting bookCover onto each of the images using affine transformation found.

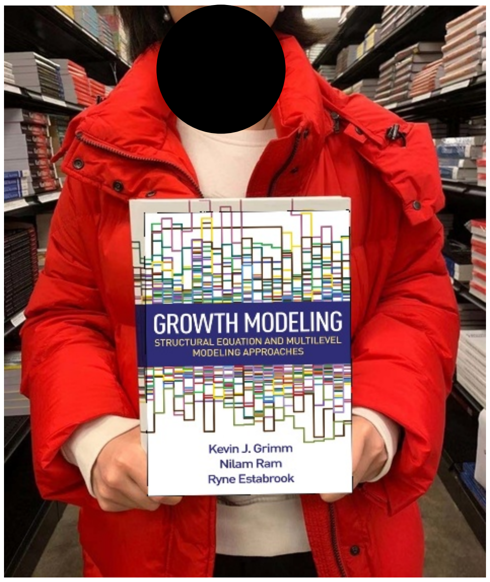
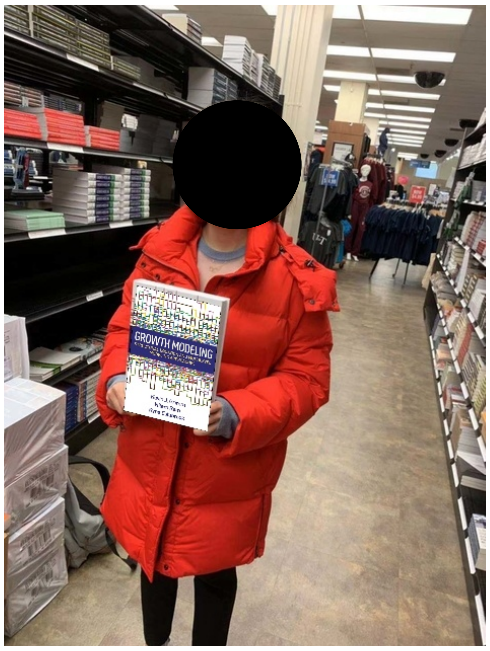


Figure 12: Paste bookCover Onto im1 Using Affine Transformation



[redacted]

Figure 13: Paste bookCover Onto im2 Using Affine Transformation

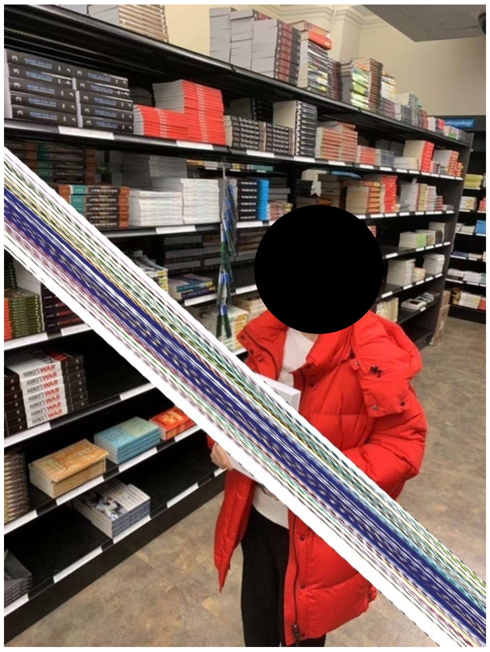

[redacted]

Figure 14: Paste bookCover Onto im3 Using Affine Transformation

Based on the output images we got, we are able to see that the bookCover pasted on im1 and im2 seems more matched than the one pasted on im3. This makes sense since im3 is taken further away and have larger degrees of out of plane rotation comparing with the im1 and im2, which causes the less number of good matches (matching features) found when a ratio test is applied and the higher percentage of outliers. Thus, bookCover and im3 are not matched properly. Hence, we can conclude that we need to achieve the larger number of good matches and the lower percentage of outliers in order to make the method successful. Also, the number of good matches and the percentage of outliers are affected by the distance of taking a photo and the degrees of out of plane rotation. The further the distance and the higher degrees of out of plane rotation would cause the less good matches found and the higher percentage of outliers, which counts as a failure of the method.

(d) Use a homography (protective transformation) to do the same. Explain when the method is successful and when it might fail. Compare and discuss the differences.

Solution:

Please see the detailed code implementation for this question in the file, **a3.py**.

In this question, open source implementation has been employed for finding homography by using RANSAC. We show the output images below obtained by pasting bookCover onto each of the images using homography found.

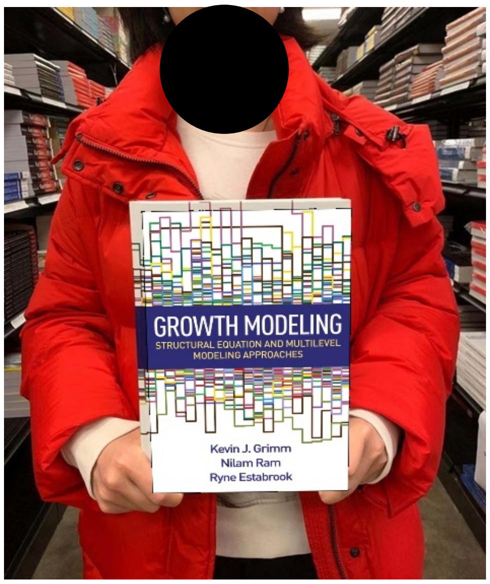

[redacted]

Figure 15: Paste bookCover Onto im1 Using Homography

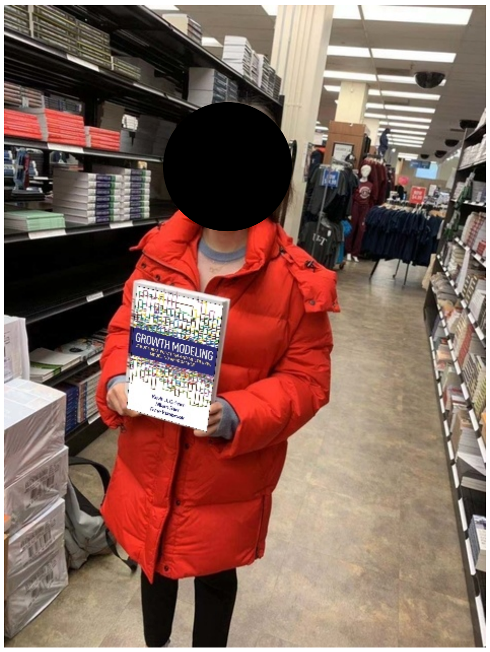

[redacted]

Figure 16: Paste bookCover Onto im2 Using Homography

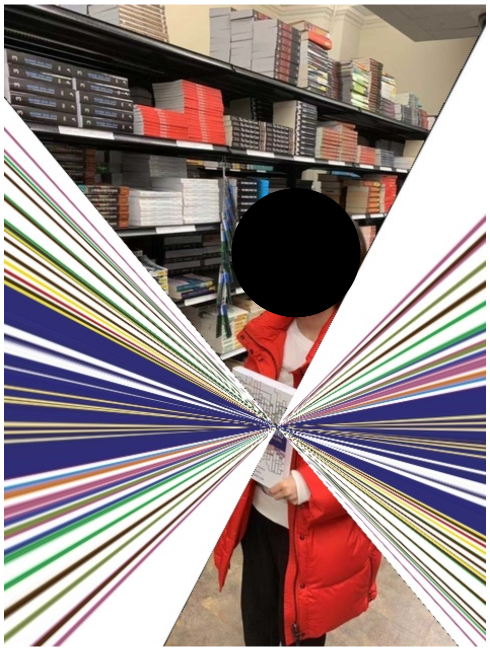


Figure 17: Paste bookCover Onto im3 Using Homography

As you can see, based on the output images we obtained, we face a similar result as we use affine transformation to paste the bookCover onto each of the images (im1, im2, and im3). The bookCover pasted on im1 and im2 are more mathced in comparison with the bookCover pasted on im3 and the bookCover pasted on im3 does not look like a bookCover at all. This situation still makes sense in this case since the further distance and the larger degrees of out of plane rotation causes the less number of good matches found and the higher percentage of outliers, which implies that the bookCover would not be mathced properly with im3.

(e) Use a homography to map the cover of the second book (anotherBookCover) onto each image. Discuss your results.

Solution:

Please see the code implementation for this question in the file, **a3.py**.

In this question, open source implementation has been employed for finding homography by using RANSAC. We show the output images below obtained by pasting anotherBookCover onto each of the images using homography found.

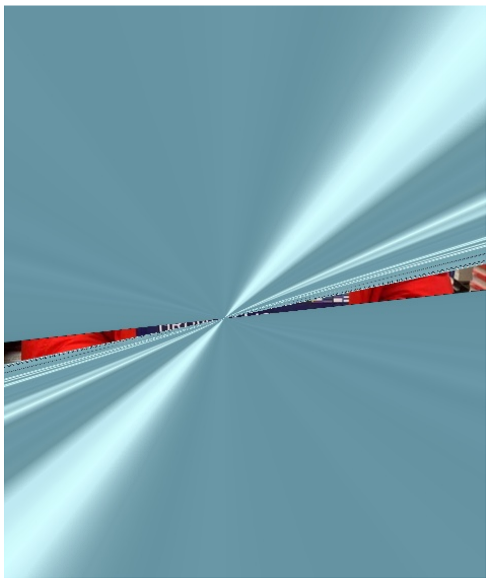


Figure 18: Paste anotherBookCover Onto im1 Using Homography

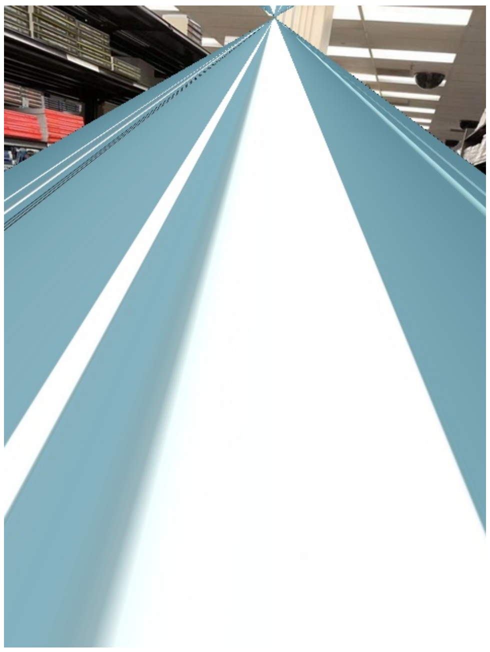


Figure 19: Paste anotherBookCover Onto im2 Using Homography

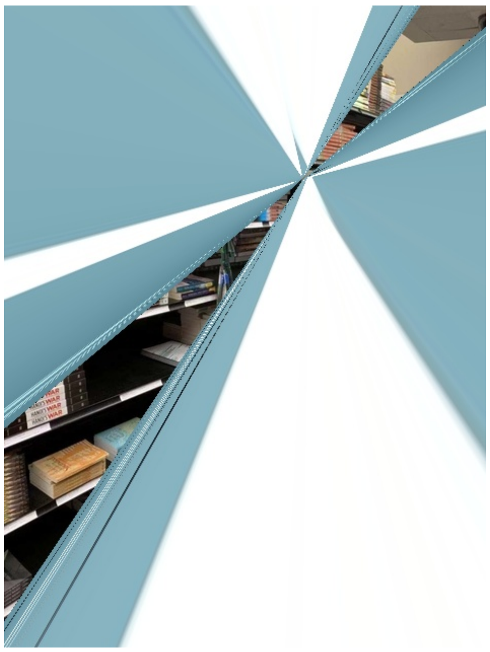


Figure 20: Paste anotherBookCover Onto im3 Using Homography

Based on output images we obtained, we are able to see that no good results obtained when we paste anotherBookCover onto each of the images (im1, im2, and im3). This makes sense since we barely can find good matches by doing ratio test between anotherBookCover and one of the images (im1, im2, and im3). This is reasonable since anotherBookCover is not shown on each of the images (im1, im2, and im3). Since the total number of good matches found are less and those found are also outliers, so, we would not be able to find a reasonable transformed anotherBookCover by using Homography and pasted it onto each of the images (im1, im2, and im3).

3. Using your phone or any camera you have(select a specific resolution), estimate the internal parameter matrix **K** for your camera. Show your plan, formula derivation, captured picture and result. To be simple, assume there is no distortion and focal length is the same for both x axis and y axis. If you use libraries, you can get at most 1.5 points.

Solution:

Please see the detailed code implementation for this question in the file, **a3.py**.

First, show the image of the bottle taken below. This is the image used to make the estimation of the internal parameter matrix **K**.

Figure 21: Image of the Bottle

For this question, we first assume that the principal point is the center of the image. Then, we are able to easily compute the coordinates of the principal point where p_x is the half of the width of the image and p_y is the half of the height of the image, where we have by computation is $p_x = \frac{3024}{2} = 1512(pixel)$ and $p_y = \frac{4032}{2} = 2016(pixel)$. Then, we show a graph based on which we make our calculation to find the focal length first and consequently estimate the internal parameter matrix $\mathbf{K}$ for my camera.

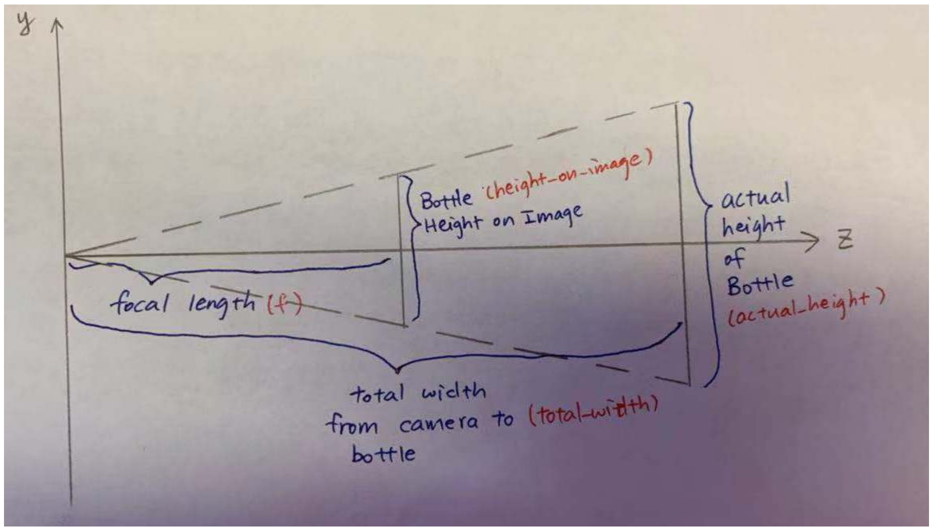
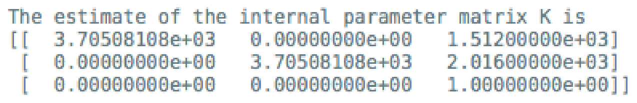

y
Z
Bottle (height-on-image)
Height on Image
actual height of Bottle (actual-height)
focal length (f)
total width from camera to bottle (total-width)

Figure 22: Visualization

Since we assume that the principal point is at the center of the image, then the *z-axis* pass through the center of the bottle and the center of the virtual image. Therefore, we are able to compute the focal length, f, as follows, $f = (\frac{height\_on\_image/2}{actual\_height/2}) \times (total\_width)$. Based on my measurements, the $height\_on\_image$ is $4032(pixel)$, the $actual\_height$ is $18.5(cm) * 10 = 185(mm)$, and the $total\_width$ is $17(cm) * 10 = 170(mm)$. By substitution, we have that $f = (\frac{4032/2}{185/2})(170) = 3705.081081(pixel) \approx 3705(pixel)$. Therefore, we have the estimation for the internal parameter matrix $\mathbf{K}$,

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3705 & 0 & 1512 \\ 0 & 3705 & 2016 \\ 0 & 0 & 1 \end{bmatrix}$$

Also, show the result obtained by my code which is just the implementation of what I mentioned above.



```
The estimate of the internal parameter matrix K is
[[  3.70508108e+03   0.00000000e+00   1.51200000e+03]
 [  0.00000000e+00   3.70508108e+03   2.01600000e+03]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00]]
```

Figure 23: Estimation of the Internal Parameter Matrix, $\mathbf{K}$