CSC420

Introduction to Image Understanding

Assignment 3

Zhili Xu

Part A

[3.5 points] 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.

```
im = cv2.imread('image1.jpg')
rows, cols, ch = im.shape
w = 210
h = 297

c = np.array([[0+500, 0+500], [w-1+500, 0+500], [w-1+500,h-1+500], [0+500,h-
1+500]])

plt.imshow(im, origin='lower')

plt.xlim(500, 700)
plt.ylim(600, 200)
x = plt.ginput(4)
print("clicked", x)
plt.show()


x = np.array(x)
print(x)


h, status = cv2.findHomography(x, c)
print h


dst = cv2.warpPerspective(im, h, (rows,cols*4))
plt.imshow(dst)
x = plt.ginput(4)
print("clicked", x)
plt.show()
```

After transformation, the width of A4 is 210 pixel and width of the door is 735 pixel, the length of A4 is 297 pixel and the length of door is 2206. Therefore, the width of the door is 73.5 cm and the length of the door is 220.6 cm.

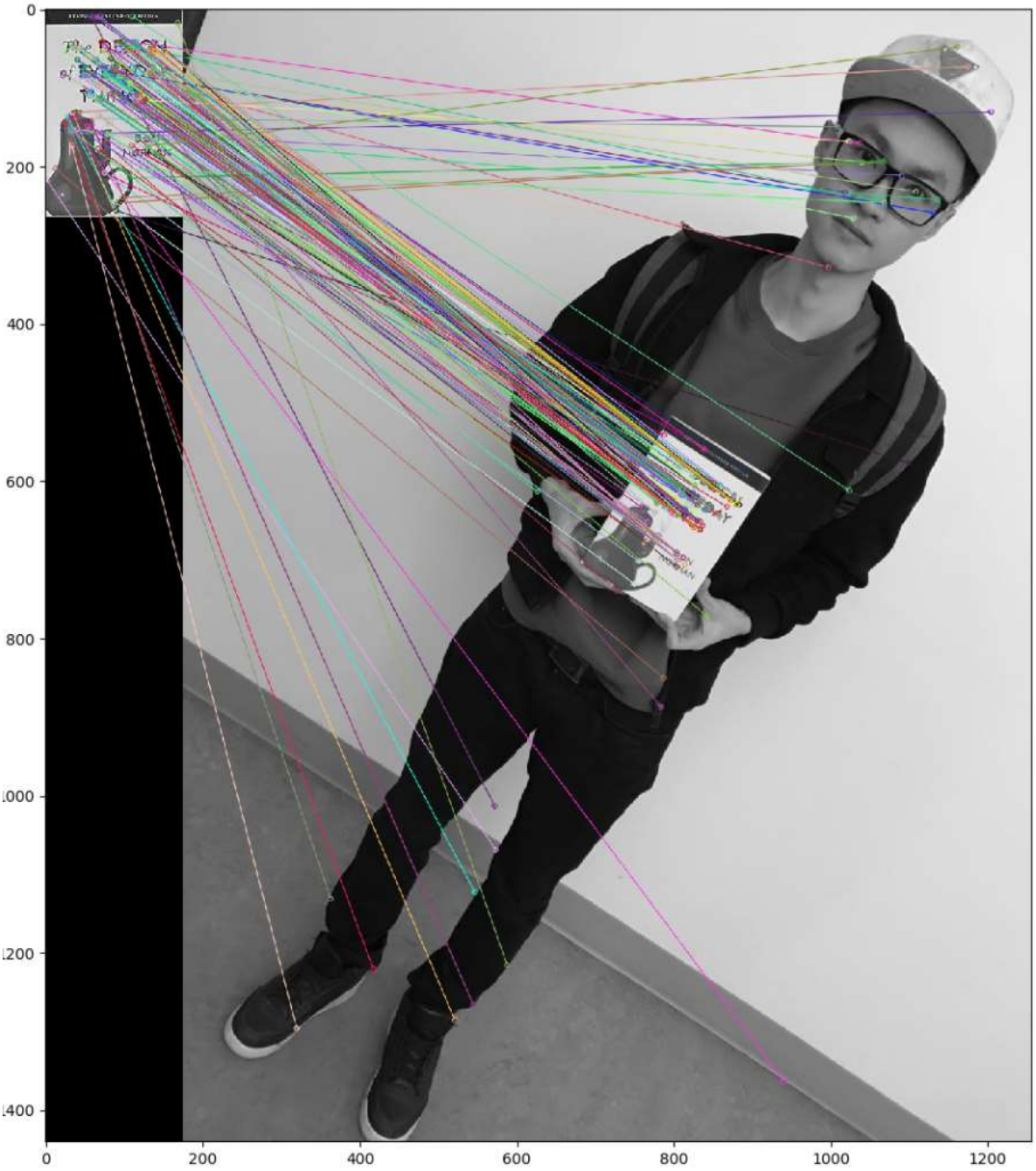

0
500
1000
1500
2000
2500
3000
3500

Part B

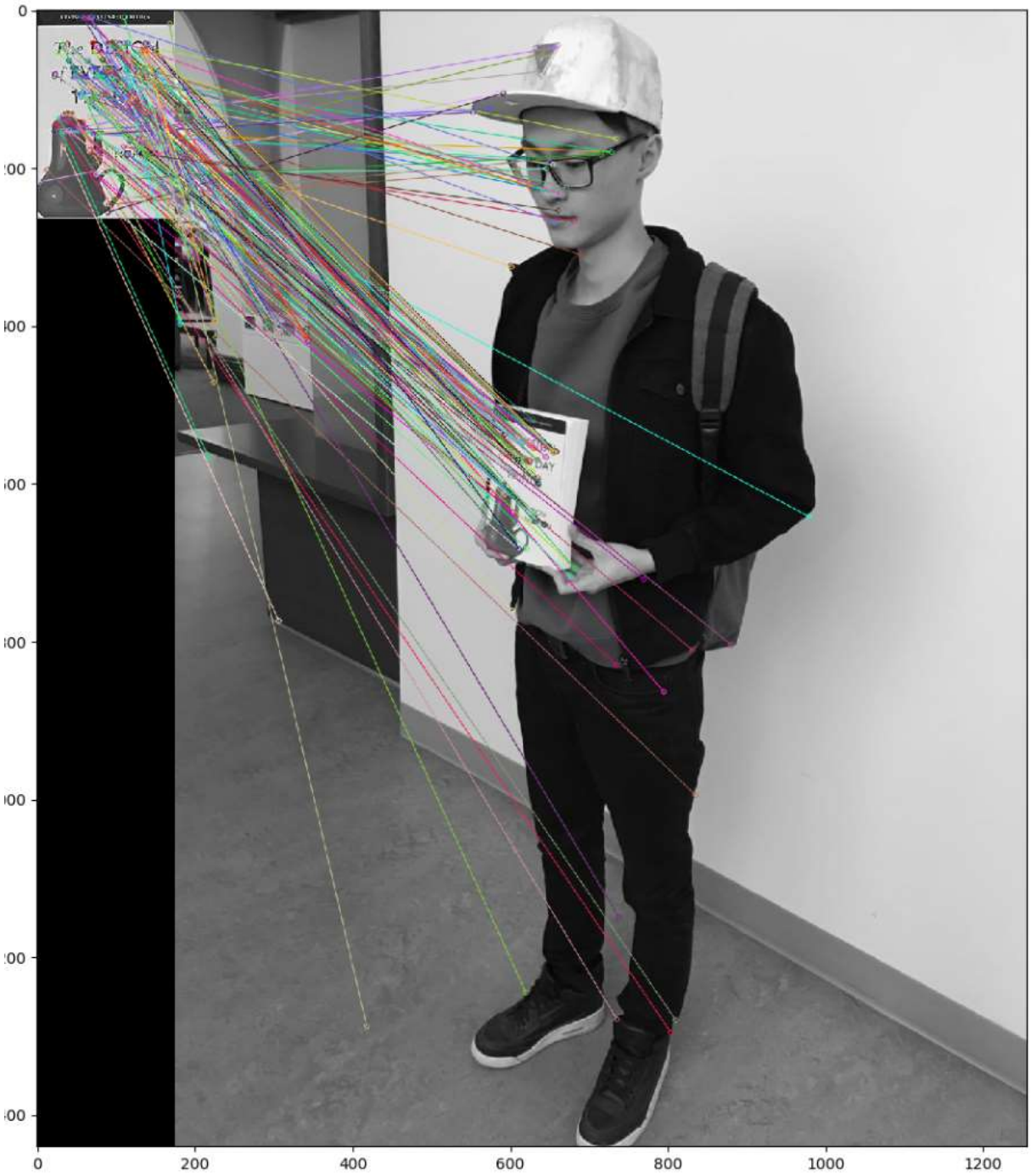
a. Using the opencv SIFT detection.

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

im1 = cv2.imread('bookCover.jpg',0)          # queryImage
im2 = cv2.imread('im3.jpg',0) # trainImage

# Initiate SIFT detector
sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(im1,None)
kp2, des2 = sift.detectAndCompute(im2,None)

# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.match(des1,des2)


# cv2.drawMatchesKnn expects list of lists as matches.
img3 = cv2.drawMatches(im1,kp1,im2,kp2,matches,flags=2, outImg = None)

plt.imshow(img3),plt.show()
```

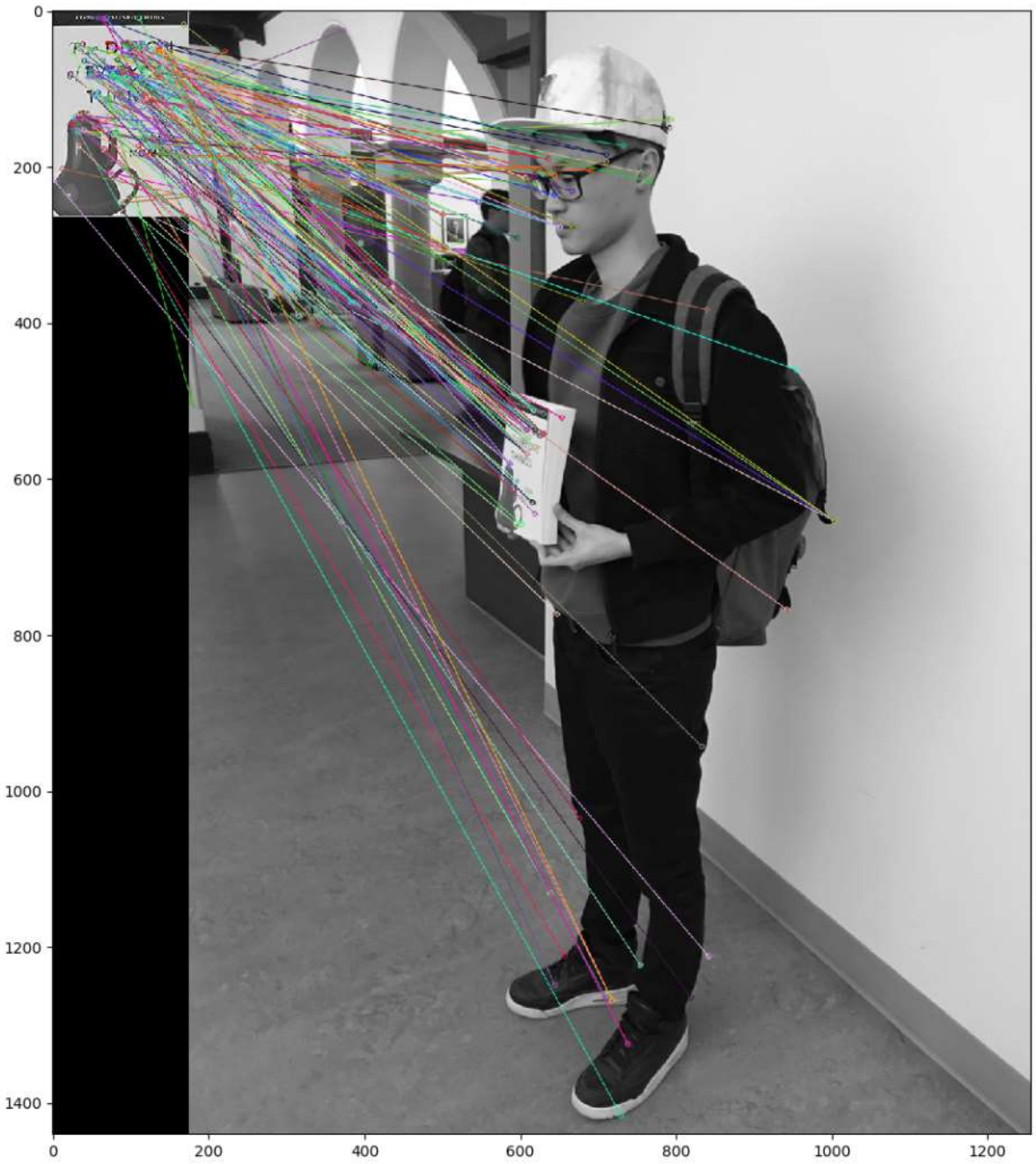
im1:



Im2:



Im3:



b:

```
P = 0.99
p = 0.8
k = 3
# k = 4 #for homography


S = np.log(1-P) / np.log(1 - p**k)
```

Im1: ~20% outliers. Number of iteration for affine transformation: 6.4. Number of iterations for homography: 8.7

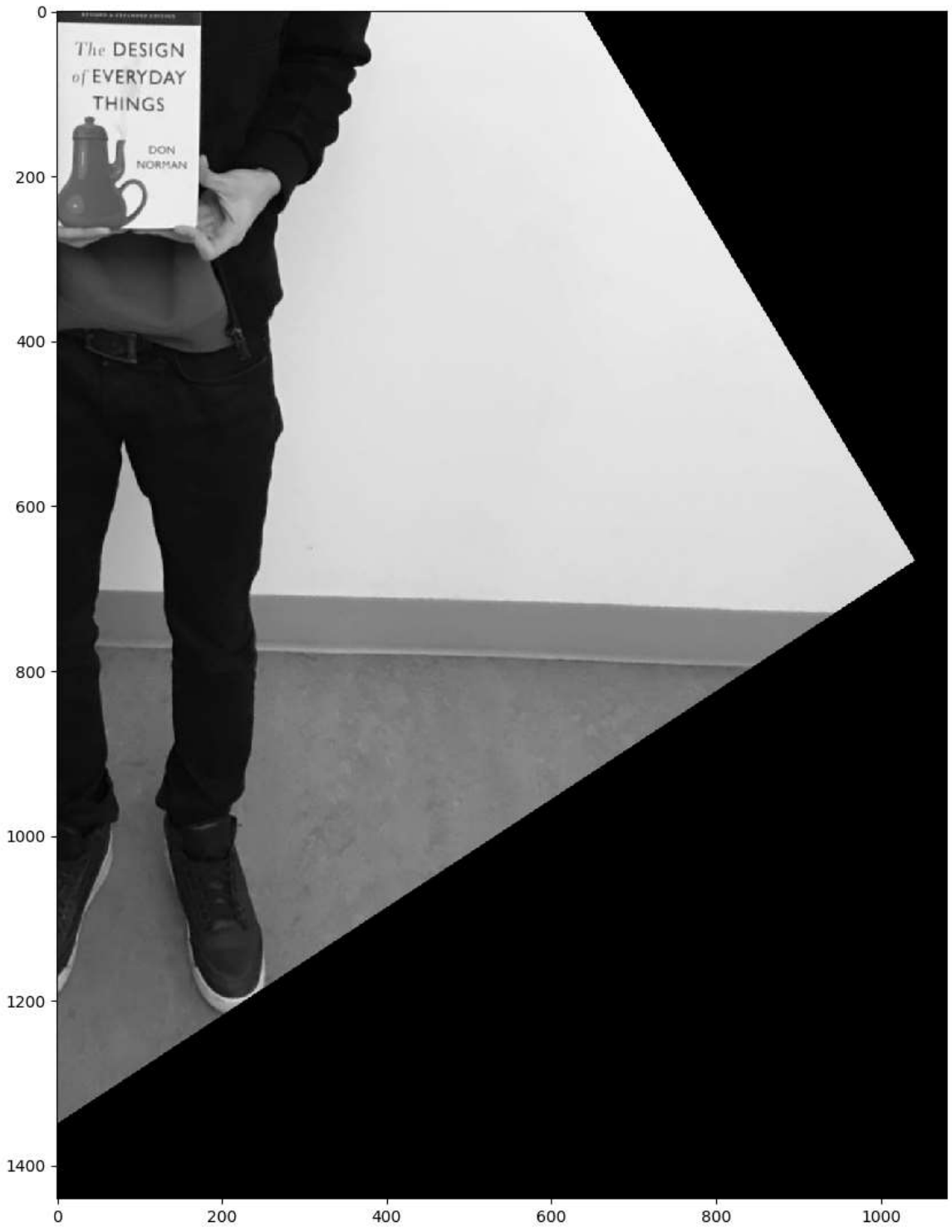
Im2: ~40% outliers. Number of iteration for affine transformation: 18.9. Number of iterations for homography: 33.1

Im3: ~60% outliers. Number of iteration for affine transformation: 69.2. Number of iterations for homography: 177.5

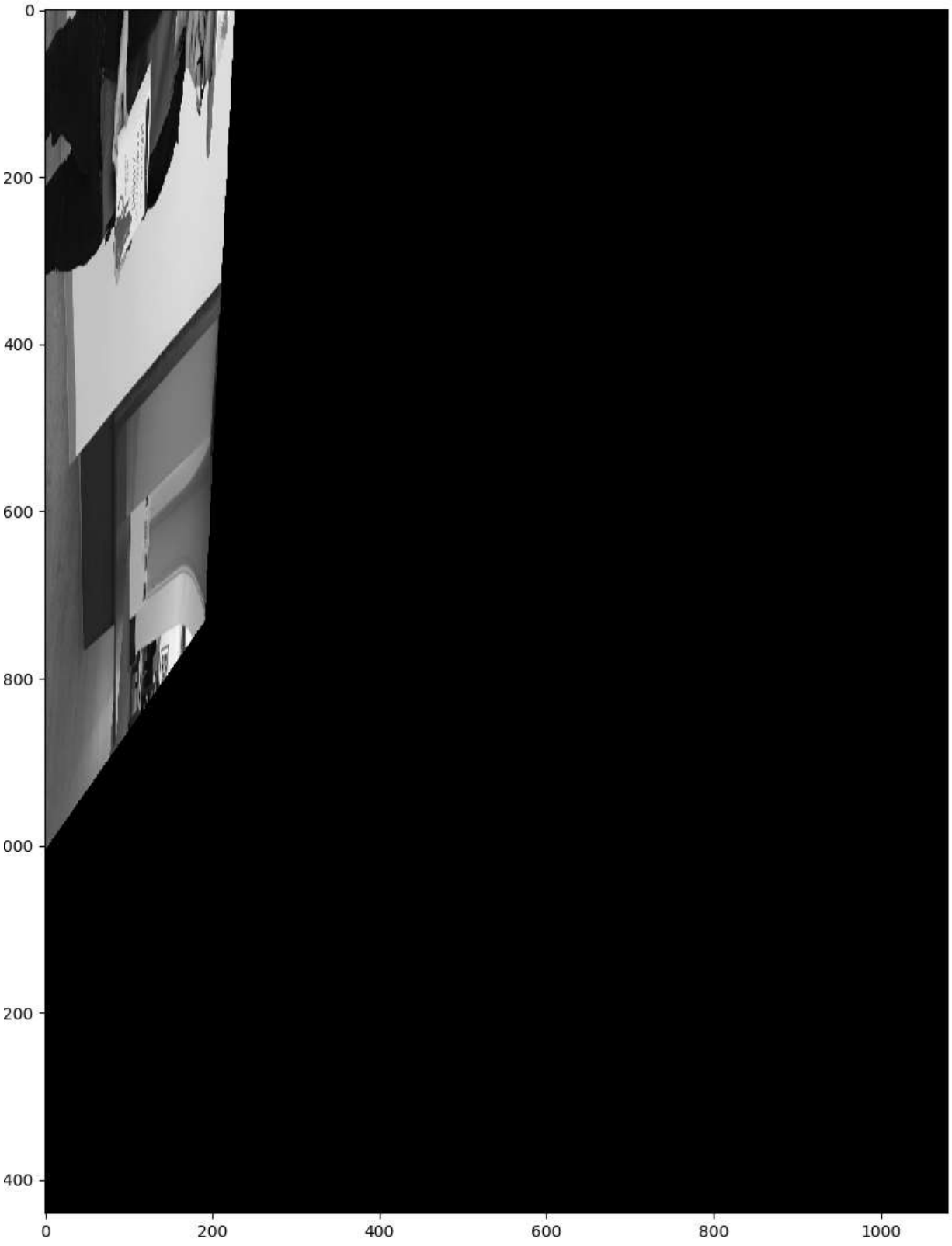
c)Using opencv sift and skimage ransac function.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from skimage.measure import LineModelND, ransac
from skimage.transform import AffineTransform, warp
from skimage.feature import match_descriptors, ORB, plot_matches

im1 = cv2.imread('bookCover.jpg',0)          # queryImage
im2 = cv2.imread('im2.jpg',0) # trainImage



sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(im1,None)
kp2, des2 = sift.detectAndCompute(im2,None)


bf = cv2.BFMatcher()

# Match descriptors.
matches = bf.match(des1,des2)




src_pts = np.array([ kp1[m.queryIdx].pt for m in matches ])
dst_pts = np.array([ kp2[m.trainIdx].pt for m in matches ])




model, inliers = ransac((src_pts,
                         dst_pts),
                        AffineTransform, min_samples=3,
                        residual_threshold=1)



warped = warp(im2, model)
plt.imshow(warped, cmap='gray'), plt.show()
```

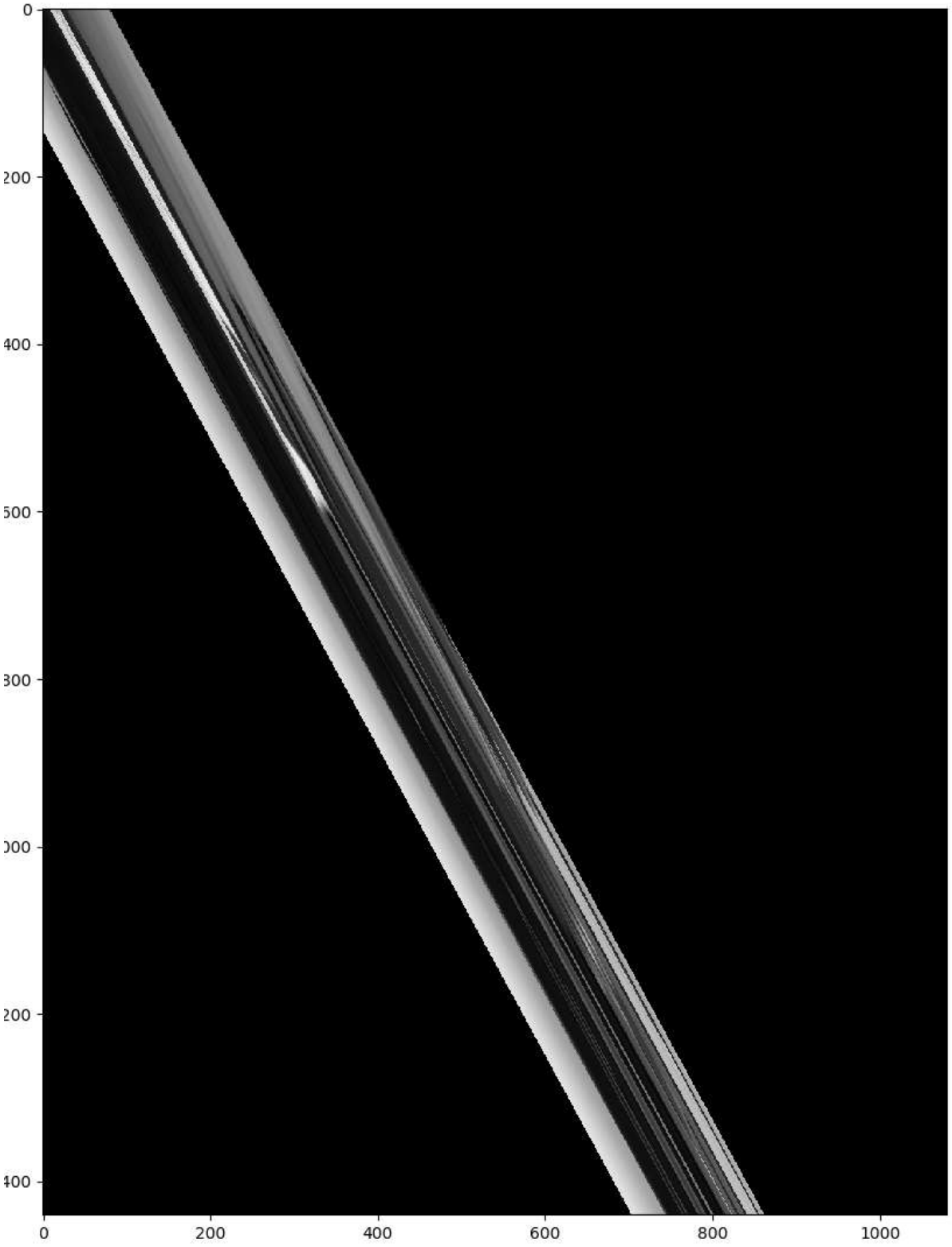
im1:


The DESIGN of EVERYDAY THINGS
DON NORMAN
0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000

Im2:



Im3:



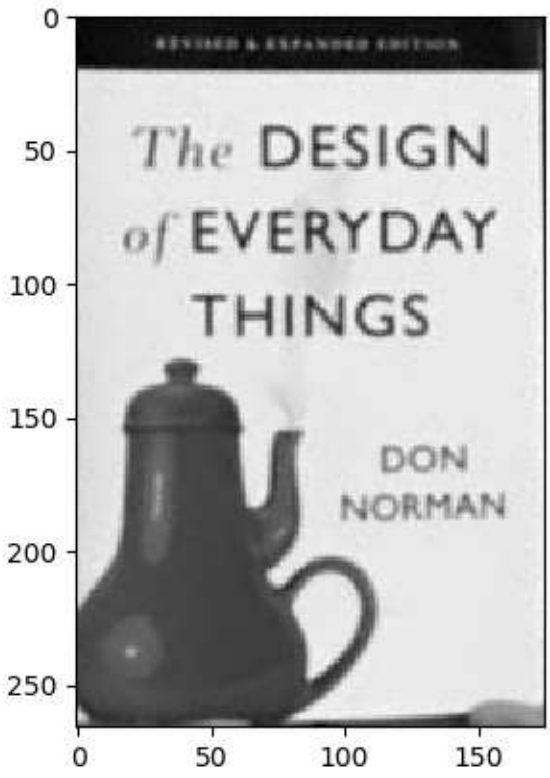
Affine transformation is good when there is little out-of-plane rotation, but is not able to transform lines that are not parallel. Therefore, affine transformation is able to identify the book in im1.jpg, but not in either im2.jpg nor im3.jpg.

(d)

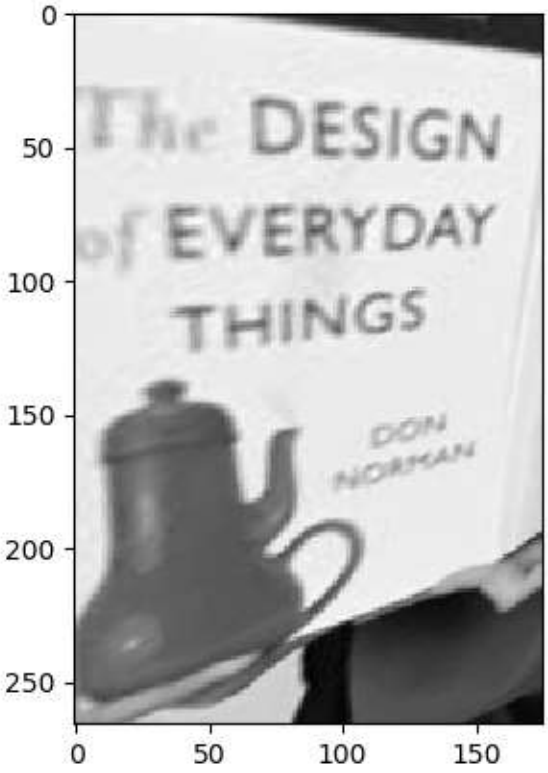
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import numpy as np
import cv2
from matplotlib import pyplot as plt
from skimage.measure import LineModelND, ransac
from skimage.transform import AffineTransform, warp
from skimage.feature import match_descriptors, ORB, plot_matches



img1 = cv2.imread('bookCover.jpg')          # queryImage
img2 = cv2.imread('im3.jpg') # trainImage

# Initiate SIFT detector
sift = cv2.xfeatures2d.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)


FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)

flann = cv2.FlannBasedMatcher(index_params, search_params)

matches = flann.knnMatch(des1,des2,k=2)

# store all the good matches as per Lowe's ratio test.
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)

src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
```

```
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 3.0)
matchesMask = mask.ravel().tolist()

h,w,ch= img1.shape
pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
dst = cv2.perspectiveTransform(pts,M)

img3 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
# H, status = cv2.findHomography(dst, pts)
# dst = cv2.warpPerspective(img2, H, (w,h))

plt.imshow(cv2.cvtColor(img3, cv2.COLOR_BGR2RGB))
plt.show()
```

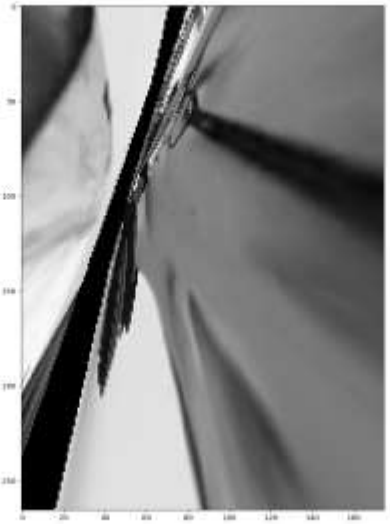
Im1:



Im2:


0
50
100
150
200
250
0
50
100
150
The DESIGN
of EVERYDAY
THINGS
DON
NORMAN

Im3:



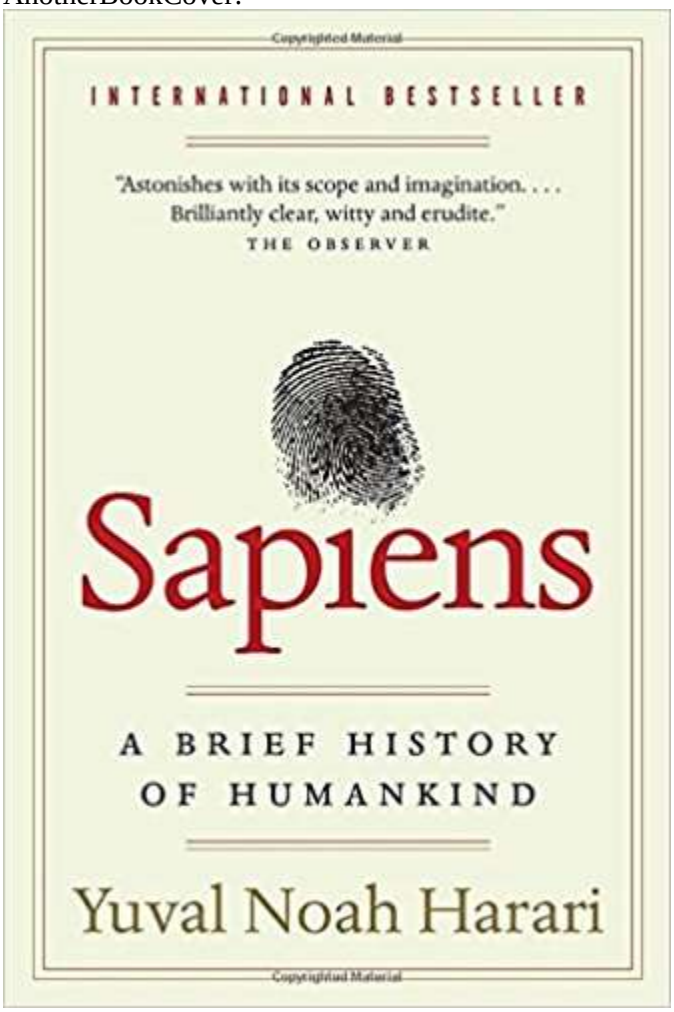
Homography, Projective transformation, is able to identify the book with some degree of out-of-plane rotation. However, when the rotation is too large, it cannot identify the object. Compares to affine transformation, it is able to identify im1.jpg and im2.jpg, but fails to identify im3.jpg.

Note: I have also submitted images with a complete view with the original im1, im2, im3 images on markus (wrap1.jpg, wrap2.jpg, wrap3.jpg)

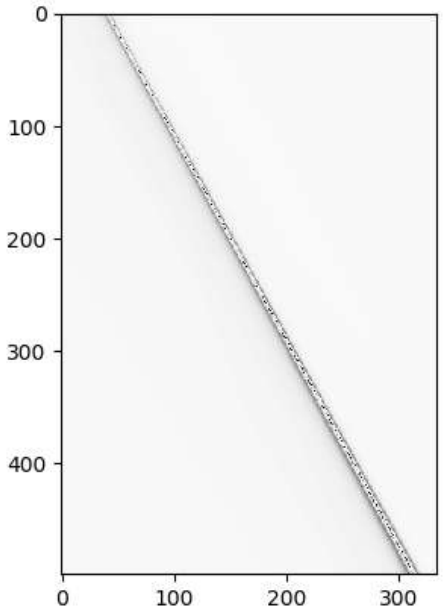
e) [1 point] use a homography to map the cover of the second book (another BookCover) onto each image. Discuss your results.

Using the same function as d), and use the below book cover.

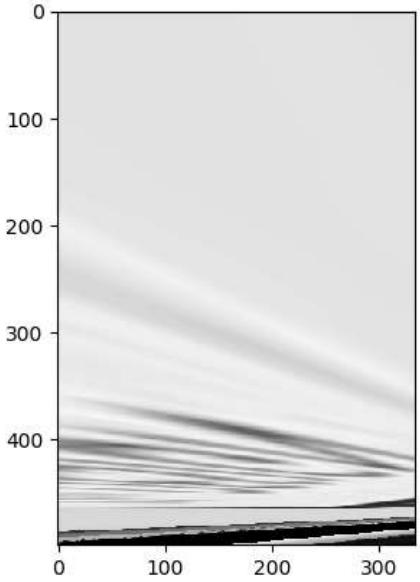
AnotherBookCover:



Im1:



Im2:



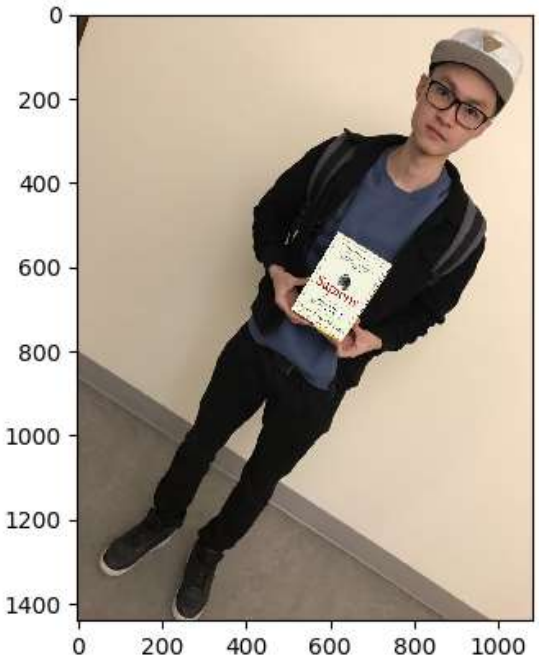
Im3:
Program fails because there is not enough match.

Because there is no similarity between two book cover, the program is unable to find the correct match.

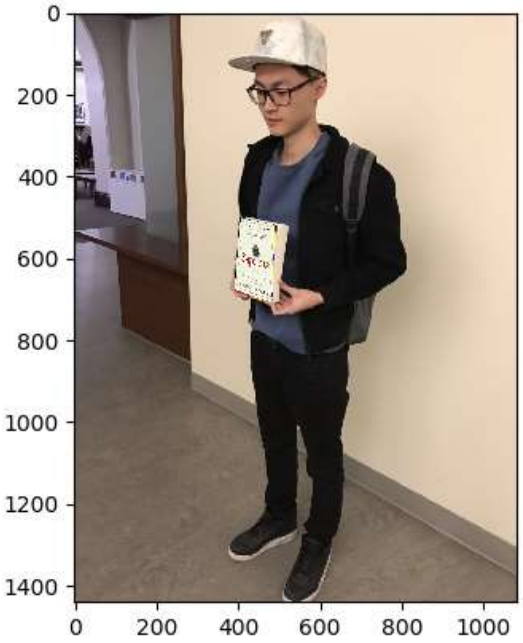
I later use manual input to past cover on the image.

```
im_src = cv2.imread('anotherBookCover.jpg')
size = im_src.shape

# Create a vector of source points.
pts_src = np.array(
                    [
                    [0,0],
                    [size[1] - 1, 0],
                    [size[1] - 1, size[0] -1],

                    [0, size[0] - 1 ]
                    ],dtype=float
                    )


# Read destination image
im_dst = cv2.imread('im3.jpg')


plt.imshow(im_dst)
pts_dst = np.array(plt.ginput(4))
plt.show()

h, status = cv2.findHomography(pts_src, pts_dst)


im_temp = cv2.warpPerspective(im_src, h, (im_dst.shape[1],im_dst.shape[0]))

cv2.fillConvexPoly(im_dst, pts_dst.astype(int), 0, 16)

im_dst = im_dst + im_temp
plt.imshow(cv2.cvtColor(im_dst, cv2.COLOR_BGR2RGB)), plt.show()
```

im1:


0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000

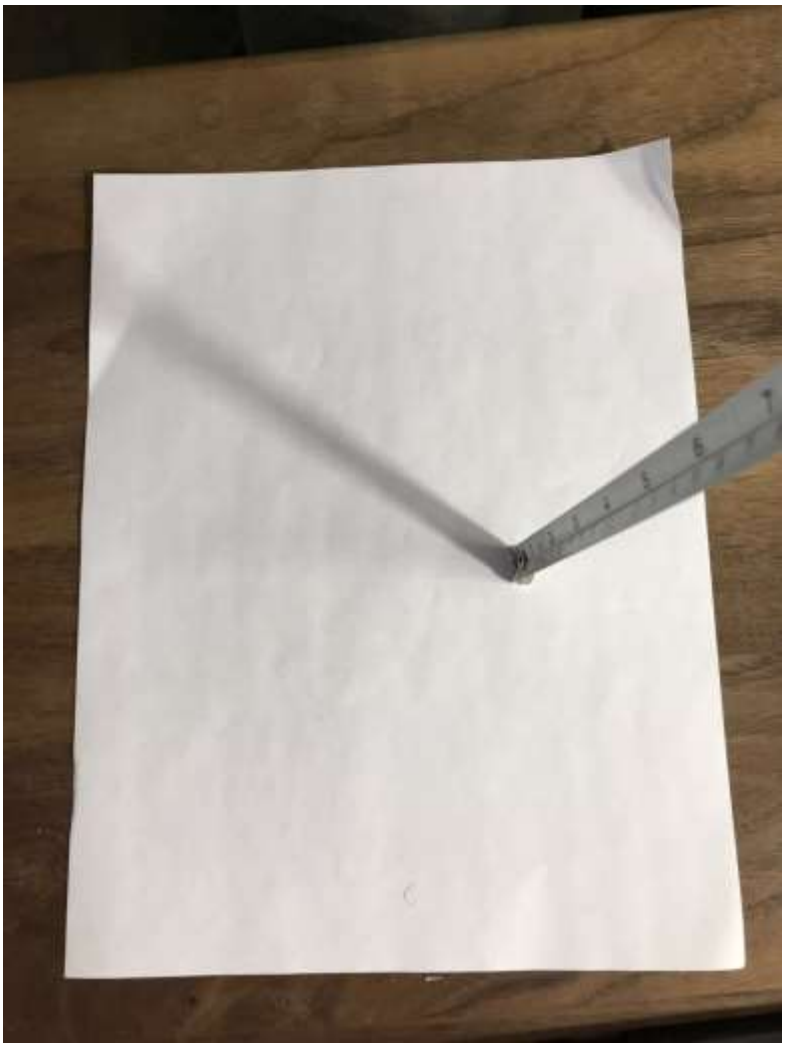
im2


0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000

Im3:

Part C:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import numpy as np
import cv2
from matplotlib import pyplot as plt
from skimage.measure import LineModelND, ransac
from skimage.transform import AffineTransform, warp
from skimage.feature import match_descriptors, ORB, plot_matches



img1 = cv2.imread('30cm.jpg')          # queryImage

# Initiate SIFT detector


h,w,ch= img1.shape

plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
x = np.array(plt.ginput(2))
plt.show()

print w, h
print x
```



I took the above picture 300mm away from the A4 paper, and the width of A4 paper in the image is 2344.6 pixel, therefore the focal length of the camera is $f = P \times D \ / \ W \ = \ 2344.6 \times 300/210$=3349.4

The dimension of the image is 3024 * 4032

Therefore, the internal parameter is

$$K = \begin{bmatrix} 3349.4 & 0 & 1512 \\ 0 & 3349.4 & 2016 \\ 0 & 0 & 1 \end{bmatrix}$$