CSC420

Introduction to Image Understanding

Assignment 2

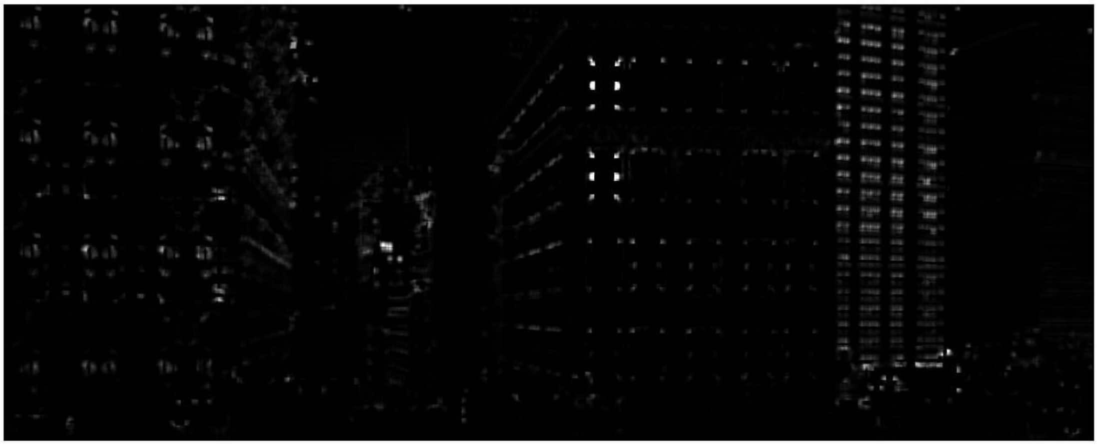
Zhili Xu
xuzhili

Question 1

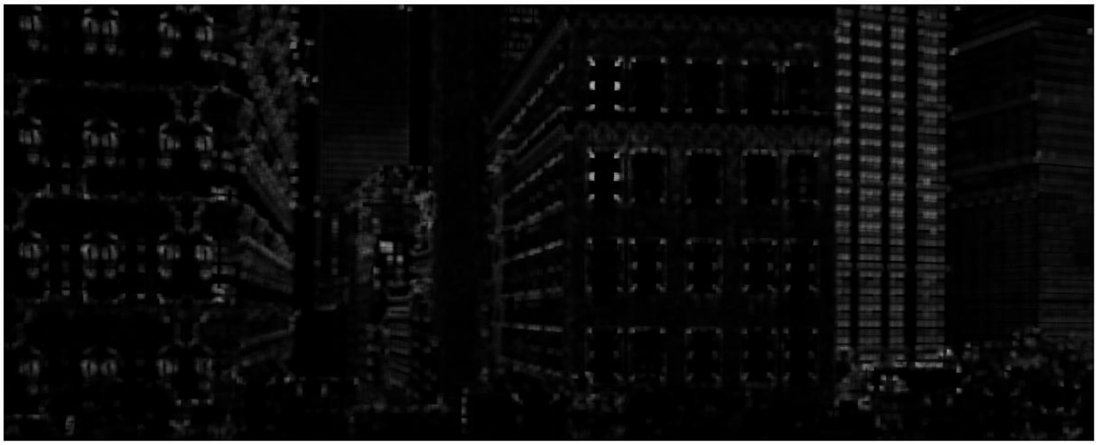
(a)

```
im = imread('./images/building.jpg');
image = im2double(rgb2gray(im));
h = fspecial('gaussian',[5 5],7);
blur = imfilter(image, h, 'same');



[Ix, Iy] = imgradientxy(image);

IxIy = Ix.*Iy;
Ix2 = Ix.^2;
Iy2 = Iy.^2;


h2 = fspecial('gaussian',[7 7],10);
Ix2_blur = imfilter(Ix2, h2, 'same');
Iy2_blur = imfilter(Iy2, h2, 'same');
IxIy_blur = imfilter(IxIy, h2, 'same');
%

det = Ix2_blur.*Iy2_blur - IxIy_blur.*IxIy_blur;
trace = Ix2_blur + Iy2_blur;

% Harris
% R =  det - 0.05 * (trace.^2);
% Brown
R = det ./ trace;

imshow(R);
```

Harris:



Brown:

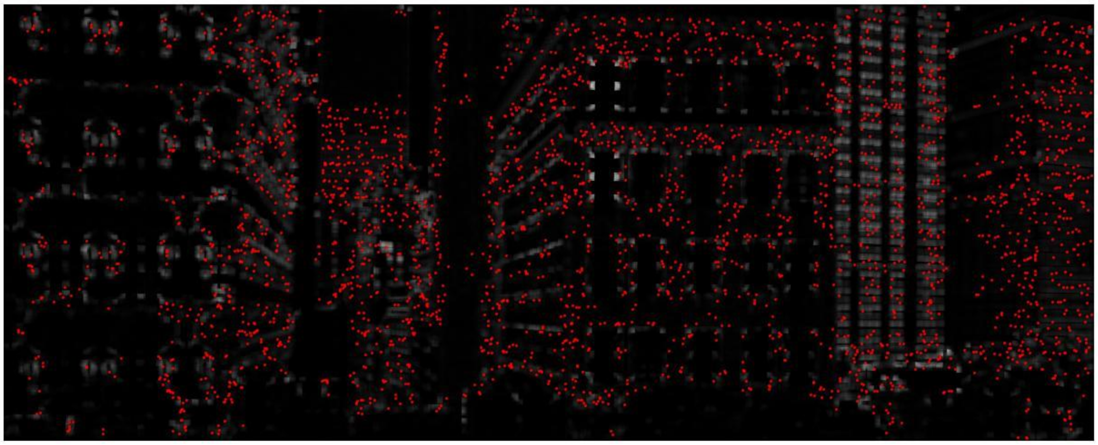


By comparing these two pictures, Harris Conner method focus more on the clearly comers, as the two windows corners are brighter than Brown’s method. On the other hand, Brown’s method has clear outlines, it shows the background corners as well.

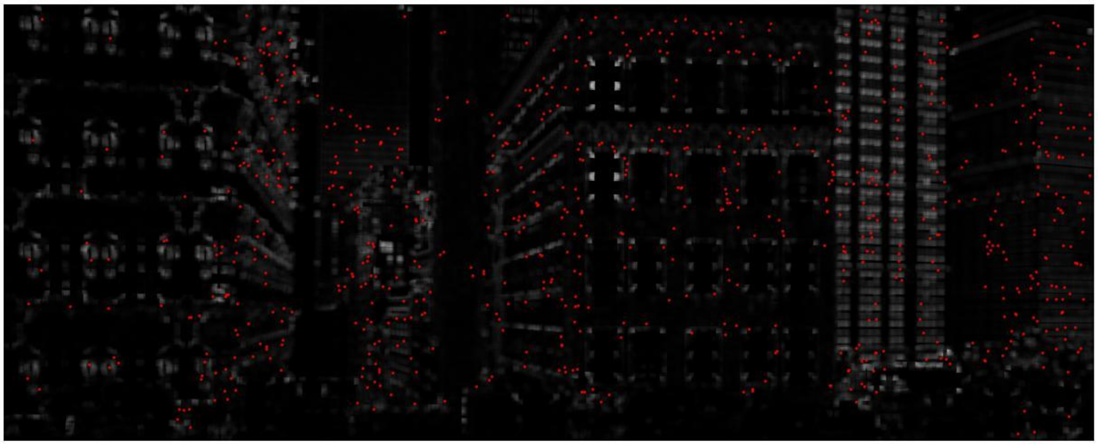
(b)

```
r = 5;
[x, y] = meshgrid(1:r*2, 1:r*2);
mask = (x - r).^2+(y - r).^2 <= r.^2;
local = ordfilt2(R, r ^ 2, mask);
[Y, X] = find(local > 0.01 & R == local);
imshow(R);
hold on;
plot(X, Y, 'r.');
```

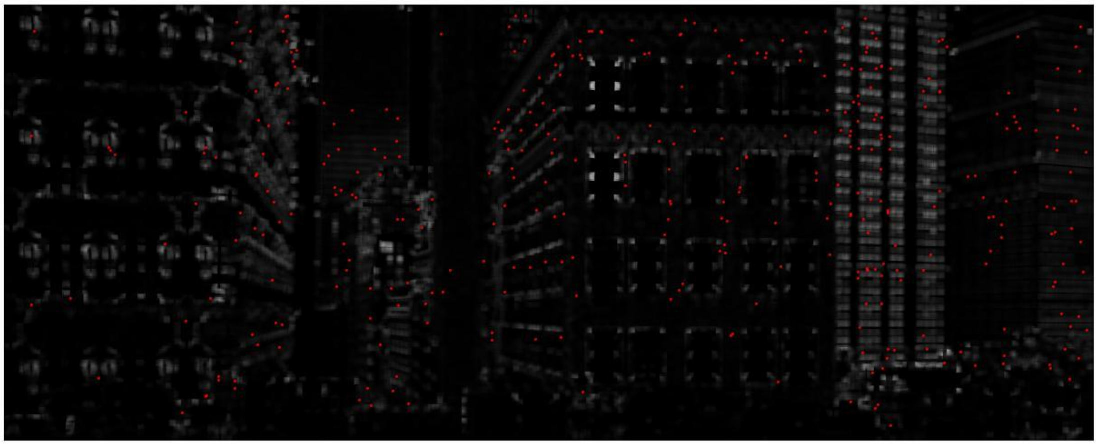
r = 5



r = 10



r = 15

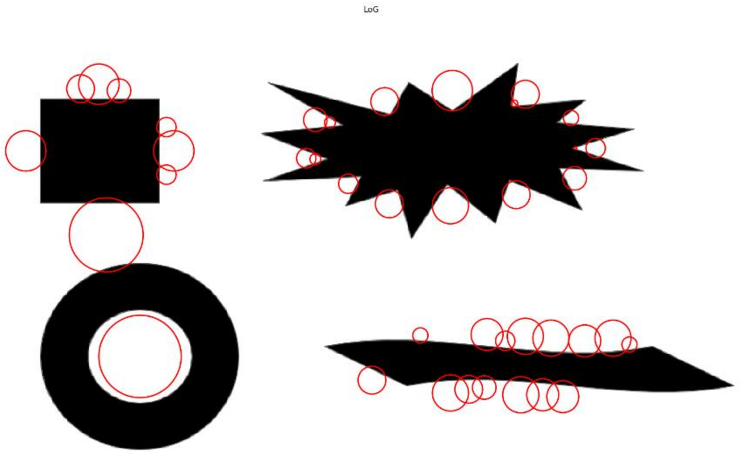


When r is increasing, less dots are plotted which means less corners are marked. This is because when r is increasing, the size of filter mask is increasing, therefore more non-maximal corners are suppressed.

(c)

```
from math import sqrt
from skimage import data
from skimage.feature import blob_log
from skimage.color import rgb2gray
import cv2
import matplotlib.pyplot as plt


image = cv2.imread("./images/synthetic.png")
image_gray = rgb2gray(image)

blobs_log = blob_log(image_gray, min_sigma = 1, max_sigma=40, num_sigma=25, 
threshold=.242)


blobs_log[:, 2] = blobs_log[:, 2] * sqrt(2)

color = 'red'
title = 'Laplacian of Gaussian'


fig= plt.subplots(figsize=(9, 3), sharex=True, sharey=True)
fig[1].set_title("LoG")
fig[1].imshow(image, interpolation='nearest')
```

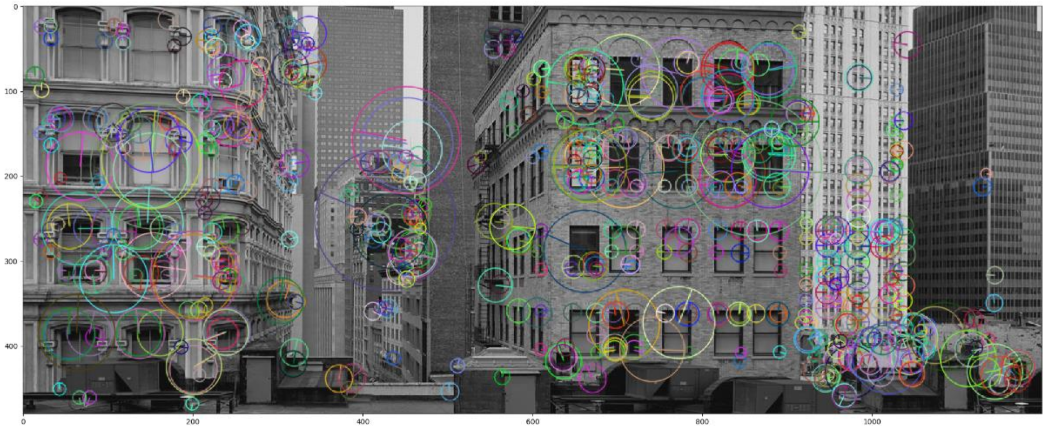
```
for blob in blobs_log:
    y, x, r = blob
    c = plt.Circle((x, y), r, color=color, linewidth=2, fill=False)
    fig[1].add_patch(c)
fig[1].set_axis_off()

plt.tight_layout()
plt.show()
```



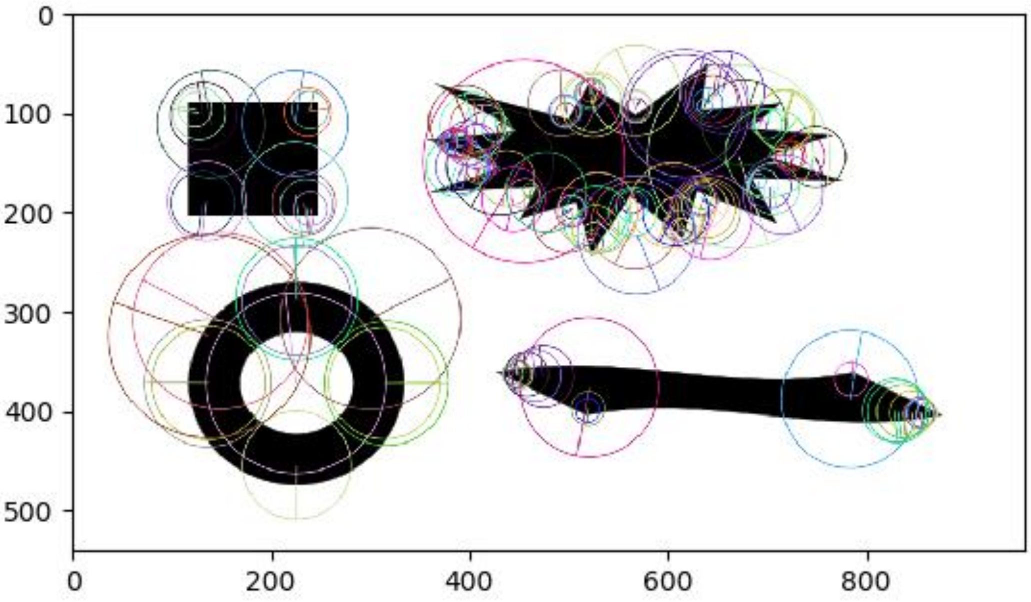
(d)

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

if __name__ == '__main__':
  img = cv2.imread('./images/building.jpg')
  gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)


  surf = cv2.xfeatures2d.SURF_create(hessianThreshold = 1000)
  kp = surf.detect(gray,None)


  img=cv2.drawKeypoints(gray,kp,None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
  plt.imshow(img),plt.show()
```

building.jpg:



Synthetic.png



For this question, I am using opencv SURF detection (https://docs.opencv.org/3.1.0/d5/df7/classcv_1_1xfeatures2d_1_1SURF.html).

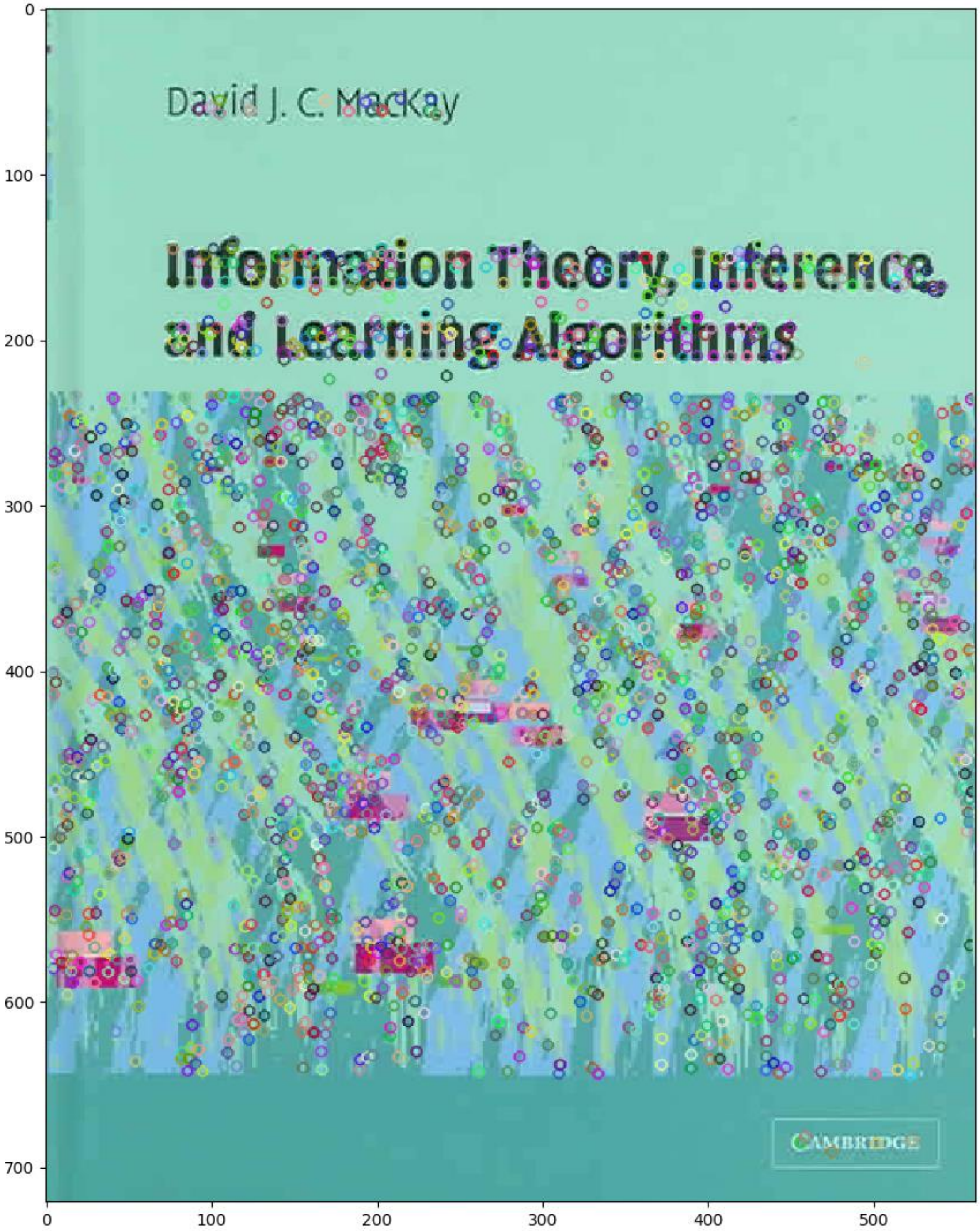
For orientation assignment, SURF uses wavelet responses in horizontal and vertical direction for a neighbourhood of size 6s. For feature description, SURF uses Wavelet responses in horizontal and vertical direction.

2. SIFT Matching

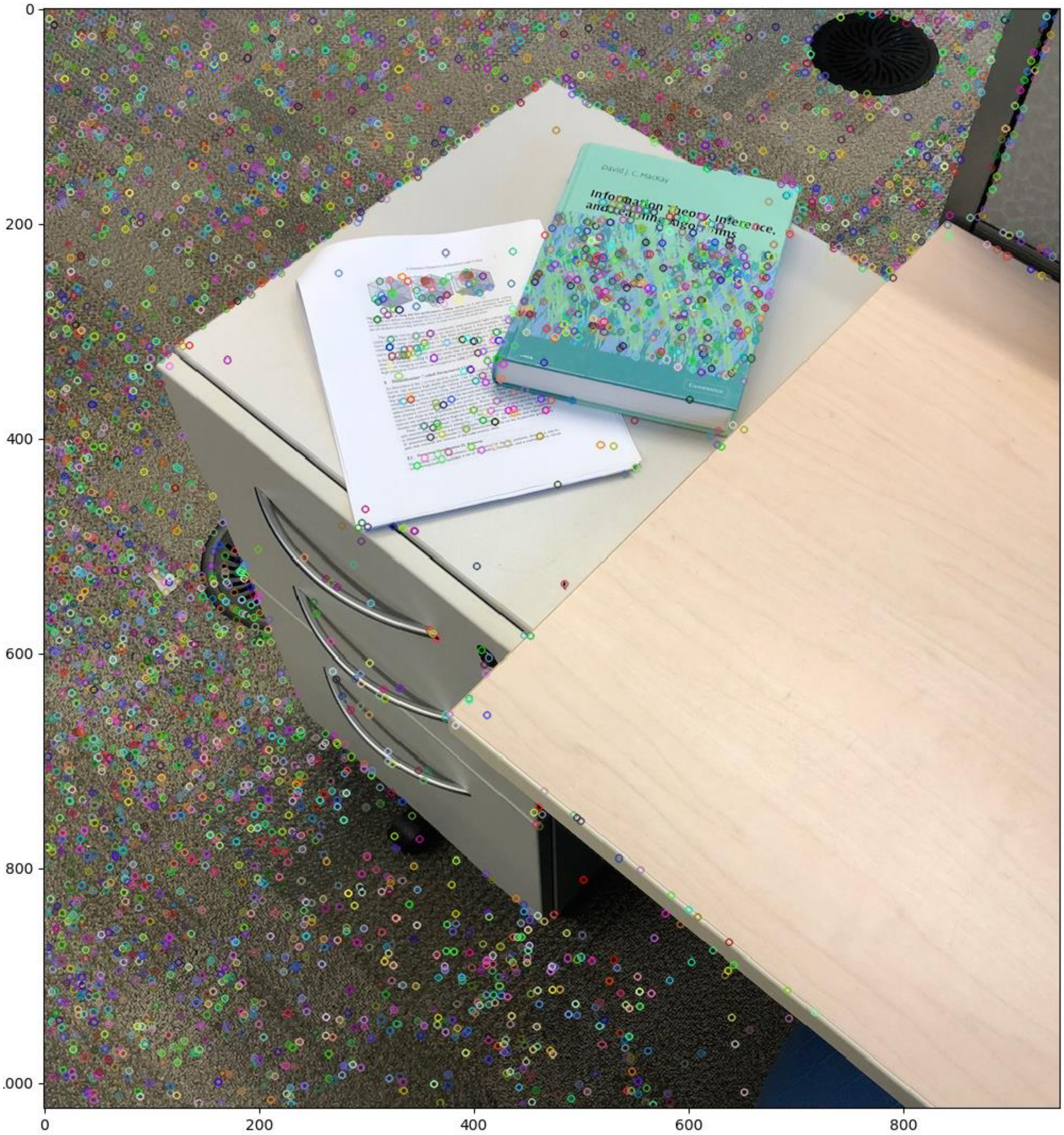
(a)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance
from matplotlib.patches import ConnectionPatch
import matplotlib
from scipy.spatial.distance import cdist
from numpy.linalg import inv


if __name__ == '__main__':

    img1 = cv2.imread('./images/book.jpeg')
    img2 = cv2.imread('./images/findbook.png')
    gray= cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    gray2= cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

    sift1 = cv2.xfeatures2d.SIFT_create(sigma = 2)
    sift2 = cv2.xfeatures2d.SIFT_create( sigma = 1.6)

    kp, des = sift1.detectAndCompute(gray,None)
    kp2, des2 = sift2.detectAndCompute(gray2,None)

    img1=cv2.drawKeypoints(img1,kp, None)
    plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
    plt.show()
    img2=cv2.drawKeypoints(img2,kp2, None)
    plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
    plt.show()
```

Book


David J. C. MacKay
Information Theory, Inference, and Learning Algorithms
CAMBRIDGE
0
100
200
300
400
500
600
700
0
100
200
300
400
500

Findbook

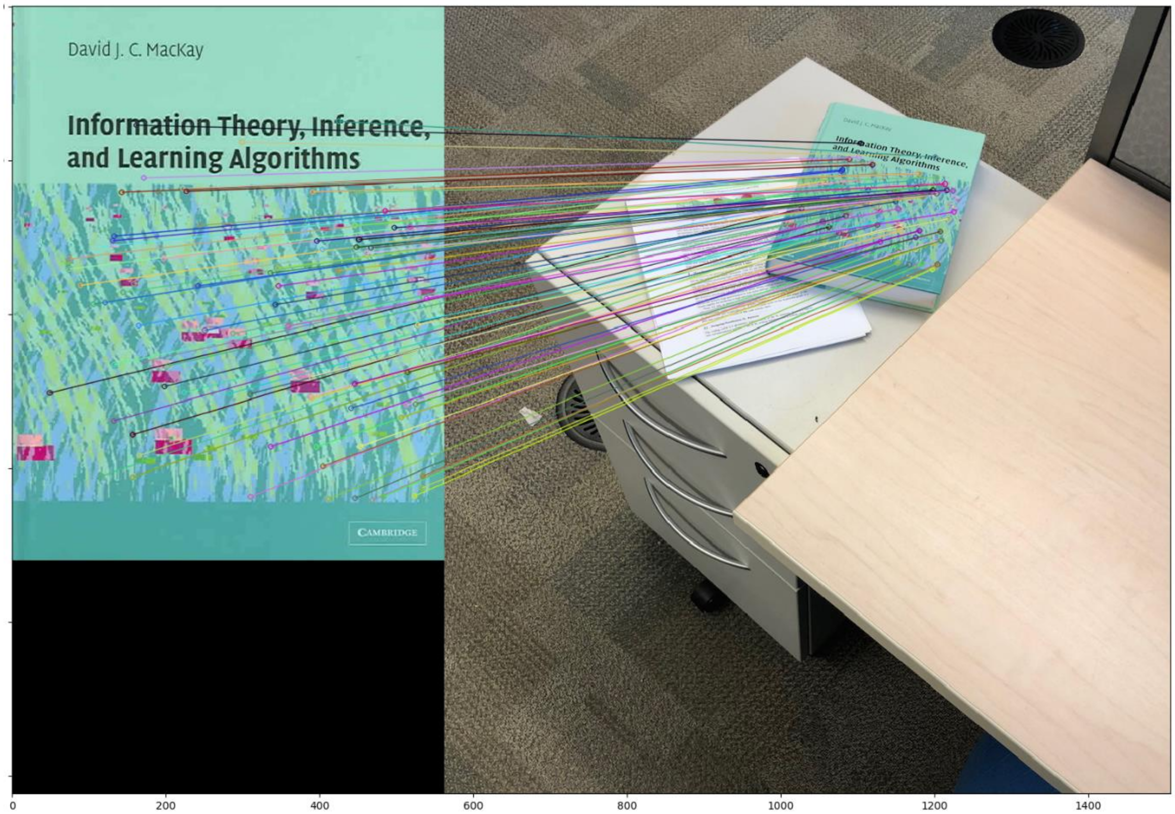


For this question, I am using opencv SIFT detection (https://docs.opencv.org/3.4/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html).

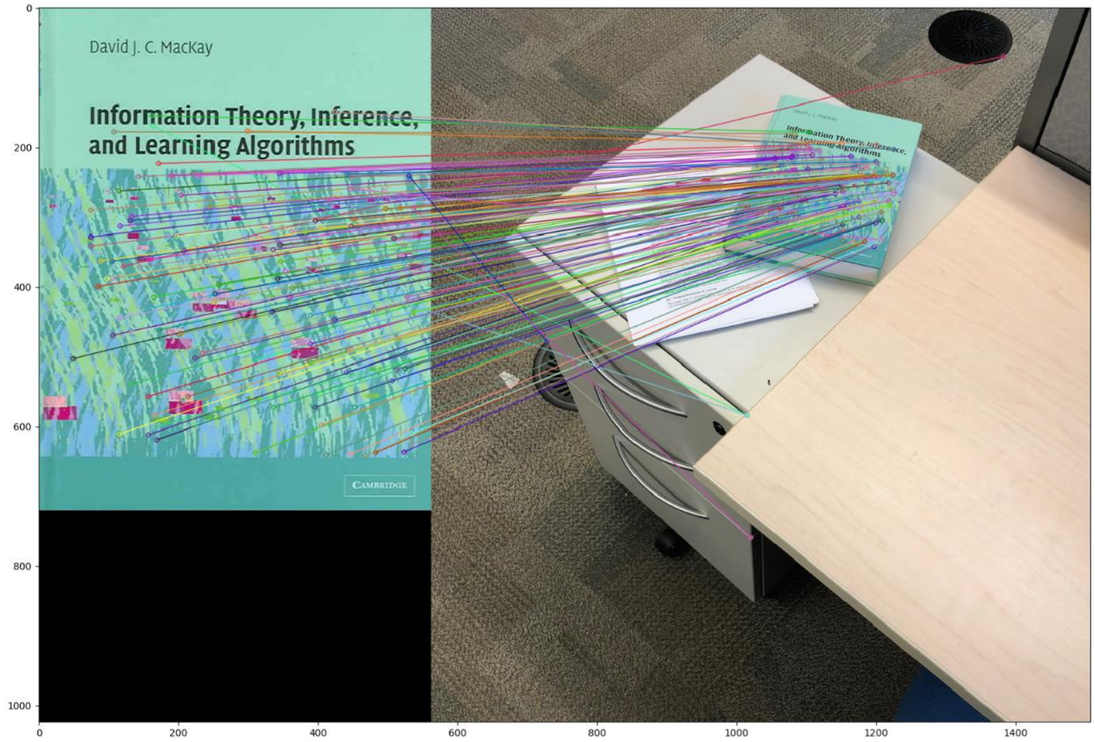
(b)

```
if __name__ == '__main__':
    img1 = cv2.imread('./images/book.jpeg')
    img2 = cv2.imread('./images/findbook.png')
    gray= cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    gray2= cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

    sift1 = cv2.xfeatures2d.SIFT_create(sigma = 2)
    sift2 = cv2.xfeatures2d.SIFT_create( sigma = 1.6)

    kp, des = sift1.detectAndCompute(gray,None)
    kp2, des2 = sift2.detectAndCompute(gray2,None)
    threshold = 0.6
    dist = cdist( des, des2, 'euclidean')
    cord = []
    newkp=[]
    newkp2=[]
    for i in range(len(dist)):
        smallest = float("inf")
        second = float("inf")
        smallest_index = 0
        second_index =0
        for j in range(len(dist[i])):
            value = dist[i][j]
            if (value < second):
                if (value < smallest):
                    second = smallest
                    second_index = smallest_index
                    smallest = value
                    smallest_index = j
                else:
                    second = value
                    second_index = j
        if (smallest/second < threshold):
            newkp.append(kp[i])
            newkp2.append(kp2[smallest_index])
            cord.append((i, smallest_index, smallest))

    matches = []
    for i in range(len(newkp)):
        matches.append(cv2.DMatch(_queryIdx = i, _trainIdx = i, _distance =
cord[i][2]))
    img3 = cv2.drawMatches(img1,newkp,img2,newkp2,matches,outImg = None)
    plt.imshow(img3),plt.show()
```

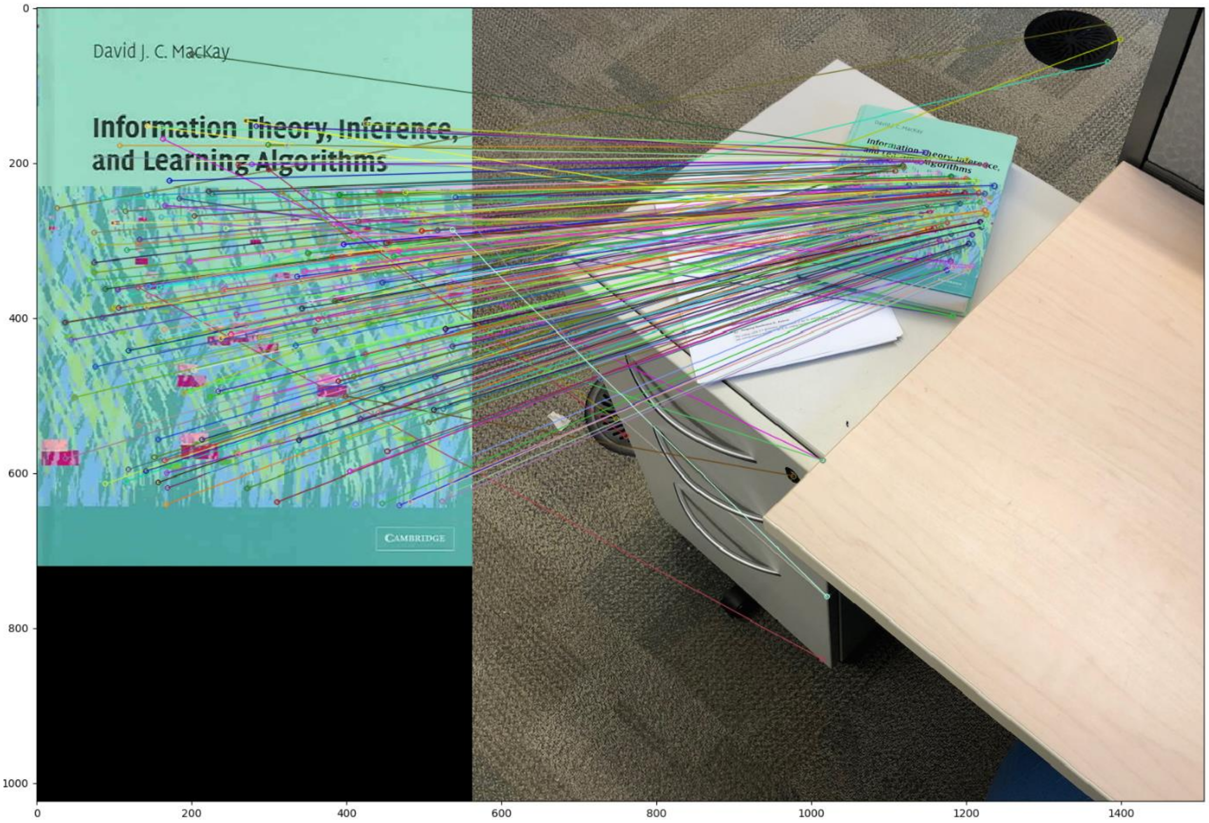
0.6:


David J. C. MacKay
Information Theory, Inference, and Learning Algorithms
CAMBRIDGE
0
200
400
600
800
1000
1200
1400

0.7


David J. C. MacKay
Information Theory, Inference, and Learning Algorithms
CAMBRIDGE
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

0.8



The percentage of true match for threshold:

0.6 is “0.0290835306053”

0.7 is “0.0480216435577”

0.8 is “0.0760906323977”

The best value is threshold 0.6

(c) follow part(b) code:

```
matches.sort(key=lambda tup: tup.distance)
k = 5
best = matches[:k]
M = np.zeros((2*k, 6))
M2 = np.zeros((2*k,1))
A = np.zeros((2*k,1))
j = 0
for i in range(len(best)):
    x = newkp[best[i].queryIdx].pt[0]
    y = newkp[best[i].queryIdx].pt[1]
    x2 = newkp2[best[i].trainIdx].pt[0]
    y2 = newkp2[best[i].trainIdx].pt[1]
    M[j,0] = x
    M[j,1] = y
    M[j+1,2] = x
    M[j+1,3] = y
    M[j,4] = 1
    M[j+1,5] = 1
    M2[j] = x2
    M2[j+1] = y2
    j +=2

np.set_printoptions(suppress=True)
A = np.dot(np.dot(inv(np.dot( np.transpose(M) , M) ), np.transpose(M)) , M2)
```

k = 5:

```
[[  0.38840657]
 [ -0.09916932]
 [  0.07891803]
 [  0.28025115]
 [494.01779715]
 [121.56470857]]
```

k = 6:

```
[[  0.3897687 ]
 [ -0.09951994]
 [  0.08018606]
 [  0.27992476]
 [493.78581467]
 [121.34875238]]
```

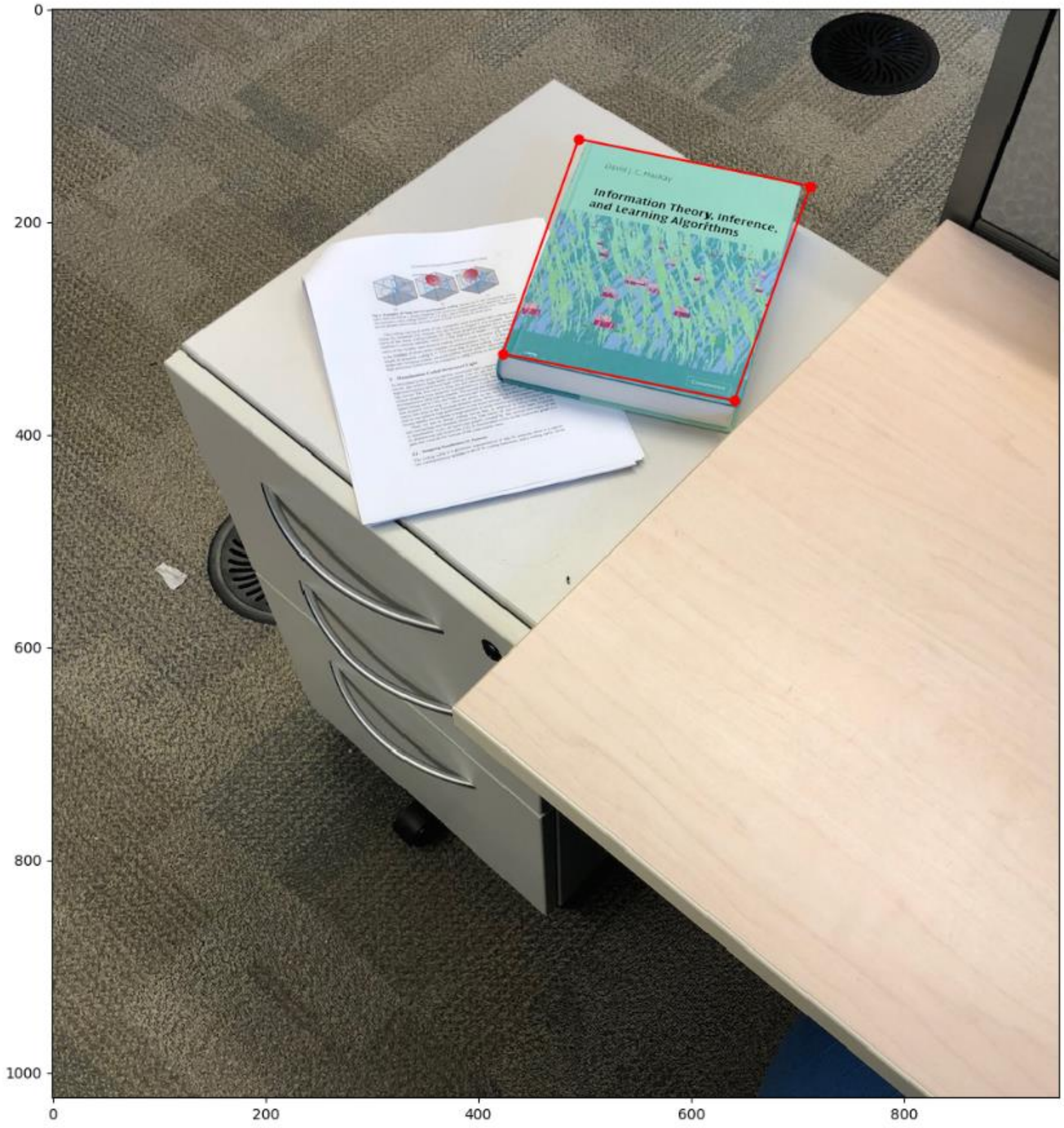
k = 4:

```
[[  0.39122639]
 [ -0.10193564]
 [  0.08373004]
 [  0.27553046]
 [494.73279813]
 [122.78485121]]
```

The minimum “k” required is 1.

(d)

Visualize affine

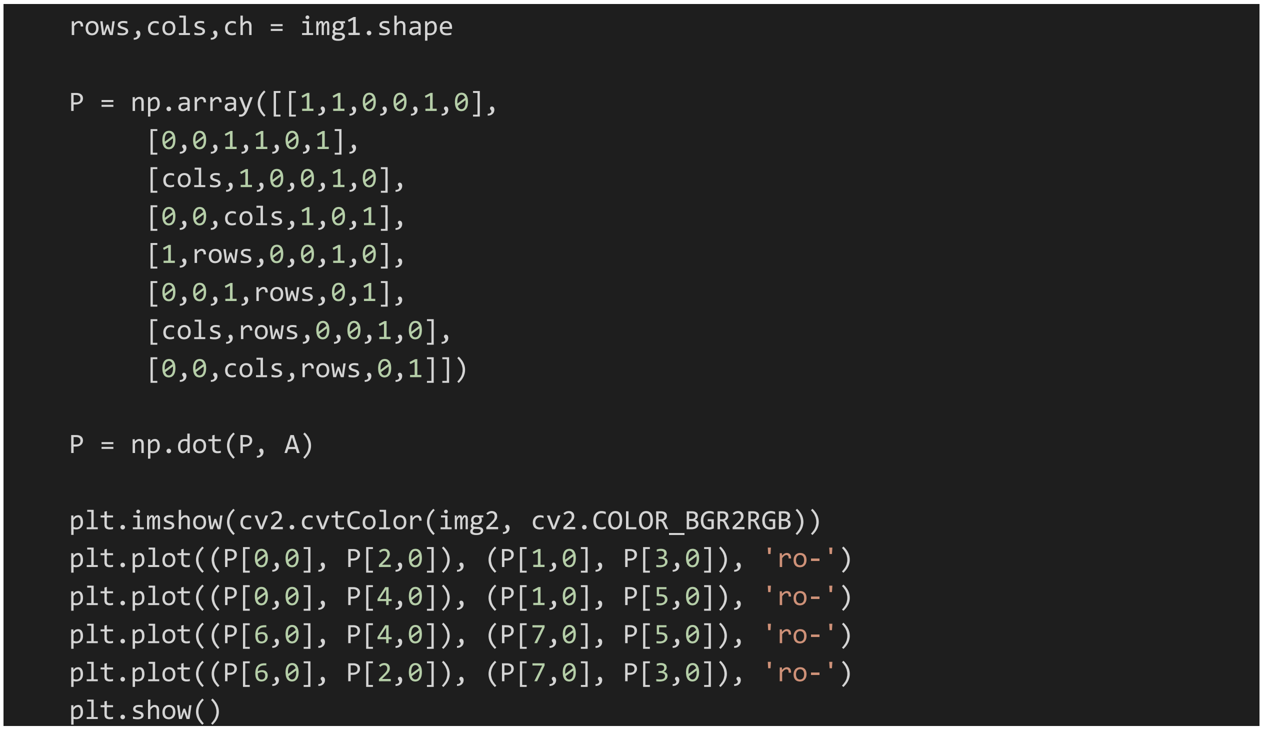


(e)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance
from matplotlib.patches import ConnectionPatch
import matplotlib
from scipy.spatial.distance import cdist
from numpy.linalg import inv



def matching(dist,kp, kp2):
    cord = []
    newkp=[]
    newkp2=[]
    threshold = 0.6
    for i in range(len(dist)):
        smallest = float("inf")
        second = float("inf")
        smallest_index = 0
        second_index =0
        for j in range(len(dist[i])):
            value = dist[i][j]

            if (value < second and value != 0):
                if (value < smallest):
                    second = smallest
                    second_index = smallest_index
                    smallest = value
                    smallest_index = j
                else:
                    second = value
                    second_index = j
            # if (smallest == 0 or second == 0):
        if ( smallest/second < threshold):
            newkp.append(kp[i])
            newkp2.append(kp2[smallest_index])
            cord.append((i, smallest_index, smallest))

    matches = []
    for i in range(len(newkp)):
        matches.append(cv2.DMatch(_queryIdx = i, _trainIdx = i, _distance = 
cord[i][2]))
    temp = []
```

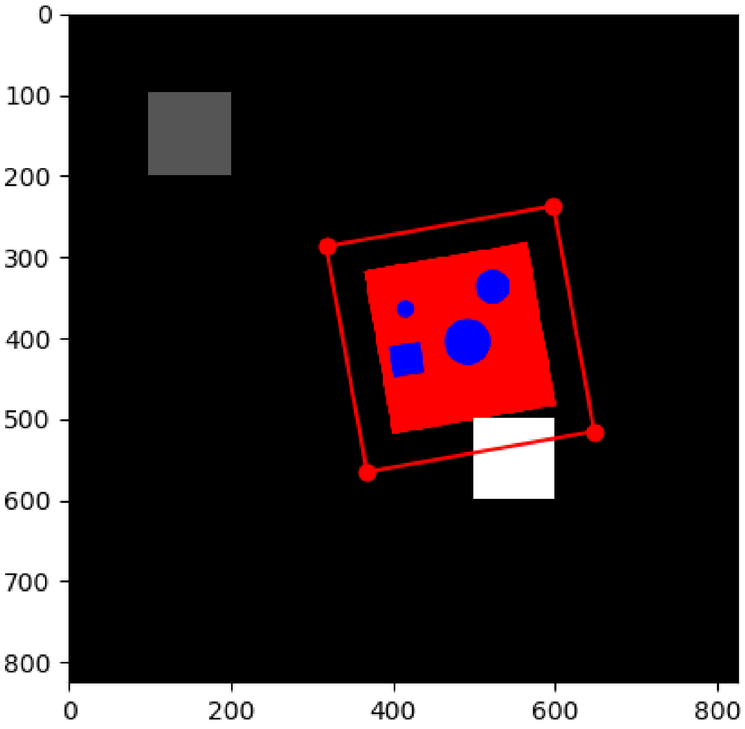
```python
    for i in range(len(matches)):
        temp.append((newkp[i],newkp2[i],matches[i]))
    return temp


if __name__ == '__main__':

    img1 = cv2.imread('./images/colourTemplate.png')
    img2 = cv2.imread('./images/colourSearch.png')





    img1blue = img1.copy()
    img1blue[:,:,1] = 0
    img1blue[:,:,2] = 0
    img1green = img1.copy()
    img1green[:,:,0] = 0
    img1green[:,:,2] = 0
    img1red = img1.copy()
    img1red[:,:,0] = 0
    img1red[:,:,1] = 0

    img2blue = img2.copy()
    img2blue[:,:,1] = 0
    img2blue[:,:,2] = 0
    img2green = img2.copy()
    img2green[:,:,0] = 0
    img2green[:,:,2] = 0
    img2red = img2.copy()
    img2red[:,:,0] = 0
    img2red[:,:,1] = 0



    sift1 = cv2.xfeatures2d.SIFT_create(sigma = 2)
    sift2 = cv2.xfeatures2d.SIFT_create( sigma = 1.6)

    kp1b, des1b = sift1.detectAndCompute(img1blue,None)
    kp1g, des1g = sift1.detectAndCompute(img1green,None)
    kp1r, des1r = sift1.detectAndCompute(img1red,None)
    kp2b, des2b = sift2.detectAndCompute(img2blue,None)
    kp2g, des2g = sift2.detectAndCompute(img2green,None)
    kp2r, des2r = sift2.detectAndCompute(img2red,None)
```

```
des1 = [des1b, des1g, des1r]
des2 = [des2b, des2g, des2r]

matchB = []
matchG = []
matchR = []

if (des1[0] is not None and des2[0] is not None):
    distB = cdist(des1[0], des2[0], 'euclidean')
    matchB = matching(distB, kp1b, kp2b)
if (des1[1] is not None and des2[2] is not None):
    distG = cdist(des1[1], des2[2], 'euclidean')
    matchG = matching(distG, kp1g, kp2g)
if (des1[2] is not None and des2[2] is not None):
    distR = cdist(des1[2], des2[2], 'euclidean')
    matchR = matching(distR, kp1r, kp2r)

matches = (matchB + matchG) + matchR
matches.sort(key=lambda tup: tup[2].distance)
k = 5
best = matches[:k]
M = np.zeros((2*k, 6))
M2 = np.zeros((2*k,1))
A = np.zeros((2*k,1))
j = 0
for i in range(len(best)):
    x = best[i][0].pt[0]
    y = best[i][0].pt[1]
    x2 = best[i][1].pt[0]
    y2 = best[i][1].pt[1]
    M[j,0] = x
    M[j,1] = y
    M[j+1,2] = x
    M[j+1,3] = y
    M[j,4] = 1
    M[j+1,5] = 1
    M2[j] = x2
    M2[j+1] = y2
    j +=2

np.set_printoptions(suppress=True)
A = np.dot(np.dot(inv(np.dot( np.transpose(M) , M) ), np.transpose(M)) , M2)
```



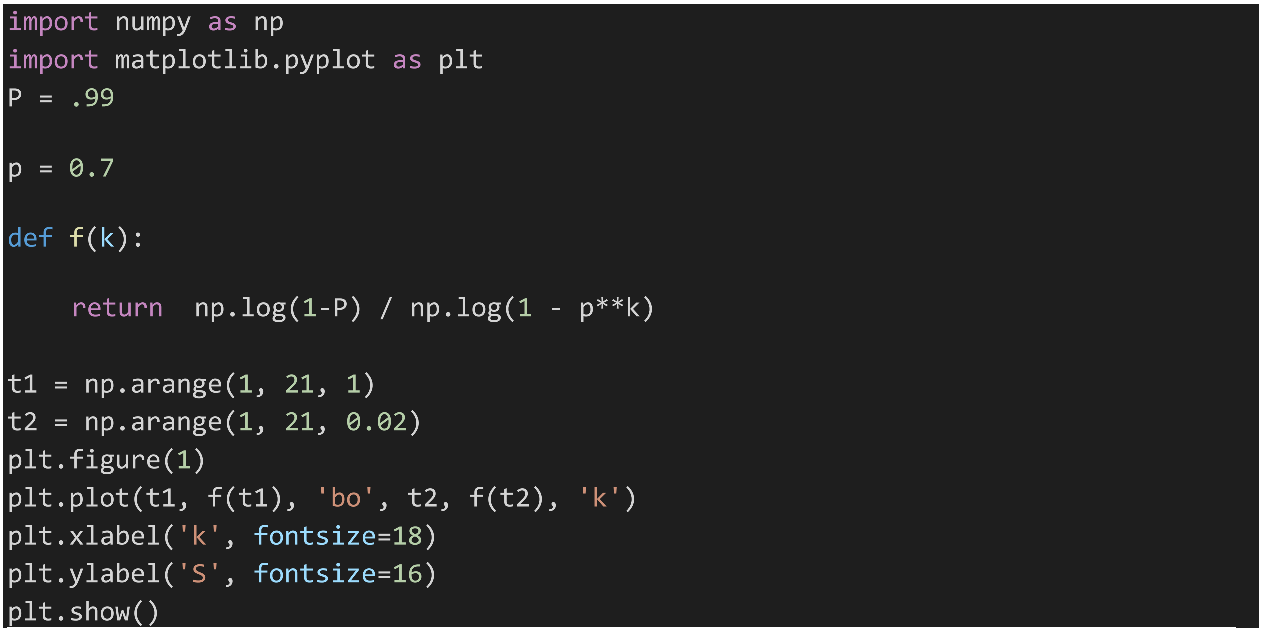
```
rows,cols,ch = img1.shape

P = np.array([[1,1,0,0,1,0],
     [0,0,1,1,0,1],
     [cols,1,0,0,1,0],
     [0,0,cols,1,0,1],
     [1,rows,0,0,1,0],
     [0,0,1,rows,0,1],
     [cols,rows,0,0,1,0],
     [0,0,cols,rows,0,1]])

P = np.dot(P, A)

plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
plt.plot((P[0,0], P[2,0]), (P[1,0], P[3,0]), 'ro-')
plt.plot((P[0,0], P[4,0]), (P[1,0], P[5,0]), 'ro-')
plt.plot((P[6,0], P[4,0]), (P[7,0], P[5,0]), 'ro-')
plt.plot((P[6,0], P[2,0]), (P[7,0], P[3,0]), 'ro-')
plt.show()
```

colourSearch

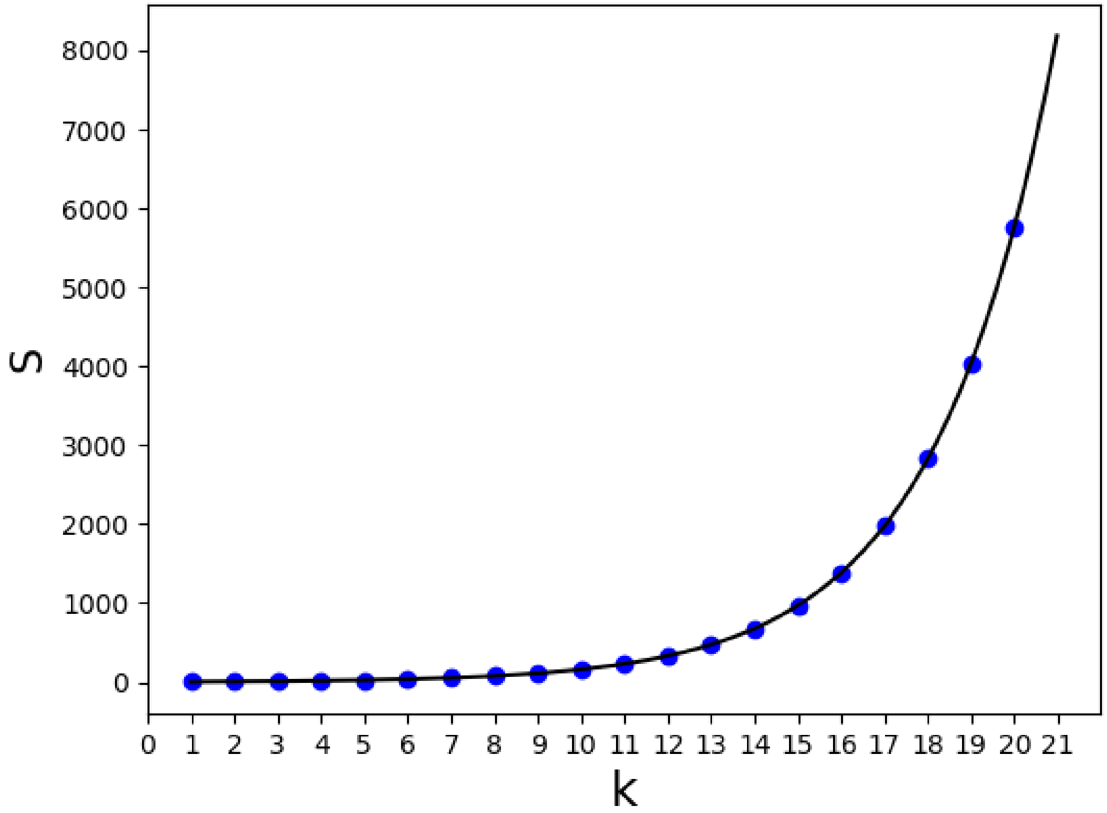


Question 3 RANSAC

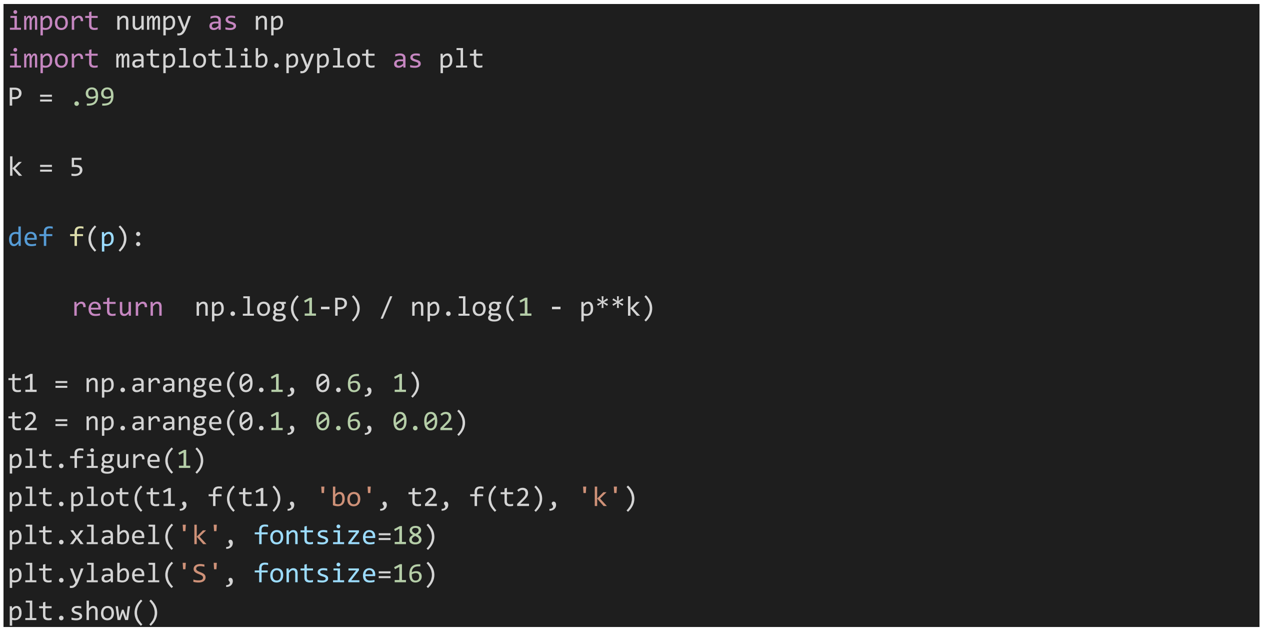
(a):



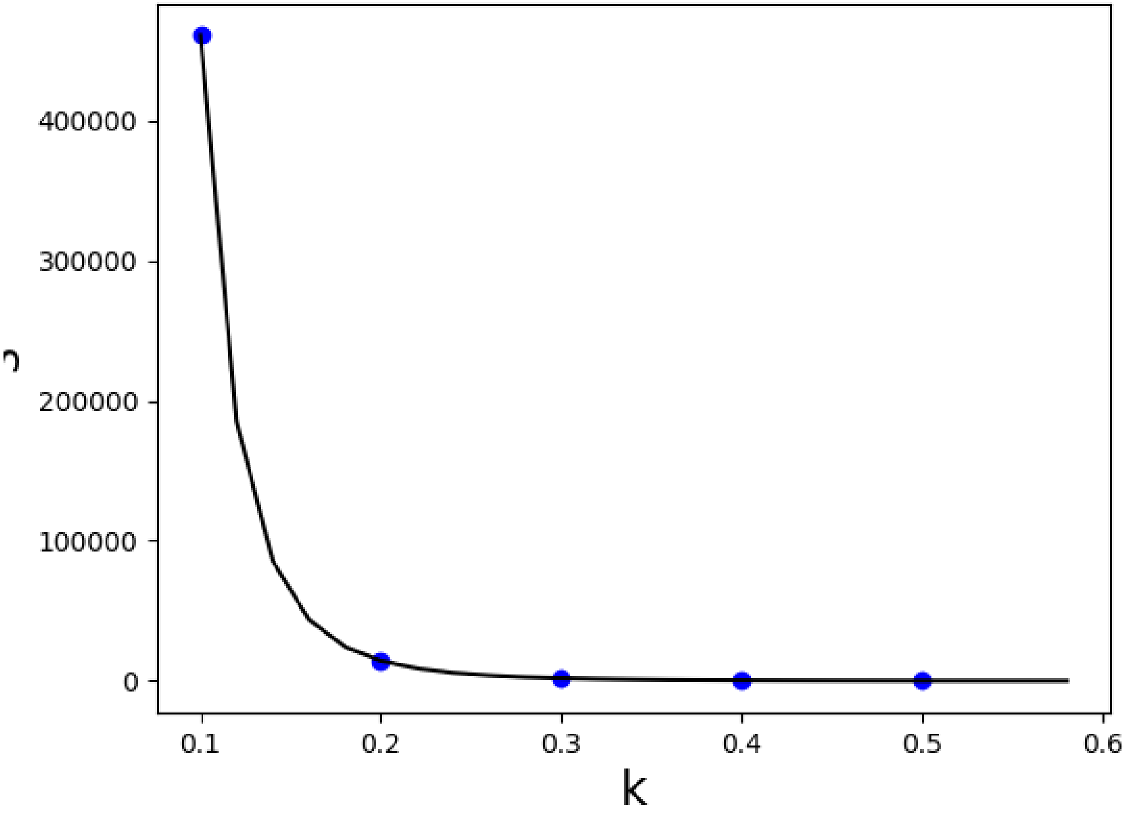
```
import numpy as np
import matplotlib.pyplot as plt
P = .99

p = 0.7

def f(k):

    return  np.log(1-P) / np.log(1 - p**k)

t1 = np.arange(1, 21, 1)
t2 = np.arange(1, 21, 0.02)
plt.figure(1)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.xlabel('k', fontsize=18)
plt.ylabel('S', fontsize=16)
plt.show()
```



(b)



```
import numpy as np
import matplotlib.pyplot as plt
P = .99

k = 5

def f(p):

    return  np.log(1-P) / np.log(1 - p**k)

t1 = np.arange(0.1, 0.6, 1)
t2 = np.arange(0.1, 0.6, 0.02)
plt.figure(1)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.xlabel('k', fontsize=18)
plt.ylabel('S', fontsize=16)
plt.show()
```



(c)

```
P = 0.99
p = 0.2
k = 5

S = np.log(1-P) / np.log(1 - p**k)
print "number of iteration need is: " , S

>> number of iteration need is:  14388.854123296185
```

The required number of iterations is 14389.

In iteration #15, the number still needs iterations, because required minimum iteration is 14389 in order to recover $P \geq 0.99$ chance.