นาย สัณหณัฐ พรมจรรย์ 63070501069

1) Source Code

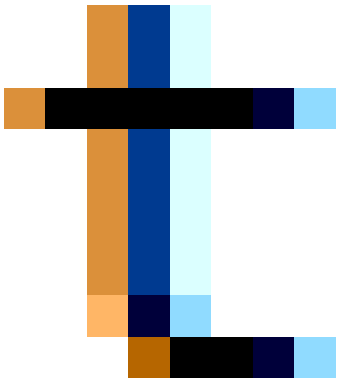
must upload image files: text_image and t_character

```
[11] import cv2
     import numpy as np
     from google.colab.patches import cv2_imshow
     def my_imshow(title, img ):
       cv2_imshow(img)
     def myImshow(title, img):
       cv2.startWindowThread()
       my_imshow(title, img)
       cv2.waitKey(0)
       cv2.destroyAllWindows()

     # Load the text image and the template image
     template = cv2.imread("t_character.png")
     text_image = cv2.imread("text_image.png")

     # Convert both images to grayscale
     text_image_gray = cv2.cvtColor(text_image, cv2.COLOR_BGR2GRAY)
     t_template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

     # Template matching using Euclidean norm
     result = cv2.matchTemplate(text_image, template, cv2.TM_SQDIFF_NORMED)

     def match_template(template, image):
        t_height, t_width = t_template_gray.shape
        i_height, i_width = text_image_gray.shape

        res = np.zeros((i_height - t_height + 1, i_width - t_width + 1))

     # Loop through different regions of the text image
        for i in range(i_height - t_height + 1):
           for j in range(i_width - t_width+1):
              img_region = text_image_gray[i:i+t_height, j:j+t_width]
              dist = np.sqrt(np.sum((img_region - t_template_gray)**2))
              res[i, j] = dist
        return res

     result = match_template(t_template_gray, text_image_gray)

     ''' Find the location of the best match
       min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
       top_left = min_loc '''
```

```
# Threshold for matching
threshold = 0.8

# Get the location of the matches
locations = np.where(result <= threshold)

# Draw a bounding box around the found 't'
for point in zip(*locations[::-1]):
  tl = point
  bottom_right = (tl[0] + template.shape[1], tl[1] + template.shape[0])
  cv2.rectangle(text_image, tl, bottom_right, (255,255,0), 1)

# Show output image
cv2_imshow(text_image)

# Save the output image
cv2.imwrite('output.png',text_image)
```

Input

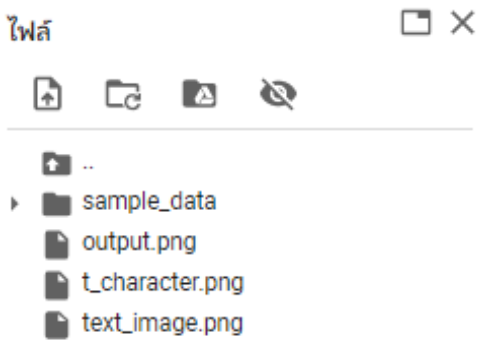
```
import numpy as np
import cv2 as cv2
import matplotlib.pyplot as plt


# In[9]:


def myImshow(title, img):
    """
    function to make windows display work in jupyter notebook
    - shows image in a separate window,
    - waits for any key to close the window.

    """

    cv2.startWindowThread()
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


# In[10]:


path = "D:/data/Dropbox/ML/"
#RGB images in BGR order in penCV
img1 = cv2.imread(path+'box.png',cv2.IMREAD_GRAYSCALE)          # queryImage
# Print error message if image is null
if img1 is None:
    print('Could not read query image')
else:
    print("Query Image read success...")

img2 = cv2.imread(path+'box_in_scene.png',cv2.IMREAD_GRAYSCALE) # targetImage
# Print error message if image is null
if img2 is None:
    print('Could not read training image')
else:
    print("Target Image read success...")


# In[11]:


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



t_character.png

text_image.png

Output



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


# In[9]:


def myImshow(title, img):
    """
    function to make windows display work in jupyter notebook
    - shows image in a separate window,
    - waits for any key to close the window.

    """

    cv2.startWindowThread()
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


# In[10]:


path = "D:/data/Dropbox/ML/"
#RGB images in BGR order in penCV
img1 = cv2.imread(path+'box.png',cv2.IMREAD_GRAYSCALE)          # queryImage
# Print error message if image is null
if img1 is None:
    print('Could not read query image')
else:
    print("Query Image read success...")

img2 = cv2.imread(path+'box_in_scene.png',cv2.IMREAD_GRAYSCALE) # targetImage
# Print error message if image is null
if img2 is None:
    print('Could not read training image')
else:
    print("Target Image read success...")


# In[11]:


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

2.1) Source Code

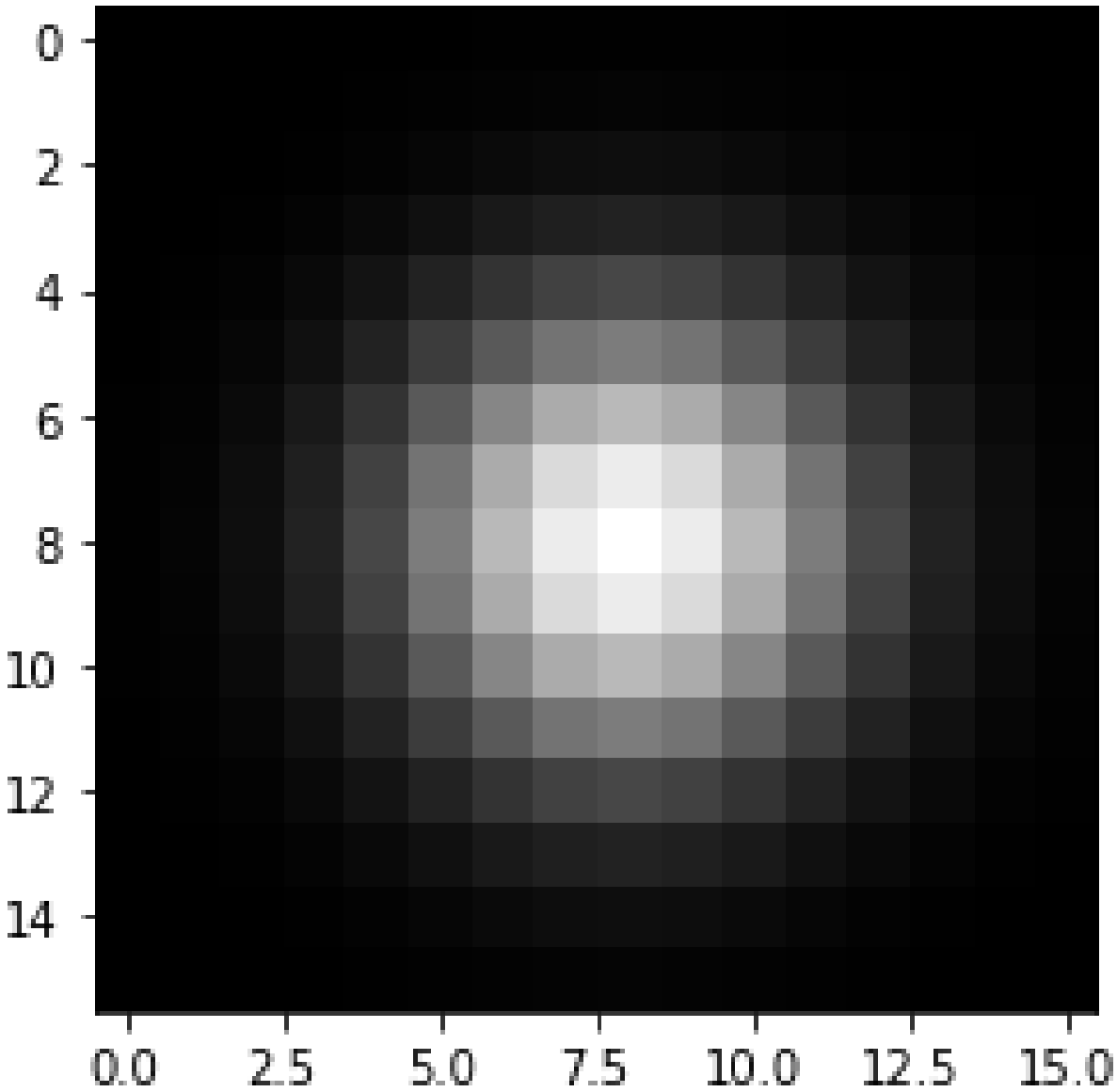
```
import numpy as np
import matplotlib.pyplot as plt

def gaussian_kernel(sigma, size):
    # Calculate width from sigma
    width = int(np.ceil(6 * sigma))
    # Ensure the size is an odd number
    size = 2 * (width // 2) + 1
    # Generate mesh grid
    x, y = np.meshgrid(np.arange(-width//2, width//2+1), np.arange(-width//2, width//2+1))
    # Calculate Gaussian function
    kernel = np.exp(-(x**2 + y**2) / (2 * sigma**2)) / (2 * np.pi * sigma**2)
    # Normalize the kernel
    kernel = kernel / np.sum(kernel)
    return kernel

# Define sigma and size
sigma = 2.5
size = 15

# Generate Gaussian kernel
kernel = gaussian_kernel(sigma, size)

# Show kernel
plt.imshow(kernel, cmap='gray')
plt.title('Karnel ( 15X15 )')
plt.show()
```

Input

matrix for Gaussian kernel with **σ** = 2.5 using kernel size of 15 x 15

(We use width = ceiling (6***σ**))

Output



2.2) Source Code

must upload image file: Lenna.png

```
[1] import cv2
    import numpy as np
```

```
[2] #this was created because Google Colab does not allow cv2.imshow, so must patch by cv2_imshow.
    #If we switch over to regular jupyter notebook not on Colab, we can change c2_imshow to cv2.imshow.
    from google.colab.patches import cv2_imshow #only used when running in Google Colab
    def my_imshow(title, img ):
      cv2_imshow(img) #should be changed to c2.imshow when not in Colab
```

```
[3] def gaussian_kernel(sigma, size):
        # Calculate width from sigma
        width = int(np.ceil(6 * sigma))
        # Ensure the size is an odd number
        size = 2 * (width // 2) + 1
        # Generate mesh grid
        x, y = np.meshgrid(np.arange(-width//2, width//2+1), np.arange(-width//2, width//2+1))
        # Calculate Gaussian function
        kernel = np.exp(-(x**2 + y**2) / (2 * sigma**2)) / (2 * np.pi * sigma**2)
        # Normalize the kernel
        kernel = kernel / np.sum(kernel)
        return kernel

    # Define sigma and size
    sigma = 2.5
    size = 15

    # Generate Gaussian kernel
    kernel = gaussian_kernel(sigma, size)
```

```
[4] path = ""
fileName = path + "Lenna.png" #+ "cat_dog_hug.jfif" #+ "Lenna.png" #  + "printed.jpg"  + "cat_dog_hug.jfif" + "Lenna.png"

#RGB images in BGR order in OpenCV
image = cv2.imread(fileName, cv2.IMREAD_COLOR)

# Print error message if image is null
if image is None:
    print('Could not read image')
else:
    print("Image file read success...")
```

```
Image file read success...
```

```
[5] def myImshow(title, img):
    """
    function to make windows display work in jupyter notebook
    - shows image in a separate window,
    - waits for any key to close the window.

    """
    cv2.startWindowThread()
    my_imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

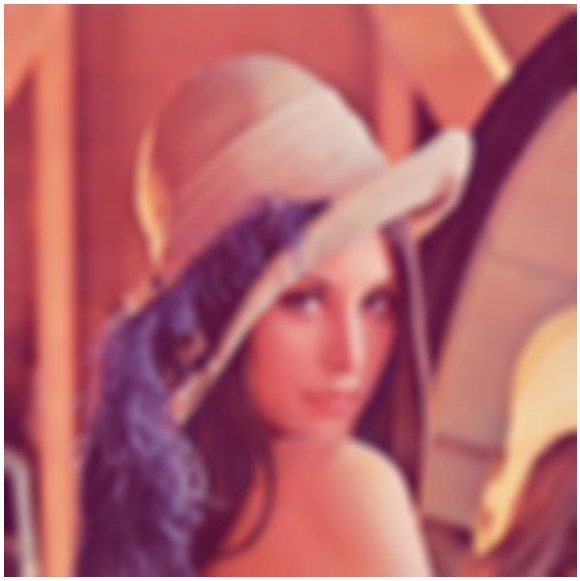
```
[6] #writes to file in current path
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)
cv2.imwrite(path+'Lenna.png', img)
```

```
True
```

```
[7] # Apply the convolution
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)
my_imshow('Original', image)
my_imshow('Kernel Blur', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input

Output



3) Source Code

```
[2]  import cv2 as cv
     import numpy as np
     import matplotlib.pyplot as plt
```

Create Training

```
[3]  # Feature set containing 25 * 2 for 25 (x,y) values of known/training data that are random integers 0-99
     trainData = np.random.randint(0,100,(25,2)).astype(np.float32)
     # Label each one either red, blue, or yellow with numbers 0, 1, or 2.
     # Response is a random integers 0-2 of 25 * 1 values
     responses = np.random.randint(0,3,(25,1)).astype(np.float32) #responses 25 x 1 matrix
```

```
[4]  print ("Training Data:\n", trainData)
     print("\n Responses:\n", responses)
     print("Responses Ravel or flattened as 1-D:\n", responses.ravel()) #Method .ravel flattens the np array to 1-D.
     print("Red Responses: \n ", responses.ravel()== 0) #color 0 is red, color 1 is blue, color 2 is yellow
```

```
Training Data:
 [[56. 74.]
 [32. 94.]
 [64. 82.]
 [72. 33.]
 [67. 46.]
 [73. 18.]
 [11. 23.]
 [59. 48.]
 [21. 34.]
 [97. 21.]
 [57. 13.]
 [16. 40.]
 [35. 62.]
 [97. 58.]
 [40. 21.]
 [40. 62.]
 [50. 12.]
 [23. 14.]
 [80. 16.]
 [31. 27.]
 [55.  0.]
 [81. 64.]
 [35. 20.]
 [67. 14.]
 [15. 63.]]
```

[4]

```
Responses:
[[2.]
[1.]
[1.]
[2.]
[1.]
[2.]
[2.]
[1.]
[2.]
[1.]
[1.]
[1.]
[1.]
[2.]
[2.]
[2.]
[0.]
[1.]
[0.]
[2.]
[1.]
[2.]
[0.]
[0.]
[1.]]
Responses Ravel or flattened as 1-D:
 [2. 1. 1. 2. 1. 2. 2. 1. 2. 1. 1. 1. 1. 2. 2. 2. 0. 1. 0. 2. 1. 2. 0. 0.
 1.]
Red Responses:
  [False False False False False False False False False False False False
 False False False False  True False  True False False False  True  True
 False]
```

[ ]

```
# Make red, blue, and yellow
red = trainData[responses.ravel()==0] #red is now trained data with responses of 0
print(red)
```

```
[[94. 38.]
 [59. 57.]
 [ 5. 83.]
 [11. 15.]
 [74. 65.]
 [83. 48.]
 [70. 48.]
 [98. 44.]]
```

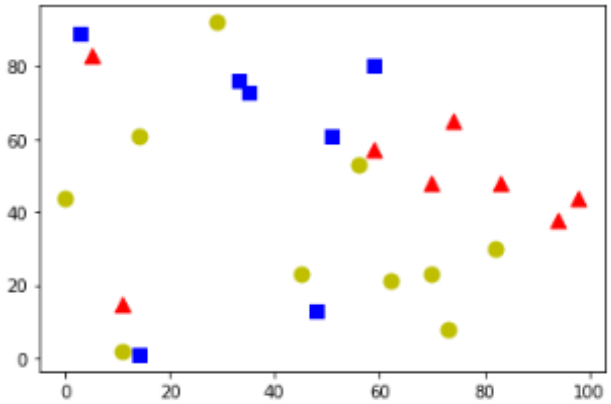
```
[ ]  blue = trainData[responses.ravel()==1] #blue is now the trained data with responses of 1
     print(blue)
```

```
[[ 3. 89.]
 [51. 61.]
 [14.  1.]
 [48. 13.]
 [35. 73.]
 [59. 80.]
 [33. 76.]]
```

```
[ ]  yellow = trainData[responses.ravel()==2] #yellow is now the trained data with responses of 2
     print(yellow)
```

```
[[62. 21.]
 [29. 92.]
 [73.  8.]
 [11.  2.]
 [14. 61.]
 [ 0. 44.]
 [82. 30.]
 [56. 53.]
 [45. 23.]
 [70. 23.]]
```

```
[ ]  # Plot the training data as scatter plots
     plt.scatter(red[:,0],red[:,1],80,'r','^') #size 80, red, triangle
     plt.scatter(blue[:,0],blue[:,1],80,'b','s') #size 80, blue, square
     plt.scatter(yellow[:,0],yellow[:,1],80,'y','o') #size 80, yellow, circle
     plt.show()
```

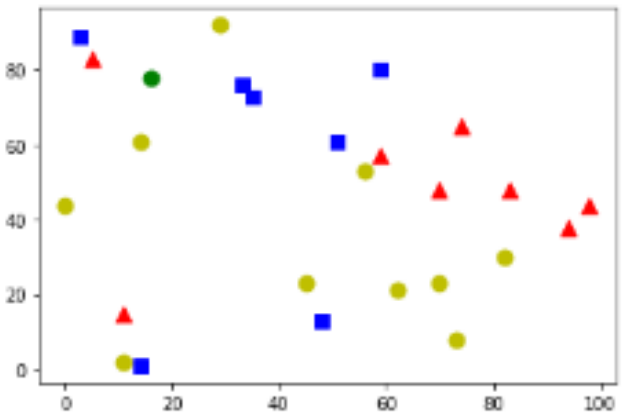


Test Unknown Data

```
[ ]  # Create a new sample with 1 * 2 or 1 (x,y) value with random integer 0-99
     newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)
     print(newcomer)
```

```
[[16. 78.]]
```

```
[ ]  # Plot the new sample with the training data
     plt.scatter(red[:,0],red[:,1],80,'r','^') #red, triangle
     plt.scatter(blue[:,0],blue[:,1],80,'b','s') #blue, square
     plt.scatter(yellow[:,0],yellow[:,1],80,'y','o') #yellow, circle
     plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o') #green, circle
     plt.show()
```



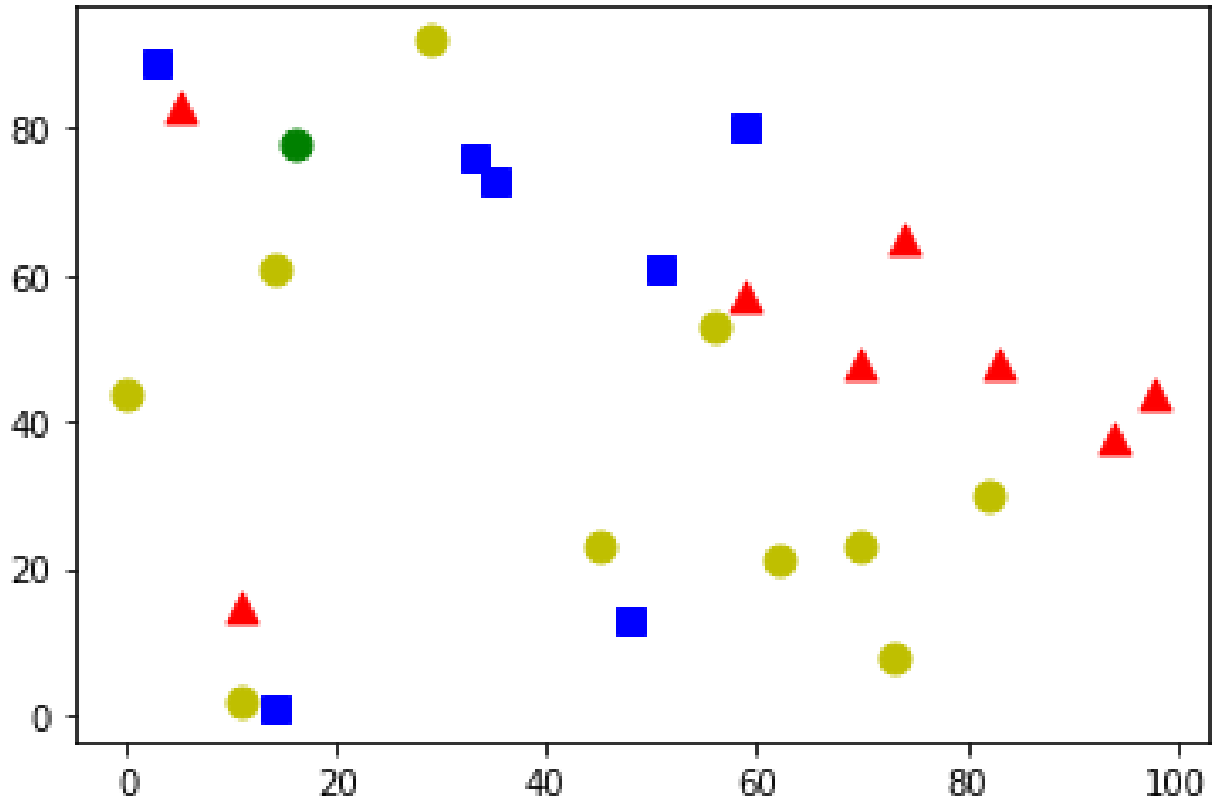

```
[ ]  # Create a list of color names
     colorName = np.array(['Red', 'Blue', 'Yellow'])
```

```
[ ]  # Train the KNN classifier
     knn = cv.ml.KNearest_create()
     knn.train(trainData, cv.ml.ROW_SAMPLE, responses)
     k = 4 # Set K=4
     ret, results, neighbors, dist = knn.findNearest(newcomer, k) # Classify the new sample
```

```
[ ]  # Get the color name for the result and neighbors
     resultColor = colorName[results[0].astype(int)]
     neighborColors = colorName[neighbors[0].astype(int)]
```

```
[ ]  print( "result color:  {}\n".format(resultColor) )
     print( "neighbors:  {}\n".format(neighbors) )
     print( "neighbor colors:  {}\n".format(neighborColors) )
     print( "neighbor distances:  {}\n".format(dist[0]) )
```

Input



Output

result color: ['Blue']

neighbors: [[0. 1. 2. 1.]]

neighbor colors: ['Red' 'Blue' 'Yellow' 'Blue']

neighbor distances: [146. 290. 293. 293.]

4) Source Code

```
[1]  import numpy as np
     import cv2 as cv2
     import matplotlib.pyplot as plt
```

```
[2]  #this was created because Google Colab does not allow my_imshow, so must patch by cv2_imshow.
     #If we switch over to regular jupyter notebook not on Colab, we can change c2_imshow to my_imshow.
     from google.colab.patches import cv2_imshow #only used when running in Google Colab
     def my_imshow(title, img ):
       print(title)
       cv2_imshow(img) #should be changed to c2.imshow(img, title) when not in Colab
```

```
     def myImshow(title, img):
         """
         function to make windows display work in jupyter notebook
         - shows image in a separate window,
         - waits for any key to close the window.

         """

         cv2.startWindowThread()
         my_imshow(title, img)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
```

```
[5]  path = ""
     #RGB images in BGR order in penCV
     img1 = cv2.imread(path+'book.jpg',cv2.IMREAD_GRAYSCALE)          # queryImage
     # Print error message if image is null
     if img1 is None:
         print('Could not read query image')
     else:
         print("Query Image read success...")

     img2 = cv2.imread(path+'book_in_scene.jpg',cv2.IMREAD_GRAYSCALE) # targetImage
     # Print error message if image is null
     if img2 is None:
         print('Could not read training image')
     else:
         print("Target Image read success...")
```

```
Query Image read success...
Target Image read success...
```

[6]
```
# Initiate SIFT detector
sift = cv2.SIFT_create()
# find the keypoints and descriptors with SIFT
#cv.KeyPoint(pt, size[, angle[, response[, octave[, class_id]]]])
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
```

[7]
```
print(kp1[0], '\n',des1[0], '\n Size = ', des1[0].size)
```

```
< cv2.KeyPoint 0x7fe66a503de0>
[ 2.   3.   3.   2.   0.   0.   0.   1. 127.   0.   0.   1.   0.   0.
  0.  11. 207.   7.   0.   0.   0.   0.   0.  11.  37.   3.   1.   1.
  1.   0.   0.   2.  10.   2.   0.   0.   0.   0.   0.   4. 159.   6.
  0.   0.   0.   0.   0.   8. 207.  10.   0.   0.   0.   0.   0.   4.
 50.   3.   2.   1.   3.   2.   1.   3.  11.   6.   0.   0.   0.   0.
  0.   2. 157.   9.   0.   0.   0.   0.   0.   2. 207.   4.   0.   0.
  0.   0.   0.  10.  53.   0.   0.   0.   7.   9.   1.   5.   5.   1.
  1.   0.   0.   1.   1.   5. 122.   5.   1.   0.   0.   0.   1.   4.
207.  12.   0.   0.   0.   0.   0.   2.  42.   6.   3.   5.   4.   3.
  0.   1.]
Size =  128
```

[8]
```
kp1[0].pt, kp1[0].size, kp1[0].angle
```

```
((115.01350402832031, 1297.4268798828125),
 5.271853923797607,
 359.9083251953125)
```

[9]
```
#Show SIFT keypoints kp1 as circles with major orientation direction overlaid onto img1
```

[10]
```
#Show SIFT keypoints kp2 as circles with major orientation direction overlaid onto img2
```

[11]
```
# BFMatcher with default params
bf = cv2.BFMatcher()
#if you use knnMatch, it will return a list of (the best) k matches instead of a single DMatch.
#in our example k=2, so will get a list of best 2 matches per feature point
matches = bf.knnMatch(des1,des2,k=2)
```

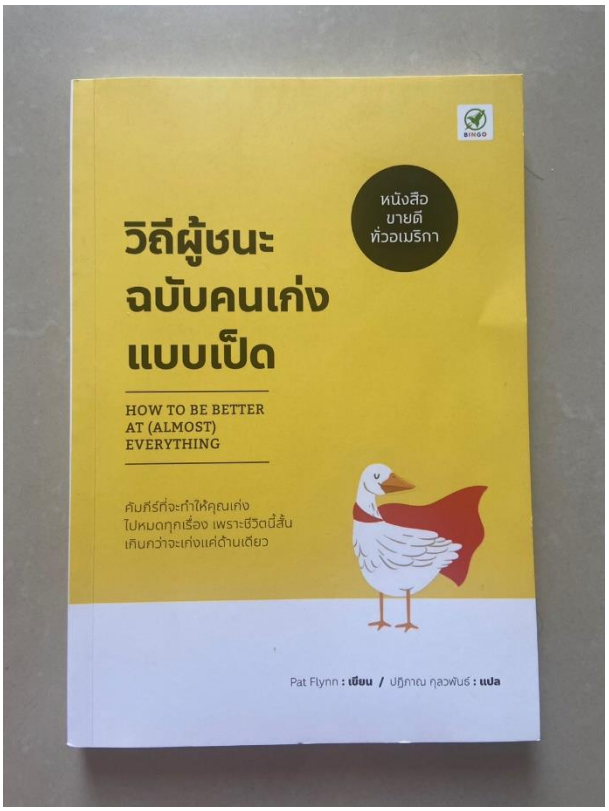
```
[12] print (matches)

((< cv2.DMatch 0x7fe659590750>, < cv2.DMatch 0x7fe659590770>), (< cv2.DMatch 0x7fe659590370>, < cv2.DMatch 0x7fe6595905b0>), (< cv2.DMatch 0x7fe659590590>, <
```

```
[13] # Apply ratio test
     good = []
     for m,n in matches:
         if m.distance < 0.75*n.distance:
             good.append([m])
```

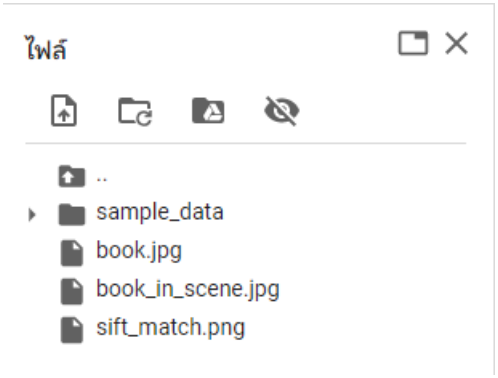
```
[14] # cv2.drawMatchesKnn expects list of lists as matches.
     #DrawMatchesFlags_DEFAULT
     #DrawMatchesFlags_DRAW_OVER_OUTIMG
     #DrawMatchesFlags_DRAW_RICH_KEYPOINTS
     #DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
     img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)
```

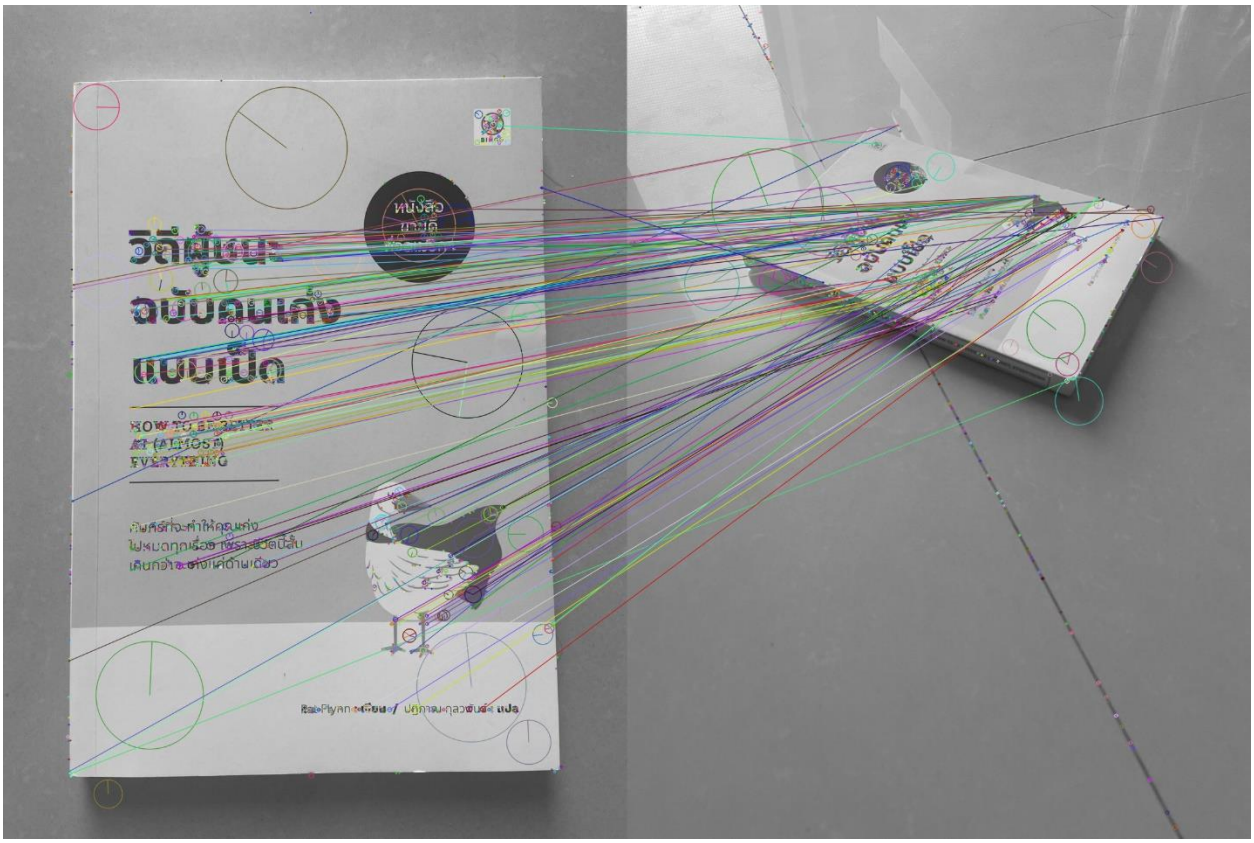
```
[15] # Show Image
     myImshow('Original directly', img3)

     # Writes to file
     cv2.imwrite(path+'sift_match.png', img3)
```

Input



Output


ไฟล์
..
sample_data
book.jpg
book_in_scene.jpg
sift_match.png


วิถีผู้ชนะ
ฉบับคนเคี้ยง
ผมเป็ด

AGREEMENT: MACHINE LEARNING COURSE HONOR CODE

Honor Code … the code we live by

King Mongkut's University of Technology Thonburi

This Machine Learning Course will consist of many assignments and projects which will determine a student's grade. All your work will be done at home. Thus, a student may be able to get someone else to do all of his/her work and yet get a good grade in this course. This would be quite unfair. This is an elective course and only those highly interested should take it. To discourage students from taking advantage of the "no-exam" nature of this course, we must follow a strict honor code for this course. The instructors will make the best attempt to check for students who violate this honor code to ensure we preserve the integrity of this course. Students may be called upon to explain their assignments and they must be able to do it confidently.

The Honor Code for this course states that:

1. You will work on all of an assignment by yourself, as if it were an in-class exam, but without a time limit. You may ask your friend to explain an algorithm or explain the problem, but he/she must not provide you the answer for any problem in an assignment. Simply stated, you must not work together with anyone else on your assignments.

2. You will not assist your friend on a project or assignment by providing your source code or writing any part of the program for him/her. Knowingly providing your work to a friend so s/he can copy it is considered dishonest. You may help your friend by explaining the algorithm so s/he understands it and learns.

I understand the above expected ethical conduct for this course. I also understand that the penalty for violating the honor code will be an immediate "F" grade. I also understand that the instructor may, on occasion, meet with me to audit my code to make sure I truly own it. *I understand that even if I were not caught for violating the above honor code, the other penalties include a reduction in self-respect, a decreased sense of accountability in our society, and a grave sense of dishonesty towards myself, my family, my class, my department, my university, my country, and mother earth as a whole.*

Signature:	สัณหณัฐ	Date:	31/ม.ค./2566
Name:	นาย สัณหณัฐ พรมจรรย์	ID.:	63070501069