Nomor 3A

Import Library

We use the opencv library (image manipulation) and matplotlib (plotting diagram)

In [20]:
```
import cv2
from matplotlib import pyplot as plt
import numpy as np
```

Read Images

In [21]:
```
image_one = cv2.imread("COMP7116_Computer Vision_REGULER & GLOBAL_UAS - Dataset/Dataset_P3/test1.jpeg")
image_two = cv2.imread("COMP7116_Computer Vision_REGULER & GLOBAL_UAS - Dataset/Dataset_P3/test2.jpeg")
collage_image = cv2.imread("COMP7116_Computer Vision_REGULER & GLOBAL_UAS - Dataset/Dataset_P3/collage.jpg")
```

Convert Images' Color

To make it easier to search or detect, the image must first be converted into a Greyscale format. If we use a colored image it will be less accurate

In [22]:
```
image_one_grey = cv2.cvtColor(image_one, cv2.COLOR_BGR2GRAY)
image_two_grey = cv2.cvtColor(image_two, cv2.COLOR_BGR2GRAY)
grey_collage_image = cv2.cvtColor(collage_image, cv2.COLOR_BGR2GRAY)
```

Create SIFT Feature Detector and Descriptor

We will use SIFT because it results better than SURF I already tried

In [23]:
```
sift = cv2.xfeatures2d.SIFT_create()
```

Get the Keypoint and Descriptor

In [24]:
```
kp1, des1 = sift.detectAndCompute(image_one_grey, None)
kp2, des2 = sift.detectAndCompute(image_two_grey, None)
kp_coll, des_coll = sift.detectAndCompute(grey_collage_image, None)
```

Create the Matcher

Now that we have all the features of our image, all we need to do now is create a matcher using FLANN (Fast Library for Approximate Nearest Neighbors)

In [25]:
```
index_param = dict(algorithm=0)
search_param = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_param, search_param)
```

Match the Image and Building Image Mask

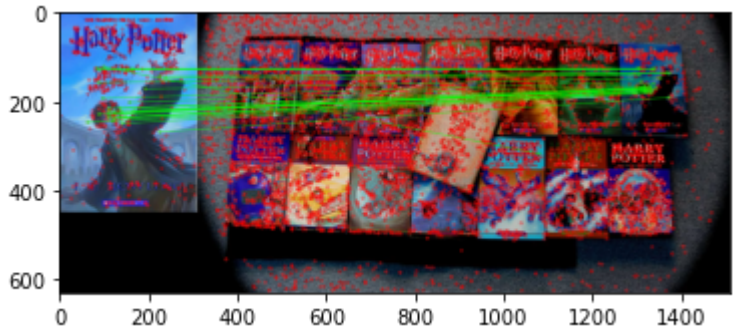
Match the target image with the image we are looking and create a mask based on the results obtained. We can change the value of the mask when the distance between the first best matches is less than the second best match distance multiplied by constant based on Lowe's paper.

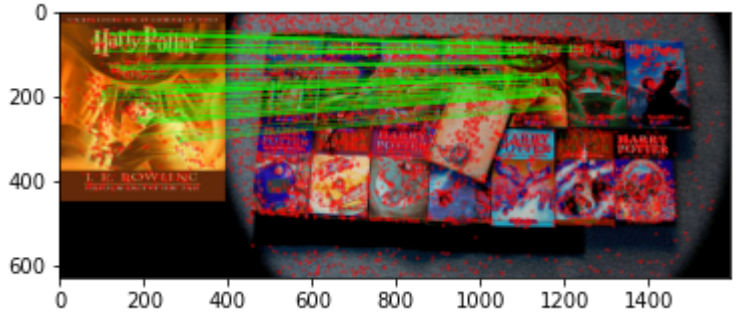
In [26]:
```
def matching(des_coll, des_test):
    matches = flann.knnMatch(des_test, des_coll, 2)

    matches_mask = []
    first_best_list = []
    for i in range(len(matches)):
        matches_mask.append([0, 0])

    for index, (first_best, second_best) in enumerate(matches):
        if (first_best.distance < 0.65 * second_best.distance):
            matches_mask[index] = [1, 0]
            first_best_list.append(first_best)

    return matches, matches_mask, first_best_list
```

In [27]:
```
matches1, matches_mask1 ,first_best_list1 = matching(des_coll, des1)
matches2, matches_mask2 ,first_best_list2 = matching(des_coll, des2)
```

Draw Matches and Show Result

In [28]:
```
result1 = cv2.drawMatchesKnn(image_one,
                             kp1,
                             collage_image,
                             kp_coll,
                             matches1, None,
                             matchColor=[0, 255, 0],
                             singlePointColor=[255, 0, 0],
                             matchesMask=matches_mask1)

result2 = cv2.drawMatchesKnn(image_two, kp2,
                             collage_image, kp_coll,
                             matches2, None,
                             matchColor=[0, 255, 0],
                             singlePointColor=[255, 0, 0],
                             matchesMask=matches_mask2)
plt.imshow(result1)
plt.show()
plt.imshow(result2)
plt.show()
```





Nomor 3B

Calculate Homography Matrix For Image 1 and 2

In [30]:
```
def findHomography(kp_test, kp_coll, first_best_list):
    image_1_points = np.float32([kp_test[m.queryIdx].pt for m in first_best_list]).reshape(-1, 1, 2)
    image_2_points = np.float32([kp_coll[m.trainIdx].pt for m in first_best_list]).reshape(-1, 1, 2)
    for i in range(0, len(first_best_list)):
        image_1_points[i] = kp_test[first_best_list[i].queryIdx].pt
        image_2_points[i] = kp_coll[first_best_list[i].trainIdx].pt

    homography, mask = cv2.findHomography(image_1_points, image_2_points, cv2.RANSAC, 5.0)
    return homography, mask

H1, mask1 = findHomography(kp1, kp_coll, first_best_list1)
H2, mask2 = findHomography(kp2, kp_coll, first_best_list2)
print("Image 1")
print(H1)
print("Image 2")
print(H2)
```

```
Image 1
[[ 5.15438123e-01 -1.21370290e-01  9.51099738e+02]
 [ 1.98625803e-02  4.21492370e-01  6.91716534e+01]
 [ 6.44629165e-05 -1.47421773e-04  1.00000000e+00]]
Image 2
[[ 7.00604514e-01 -4.41398525e-01  6.19479458e+02]
 [ 3.46281820e-02  4.53801111e-01  5.01371669e+01]
 [ 1.67048373e-04 -5.96960366e-04  1.00000000e+00]]
```

Nomor 3C

Draw the bounding box

Our result is not really good enough yet, maybe we can change the feature extractor for better result

In [32]:
```
h, w = grey_collage_image.shape
pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

dst1 = cv2.perspectiveTransform(pts, H1)
boundingBox1 = cv2.polylines(collage_image, [np.int32(dst1)], True, (0, 255, 0), 3)
plt.imshow(boundingBox1)
plt.show()
plt.cla()
dst2 = cv2.perspectiveTransform(pts, H2)
boundingBox2 = cv2.polylines(collage_image, [np.int32(dst2)], True, (0, 255, 0), 3)
plt.imshow(boundingBox2)
plt.show()
```