```
!pip install opencv-contrib-python
```

```
    Requirement already satisfied: opencv-contrib-python in /usr/local/lib/python3.10/dist-packages (4.8.0.76)
    Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-contrib-python) (1.25.2
```

```
import numpy as np
import cv2
import random
from scipy.ndimage import gaussian_filter
from matplotlib import pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

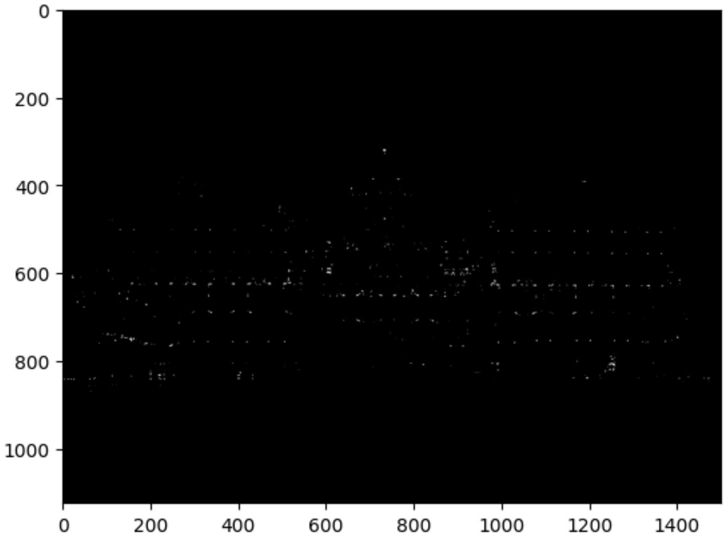
```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
def gradient(image):
    return cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5), cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
```

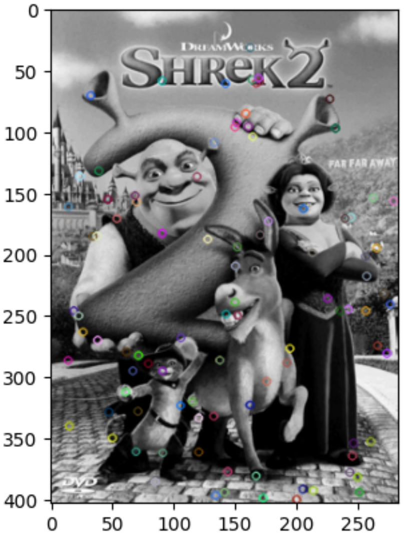
```
# Question 1a
def harris_keypoint(image):

    if len(image.shape) == 3:
        image = np.dot(image[..., :3], [0.2989, 0.5870, 0.1140])

    # Step 1
    image_x, image_y = gradient(image)

    # Step 2
    image_x_sq, image_y_sq, image_x_times_y = image_x * image_x, image_y * image_y, image_x * image_y

    # Step 3
    avg_gaussian_x_sq = gaussian_filter(image_x_sq, sigma=1)
    avg_gaussian_y_sq = gaussian_filter(image_y_sq, sigma=1)
    avg_gaussian_xy_sq = gaussian_filter(image_x_times_y, sigma=1)

    # Step 4
    determinants = avg_gaussian_x_sq * avg_gaussian_y_sq - avg_gaussian_xy_sq * avg_gaussian_xy_sq
    traces = avg_gaussian_x_sq + avg_gaussian_y_sq
    r_matrix = determinants - 0.05 * traces * traces

    # Step 5
    corners = np.zeros_like(r_matrix)
    # Setting threshold to 0.1
    corners[r_matrix > 0.1 * r_matrix.max()] = 255

    return corners
```

```
building = cv2.imread("/content/drive/MyDrive/building.jpg")
plt.imshow(harris_keypoint(building), cmap='gray')
```

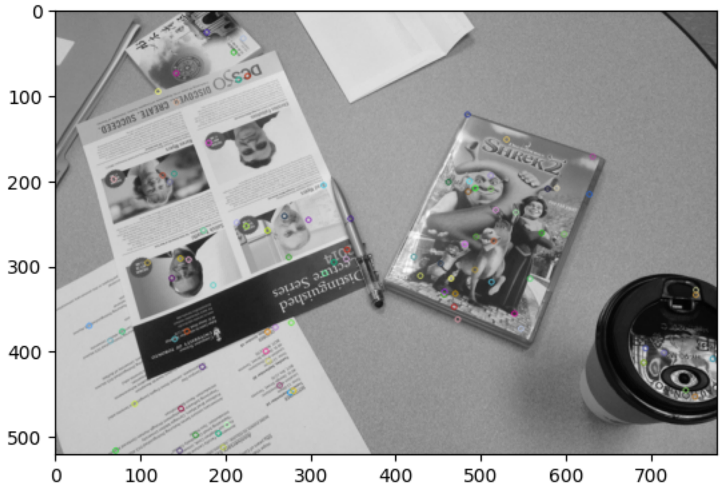
```
<matplotlib.image.AxesImage at 0x7dd4c780ead0>
```



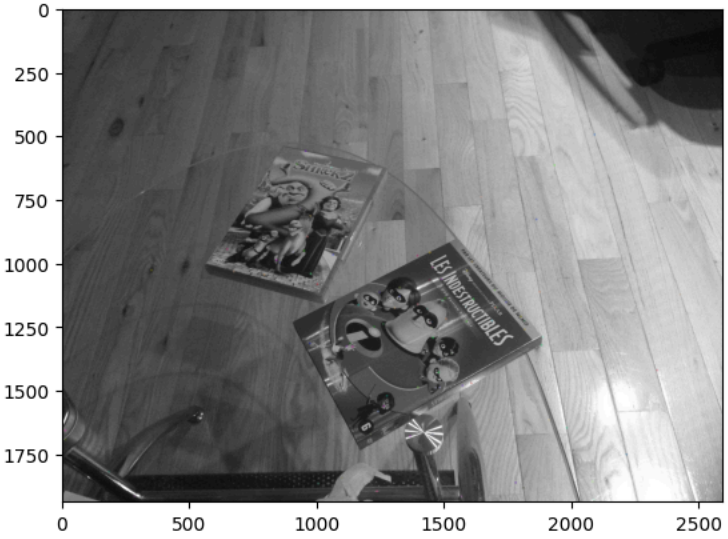
```
# Question 2a
reference = cv2.imread("/content/drive/MyDrive/reference.png")
gray_reference = cv2.cvtColor(reference, cv2.COLOR_BGR2GRAY)

test = cv2.imread("/content/drive/MyDrive/test.png")
gray_test = cv2.cvtColor(test, cv2.COLOR_BGR2GRAY)

test2 = cv2.imread("/content/drive/MyDrive/test2.png")
gray_test2 = cv2.cvtColor(test2, cv2.COLOR_BGR2GRAY)

sift = cv2.xfeatures2d.SIFT_create()

kps_reference, des_reference = sift.detectAndCompute(gray_reference, None)

# Function to
def visualize_100(kps, des, image):
    kps_reference1, des_reference1 = [], []
    for _ in range(100):
        i = random.randint(0, len(kps) - 1)
        kps_reference1.append(kps[i])
        des_reference1.append(des[i])

    img_kps = cv2.drawKeypoints(image, kps_reference1[:], outImage=None)
    plt.imshow(img_kps)

visualize_100(kps_reference, des_reference, gray_reference)
```



```
kps_test, des_test = sift.detectAndCompute(gray_test, None)

visualize_100(kps_test, des_test, gray_test)
```

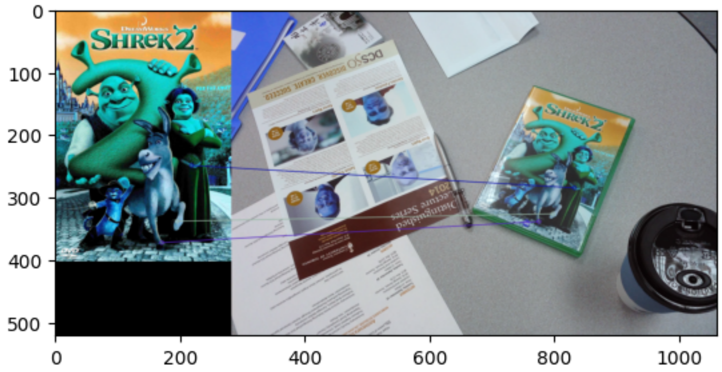


```
# THIS WORKS THE POINTS ARE SO SMALL
kps_test2, des_test2 = sift.detectAndCompute(gray_test2, None)

visualize_100(kps_test2, des_test2, gray_test2)
```



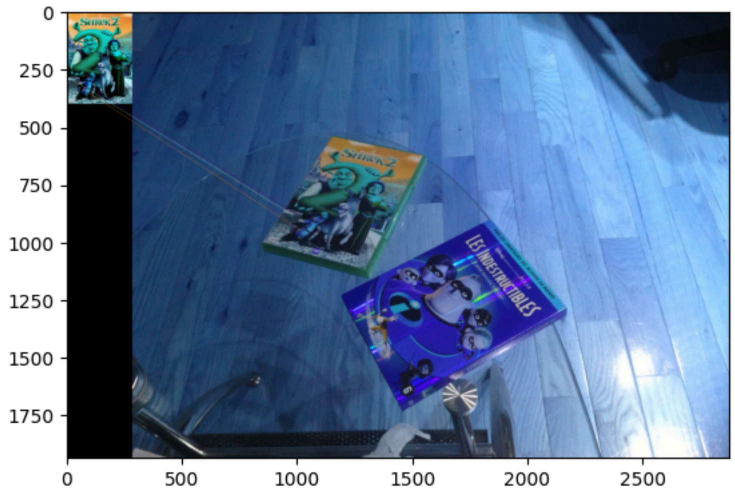
```
# Question 2b

# The best matches here have been defined by Euclidean Distance. Each descriptor
# in reference is compared to each descriptor in test. The best match for each
# descriptor is the one with the smallest Euclidean Distance. The top 3 are
# visualized by the same benchmark of Euclidean Distance

# The criteria used to determine the best matches is to be the smallest
# Euclidean Distance amongst all possible matches

def matching(threshold, descriptor_1, descriptor_2):
    matches = []
    for i, descriptor_ref in enumerate(descriptor_1):
        first_best_match_index, second_best_match_index = None, None
        first_best_match, second_best_match = float('inf'), float('inf')

        for j, descriptor_test in enumerate(descriptor_2):
            euclidean_distance = np.linalg.norm(descriptor_ref - descriptor_test)

            if euclidean_distance < first_best_match:
                second_best_match, second_best_match_index = first_best_match, first_best_match_index
                first_best_match, first_best_match_index = euclidean_distance, j
            elif euclidean_distance < second_best_match:
                second_best_match, second_best_match_index = euclidean_distance, j

        phi_ratio = first_best_match / second_best_match
        if phi_ratio < threshold:
            matches.append((first_best_match, i, first_best_match_index))

    matches.sort()
    return matches

top_matches_1 = matching(0.8, des_reference, des_test)
top_3_matches_1 = [cv2.DMatch(_queryIdx=i, _trainIdx=j, _distance=k) for k, i, j in top_matches_1[:3]]

matches_img = cv2.drawMatches(reference, kps_reference, test, kps_test, top_3_matches_1, reference, flags=2)
plt.imshow(matches_img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7dd4c9375060>
```



```
top_matches = matching(0.8, des_reference, des_test2)
top_3_matches = [cv2.DMatch(_queryIdx=i, _trainIdx=j, _distance=k) for k, i, j in top_matches[:3]]

matches_img = cv2.drawMatches(reference, kps_reference, test2, kps_test2, top_3_matches, reference, flags=2)

plt.imshow(matches_img)
```

```
<matplotlib.image.AxesImage at 0x7dd4c7b56860>
```

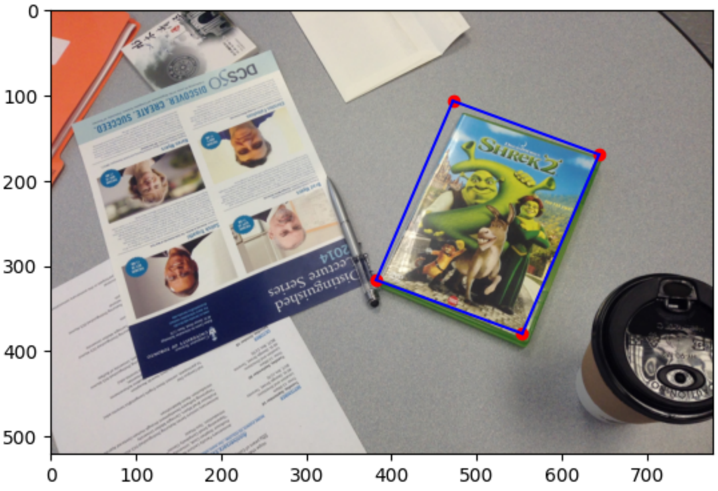


```
# Question 2c
p_1, p_1_prime = [], []

for _, i, j in top_matches_1[:3]:
    p_1.append(list(kps_reference[i].pt) + [0, 0, 1, 0])
    p_1.append([0, 0] + list(kps_reference[i].pt) + [0, 1])
    p_1_prime.extend(list(kps_test[j].pt))
    # p_1.append(list(kps_reference[i].pt))
    # p_1_prime.append(list(kps_test[j].pt))

p_2, p_2_prime = [], []
for _, i, j in top_matches[:3]:
    p_2.append(list(kps_reference[i].pt) + [0, 0, 1, 0])
    p_2.append([0, 0] + list(kps_reference[i].pt) + [0, 1])
    p_2_prime.extend(list(kps_test2[j].pt))
    # p_2.append(list(kps_reference[i].pt))
    # p_2_prime.append(list(kps_test2[j].pt))


p_1 = np.float32(p_1)
p_1_prime = np.float32(p_1_prime)

p_2 = np.float32(p_2)
p_2_prime = np.float32(p_2_prime)

# Answer for 2c
A_1 = np.linalg.inv(p_1) @ p_1_prime
A_2 = np.linalg.inv(p_2) @ p_2_prime

A_1 = np.array([[A_1[0], A_1[1], A_1[4]], [A_1[2], A_1[3], A_1[5]]])
A_2 = np.array([[A_2[0], A_2[1], A_2[4]], [A_2[2], A_2[3], A_2[5]]])


print("The Affine Transformation For test.png is:")
print(A_1)
print("=====================")
print("The Affine Transformation For test2.png is:")
print(A_2)
```

```
The Affine Transformation For test.png is:
[[ 6.0849571e-01 -2.2437000e-01  4.7229785e+02]
 [ 2.2278404e-01  5.2296352e-01  1.0603906e+02]]
=====================
The Affine Transformation For test2.png is:
[[ 1.4342003e+00 -7.6163673e-01  8.8210303e+02]
 [ 3.6244965e-01  1.2537918e+00  4.9581348e+02]]
```

```
corners = (np.float32([0, 0, 1]), np.float32([reference.shape[1] - 1, 0, 1]), np.float32([reference.shape[1] - 1, reference.shap

transformed_corners_A_1 = [list(A_1 @ corner) for corner in corners]
transformed_corners_A_2 = [list(A_2 @ corner) for corner in corners]

x_1 = [point[0] for point in transformed_corners_A_1]
y_1 = [point[1] for point in transformed_corners_A_1]

x_2 = [point[0] for point in transformed_corners_A_2]
y_2 = [point[1] for point in transformed_corners_A_2]

plt.plot(x_1, y_1, 'ro')
for i in range(len(transformed_corners_A_1)):
    plt.plot(
        [transformed_corners_A_1[i][0], transformed_corners_A_1[(i+1) % len(transformed_corners_A_1)][0]],
        [transformed_corners_A_1[i][1], transformed_corners_A_1[(i+1) % len(transformed_corners_A_1)][1]],
        'b-'
    )
plt.imshow(cv2.cvtColor(test, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7dd4c7b320b0>
```



```
plt.plot(x_2, y_2, 'ro')
for i in range(len(transformed_corners_A_2)):
    plt.plot(
        [transformed_corners_A_2[i][0], transformed_corners_A_2[(i+1) % len(transformed_corners_A_2)][0]],
        [transformed_corners_A_2[i][1], transformed_corners_A_2[(i+1) % len(transformed_corners_A_2)][1]],
        'b-'
    )
plt.imshow(cv2.cvtColor(test2, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7dd4c7b30820>
```