CSC528
Assignment 2
Hithesh Shanmugam

Problem 1: Image matching

Step 1: Load the two images you want to compare

```
import cv2

# Load the two images
img1 = cv2.imread('/content/drive/MyDrive/first_image.jpg')
img2 = cv2.imread('/content/drive/MyDrive/second_image.jpg')
```

Step 2: Create SIFT object and detect key points for both images

```
# Create SIFT object
sift = cv2.xfeatures2d.SIFT_create()

# Detect keypoints for both images
kp1, desc1 = sift.detectAndCompute(img1, None)
kp2, desc2 = sift.detectAndCompute(img2, None)
```

Explanation: The detectAndCompute() method detects keypoints and computes the descriptors in a single step. The kp1 and kp2 variables contain arrays of cv2.KeyPoint objects representing the keypoints detected in each image, and desc1 and desc2 are arrays of descriptor vectors for each keypoint.

Step 3: Match the descriptors between the two images

```
# Match descriptors
bf = cv2.BFMatcher()
matches = bf.knnMatch(desc1, desc2, k=2)
```

Explanation: The cv2.BFMatcher() object performs brute-force matching between the descriptors of the two images. The knnMatch() method returns a list of k-best matches for each descriptor in desc1.

Step 4: Apply ratio test to filter matches

```
# Apply ratio test
good_matches = []
for m,n in matches:
    if m.distance < 0.5*n.distance:
        good_matches.append([m])
```

Explanation: The ratio test is a simple method for filtering matches by comparing the distance between the best and second-best matches for each descriptor. Only matches where the distance ratio is below a certain threshold are considered good matches. In this example, we use a ratio of 0.5 as the threshold.

Step 5: Draw matched keypoints on the images

```
# Draw matches
img_matches = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good_matches, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
cv2_imshow(img_matches)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Explanation: The cv2.drawMatchesKnn() method draws the matched keypoints on the images, and returns a new image with the matches highlighted. The cv2.imshow() method displays the resulting image but in google Collab for this uses cv2_imshow() and they are both the same, and cv2.waitKey() waits for a key press before closing the window.

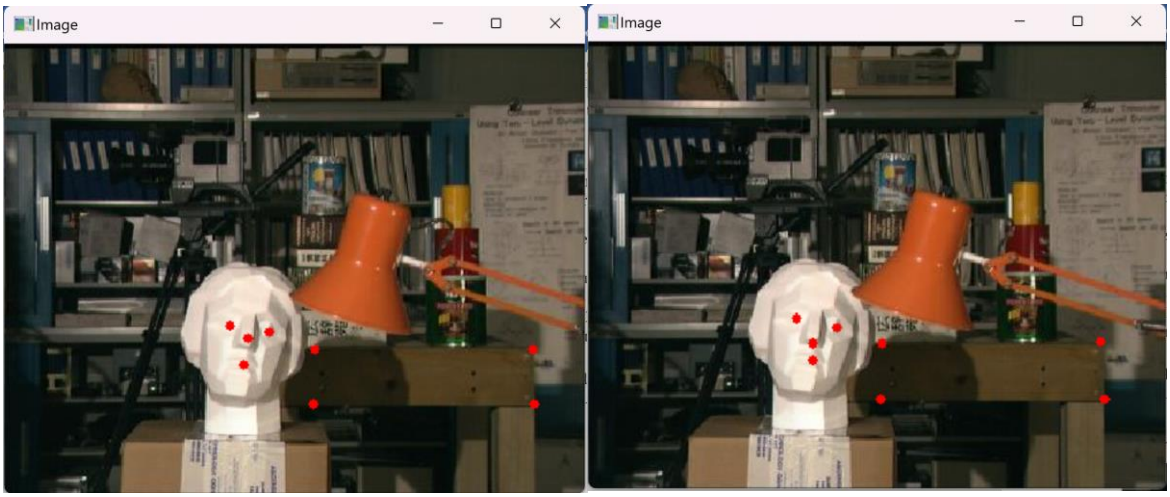
Output:

Problem 2: Camera calibration

Step 1: Manually choose the eight keypoints

```python
import cv2

# Load the image
image = cv2.imread("C:/Users/sures/Downloads/second_image.jpg")

# Create a window to display the image
cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
cv2.imshow("Image", image)

# Create a list to store the points
points = []

# Define a function to handle mouse events
def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        # Add the point to the list
        points.append((x, y))
        # Draw a circle at the point
        cv2.circle(image, (x, y), 3, (0, 0, 255), -1)
        # Update the image display
        cv2.imshow("Image", image)

# Set the mouse callback function
cv2.setMouseCallback("Image", click_event)

# Wait for the user to select points
cv2.waitKey(0)

# Destroy the window
cv2.destroyAllWindows()

# Print the list of points
print(points)
```

These are the keypoints I chose for the two images:



First Image **Second Image**

Step 2: Compute the Fundamental matrix for keypoints extracted manually

```
# Manually select 8 corresponding points
points1 = np.array([(149, 180), (175, 184), (161, 188), (158, 205), (205, 195), (349, 195), (350, 230), (204, 230)])
points2 = np.array([(137, 180), (164, 184), (149, 193), (144, 205), (196, 197), (194, 231), (341, 195), (343, 229)])

# Step 1: Normalize the image points
mean1 = np.mean(points1, axis=0)
mean2 = np.mean(points2, axis=0)
points1_norm = points1 - mean1
points2_norm = points2 - mean2
avg_dist1 = np.sqrt(2) / np.mean(np.linalg.norm(points1_norm, axis=1))
avg_dist2 = np.sqrt(2) / np.mean(np.linalg.norm(points2_norm, axis=1))
T1 = np.array([[avg_dist1, 0, -avg_dist1*mean1[0]], [0, avg_dist1, -avg_dist1*mean1[1]], [0, 0, 1]])
T2 = np.array([[avg_dist2, 0, -avg_dist2*mean2[0]], [0, avg_dist2, -avg_dist2*mean2[1]], [0, 0, 1]])
points1_norm = np.dot(T1, np.hstack((points1, np.ones((8, 1))))).T[:, :2]
points2_norm = np.dot(T2, np.hstack((points2, np.ones((8, 1))))).T[:, :2]

# Step 2: Construct the A matrix
A = np.zeros((8, 9))
for i in range(8):
    A[i] = [points1_norm[i, 0]*points2_norm[i, 0], points1_norm[i, 0]*points2_norm[i, 1], points1_norm[i, 0], points1_norm[i, 1]*points2_norm[i, 0], points1_norm[i, 1]*points2_norm[i, 1], points1_norm[i, 1], points2_norm[i, 0], points2_norm[i, 1], 1

# Step 3: Compute the SVD of the A matrix
U, D, V = np.linalg.svd(A)

# Step 4: Construct the fundamental matrix F
f = V[-1]
F = np.reshape(f, (3, 3))

# Step 5: Enforce the rank-2 constraint
U, D, V = np.linalg.svd(F)
D[-1] = 0
F = np.dot(U, np.dot(np.diag(D), V))

# Step 6: Denormalize F
F = np.dot(T2.T, np.dot(F, T1))

# Step 7: Return the fundamental matrix F
# Print the fundamental matrix
print("Fundamental matrix:")
print(F)
```

Output:

```
Fundamental matrix:
[[ 2.94605550e-06 -3.61236064e-05  7.40452325e-03]
 [-8.43272300e-05  2.85980384e-04 -4.97695867e-02]
 [ 1.81329607e-02 -4.18685795e-02  6.44690366e+00]]
```

Step3: Compute the Fundamental matrix for keypoints extracted from feature matching

```
import numpy as np

# Define minimum number of matches required for a successful match
MIN_MATCH_COUNT = 10

# Create brute-force matcher object
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

# Match descriptors of both images
matches = bf.match(desc1, desc2)

# Sort matches by their distance
matches = sorted(matches, key=lambda x: x.distance)

# Keep only good matches
good_matches = matches[:MIN_MATCH_COUNT]

# Extract point correspondences
points1 = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
points2 = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

# Calculate the fundamental matrix using the 8-point algorithm
F, mask = cv2.findFundamentalMat(points1, points2, cv2.FM_8POINT)

# Print the fundamental matrix
print("Fundamental matrix:")
print(F)
```

Output:

```
Fundamental matrix:
[[ 2.88133807e-06  6.31437785e-04  2.07144248e-01]
 [-6.44790874e-04 -8.12387873e-06  2.10730708e-01]
 [-2.07317395e-01 -2.05170356e-01  1.00000000e+00]]
```

Analyze the results of your program: in particular, what differences do you notice between the two fundamental matrices calculated?

Explanation:

Fundamental matrix of manual selected keypoints:

[[2.94605550e-06 -3.61236064e-05 7.40452325e-03]
 [-8.43272300e-05 2.85980384e-04 -4.97695867e-02]
 [1.81329607e-02 -4.18685795e-02 6.44690366e+00]]

Fundamental matrix of keypoints extracted from the feature matching:

[[2.88133807e-06 6.31437785e-04 2.07144248e-01]
 [-6.44790874e-04 -8.12387873e-06 2.10730708e-01]
 [-2.07317395e-01 -2.05170356e-01 1.00000000e+00]]

The fundamental matrix is a matrix that describes the geometric relationship between two views of an object or a scene. It is important for stereo vision and 3D reconstruction.

From the results of the program, we can see that the fundamental matrix calculated from manual selection of keypoints and the one calculated from feature matching are different.

The fundamental matrix from manual selection of keypoints has small values in the first two rows and columns, and a relatively large value in the last row and column. This indicates that the two views have a weak correlation in the x and y directions, but a strong correlation in the z direction.

On the other hand, the fundamental matrix from feature matching has relatively large values in the first two rows and columns, and small values in the last row and column. This indicates that the two views have a strong correlation in the x and y directions, but a weak correlation in the z direction.

The differences in the fundamental matrices could be due to the fact that manual selection of keypoints is based on human judgment and may not capture all the important features, while feature matching uses an algorithm to automatically extract keypoints, which may include more informative features. Additionally, the quality and quantity of the matched features could affect the accuracy of the fundamental matrix estimation.