ECE 8410: Computer Vision

Assignment 3

March 4, 2021

Mark Belbin (201618477)
Raylene Mitchell (201415247)
Rodney Winsor (200433886)

Code Explanation

For this assignment, our group decided to use python and the opencv library. The first part of our python script imported the two images, im1 and im2, and then converted them to grayscale. As well, a feature that was added later essentially padded each image with zeros so that the full panoramic image could be seen. (Initially the panoramic extended beyond the edges of the image and was cropped). The imported images, converted to grayscale and shifted, are shown in Figure 1, and the code segment is shown in Figure 2.

Figure 1: Grayscale im1 (left) and im2 (right). The images are padded with zeros.

```
import cv2
import numpy as np

# Read images and convert to grayscale
im1 = cv2.imread('im1.jpg')
im2 = cv2.imread('im2.jpg')

# Shift images towards the center so after transform all of the pano is
shown
T = np.float32([[1, 0, im2.shape[1]/2],[0, 1, im2.shape[0]/2]])
im2 = cv2.warpAffine(im2, T,  (im1.shape[1]*2,  im1.shape[0]*2))
im1 = cv2.warpAffine(im1, T,  (im1.shape[1]*2,  im1.shape[0]*2))

im1_gray = cv2.cvtColor(im1,cv2.COLOR_BGR2GRAY)
im2_gray = cv2.cvtColor(im2,cv2.COLOR_BGR2GRAY)
```

Figure 2: Code for importing the images and padding zeros.

Next, keypoints and feature descriptors were calculated for each grayscale image. This was done using the BRISK feature detector, which is a similar method to SIFT or SURF. However, recently SIFT and SURF detectors have been removed from opencv due to copyright issues.

Another descriptor method called ORB was also tried but it proved to produce inconsistent results compared to BRISK.

The code for the keypoint and feature descriptors is shown in Figure 3, and the resulting keypoints and features are plotted on the original grayscale images in Figures 4 and 5.

```
# Find features using BRISK.
# SURF/SIFT are not included in opencv pypi due to a patent, BRISK worked
better than ORB
brisk = cv2.BRISK_create()
keypoints1, descriptors1 = brisk.detectAndCompute(im1,None)
keypoints2, descriptors2 = brisk.detectAndCompute(im2,None)
```

Figure 3: Code for importing the images and padding zeros.

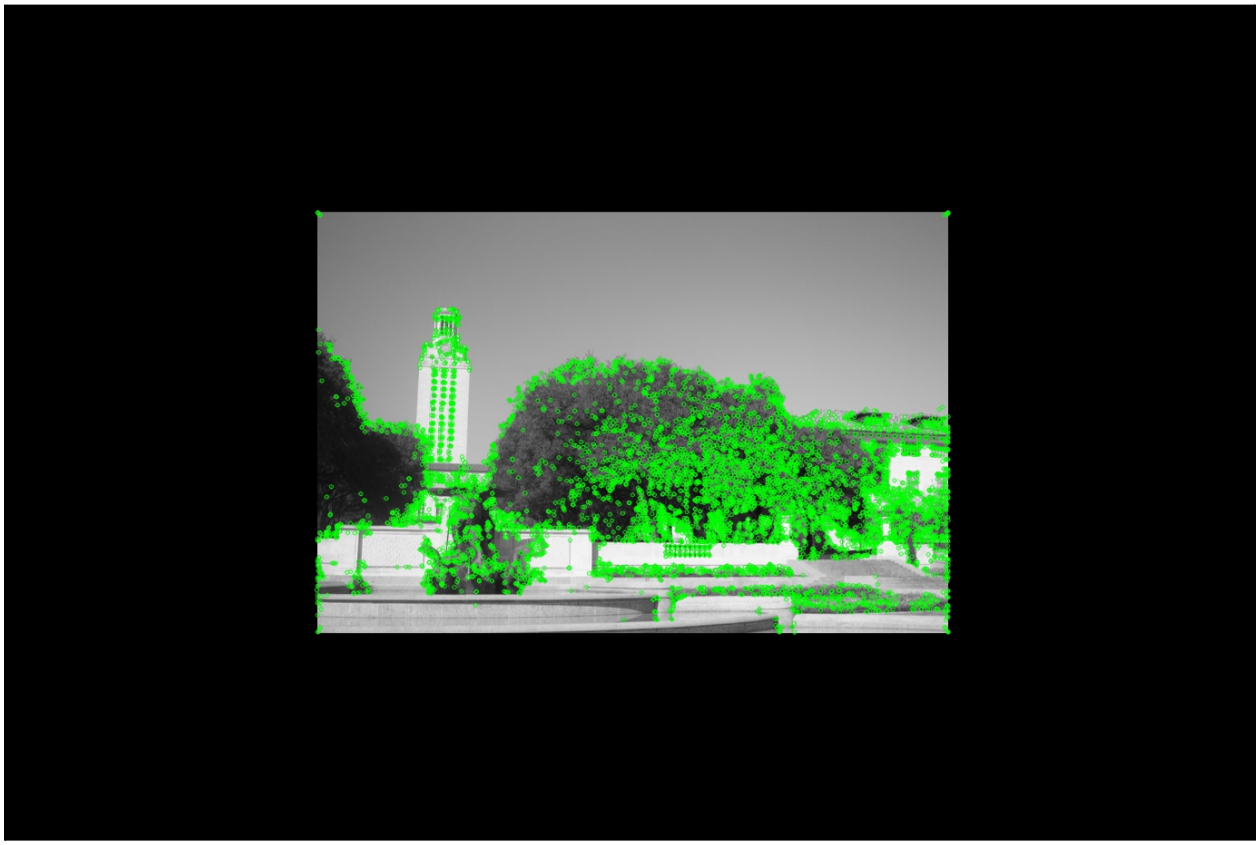

Figure 4: Key points for im1 using BRISK.

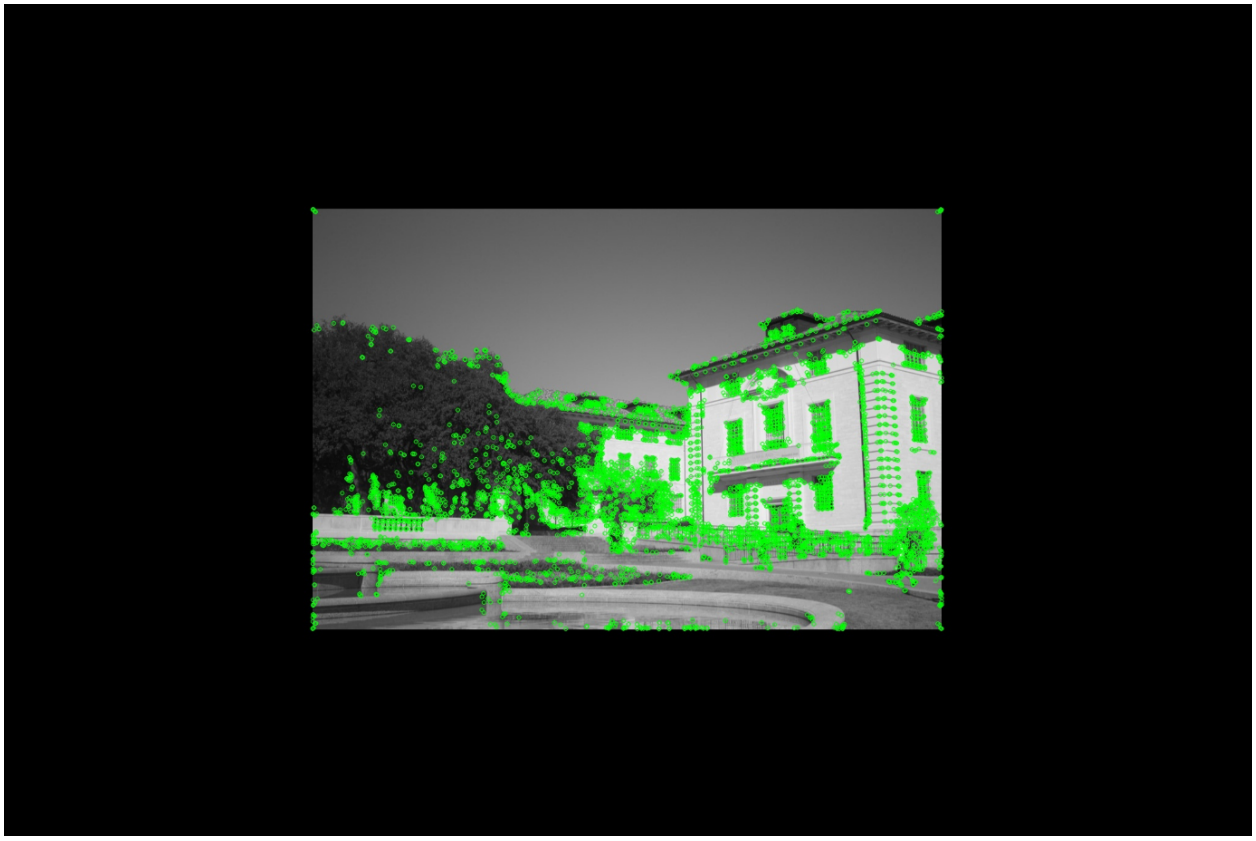

Figure 5: Key points for im2 using BRISK.

Next, key points were matched based on their feature descriptors. This was accomplished using a brute-force method provided by opencv. The final matched key points were then stored in the ‘matches’ structure after being sorted according to the best match. This code segment is shown in Figure 6, and the best 75 matches are shown across both images in Figure 7.

```
# Brute-force match the descriptors
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(descriptors1, descriptors2)
matches = sorted(matches, key = lambda x:x.distance)
```

Figure 6: Code segment for key point matching.

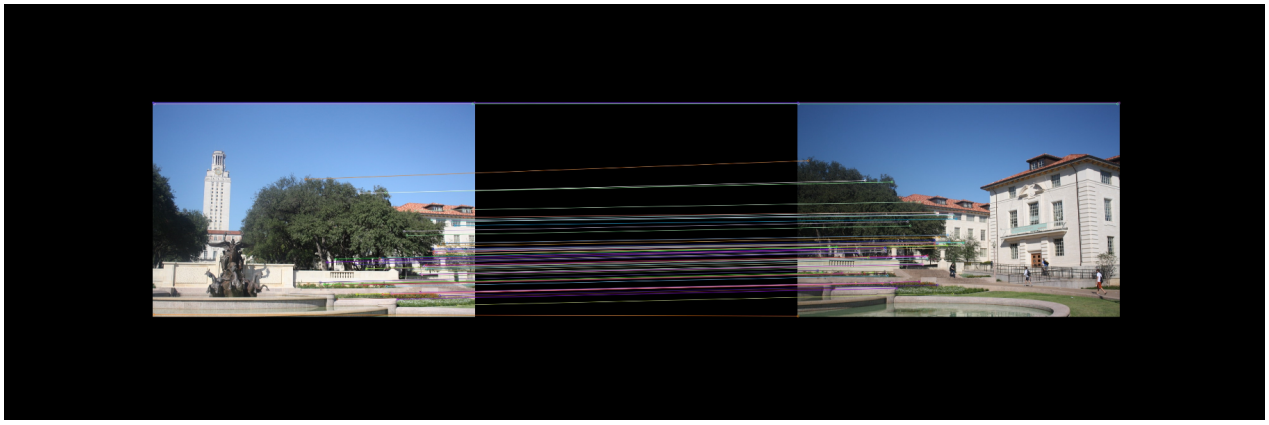

Figure 7: Matched key points connected across both images.

Finally, a homography transform was calculated using the best matched keypoints. The underlying method for this uses RANSAC, and is provided by opencv. The code segment for this is shown in Figure 8.

```
# convert the keypoints to numpy arrays
keypoints1 = np.float32([kp.pt for kp in keypoints1])
keypoints2 = np.float32([kp.pt for kp in keypoints2])

# construct the two sets of points
pts1 = np.float32([keypoints1[m.queryIdx] for m in matches])
pts2 = np.float32([keypoints2[m.trainIdx] for m in matches])

# estimate the homography between the sets of points using RANSAC
(homography, status) = cv2.findHomography(pts2, pts1, cv2.RANSAC, 4)
```

Figure 8: Homography calculated using RANSAC and opencv's findHomography function.

Finally, the panoramic image was created by applying the transform to im2, and then combining the resulting image with im1. This was completed in color and RGB, where the colour version im1 is simply cropped over top of the transformed im2, and in the grayscale result the two images are averaged together so the overlap is clearly visible. The code segment for the image transforms and combinations is shown in Figure 9, and the final RGB and grayscale images are shown in Figure 10 and 11.

```
# Generate result for RGB cropping
pano   =   cv2.warpPerspective(im2,   homography,   (int(im2.shape[1]*1.25),
im2.shape[0]))

for y in range(im1.shape[0]):
    for x in range(im1.shape[1]):
        if im1[y, x].any():
            pano[y, x] = im1[y, x]

# Generate result for grayscale overlapping
im1_gray = im1_gray.astype(float)
im2_gray = im2_gray.astype(float)

gray_pano        =        cv2.warpPerspective(im2_gray,        homography,
(int(im2_gray.shape[1]*1.25), im2_gray.shape[0])).astype(float)

for y in range(im1_gray.shape[0]):
    for x in range(im1_gray.shape[1]):
        if gray_pano[y, x] != 0:
            gray_pano[y, x] = (gray_pano[y, x] + im1_gray[y,x])
        else:
            gray_pano[y,x] = im1_gray[y,x]

gray_pano = gray_pano / np.amax(gray_pano) * 255.0 # Normalize
gray_pano = gray_pano.astype(np.uint8)
```

Figure 9: Code segment for transforming im2 and combining both images in RGB and grayscale.

Final Results

Figure 10: Final image panoramic in RGB. Im1 is placed over im2 completely.

Figure 11: Final image panoramic in grayscale. The images are averaged where they overlap.