Graphics Programming

Assignment 04

Taken Images

Left

Right

Middle

Finding Homography

To find homography, an ORB object detects keypoints in two images and then a BruteForce(BF) Matcher finds the best two matches for each keypoint in the images. If the distance between one match is bigger than another, specified by a value, N, then it is considered a good match. Using the good matches, the points on the images for these matches are found and used to find a homography between the two images.

Finding Homography (code)

```
orb = cv2.ORB_create()
kp1, des1 = orb.detectAndCompute(img1, None)
```

Creating the orb and detecting keypoints. (for one image)

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING)
matches = bf.knnMatch(des1, des2, k=2)
```

The best matches are found with both descriptors

```
goodMatches = []
    for match1, match2 in matches:
        if match1.distance < N*match2.distance:
            goodMatches.append(match1)
```

N is a percentage (0.75 in this). For one match, it has to be far enough from its counterpart to be considered ‘good’.

```
        index1 = match.queryIdx
        index2 = match.trainIdx
```

```
        srcPoint = kp1[index1].pt
        dstPoint = kp2[index2].pt
```

An example of finding the image point corresponding to the match. The srcPoint and dstPoint are added to their respective lists.

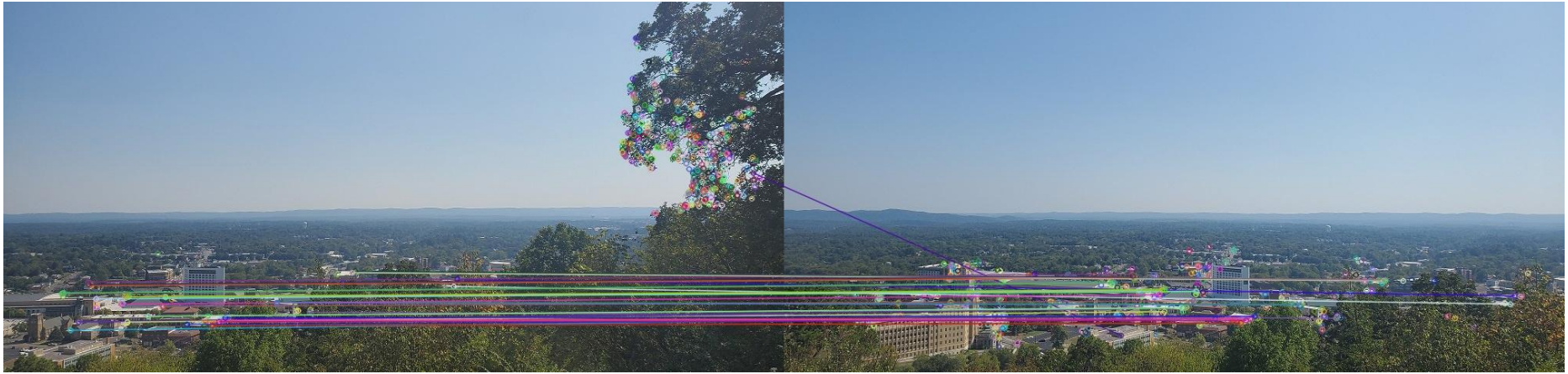
```
H, _ = cv2.findHomography(srcPoints, dstPoints,
cv2.RANSAC, 5.0)
```

Then the homography between two images is found with those points.

Keypoint Matches

Keypoint matches for the left and middle image.

Keypoint Matches



Keypoint matches for the right and middle images. Notice: in both images there are little overlapping lines which is good. However, there is an outlier in this one.

Image Stitching

Once the homography is found, the images have to be warped and then stitched together. This is done by translating each image so they are aligned, and warping the perspective with the homography for each image.

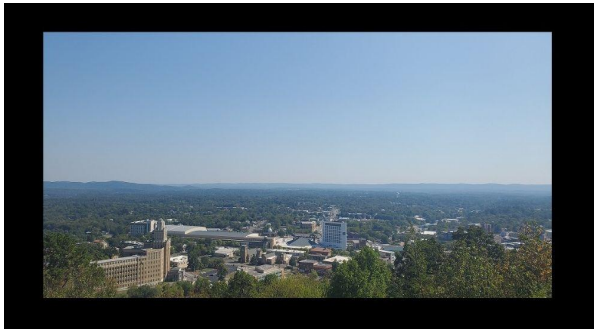
```
srcPoints = np.float64([[[0, 0]], [[0, h]], [[w, h]], [[w, 0]]])
srcPoints2 = np.float64([[[0, 0]], [[0, h2]], [[w2, h2]], [[w2, 0]]])
dstPoints = cv2.perspectiveTransform(srcPoints, H) #matches points from img2 to img1
dstPoints2 = cv2.perspectiveTransform(srcPoints2, G) #matches points from img3 to img1

points = np.concatenate((srcPoints,dstPoints, srcPoints2, dstPoints2), axis=0)

x_min, y_min = np.int32(points.min(axis=0).ravel())
x_max, y_max = np.int32(points.max(axis=0).ravel())
T = np.float32([[1,0,-x_min],[0,1,-y_min],[0,0,1]])
out1 = cv2.warpPerspective(img2,T,(new_w, new_h)) #original image but
realigned to be stitched
out2 = cv2.warpPerspective(img1,T@H,(new_w, new_h)) #left image,
realigned
out3 = cv2.warpPerspective(img3, T@G, (new_w, new_h)) #right image,
realigned
```

Image Stitching

Realigned left image.



Realigned middle image

Realigned right image

These are the realigned and warped images.

Image Stitching p.2

After being stitched, the images have some artifacting; there are ‘seams’ where the images connect. This can be fixed by creating a mask, and then blurring and eroding the mask of the image. The blur kernel was a simple box blur.

```
mask = cv2.warpPerspective(img2*0+255,T@H,(new_w, new_h))/255.0
mask = cv2.erode(mask, kernel)
mask = cv2.blur(mask,(30,30))

mask2 = cv2.warpPerspective(img2*0+255, T@G, (new_w, new_h))/255.0
mask2 = cv2.erode(mask2, kernel)
mask2 = cv2.blur(mask2,(30,30))

out = np.uint8((out1*(1-mask)+out2*mask + out1*(1-mask2)+out3*mask2)-out1)
```

Uncropped Pano

There was still a tiny bit of artifacting in the center, as seen above.

Cropped Pano

I just easy cropped the image in paint.