Computer Vision

Assignment 4

Keio University

STEFAN Vlad 82123434
15/05/2022

1 Panography

Figures 1a and 1b show 2 pictures that share common elements. These elements can be used to stitch these pictures together. Both picture were provided with paper [1].

(a) First picture

(b) Second picture

To do so, features points and their descriptors are extracted and matched using the method described in the previous assignment

(a) Keypoints in the first picture

(b) Keypoints in the second picture

Figure 3: Matches between keypoints

By assuming the displacement between feature points to be a simple translation, it is possible to apply Linear Least Squares to compute the translation parameters.
Translation parameters can then be used to compute the dimensions of the final composite canvas upon which both pictures are combined by averaging pixel values. The final result is shown in figure 4

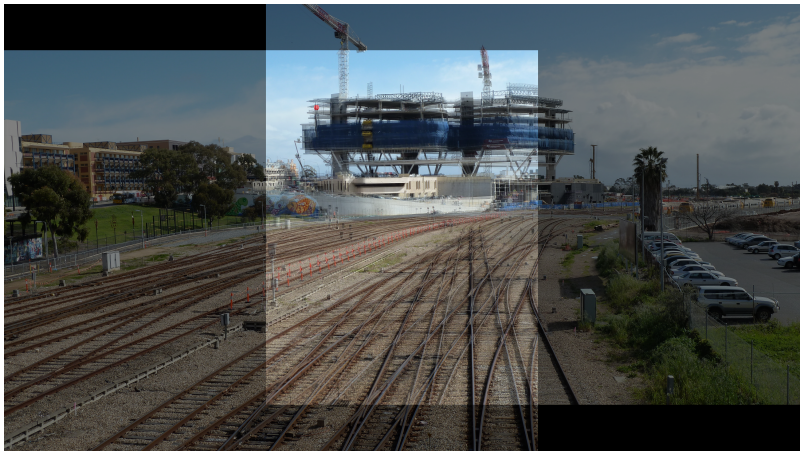

Figure 4: Stitched pictures

Obviously, the result is not perfect. The assumption that the feature displacement can be explained by a translation is an oversimplification. The features from both images are thus not perfectly superimposed, resulting in the presence of an after-image. Also, averaging is a very crude way of merging both pictures.
However, the resulting image is a nice estimate showing that even simple techniques can be used to roughly fuse 2 images together

2 Code

The previous results were obtained with the following code [1].

```
import cv2 as cv
import numpy as np

# Read both images
img1 = cv.imread("img1.jpg")
img2 = cv.imread("img2.jpg")

# Initiate an ORB detector
orb = cv.ORB.create()

# Find the keypoints and descriptors within each image
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)

# Draw keypoint location without size or orientation
img3 = cv.drawKeypoints(img1, kp1, None, color=(255, 0, 0), flags=0)
img4 = cv.drawKeypoints(img2, kp2, None, color=(0, 255, 0), flags=0)

# Save resulting images
cv.imwrite("img1_kp.png", img3)
cv.imwrite("img2_kp.png", img4)

# Create BFMatcher object
matcher = cv.BFMatcher()

# Use NNDR to match keypoints with their closest neighbour
matches = matcher.knnMatch(des1, des2, k=2)
good = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good.append([m])

img6 = cv.drawMatchesKnn(img1, kp1, img2, kp2, good, None,
                         flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv.imwrite("matches2.png", img6)

# Find the coordinates of matched keypoints
kp1_coord = [kp1[m[0].queryIdx].pt for m in good]
kp2_coord = [kp2[m[0].trainIdx].pt for m in good]
y, x = kp1_coord[0]
y, x = int(y), int(x)

img7 = cv.circle(img1, (y, x), 10, (0, 0, 255), -1)
cv.imwrite("circle.png", img7)

# Compute the translation matrix between both images
N = len(kp1_coord)
A = np.array([[N, 0],
              [0, N]])
b = np.array([[0],
              [0]])
for i in range(N):
    x, y = kp1_coord[i]
    x, y = round(x), round(y)
    x_, y_ = kp2_coord[i]
```

[1] The code and test pictures are available at this link

```
    x_, y_ = round(x_), round(y_)
    b[0, 0] = b[0, 0] + (x_ - x)
    b[1, 0] = b[1, 0] + (y_ - y)

p = np.linalg.solve(A, b)
print(p)
t_x = round(p[0, 0])
t_y = round(p[1, 0])

m = np.float32([[1, 0, t_x],
                [0, 1, t_y]])
# Translate the images by adding borders
if t_x >= 0:
    img1 = cv.copyMakeBorder(img1, 0, 0, t_x, 0, cv.BORDER_CONSTANT)
    img2 = cv.copyMakeBorder(img2, 0, 0, 0, t_x, cv.BORDER_CONSTANT)
else:
    img1 = cv.copyMakeBorder(img1, 0, 0, 0, abs(t_x), cv.BORDER_CONSTANT)
    img2 = cv.copyMakeBorder(img2, 0, 0, abs(t_x), 0, cv.BORDER_CONSTANT)

if t_y >= 0:
    img1 = cv.copyMakeBorder(img1, t_y, 0, 0, 0, cv.BORDER_CONSTANT)
    img2 = cv.copyMakeBorder(img2, 0, t_y, 0, 0, cv.BORDER_CONSTANT)
else:
    img1 = cv.copyMakeBorder(img1, 0, abs(t_y), 0, 0, cv.BORDER_CONSTANT)
    img2 = cv.copyMakeBorder(img2, abs(t_y), 0, 0, 0, cv.BORDER_CONSTANT)

# Combine both images by taking the average
dst = cv.addWeighted(img1, 0.5, img2, 0.5, 0.0)
cv.imwrite("dst2.png", dst)
```

The first part of the code is used to detect and match feature points. Then the coordinates of those feature points are used to compute the translation parameters by applying Linear Least Squares. Afterwards, the translation is applied by adding borders to the original images. Finally, both images are combined by averaging their pixel values.

References

[1] J. Zaragoza et al. “As-projective-as-possible image stitching with Moving DLT”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 36.7 (2014), pp. 1285–1298.