4TN4 Assignment 5

Adam Bujak (400113347)

March 13, 2022

1 Theory

1.1 Harris Corner Detection

1.2 Scale selection, LoG

The LoG of an image is represented by the formula

$$LoG(x, y) = -\frac{1}{\pi\sigma^4} \cdot (1 - \frac{x^2 + y^2}{2\sigma^2}) \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

To achieve the maximum response, the zeros of the Laplacian must be aligned with the edges of the circle, therefore:

$$0 = -\frac{1}{\pi\sigma^4} \cdot (1 - \frac{x^2 + y^2}{2\sigma^2}) \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

and since

$$x^2 + y^2 = r^2$$

$$0 = (1 - \frac{r^2}{2\sigma^2}) \cdot e^{-\frac{r^2}{2\sigma^2}}$$

$$0 = 1 - \frac{r^2}{2\sigma^2}$$

$$1 = \frac{r^2}{2\sigma^2}$$

$$\sigma = \pm\sqrt{\frac{r^2}{2}}$$

$$\sigma = \frac{r}{\sqrt{2}}$$

1.3 RANSAC

p is the probability that at least one of the sets does not include an outlier. Therefore $1-p$ is the probability every set includes outliers. u is the probability of any point being an inlier, therefore:

$$1 - p = (1 - u^m)^N$$

$$N = \frac{log(1-p)}{log(1-u^m}$$

$$N = \frac{log(1-0.99)}{log(1-0.7^2)}$$

$$N = 6.83$$

$$N \simeq 7$$

2 Implementation

3 Implementation

3.1 Change point of view

Figure 1: Points selected for transformation

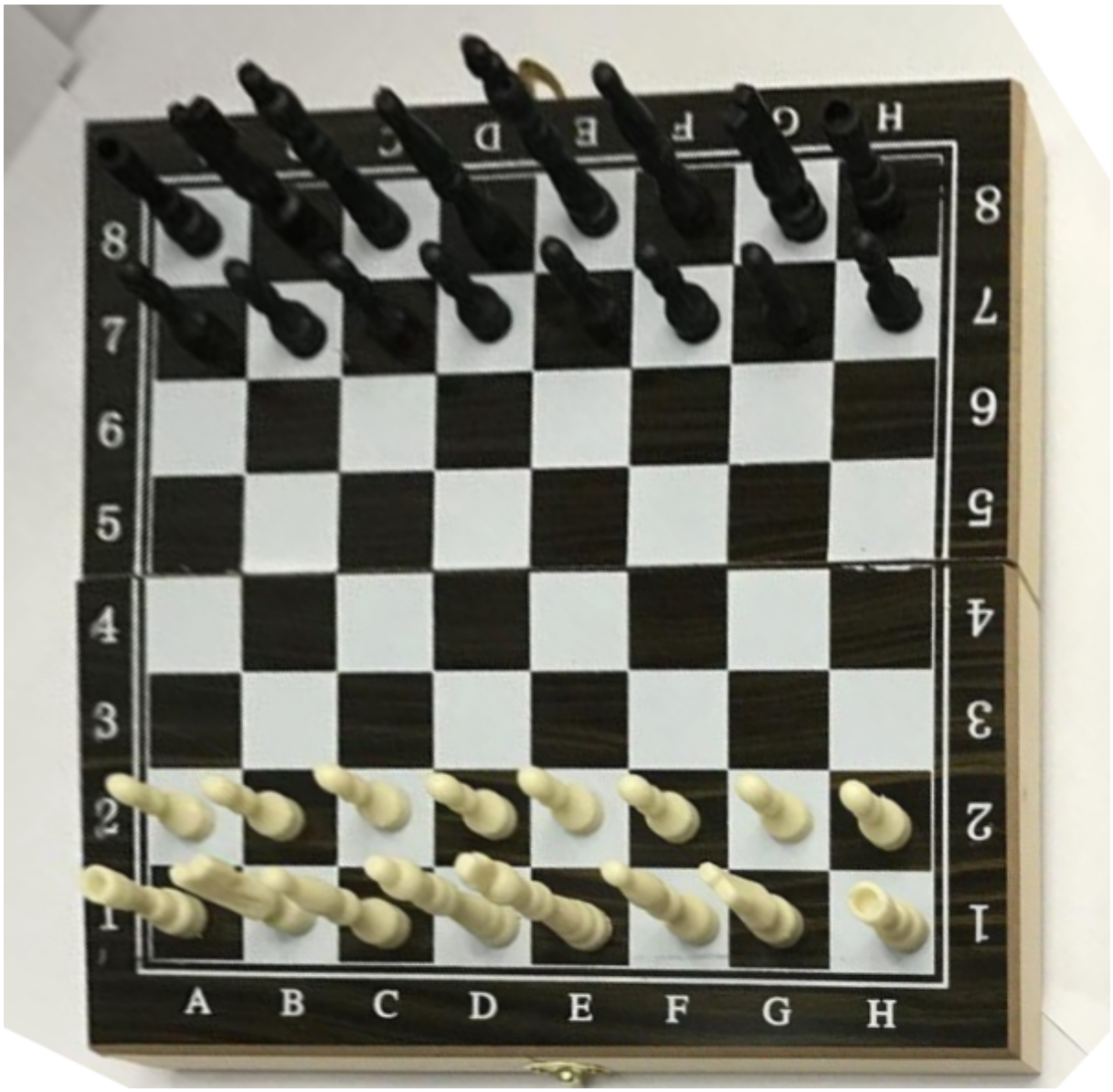

Figure 2: Transformed image

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt


def get_distance(p1, p2):
    return np.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)

img = cv2.imread('chess.png')
height, width, channels = img.shape

temp = np.full((height + 200, width + 200, 3), 255, dtype='uint8')
temp[100:height+100, 100:width+100] = img

#plt.imshow(temp)
#plt.show()

img = temp


img_cpy = np.copy(img)


# x, y
BOTTOM_LEFT = (66, 358)
BOTTOM_RIGHT = (475, 801)
TOP_LEFT = (511, 170)
TOP_RIGHT = (916, 408)

IMAGE_WIDTH = int(max(get_distance(BOTTOM_LEFT, BOTTOM_RIGHT), get_distance(TOP_LEFT, TOP_RIGHT)))
IMAGE_HEIGHT = int(max(get_distance(BOTTOM_LEFT, TOP_LEFT), get_distance(BOTTOM_RIGHT, TOP_RIGHT)))

start_point_arr = [TOP_LEFT, BOTTOM_LEFT, BOTTOM_RIGHT, TOP_RIGHT]
start_points = np.float32(start_point_arr)
end_points = np.float32([[0,0], [0, IMAGE_HEIGHT], [IMAGE_WIDTH, IMAGE_HEIGHT], [IMAGE_WIDTH, 0]])

for pt in start_point_arr:
    cv2.circle(img_cpy, pt, 3, (255, 155, 100), cv2.FILLED)

cv2.imwrite('chesspts.png', img_cpy)

T = cv2.getPerspectiveTransform(start_points, end_points)

img = cv2.warpPerspective(img, T, (IMAGE_WIDTH, IMAGE_HEIGHT), flags=cv2.INTER_LINEAR)


cv2.imwrite('new_chess.png',img)
```

Figure 3: Code used to transform image

I needed 4 points to execute this transformation.

3.2 Visualize Matched points

Using SIFT:

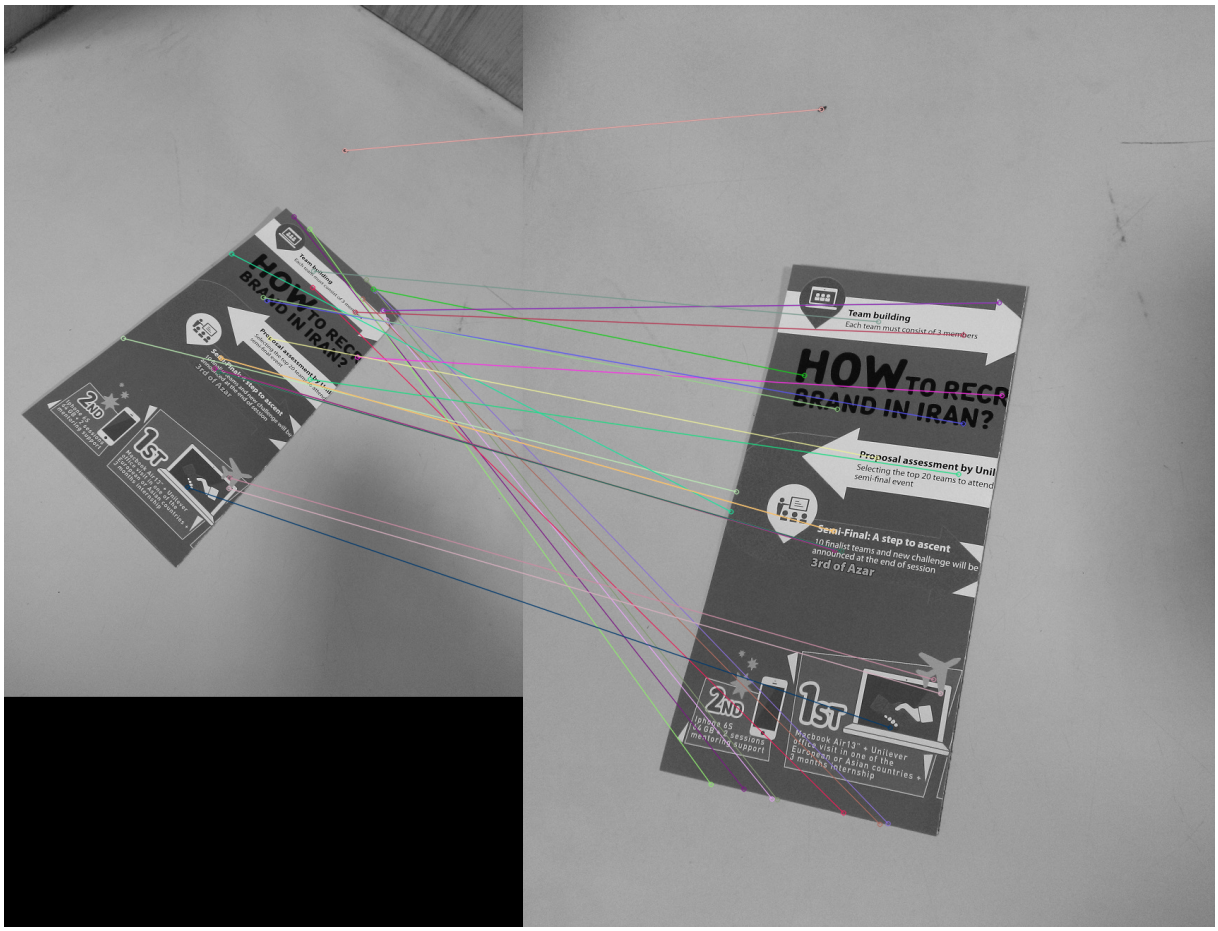

Figure 4: Matched images

```
import cv2
import matplotlib.pyplot as plt

# read images
img1 = cv2.imread('image1_1.jpg', 0)
img2 = cv2.imread('image1_2.jpg', 0)

sift = cv2.SIFT_create()

keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)

# exectue feature matching
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

matches = bf.match(descriptors_1,descriptors_2)

matches = list(matches)
# sort matches by distance
for i in range(len(matches)):
    for j in range(len(matches)):
        if matches[j].distance > matches[i].distance:
            temp = matches[j]
            matches[j] = matches[i]
            matches[i] = temp

# only use best 30 matches
matches = matches[0:30]

out = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2, matches, img2, flags=2)

cv2.imwrite('image_1_matches.png', out)
```

Figure 5: Code used to match features

3.3 Solving a puzzle

Figure 6: Solved Puzzle

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

def find_matches(img1, img2):
    sift = cv2.SIFT_create()
    keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)

    # exectue feature matching
    bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

    matches = bf.match(descriptors_1,descriptors_2)

    return matches, keypoints_1, keypoints_2


def warp(img1, img2):
    matches, k1, k2 = find_matches(img1, img2)

    k1 = np.float32([kp.pt for kp in k1])
    k2 = np.float32([kp.pt for kp in k2])

    ptsA = np.float32([k1[m.queryIdx] for m in matches])
    ptsB = np.float32([k2[m.trainIdx] for m in matches])


    H, status = cv2.findHomography(ptsA,ptsB, cv2.RANSAC, 4)

    rows1, cols1 = img1.shape[:2]
    rows2, cols2 = img2.shape[:2]

    list_of_points_1 = np.float32([[0,0], [0, rows1], [cols1, rows1], [cols1, 0]]).reshape(-1, 1, 2)
    temp_points = np.float32([[0,0], [0, rows2], [cols2, rows2], [cols2, 0]]).reshape(-1, 1, 2)

    list_of_points_2 = cv2.perspectiveTransform(temp_points, H)

    list_of_points = np.concatenate((list_of_points_1, list_of_points_2), axis=0)

    [x_min, y_min] = np.int32(list_of_points.min(axis=0).ravel() - 0.5)
    [x_max, y_max] = np.int32(list_of_points.max(axis=0).ravel() + 0.5)
    translation_dist = [-x_min, -y_min]

    H_translation = np.array([[1, 0, translation_dist[0]], [0, 1, translation_dist[1]], [0, 0, 1]])

    out = cv2.warpPerspective(img1, H_translation.dot(H), (x_max-x_min, y_max-y_min))
    out[translation_dist[1]:rows1+translation_dist[1], translation_dist[0]:cols1+translation_dist[0]] = img2

    return out


# read images
img1 = cv2.imread('image2_1.jpg', 0)
img2 = cv2.imread('image2_2.jpg', 0)
img3 = cv2.imread('image2_3.jpg', 0)
img4 = cv2.imread('image2_4.jpg', 0)


top = warp(img1, img2)
bottom = warp(img3, img4)
out = warp(top, bottom)
cv2.imwrite('out3.png', out)
```

Figure 7: Code used to merge images