Assignment 5

Theory

1 Harris Corner Detection

Step 1: Horizontal and Vertical gradient extraction
Sample Calculation for Ix on pixel (1,1): -1*255 + 1*255 = 0

$$Ix=\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 255 & 0 & 0 & -255 & 0 \\ 255 & 0 & 0 & -255 & 0 \\ 255 & 0 & 0 & -255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Sample Calculation for Iy on pixel (1,1): -1*0 + 1*255 = 255

$$Iy=\begin{pmatrix} 0 & 255 & -255 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -255 & -255 & -255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Step 2: Second Moment Matrix

$$M = \sum_{x,y} w(x,y) * \begin{bmatrix} I_x^2 & I_x * I_y \\ I_x * I_y & I_y^2 \end{bmatrix}$$

Sample Calculation for pixel (1,1): $M_{11} = \begin{bmatrix} I_x^2 = 65025 & I_x * I_y = 0 \\ I_x * I_y = 0 & I_y^2 = 65025 \end{bmatrix}$

$M_{11} = \begin{bmatrix} 65025 & 0 \\ 0 & 65025 \end{bmatrix}$

$M_{12} = \begin{bmatrix} 0 & 0 \\ 0 & 130050 \end{bmatrix}$

$M_{13} = \begin{bmatrix} 65025 & 0 \\ 0 & 130050 \end{bmatrix}$

$M_{21} = \begin{bmatrix} 130050 & 0 \\ 0 & 0 \end{bmatrix}$

$M_{22} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

$M_{23} = \begin{bmatrix} 65025 & 0 \\ 0 & 130050 \end{bmatrix}$

$M_{31} = \begin{bmatrix} 130050 & 0 \\ 0 & 65025 \end{bmatrix}$

$M_{32} = \begin{bmatrix} 0 & 0 \\ 0 & 130050 \end{bmatrix}$

$M_{33} = \begin{bmatrix} 130050 & 65025 \\ 65025 & 130050 \end{bmatrix}$

Step 3: Lambda Calculation

$$det(M - \lambda I) = 0$$

$$SampleCalculationforpixel(1,1) : \lambda_{11} = \begin{bmatrix} 65025 - \lambda I & 0 \\ 0 & 65025 - \lambda I \end{bmatrix}$$

$$(65025 - \lambda)^2 = 0$$

$$\lambda_{11} = 65025, 65025$$

$\lambda_{11} = 65025, 65025$
$\lambda_{12} = 0, 130050$
$\lambda_{13} = 65025, 130050$
$\lambda_{21} = 0, 130050$
$\lambda_{22} = 0, 0$
$\lambda_{23} = 0, 130050$
$\lambda_{31} = 65025, 65025$
$\lambda_{32} = 0, 130050$
$\lambda_{33} = 65025, 195075$

Step 4: Cornerness score computation
R= $\lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$
$R_{11} = 65025^2 - (0.04 * (65025 + 65025)^2 = 3551730525$
$R_{12} = -676520100$
$R_{13} = 6934331025$
$R_{21} = -676520100$
$R_{22} = 0$
$R_{23} = -676520100$
$R_{31} = 6934331025$
$R_{32} = -676520100$
$R_{33} = 9978671475$

All corner pixels have cornerness scores greater than 0: $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

2 Scale Selection

The Laplacian is given by:

$$\left(x^2+y^2-2\sigma^2\right)e^{-\left(x^2+y^2\right)/2\sigma^2}$$

To get maximum response, the zeros of the Laplacian have to be aligned with the circle:

$$\left(x^2+y^2-2\sigma^2\right)e^{-\left(x^2+y^2\right)/2\sigma^2}=0$$

There are two factors when solving for scale value but :

$$\left(e^{-\left(x^2+y^2\right)/2\sigma^2}!=0\right)$$

so rearranging the one valid factor:

$$\left(x^2+y^2-2\sigma^2\right)=0$$

$$\sigma=\frac{\sqrt{x^2+y^2}}{\sqrt{2}}$$

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

3 RANSAC

The probabilistic equation to determine the of iterations:

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

Substitute m=2,p=0.99,u=0.7 in the probabilistic equation:

$$N>=\frac{log(0.01)}{log(1-0.7^2)}$$

$$N>=6.84$$

Therefore, at least 7 iterations are required.

Implementation

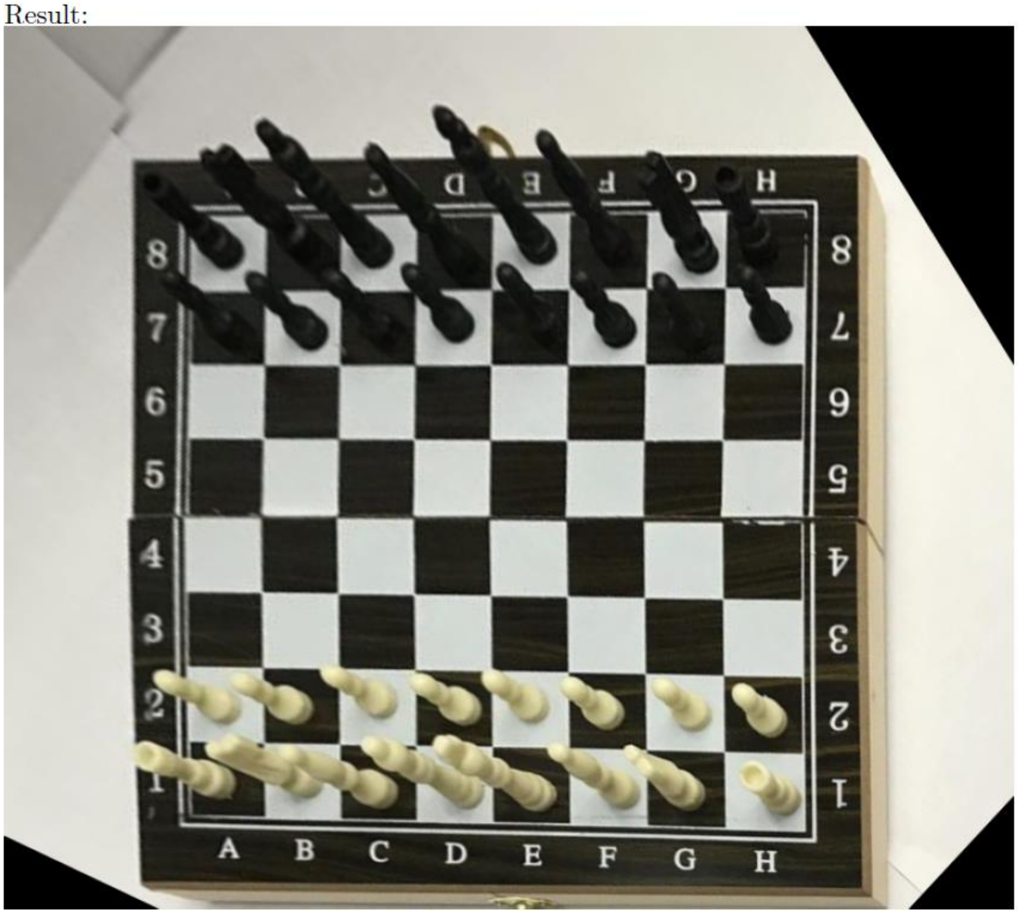
1 Change point of view

Code:

implementation1.py

```python
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

if __name__ == "__main__":
    # Read the image in
    img = cv.imread("chess.png")

    # source coordinates
    src_points = np.float32(
        [[400, 100], [750, 300], [10, 270], [350, 620]]
    )  # Top Left, Top Right, Bottom Left, Bottom Right

    # destination coordinates
    dst_points = np.float32(
        [[100, 100], [650, 100], [100, 650], [650, 650]]
    )  # Top Left, Top Right, Bottom Left, Bottom Right

    # Calculates a perspective transform from four pairs of the corresponding points
    projective_matrix = cv.getPerspectiveTransform(src_points, dst_points)

    rows, cols = img.shape[:2]
    # Apply a perspective transformation to an image
    img_output = cv.warpPerspective(img, projective_matrix, (cols, rows))

    plt.imshow(img_output)
    plt.show()
    cv.imwrite("implementation1.png", img_output)
```

Result:



2 Visualize Matched points

Code:

implementation2.py > ...

```python
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

if __name__ == "__main__":
    # Read the image in
    img1 = cv.imread("image1_1.jpg")
    img2 = cv.imread("image1_2.jpg")

    # Convert images to grayscale
    gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
    gray2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)

    # Detect ORB features and descriptors
    orb = cv.ORB_create(400)
    keypoints1, descriptors1 = orb.detectAndCompute(gray, None)
    keypoints2, descriptors2 = orb.detectAndCompute(gray2, None)

    # create Brute-Force matcher object
    # since ORB descriptor is binary stirng based, Hamming distance is used as measurement
    # crossCheck enabled provides consistent results by ensuring two features in both sets should match each other

    bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

    # Match descriptors.
    matches = bf.match(descriptors1, descriptors2)

    # Sort them in the order of their distance.
    matches = sorted(matches, key=lambda x: x.distance)
    # Draw best 10 matches
    img3 = cv.drawMatches(
        img1,
        keypoints1,
        img2,
        keypoints2,
        matches[:10],
        None,
        flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
    )
    plt.imshow(img3), plt.show()
    cv.imwrite("matches_output.jpg", img3)
```

Result:

3 Solving a puzzle

Code:

implementation3 > implementation3.py > ...

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


def stitchImage(img1, img2, debug):
    # Convert images to grayscale
    gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
    gray2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)

    # Detect SIFT features and descriptors
    descriptor = cv.SIFT_create()
    keypoints1, descriptors1 = descriptor.detectAndCompute(gray, None)
    keypoints2, descriptors2 = descriptor.detectAndCompute(gray2, None)

    # create Brute-Force matcher object
    # crossCheck enabled provides consistent results by ensuring two features in both sets should match each other
    bf = cv.BFMatcher(crossCheck=True)
    # Match descriptors.
    matches = bf.match(descriptors1, descriptors2)

    # The points with small distance (more similarity) are ordered first in the vector
    rawMatches = sorted(matches, key=lambda x: x.distance)

    # Draw best 25 matches
    img3 = cv.drawMatches(
        img1,
        keypoints1,
        img2,
        keypoints2,
        rawMatches[:25],
        None,
        flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
    )

    if debug:
        plt.imshow(img3)
        plt.show()

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

    # construct the two sets of points
    ptsA = np.float32([kpsA[m.queryIdx] for m in rawMatches])
    ptsB = np.float32([kpsB[m.trainIdx] for m in rawMatches])

    # estimate the homography between the sets of points
    H, status = cv.findHomography(ptsA, ptsB, cv.RANSAC, 4)

```

implementation3 > implementation3.py > ...

```python
    # Merging images

    # Apply panorama correction
    width = img1.shape[1] + img2.shape[1]
    height = img1.shape[0] + img2.shape[0]
    result = cv.warpPerspective(img1, H, (width, height))
    result[0 : img2.shape[0], 0 : img2.shape[1]] = img2

    if debug:
        plt.figure(figsize=(20, 10))
        plt.imshow(result)

        plt.axis("off")
        plt.show()

    # Remove extra black edges after merging
    # Transform the panorama image to grayscale and threshold it
    gray = cv.cvtColor(result, cv.COLOR_BGR2GRAY)
    thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY)[1]

    # Finds contours from the binary image
    cnts = cv.findContours(thresh.copy(), cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    # Get the maximum contour area
    c = max(cnts, key=cv.contourArea)
```

```python

    # Get a bbox from the contour area
    (x, y, w, h) = cv.boundingRect(c)

    # Crop the image to the bbox coordinates
    result = result[y : y + h, x : x + w]
    return result


if __name__ == "__main__":
    # Stich bottom half images together
    img2_4 = cv.imread("image2_4.jpg")
    img2_3 = cv.imread("image2_3.jpg")
    output2 = stitchImage(img2_4, img2_3, False)
    cv.imwrite("bottomHalf.jpg", output2)

    # Stich top half images together
    img2_2 = cv.imread("image2_2.jpg")
    img2_1 = cv.imread("image2_1.jpg")
    output = stitchImage(img2_2, img2_1, False)
    cv.imwrite("topHalf.jpg", output)

    # Stich two previous halfs together
    bottomOutput = cv.imread("bottomHalf.jpg")
    topOutput = cv.imread("topHalf.jpg")
    final_output = stitchImage(bottomOutput, topOutput, False)
    cv.imwrite("puzzledSolved.jpg", final_output)
```

Top Half: 

Bottom Half: 

Final Result: