Computer Vision Assignment Report

Keran Zhao

I. Introduction

This assignment will implement the extraction of salient visual features from images by utilising various methods in Python and OpenCV library, such as Harris corner points and SIFT feature points. Creating descriptors for these features and achieving feature matching between two images using the detected feature points and descriptors. Finally integrate the above functions to finalise the image stitching task.

In addition, a user interaction page has been developed for this assignment where the user can upload an image locally and see the extracted features, matched feature point pairs or stitched images by clicking on the buttons from the page.

The version of python used in this assignment is 3.11 and the OpenCV version is 4.8.0

II. Methodology

This chapter will introduce the methods used in the process of achieving the tasks of feature detection, feature matching, and image stitching in the assignment.

A. Harris corner detector

Harris corner detection is an image gradient based corner detection method with strong invariance to rotational transformations and illumination changes (Derpanis 2004). Harris corner detection determines whether a pixel point is a corner point or not by calculating the auto-correlation matrix of each pixel point.

The auto-correlation matrix is given by:

$$M = \sum_{u,v} w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

where $w(u,v)$ is a window function, usually a Gaussian window, used to weight the pixel points such that pixels near the centre of the window contribute more and pixels far from the centre contribute less. I_x denotes the horizontal gradient of the pixel point and I_y denotes the vertical gradient of the pixel point.

After obtaining the auto-correlation matrix for each pixel point in the local region, use corner point response function:

$$R = \det(M) - k \cdot (\text{trace}(M))^2$$

to determine if a pixel point is a corner point based on the value of R. If R has a large positive value, the point is a corner point.

In the above function, $\det(M) = \lambda_1 \lambda_2$, $\text{trace}(M) = \lambda_1 + \lambda_2$. λ_1 and λ_2 are the eigenvalues of the matrix M. k is an empirical parameter that usually takes values from 0.04 to 0.06.

B. Scale-Invariant Feature Transform (SIFT)

SIFT (Scale-Invariant Feature Transform) is an algorithm for detecting and describing local features with scale and rotation invariance.

1) Keypoints Detector: SIFT constructs the scale space of an image by Gaussian blurring and Difference of Gaussian (DoG) at different scales. In the scale space, the DOG extreme points of each pixel point are labelled as key points. Specifically, in each scale image, each pixel is compared with its 8 surrounding neighbourhood pixels and 18 pixels of the upper and lower images of the adjacent scale. If it is a maximum or minimum value, it is marked as a keypoint.

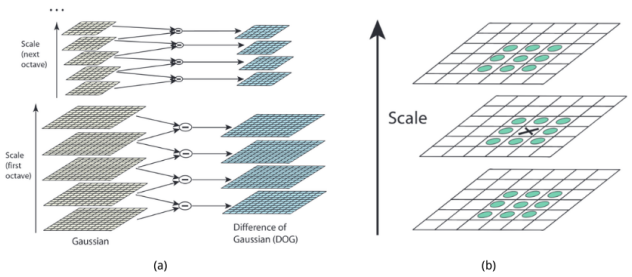


Fig. 1. Detect SIFT feature points. (a) Acquisition of image space at different scales. (b) Get SIFT feature points. (Image captured from Week 6 lecture slides)

After obtaining the SIFT feature points from the image, the gradient direction and magnitude of the pixels in the neighbourhood around the key point are computed, and a histogram of the gradient direction then constructed. The peak of the smoothed histogram is designated as the standard direction.

2) Create SIFT Descriptor: The position and orientation of the SIFT feature points can be used to create SIFT descriptors, which will be used to implement feature matching. This is one of the reasons why SIFT is widely used for feature detection. In generating the key point descriptors, the key point neighbourhood is divided into 4*4 grids, and each grid calculates the gradient histograms in 8 directions, which ultimately results in a 128-dimensional vector as a descriptor of a feature point.

C. Binary Descriptor - Oriented FAST and Rotated BRIEF (ORB)

The feature point descriptor of SIFT is a 128-dimensional high-dimensional vector. In addition to the SIFT method, the ORB (Oriented FAST and Rotated BRIEF) descriptor uses

a binary string to describe the feature point information, which also enables feature point detection and description from images.

ORB uses the FAST (Features from Accelerated Segment Test) algorithm for feature point detection and performs Harris corner point scoring on the detected points, selecting the top N points with the highest scores as keypoints. After obtaining the feature points, ORB describes the feature points using BRIEF (Binary Robust Independent Elementary Features) descriptor. The descriptor generates a binary string by performing a series of pixel intensity comparisons within the neighbourhood of the feature point. The descriptor is made rotationally invariant by rotating the BRIEF descriptor to align it with the principal direction of the feature point.

D. Feature Matching

1) Sum of squared differences (SSD): The formula for SSD is as follows:

$$d(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\|^2$$

where $\mathbf{X}$ and $\mathbf{Y}$ are vectors of two descriptors, and $\|\cdot\|$ denotes the L2 norm.

The similarity of feature points is determined by calculating the SSD value between two feature point descriptors, the smaller the SSD value the more similar the features are. The two feature points with the smallest SSD value will be matched as a pair of features.

2) Hamming Distance: For binary descriptors obtained using ORB, Hamming Distance is used to measure the similarity between two strings. Hamming Distance is calculated using the formula:

$$d_H(A, B) = \sum_{i=1}^{n} 1(A_i \neq B_i)$$

where $1(A_i \neq B_i)$ is the indicator function. When $(A_i \neq B_i)$, takes the value of 1, otherwise it takes the value of 0.

Similar to SSD, the smaller the value of Hamming distance, the more similar the two descriptors are. The feature points corresponding to the two descriptors with the smallest distance will be returned as a feature matching pair.

3) Ratio Test: Ratio test is a method to improve matching accuracy by comparing the optimal and sub-optimal matches for each feature point to filter out more reliable matching pairs.

The Ratio test calculates the ratio of the optimal match to the suboptimal match as the basis for judging the good and bad matches. Test will preset a threshold value and only matches less than this threshold will be marked.

E. Image Stitch

After obtaining the feature matching of the two images, implementing image stitching requires computing the mapping relationship between the two image planes. This mapping relation can be obtained by computing the homography of the two images. The homography H satisfies the following relation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

H is a 3*3 matrix that can be estimated by the RANSAC algorithm. Using homography, one image can be transformed into the coordinate system of another image by affine or perspective transformations, and the transformed image can be stitched together with the target image to achieve image stitching.

III. Design and Implementation

A. Image Dataset Construction

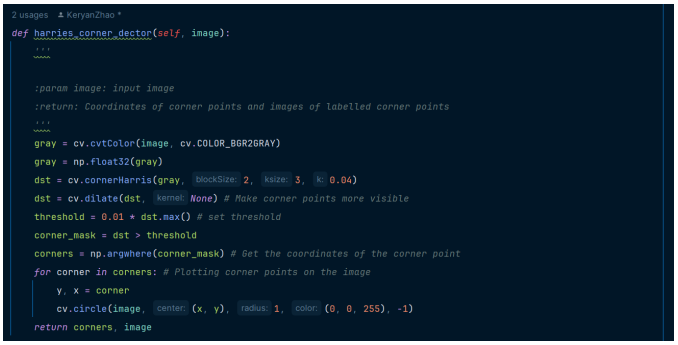
In order to test the task completion and compare the differences between the different methods, collected 5 pairs of images as the image dataset for this assignment. One pair of images is provided by the assignment document, one pair of images is taken from the lecture slides, and the remaining three pairs of images cover indoor environments, changes in lighting, and other situations. The aim is to explore more fully the invariance characteristics of the methods used in the assignment in the face of change.

Fig. 2. Constructed Image Dataset

B. Harris corner detector

The specific code implementation of Harris corner point detection is shown below:

The input BGR image is first converted to a grey scale image and Harris corner point detection is performed on the grey scale image using `cv.cornerHarris`. The parameter 2 denotes the size of the window used to calculate the auto-correlation matrix, and 0.04 is the value of the empirical parameter k. After obtaining the corner points, a threshold is set for filtering the more significant corner points. Finally,



```
2 usages  KeryanZhao *
def harries_corner_dector(self, image):
    '''
    :param image: input image
    :return: Coordinates of corner points and images of labelled corner points
    '''
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv.cornerHarris(gray, blockSize: 2, ksize: 3, k: 0.04)
    dst = cv.dilate(dst, kernel: None) # Make corner points more visible
    threshold = 0.01 * dst.max() # set threshold
    corner_mask = dst > threshold
    corners = np.argwhere(corner_mask) # Get the coordinates of the corner point
    for corner in corners: # Plotting corner points on the image
        y, x = corner
        cv.circle(image, center: (x, y), radius: 1, color: (0, 0, 255), -1)
    return corners, image
```

Fig. 3. code implementation of Harris corner detector

the intersection coordinates are obtained and labelled in the image.

C. SIFT feature detection and description

The SIFT method has been preset in OpenCV to detect feature points and compute descriptors. The SIFT detector is initialized using `cv.SIFT_create()` and `sift.detectAndCompute()` is used to perform SIFT feature point detection on the input greyscale image and compute the descriptors of the feature points.

```
6 usages  Keran Zhao +1 *
def SIFT_points_dector(self, image):
    # Converting images to greyscale
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    sift = cv.SIFT_create()
    # Detecting keypoints and descriptors using SIFT
    keypoints, descriptors = sift.detectAndCompute(gray, None)
    image_with_keypoints = cv.drawKeypoints(image, keypoints, outImage: None, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    keypoints_info = [(kp.pt, kp.angle) for kp in keypoints]

    return image_with_keypoints, keypoints, keypoints_info, descriptors
```

Fig. 4. Feature detection and feature description using SIFT

D. ORB feature detection and description

Similar to SIFT, the ORB detector is also preset in the OpenCV library. The ORB method is implemented using a similar approach to SIFT to detect keypoints and compute descriptors.

```
4 usages  Keran Zhao *
def ORB_points_dector(self, image):
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    # Initialise the ORB detector
    orb = cv.ORB_create()
    keypoints, descriptors = orb.detectAndCompute(gray, None)
    image_with_keypoints = cv.drawKeypoints(image, keypoints, outImage: None, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return image_with_keypoints, keypoints, descriptors
```

Fig. 5. Feature detection and feature description using ORB

E. Feature Matching

For the 128-dimensional descriptor vector computed by the SIFT detector, the square of the L2-paradigm distance of the feature descriptors is first computed using the SSD method, and the feature pair with the smallest distance is selected as the matching feature based on the SSD value. The ratio test will add a ratio threshold to the SSD distance and only features within the threshold will be labelled.

For binary strings computed by the ORB detector, use Hamming distance as the distance function, and again the ratio test will be applied to screen for more accurate matching pairs.

```
A Keran Zhao *
def compute_ssd(self, descriptor1, descriptor2):
    ssd = np.sum((descriptor1 - descriptor2) ** 2)
    return ssd

2 usages  new *
def match_descriptors_ssd(self, descriptors1, descriptors2, ORB = False):
    '''
    :param descriptors1: descriptors of left image
    :param descriptors2: descriptors of right image
    :return:
    '''
    if ORB:
        bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
        matches = bf.match(descriptors1, descriptors2)
        return matches

    else:
        bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
        matches = bf.match(descriptors1, descriptors2)
        return matches[:800]
        # matches = []
        # for i, desc1 in enumerate(descriptors1):
        #     best_match = None
        #     min_distance = float('inf')
        #     for j, desc2 in enumerate(descriptors2):
        #         distance = self.compute_ssd(desc1, desc2)
        #         if distance < min_distance:
        #             min_distance = distance
        #             best_match = cv.DMatch(i, j, distance)
        #     if best_match is not None:
        #         matches.append(best_match)
        # return matches
```

Fig. 6. Compute SSD and find matching pairs using SSD distance

In the implementation, calculating the SSD distance for each pair of descriptors of feature points leads to the problem of long computation time, because the L2-paradigm value and the square of the L2-paradigm does not change the ordering of the distances, so the matching results can be obtained using the `BFMatcher` and the L2-paradigm distance as well.

```
1 usages  A Keran Zhao *
def ratio_test(self, descriptors1, descriptors2, ORB = False):
    """
    Apply ratio tests to filter matching pairs
    :param descriptors1: descriptors of left image
    :param descriptors2: descriptors of right image
    :return Matches that pass the ratio test
    """
    if ORB:
        bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=False)
        matches = bf.knnMatch(descriptors1, descriptors2, k=2)
        good_matches = []
        for m, n in matches:
            if m.distance < 0.75 * n.distance:
                good_matches.append(m)
        return good_matches

    else:
        bf = cv.BFMatcher(cv.NORM_L2, crossCheck=False)
        matches = bf.knnMatch(descriptors1, descriptors2, k=2)
        good_matches = []
        for m, n in matches:
            m_ssd = m.distance ** 2
            n_ssd = n.distance ** 2
            if m_ssd < 0.75 * n_ssd:
                good_matches.append(m)
        return good_matches
```

Fig. 7. Feature Matching using Ratio test

The ratio test also initializes a `BFMatcher`, uses the `bf.knnMatch()` method with $k = 2$ to get the optimal and sub-optimal matches, calculates the square of the distance of two matches to get the SSD distance, and calculates the ratio of the two distances to determine whether the optimal match is within the set threshold, so as to get the matching result of the ratio test.

The specific code implementation process is shown in Figures 5 and 6.

F. Image Stitching

Implementing image stitching firstly the feature points and descriptors of the two images are obtained using SIFT method and after that the ratio test is applied to get the feature matching of the two images.

The homography of the image transformation is calculated using the `cv.homography()` method based on the positions of the matched feature points. The perspective transformation is performed on the right image using the `cv.warpPerspective()` method, and the left image is stitched onto the left side of the transformed image to obtain the result.

```
Image  1 Keran Zhao *
def stitch_images(self, images):
    (imageB, imageA) = images
    # Detection and description using SIFT features
    _, keypointsA, _, descriptorsA = self.SIFT_points_dector(imageA)
    _, keypointsB, _, descriptorsB = self.SIFT_points_dector(imageB)

    # Feature point matching using ratio tests
    matches = self.ratio_test(descriptorsA, descriptorsB)

    src_pts = np.float32([keypointsA[m.queryIdx].pt for m in matches])
    dst_pts = np.float32([keypointsB[m.trainIdx].pt for m in matches])

    # Calculate the homography
    H, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    # Perform a perspective transformation on an image using the homography
    # and stitch the two images together
    result = cv.warpPerspective(imageA, H, dsize: (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
    result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB

    return result
```

Fig. 8. Image Stitching

G. User Interface

The user interface page expects the user can upload images from the local area to test the performance of the programme's functions such as feature point detection, feature matching, image stitching, and so on. After uploading the left and right side images using the correct buttons, the functions can be invoked by clicking on the buttons in the UI page and the processed images will be displayed in a new page, and for the feature detection task, the position and orientation (if available) of the feature points will be displayed by clicking on the corresponding buttons.

The implemented UI page is shown below:

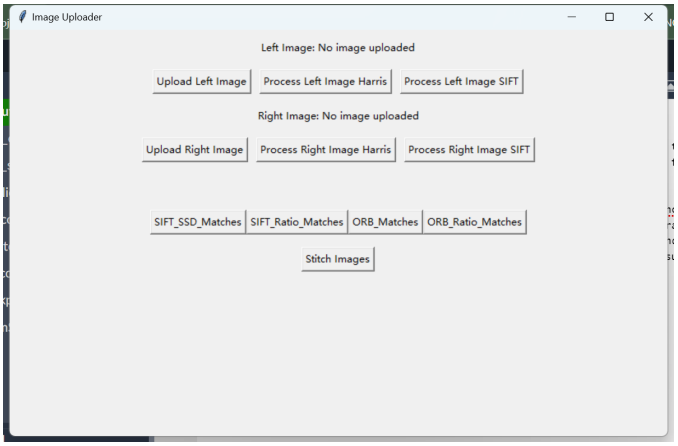


Fig. 9. User Interface

IV. Results and Discussion

A. Feature Detection

The detected Harris corner points and SIFT feature points are shown below:

Fig. 10. Detected Harris corner points and SIFT feature points

Harris detected a total of 41,519 corner points and SIFT detected a total of 6,841 feature points. According to the results it can be seen that Harris detector can detect more corner points. This is because SIFT will screen and optimise the key points during the detection process, and some less characteristic points may be ignored during the screening process, which results in fewer feature points being detected by the SIFT method. Based on the distribution of feature points in the figure, both methods detected denser feature points in the tree region in the middle of the image.

In the position information of the detected feature points, it can be seen that the positions of the SIFT feature points contain float numbers, while the Harris corner points are pixel point positions. This is because the SIFT algorithm will locate the keypoints accurately at the sub-pixel level. SIFT constructs the scale space of the image at different scales by Gaussian blurring and Difference of Gaussian (DoG). In these different scale spaces, the position of the keypoints may be in floating point numbers. So the final obtained feature point location coordinates may contain floating point numbers.

B. Feature Matching

The implementation of the two descriptors as well as the feature matching scheme has been stated in the implementation of the previous section.

1) Display the matches: The match between two images obtained using two descriptors and two distance functions is shown below:

Based on the feature matching results, it can be observed that feature matching can be achieved for both descriptor descriptions, and the SIFT method also obtained many matching pairs in image data captured under different ambient light, which also shows that SIFT has invariant features and is more robust.

2) Comparative analysis of the two descriptors: By looking at the feature matching results obtained for the two descriptors in the above figure, it can be seen that the number of matched pairs obtained by the SIFT method is significantly more than

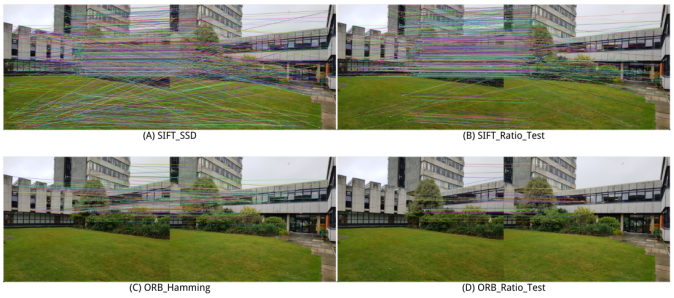


Fig. 11. Feature Matching results with different descriptor and distance function

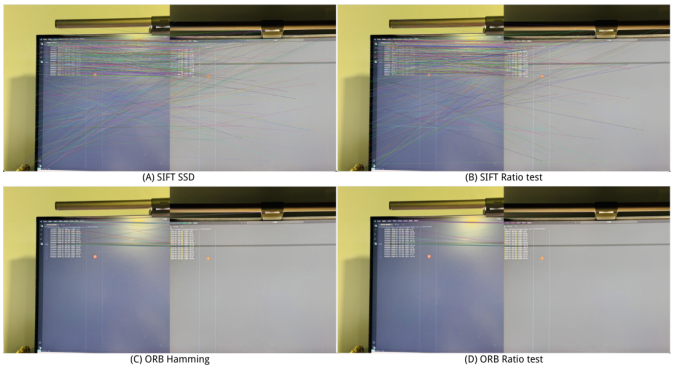


Fig. 12. Feature Matching results with different descriptor and distance function

the ORB method. This is because the descriptors generated by SIFT are 128-dimensional floating-point vectors that contain rich local image information. The descriptors generated by the ORB are binary strings, which are relatively less informative. In terms of matching accuracy, most of the matches obtained by the SIFT method contain matching feature points detected by the ORB method. However, the feature matching results obtained by both descriptors contain mismatched feature pairs. In terms of matching speed, the running times of five sets of image datasets using different descriptors and distance methods were compared as shown in the table below:

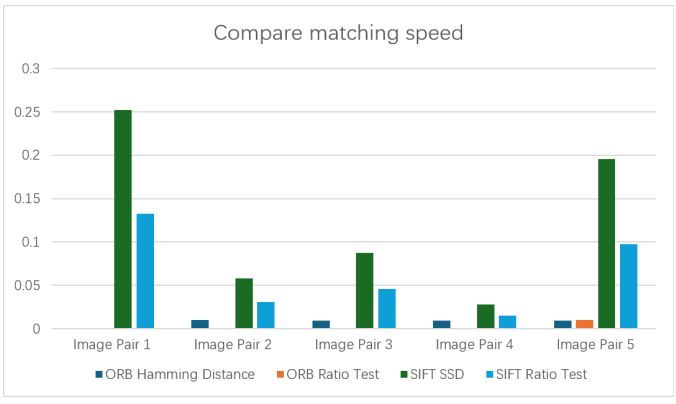


Fig. 13. Running speed of different descriptors and distance methods

It can be seen that the matching speed of ORB is significantly better than the SIFT method. This is because the amount of data information processed by the ORB method is less than the SIFT method.

3) Compare the two distance measurements: Based on the displayed matching results, it can be seen that the matching accuracy using the Ratio test method is better than that of the feature matching results obtained by using only the SSD distance because of the filtering out of some false matches of some SSDs.

C. Image Stitch

In the previous chapter the assignment was described how to implement the image stitching task using code. This chapter will show the results of the image dataset after stitching, where green circles will be used to mark the parts that were successfully docked and red circles will represent the offsets present in the stitching results.

Fig. 14. Stitch result 1

The images can be seen through the pictures that successful stitching can be achieved in the grass part, but there is an offset in the connecting part of the building, but the overall stitching is successful.

Fig. 15. Stitch result 2

As can be seen from the image, the stitching of the image in the grass, road, and building is successful, but there is an offset for the tree branch portion at the top of the image. But the overall stitching result is also successful.

Image 17 shows the final stitching results of the images captured in different lighting environments. It can be seen that the feature matching results obtained using the SIFT detector and descriptors can overcome the effect of lighting on the image and obtain more accurate stitching results.

Fig. 16. Stitching result 3

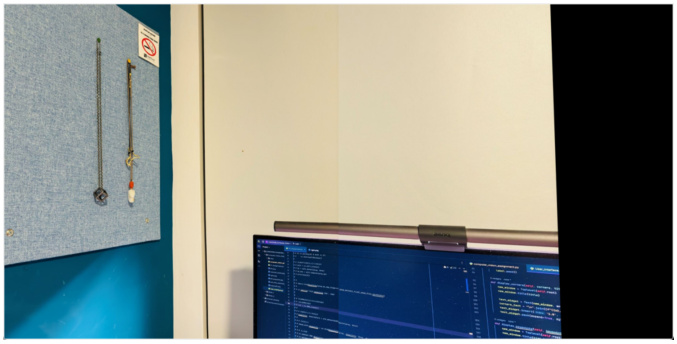


Fig. 17. Stitching result 4

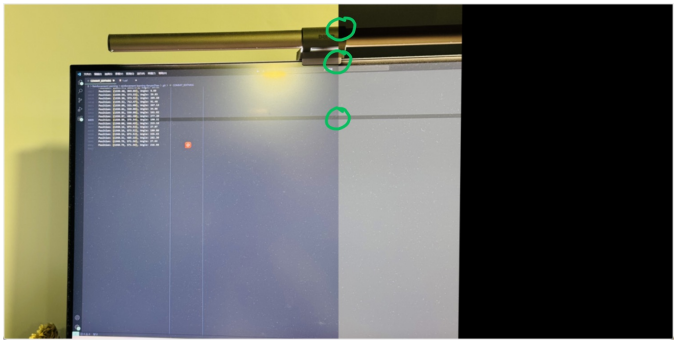


Fig. 18. Stitching result 5

V. Conclusion

This assignment uses the knowledge of computer vision combined with the preset methods in OpenCV to successfully implement feature detection, feature description and feature matching, and uses these methods in combination to achieve the stitching of two images. By analysing the feature matching results obtained from different descriptors, the SIFT descriptor can achieve more accurate feature matching by calculating more image information, and successfully achieve the image stitching task. The assignment applies Harris corner detection and SIFT feature detection and description covered in the lecture, and also applies the ratio test in feature matching. The results show that the ratio test can filter out some bad matching features and the results are more accurate.

CNN and machine learning methods were also covered in the lecture, which can be used in future work to detect and match features in images, which may require more image data to train artificial neural networks.

References

Derpanis, Konstantinos G (2004). "The harris corner detector". In: *York University* 2.1, p. 2.