Practical Assignment № 3

Features Detectors.

Instructing Professor :Sergei Shavetov & Andrei Zhdanov
Author :Xu Miao

May 20, 2022

Contents

1 Introduction

§1.1 Student information

- ★ Name: Xu Miao
- ★ ITMO Number: 293687
- ★ HDU Number: 19322103

§1.2 Purpose.

Study of the transformation to find of geometric primitives.

§1.3 Introduction

In this report, I will complete the content of the following sections:

1) **Feature points detection.** (3 images.) Perform to search for feature points using:
   a) SIFT feature point descriptor
   b) ORB feature point descriptor
2) **Feature points matching.** (2 pairs of images)
   a) Extract feature points of an object and match them with feature points of a scene containing this object.
   b) Calculate the **transformation matrix** using RANSAC method and **highlight the object position** in the scene.
   c) Show the **inlier matches**. Compare feature point descriptors for the task of image matching.
3) **simple automatic image stitching.**
   a) calculate the transformation matrix between **two images** and stitch them into a single panoramic image.
   b) Use it to stitch **three images** into a single panoramic image.

2 Notations

Table 2.1: Notation used in this report

Symbol	Definition
x	Abscissa in Cartesian coordinate system
y	Ordinate in Cartesian coordinate system
θ	Ordinate in Hough space
ρ	Abscissa in Hough space
R	the radius of the circle

⋆ Other notations instructions will be given in the text.

3 Solution of problem

§3.1 Feature points detection.

§3.1.i Theorem 1: SIFT detector.

SIFT stands for Scale-Invariant Feature Transform

Its steps can be mainly divided into three steps:

1) keypoint localisation.(DOG)

2) Calculate feature point orientation

3) feature point description.(Approximate to HOG)

SIFT detector keypoint localisation : DOG

1) **Construct scale space**

One of the properties of constructing a scale space of feature points is that they **remain invariant to changes in scale.** Therefore, the feature points are found to be detectable at different scales. According to the literature, it is known that the only kernel function that transforms to the scale space is the **Gaussian function.**

It can be calculated as a scale-space maximum response of the Laplacian of Gaussian of an image with varying the σ value, which is calculated by convolution of of the variable-scale Gaussian $G(x, y, \sigma)$ with an input image $I(x, y)$:

$$\begin{aligned} L(x, y, \sigma) &= G(x, y, \sigma) * I(x, y) \\ G(x, y, \sigma) &= \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}} \end{aligned} \tag{3.1}$$

where $*$ is a convolution operation in x and y.(x, y) is the scale coordinate, and the size of σ determines the smoothness of the image, large scale corresponds to the generalized features of the image, small scale corresponds to the detailed features of the image, large corresponds to the coarse scale (low resolution), and small corresponds to the high resolution.

To make the calculation relatively efficient, a Gaussian differential scale space is really used.

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{3.2}$$

D is the difference in scale between two adjacent scales by a multiplicative factor k.

DOG implementation:

An initial image is convolved with Gaussian functions of different values to obtain a stack of blurred images, and then the stack of images is subtracted by two to obtain

the corresponding DOG. s is the number of layers in each group, usually 3 ~ 5, and finally the relationship between s and k can be established as $k = 2^{1/s}$.The steps of the algorithm to implement DOG are shown in the following figure:

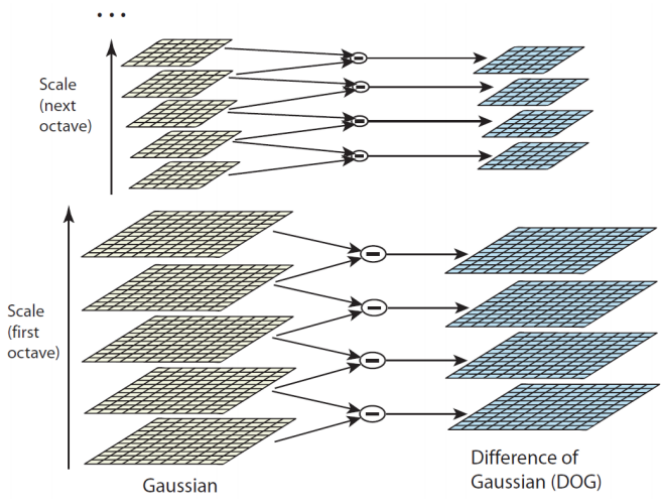

Fig. 3.1. Difference of Gaussian calculation

2) **Scale space detection of extreme value points**

A pixel on the DOG is compared with 8 pixels on this scale and 9 neighboring pixels on each of the upper and lower adjacent scales for a total of 26 pixel values to determine if it is a local maximum or minimum value. If it is the highest or lowest pixel among the neighboring pixels, it is determined that the pixel point is one of the extreme value points in the scale space,as shown in the figure below:

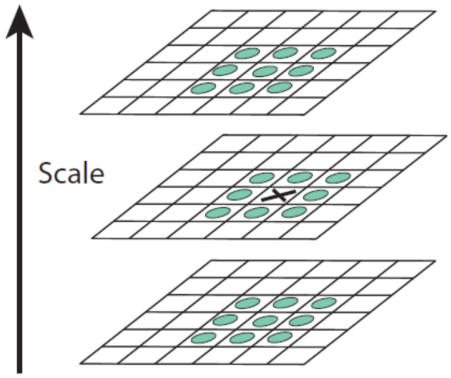

Fig. 3.2. Selecting DoG maxima

3) **Screening keypoints and eliminating unstable extreme points.**

At this point there are already scale invariant keypoints, but these keypoints should be screened to remove the keypoints of contrast degree and those at undesirable edges, and get the feature keypoints that meet the requirements.

SIFT detector : Calculate feature point orientation

1) Calculate the direction and amplitude of the neighborhood gradient

In order to achieve invariance of image rotation, an orientation reference needs to be derived from the local image structure of the detected feature points. SIFT uses the image gradient method to derive the stable orientation of this local structure.

Calculate the magnitude and amplitude of the image of the region centered on the feature point with the radius of $3 \times 1.5\sigma$. The mode $m(x, y)$ and the direction of the gradient$\theta(x, y)$ at each point$L(x, y)$ can be obtained by the following equation:

Gradient magnitude:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

Gradient direction: (3.3)

$$\theta(x, y) = \tan^{-1}\left[\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right]$$

2) Calculate the gradient direction histogram

After completing the gradient calculation of the Gaussian image of the feature point neighborhood, use the histogram to count the gradient direction and magnitude of the pixels in the neighborhood. The horizontal axis of the gradient direction histogram is the gradient direction angle, and the vertical axis is the cumulative value of the gradient amplitude corresponding to the gradient direction angle (with Gaussian weight). The gradient direction histogram divides the range from $0^\circ \sim 360^\circ$ into 36 columns, and every 10° is a column. The peak of the histogram represents the main direction of the image gradient in the neighborhood of the feature point, that is, the main direction of the feature point, as shown in the following figure.

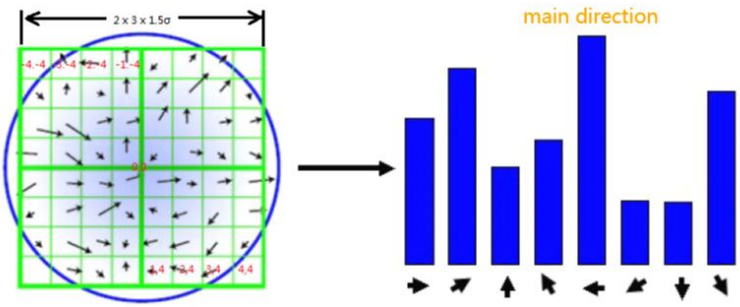

Fig. 3.3. Determination of the main direction of the feature point

3) Determine the direction of feature points

Once the histogram of gradient directions is available, the direction with the largest value in the histogram is considered as the primary direction of the feature point, and if there is another direction larger than the maximum value of 80%, the direction is considered as the secondary direction of the feature point. A feature may have more than one direction (one primary direction and more than one secondary direction), which can enhance the robustness of matching. Specifically, the feature point is copied into multiple copies ($x, y,$, are the same except for the direction θ).

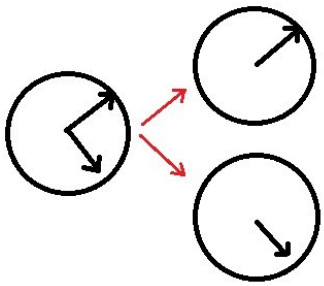

Fig. 3.4

SIFT detector feature point description (Approximate to HOG)

1) Correct the main direction of rotation to ensure rotation invariance

In order to ensure that the feature vector is rotationally invariant, it is necessary to center the $\left(m\sigma\left(B_p+1\right)\sqrt{2}\times m\sigma\left(B_p+1\right)\sqrt{2}\right)$ image gradient in the neighborhood near the feature point. is rotated by an orientation angle θ, which means that the original image x-axis is rotated to the same direction as the main direction. The rotation formula is as follows.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.4}$$

After rotating the position and direction of the neighborhood image gradient near the feature point, take the feature point as the center, and take a $m\sigma B_p \times m\sigma B_p$ size of the image area. And divide it into $B_p \times B_p$ sub-regions at equal intervals, each interval is $m\sigma$ pixel .

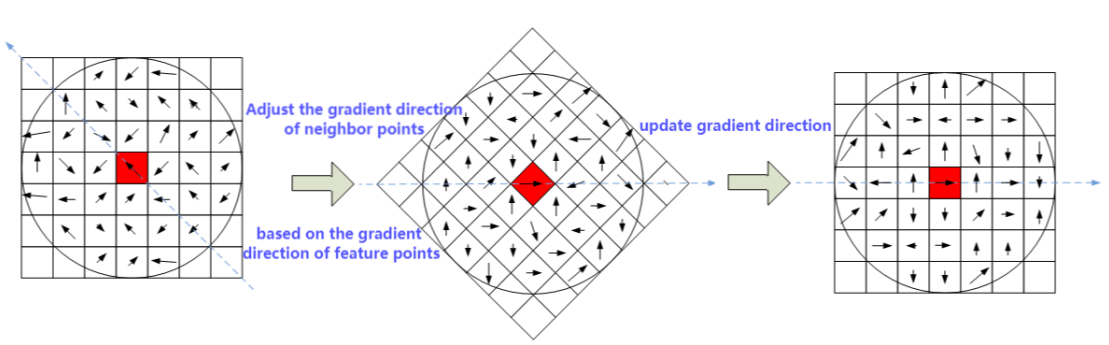

Fig. 3.5. Rotation of adjacent images around feature points

2) **Generate descriptors and finally form a 128-dimensional feature vector**

First take a 16×16 neighborhood around keypoints. This 16x16 region is further divided into 4x4 sub-blocks, and for each of these sub-blocks, a histogram is generated using magnitude and direction.

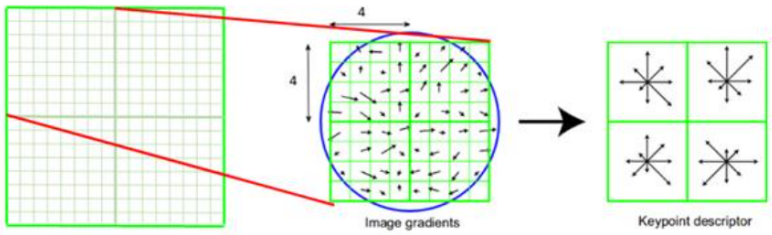

Fig. 3.6. Generation of SIFT descriptors

Different from finding the main direction, the gradient histogram of each seed area is divided into 8 direction intervals between 0 and 360, and each interval is 45 degrees, that is, each seed point has gradient strength information in 8 directions.

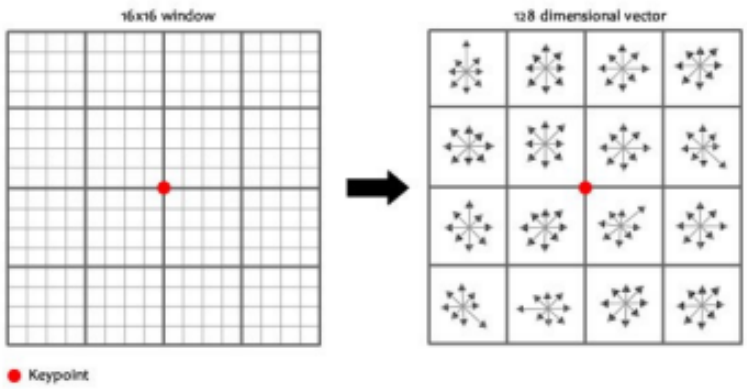

Fig. 3.7. SIFT descriptor

So, SIFT descriptor contains 16 histograms, and each histogram contains 8 bins, that gives total 128-dimensional vector for a feature point descriptor.

3) **Normalization processing(further remove the influence of lighting)**

The $4*4*8 = 128$ gradient information calculated above is the feature vector of the key point. After the feature vector is formed, in order to remove the influence of illumination on the descriptor, the gradient histogram is normalized. For the overall drift of the image gray value, the gradient of each point in the image is subtracted from the neighboring pixels, so it can also be removed.

So far, the SIFT feature description vector is generated.

§3.1.ii Theorem 2: ORB detector.

ORB detector [Rublee, Ethan, et al. "ORB: An efficient alternative to SIFT or SURF." 2011 International conference on computer vision. Ieee, 2011.] is a fusion of FAST feature point detector and BRIEF descriptor with many modifications to enhance the detector performance.

ORB detector

First it uses FAST detector to find feature points, then applies Harris corner measure to find top N-points among them. It also use pyramid to produce multiscale-features. Since the FAST feature point detector is not rotation invariant the following method is used to calculate the characteristic rotation of the point: the intensity weighted centroid of the patch with located corner at center is calculated. The direction of the vector from this corner point to centroid is considered as the orientation of the feature point. To improve the rotation invariance, moments are computed with x and y axis which should be in a circular region of radius r, where r is the size of the patch.
ORB uses BRIEF descriptors for its feature points. The BRIEF descriptor [6] is a bit string description of an image patch constructed from a set of binary intensity tests:

$$\tau(p, x, y) = \begin{cases} 1, p(x) < p(y) \\ 0, p(x) \geqslant p(y) \end{cases} \tag{3.5}$$

Then the BRIEF feature point descriptor defined as a binary can be calculated from set of simple binary intensity tests as following:

$$f_n(p) = \sum_{i=1}^{n} 2^{i-1} \tau\left(p, x_i, y_i\right) \tag{3.6}$$

To get a better performance of the BRIEF descriptor for rotated features, the descriptor is rotated according to the orientation of feature points. For any feature set of n binary tests at location (x_i, y_i), $S_{2\times n}$ matrix is defined, which contains the coordinates of these pixels.

$$S = \begin{pmatrix} x_1, \ldots, x_n \\ y_1, \ldots, y_1 \end{pmatrix} \tag{3.7}$$

Then using the orientation θ of a patch its rotation matrix R_θ is calculated and used to rotate the S matrix to get a rotated version S_θ.

$$S_\theta = R_\theta S \tag{3.8}$$

ORB quantize the angle to increments of $2\pi/30 = 12$ deg, so a lookup table of precomputed BRIEF patterns can be calculated for each possible angle. As long as the keypoint orientation θ is consistent across views, the correct set of points S_θ will be used to compute its descriptor.

$$g_n(p, \theta) = f_n(p) \mid \left(x_i, y_y\right) \in S_\theta \tag{3.9}$$

To compute a distance between two ORB descriptors a Hamming distance can be used. The multi-probe Locality-sensitive hashing (LSH) method is used for ORB descriptor matching.

§3.1.iii Solution

Feature points detection.

1.a Selection of the original image

Fig. 3.8. original image 1

Fig. 3.9. original image 2

Fig. 3.10. original image 3

1.b Image processing result

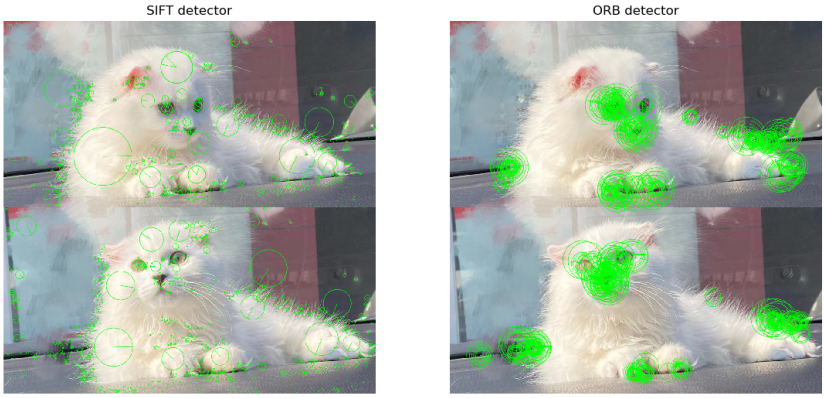

Fig. 3.11. result image 1

Fig. 3.12. result image 2

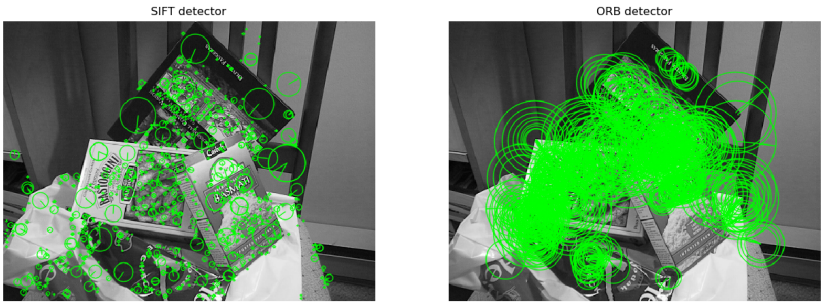

Fig. 3.13. result image 3

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ SIFT and ORB can be easily implemented based on opencv. From the experimental results, both methods detect a certain number of feature points
- ⋆ However, it can be seen from the experimental results that the feature points detected by **SIFT** are significantly **more than** those detected by **ORB**.

§3.2 Feature points matching.

§3.2.i Theorem.

Feature point descriptors matching : brute force method.
The simplest way to match two sets of feature points is by using a brute force method. In this case for each point of the first set an item of the second set is selected which have the smallest distance. For SIFT descriptor you can use the Euclidean L_2 distance. As for the ORB descriptor, since it is a binary mask, so the Hamming distance between descriptors should be used instead.
Obviously, the brute force method works slow.

Feature point descriptors matching : faster method.
To speed up the descriptor matching an accelerating structure should be build on top of the descriptors set, then when matching a descriptor from the search query it is compared not with whole set, but only with descriptors from some cluster. The simplest accelerating structure is KD-tree (kdimensional tree) that is built on the space of training set descriptors. In case if descriptor is defined as a binary mask, then it is preferable to use Locality-Sensitive Hashing (LSH) method.

filter some of the not strong matches : cross checking.
Using the first best match may result in a lot descriptor matches, however a lot of them are false matches due to the fact that some feature points are from repeating pattern (e.g., windows, water, clouds, etc.). There are two possible solution to filter some of the not strong matches.
First solution is to use the cross checking, that requires the descriptor to be matched in two directions: when matching two images it should be the best match in forward and in backward directions.

filter some of the not strong matches : k-nearest matching method.
First solution is to use the cross checking, that requires the descriptor to be matched in two directions: when matching two images it should be the best match in forward and in backward directions.
The second solution is to use the k-nearest matching method. In this case for each point several best matches are found sorted by the distance, and the match is considered to be «good if it is significantly different from the next nearest match, so the distance between first nearest match is significantly lower comparing to distance with the second nearest match:

$$D_1 < r \cdot D_2 \tag{3.10}$$

where D_1 is a distance to the first nearest match, D_2 is a distance to the second nearest match, and r is difference ratio, which is advised to be 0.75 by the SIFT method authors. Please note that the k-nearest matching method is not compatible with the cross-checking since the cross-check does not allow more than one match to be found.

calculate the geometric transformation between images(RANSAC).
Using accelerating structures and k-nearest filtering we could get a set of strong matches between images. Since when matching descriptors we did not took the feature point positions into an account, so the next step would be to calculate the geometric transformation between images taking into an account that there may still be lot of outliers or false matches. The most commonly used solution is to use the Random Sequence Consensus (RANSAC) method. The general idea of the method is to estimate not all data, but only a small sample, then build a hypothesis basing on this sample and check how correct this hypothesis is. After checking number of such hypothesis, we choose one that best fits with most of the data.

1) On input we have a set of pairs of matched feature point coordinates on two images: $S = \{(x, y)\} \mid x \in X, y \in Y$, where X is a first image, and Y is a second image.

2) For each i from 1 to N build a hypothesis and check it:

   a) We build a hypothesis θ_i by selecting random pairs $S_i = \{(x_i, y_i)\} \mid (x_i, y_i) \in S$. In our case it is enough to select 4 points from each of X and Y sets to build a matrix M for perspective transformation hypothesis.

   b) Evaluate the hypothesis θ_i by applying the perspective transformation matrix M to all points of the first X set and checking their matches with the points of the second Y set with some threshold. The number of matches is the hypothesis evaluation score $R(\theta_I)$:

$$\begin{aligned} R(\theta) &= \sum_{x \in X} p(\theta, x, Y), \\ p(\theta, x, Y) &= \begin{cases} 1, |\varepsilon(\theta, x, Y)| \leqslant T \\ 0, |\varepsilon(\theta, x, Y)| > T \end{cases} \end{aligned} \tag{3.11}$$

   where $\varepsilon(\theta, x)$ is the minimum distance from point x to points of the set Y with hypothesis θ.

   c) If this is the first hypothesis, then store it as a current best hypothesis θ_0. Else, check if the current hypothesis θ_i is better than the best one found before θ_0, and, if so, then it is stored as the new best hypothesis.

$$(i = 0) \vee (R(\theta_i) > R(\theta_0)) \Rightarrow \theta_0 = \theta_i \tag{3.12}$$

3) After finishing N iterations, the θ_0 stores the best hypothesis. In our case it is a perspective transformation matrix that transforms the first image to the coordinate system of the second image. The probability of choosing at least one sample without outliers with the RANSAC method can be estimated as following:

$$p = 1 - (1 - N(1 - e)^s)^N \tag{3.13}$$

   where p is the probability of getting a good sample in N iterations, N is the number of samples (iterations), s is the number of points in the sample, and e is the ratio of outliers.

Since after estimating at least one hypothesis we can estimate the ratio of outliers, this allows us to estimate required number of iterations basing on the currently best hypothesis:

$$N = \frac{\log(1-p)}{\log\left(1-(1-e)^s\right)} \tag{3.14}$$

The modification of RANSAC method that uses M-estimator to evaluate the hypothesis is called M-SAC. In this case each point score $p(\theta, x, Y)$ depends on the minimum distance from point x to points of the set Y with hypothesis θ :

$$\begin{aligned} R(\theta) &= \sum_{x \in X} p(\theta, x, Y), \\ p(\theta, x, Y) &= \begin{cases} \varepsilon^2(\theta, x, Y), |\varepsilon(\theta, x, Y)| \leqslant T \\ T^2, |\varepsilon(\theta, x, Y)| > T \end{cases} \end{aligned} \tag{3.15}$$

§3.2.ii Solution

1.Feature points matching.

1.a Selection of the original image

Choose two groups of pictures, one is a little easier (Fig.3.14. There are scale changes, no perspective changes, and no occlusion), and the other is more difficult (Fig.3.15. There are scale changes, perspective changes, and occlusions). See if matching images with SIFT and ORB descriptors works well:

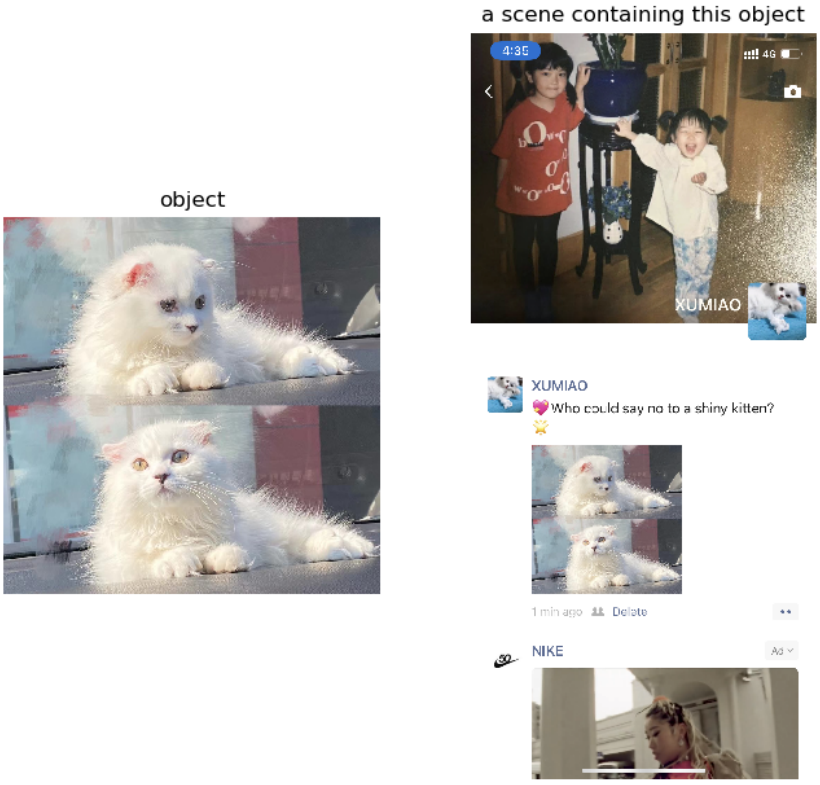

Fig. 3.14. original image 1

Fig. 3.15. original image 2

1.b Image processing result

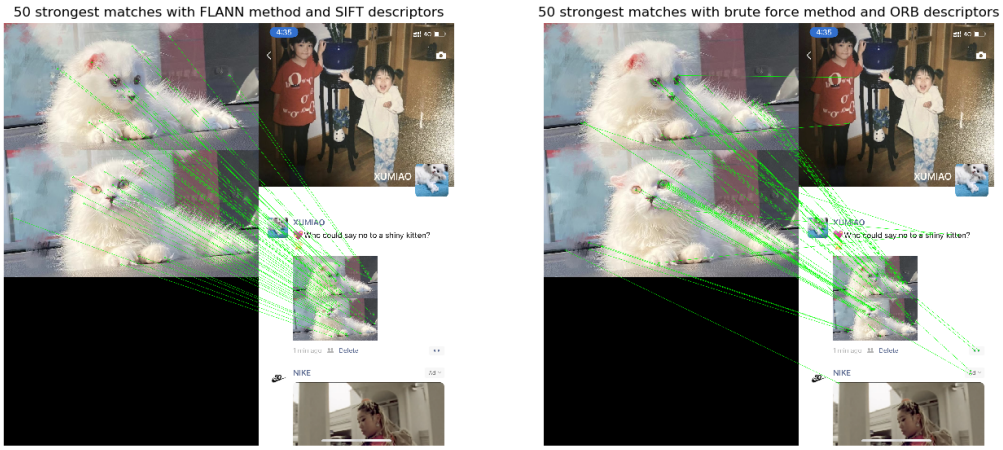

Fig. 3.16. 50 strongest matches with FLANN method(original image 1)

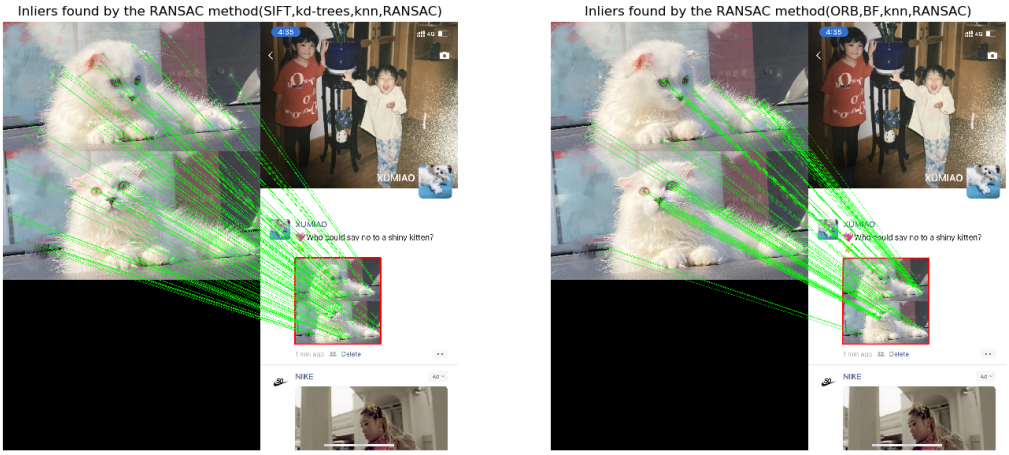

Fig. 3.17. lnliers found by the RANSAC method(original image 1)

Fig. 3.18. Detected object (original image 1)

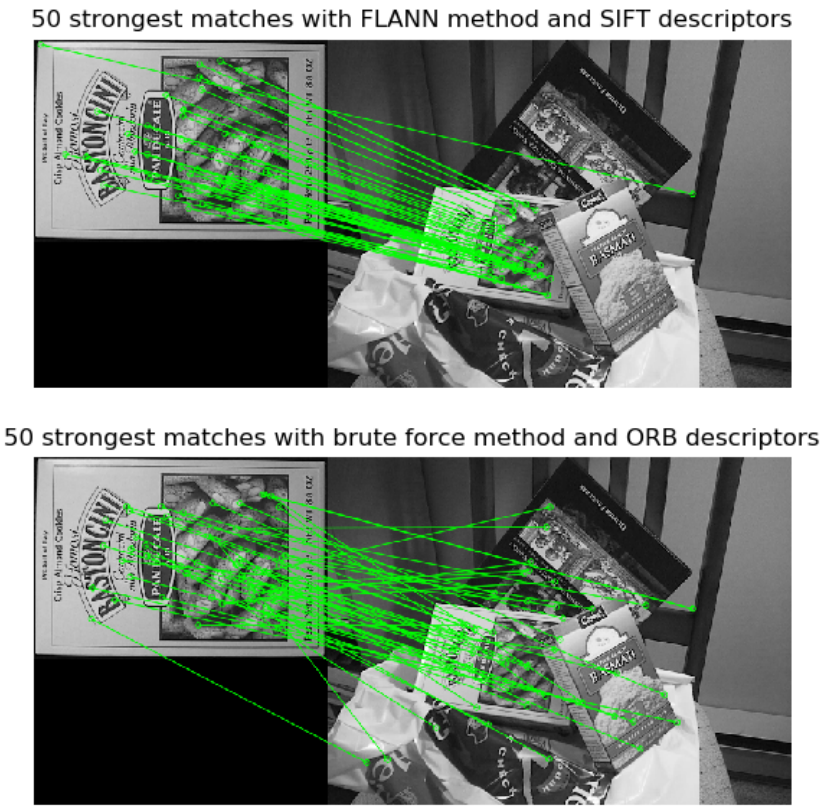

Fig. 3.19. 50 strongest matches with FLANN method(original image 2)

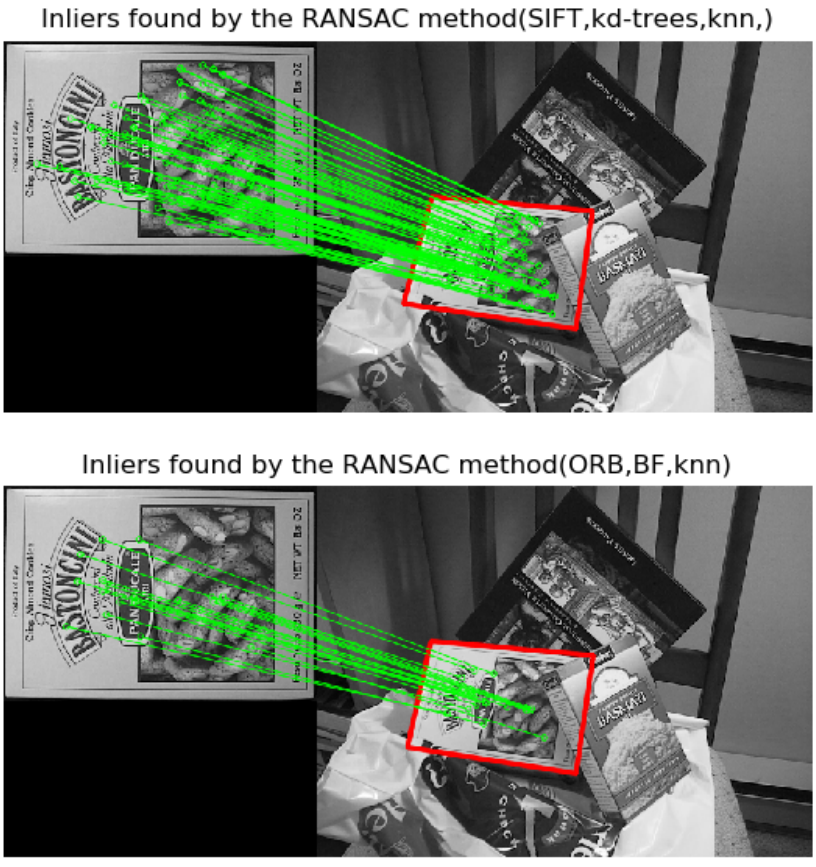

Fig. 3.20. lnliers found by the RANSAC method(original image 2)

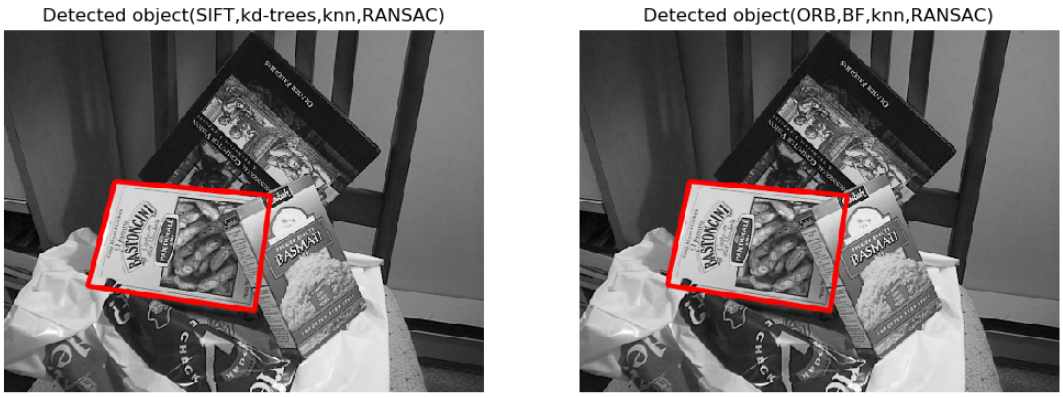

Fig. 3.21. Detected object (original image 2)

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ For image matching, both SIFT and ORB perform better. From the experimental results (Fig.3.17, Fig.3.18, Fig.3.20, Fig.3.21), **the matching results are relatively accurate.**
- ⋆ Observing the matching results without RANSAC at the beginning, SIFT is obviously due to ORB, and there are many wrong matching in the feature point matching detected by ORB (Fig3.16, Fig3.19)
- ⋆ But after RANSAC processing (Fig3.17, Fig3.20), the matching results of both descriptors are better
- ⋆ From the processing results of the original image 2 (Fig3.20, Fig3.21), it can be seen that the matching using the two descriptors can adapt to the matching object occlusion, angle change, size change, etc.

§3.3 Optional.simple automatic image stitching.

§3.3.i Solution

1.Stitch two images

1.a Selection of the original image

Pick two pictures to be stitched (with angle change Fig.3.22)

Fig. 3.22. original image

1.b Image processing result

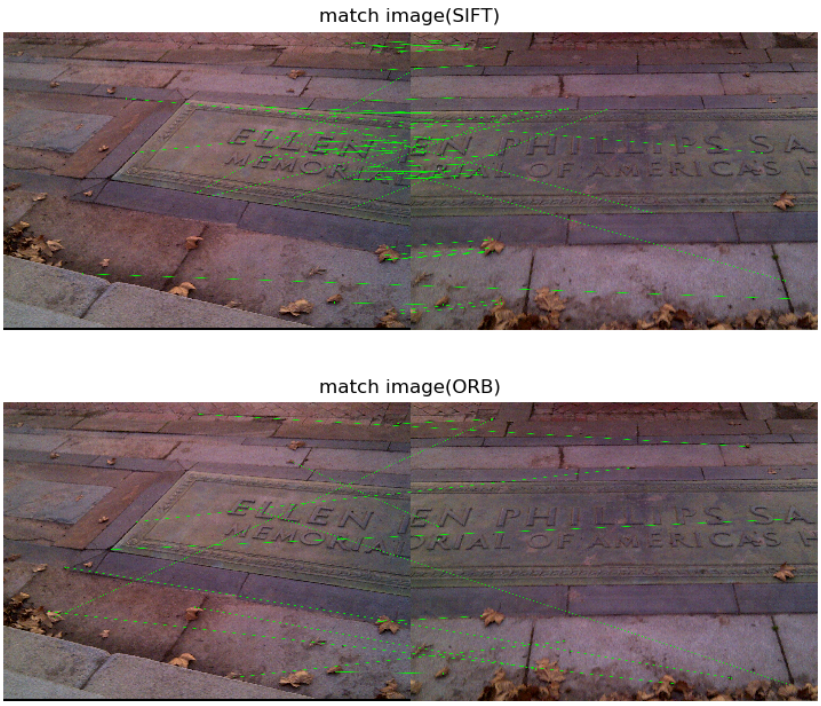

Fig. 3.23. match images

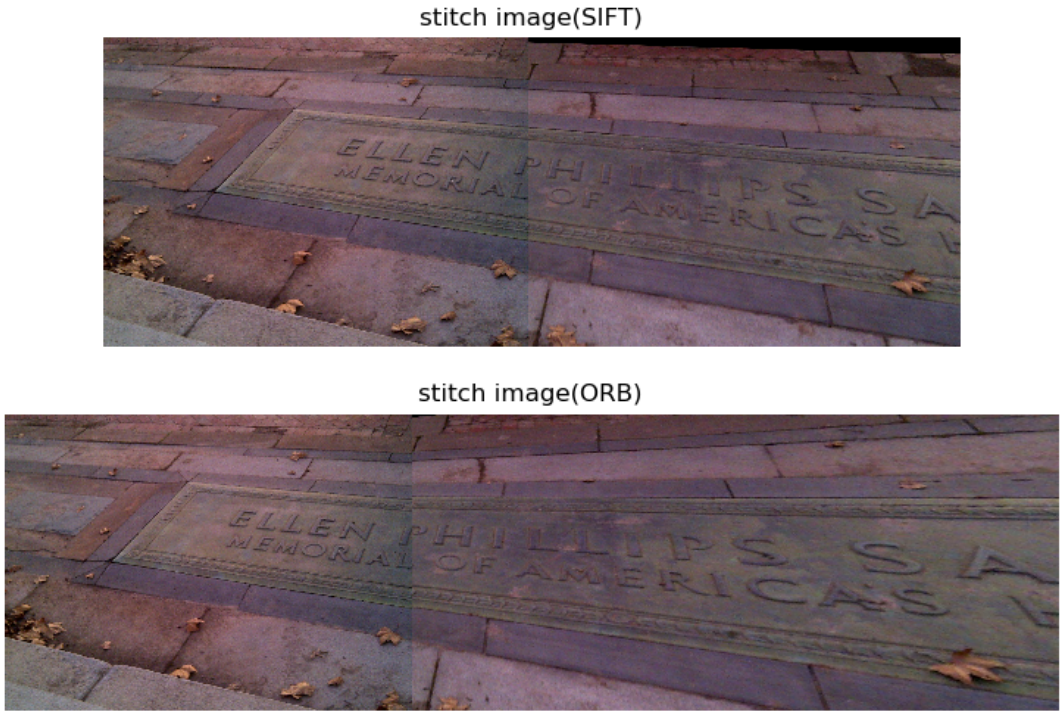

Fig. 3.24. Stitched images

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

2.Stitch three images

To stitch three images, we can refer to the stitching of two images, first stitch the two images on the left and the middle, and then stitch the remaining images

2.a Selection of the original image

Select three panoramic images to be stitched:

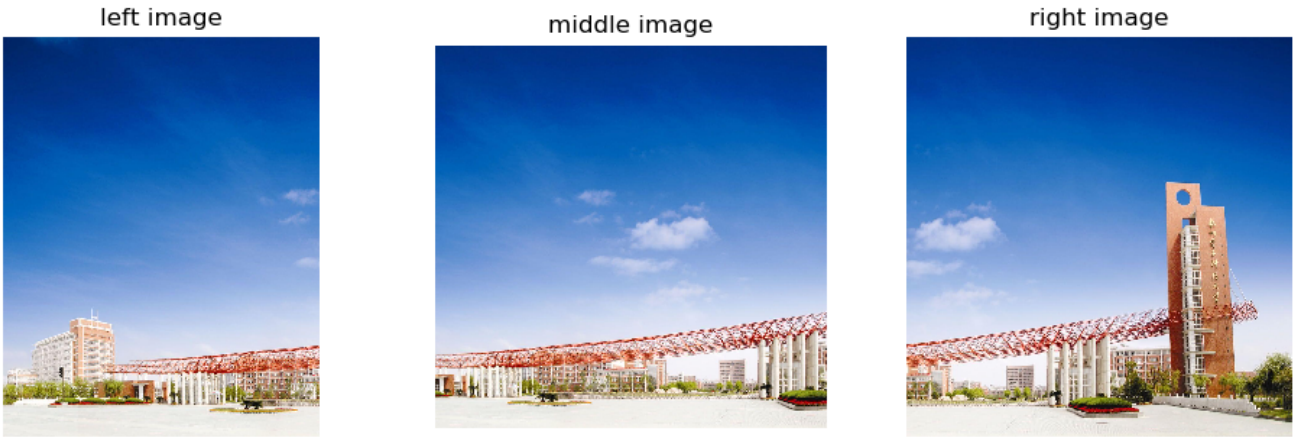

Fig. 3.25. original image

2.b Image processing result

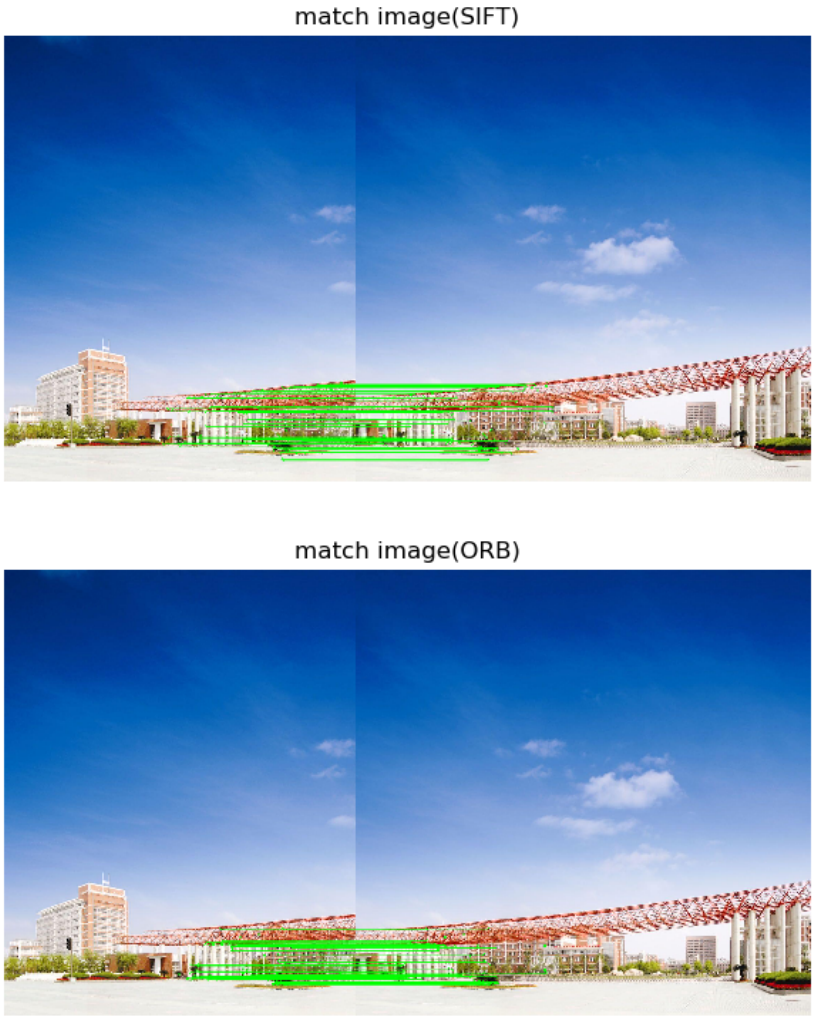

Fig. 3.26. Match the first two images

Fig. 3.27. Stitching of the first two images

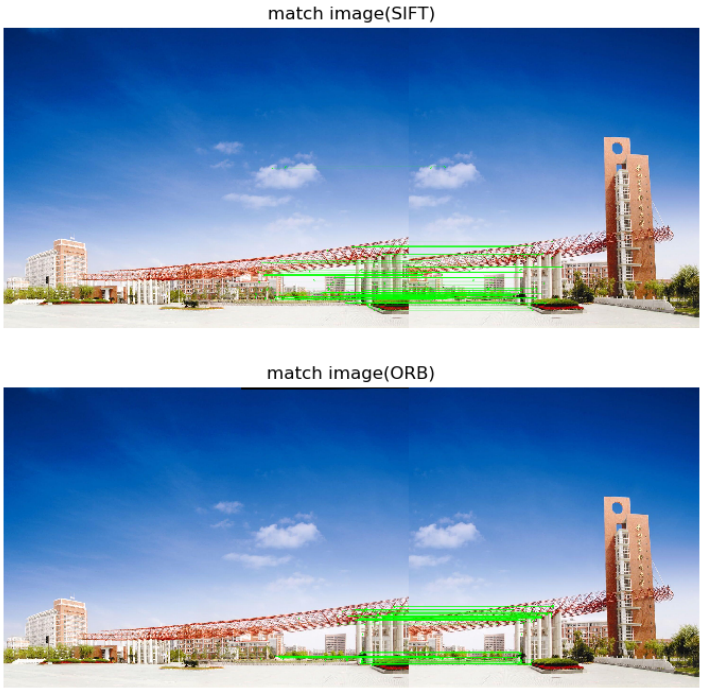

Fig. 3.28. The image on the right matches

Fig. 3.29. Three image stitching results

2.c full source code

You can click ⋆here⋆ to see the full code for this part.

3.Comments

- ⋆ From the experimental results, for the stitching of images without shooting angle changes, both SIFT and ORB descriptors perform better, and no stitching marks can be seen on the resulting images (Fig.3.27, Fig.3.29).
- ⋆ However, for images with changing shooting angles, the images matched with ORB descriptors have some angular shifts (Fig. 3.24). It may be caused by the shift in the matching positions of some feature points.

4 Conclusion

In this exercise assignment, I learned the following application methods:

- ★ There are a lot of algorithms designed to detect and descript feature points, including:
  1) Harris corner detector
  2) Shi-Tomasi corner detector
  3) Scale-Invariant Feature Transform (SIFT) detector and descriptor
  4) Speeded-Up Robust Features (SURF) detector and descriptor
  5) Features from Accelerated Segment Test (FAST) detector
  6) Binary Robust Independent Elementary Features (BRIEF) descriptor
  7) Oriented FAST and Rotated BRIEF (ORB) detector and descriptor
- ★ Feature point detection can be widely used in
  1) Moving target tracking
  2) Object Recognition
  3) Image Registration
  4) Panoramic image stitching
  5) 3D Reconstruction

Note: The characteristics of each method are described in the previous comments and will not be repeated here

5 The answer to the questions

1) How the characteristic orientation (rotation) of the feature point can be estimated?

After completing the gradient calculation of the Gaussian image of the feature point neighborhood, use the histogram to count the gradient direction and magnitude of the pixels in the neighborhood. The horizontal axis of the gradient direction histogram is the gradient direction angle, and the vertical axis is the cumulative value of the gradient amplitude corresponding to the gradient direction angle (with Gaussian weight). The gradient direction histogram divides the range from $0^{\circ} \sim 360^{\circ}$ into 36 columns, and every 10° is a column. **The peak of the histogram** represents the **main direction** of the image gradient in the neighborhood of the feature point, that is, the main direction of the feature point, as shown in the following figure.

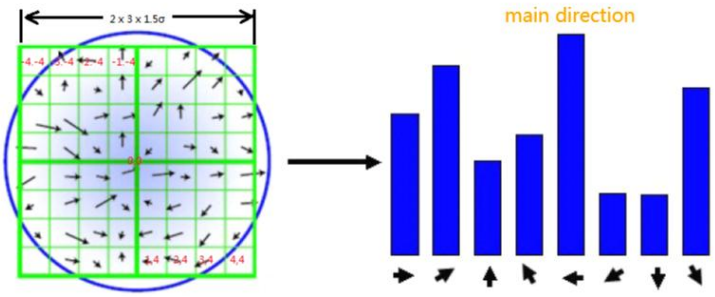

Fig. 5.1. Determination of the main direction of the feature point

The main direction is the feature point direction of the feature point

2) How to filter the not-strong feature point descriptor matches on a repeating texture (e.g., windows, water, etc.)?

After using brute force matching or knn matching, calculate the **<distance>** between two matching descriptors and **sort** them, filter and delete matching descriptors with larger distances

3) What is the minimum required sample size (the number of matched pairs of feature points) to build an affine transformation hypothesis with RANSAC method? What is the minimum required sample size to build a perspective transformation hypothesis with RANSAC method?

The formula for affine transformation is as follows:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} a_{13} \\ a_{21} \end{pmatrix} \tag{5.1}$$

As can be seen from the above formula, there are a total of 6 unknown parameters

in the transformation. Therefore, at least **3 pairs of matching points** are required for the solution (but please note: there will be mismatches in the matching points. If there are fewer matching points, the parameter calculation results will be inaccurate, and the affine transformation results will be quite different. In general, There are more strong match points the better)**(This is true for both affine transformation h and perspective transformation)**

4) How to use feature points for stitching a panoramic image?

We can perform image stitching through the following algorithm:
Image Stitching Algorithms:

1) Match image feature points

2) filter weak matches

3) Using RANSAC method to calculate the matrix M of affine transformation between matching points

4) Take two images as an example: Calculate the position of the upper right corner and lower right corner of the right image after affine matrix transformation to determine the size of the spliced image, and perform affine transformation on the right image under this size.

   The calculation formula is as follows:

$$
\begin{aligned}
&\mathrm{dst}(x, y)=\mathrm{src}\left(\frac{M_{11} x+M_{12} y+M_{13}}{M_{31} x+M_{32} y+M_{33}}, \frac{M_{21} x+M_{22} y+M_{23}}{M_{31} x+M_{32} y+M_{33}}\right) \\
&\left(x_{1 \text{ new}}^{\text{right}}, y_{1 \text{ new}}^{\text{right}}\right)=\mathrm{dst}\left(x_{1}^{\text{right}}, y_{1}^{\text{right}}\right) \\
&\left(x_{2 \text{ new}}^{\text{right}}, y_{2 \text{ new}}^{\text{right}}\right)=\mathrm{dst}\left(x_{2}^{\text{right}}, y_{2}^{\text{right}}\right) \\
&x_{\text{size}}, y_{\text{size}}=\left(\min \left(x_{1} \text{ new}, x_{2} \text{ new}\right), y_{1}^{\text{left}}\right)
\end{aligned}
\tag{5.2}
$$

5) Overlay the image on the left to the [0:w _left,0:h_left] area on the image on the right,The algorithm flow is shown in the following figure:

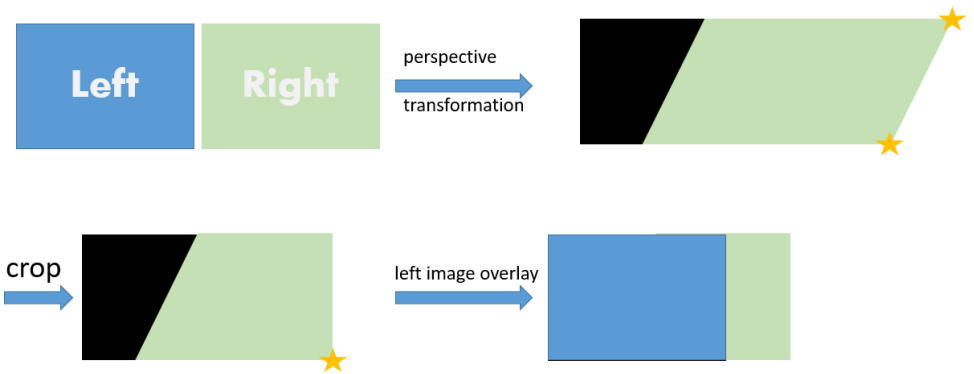

Fig. 5.2. Image Stitching Algorithms

I

Appendix

A Complete source code

§A.1 Feature points detection.

and you can click ⋆here⋆ to return to reading the report

Python

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

def SIFT_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    Ifp, Ides = sift.detectAndCompute(Igray,None)
    image_SIFT =image.copy()
    image_SIFT = cv2.drawKeypoints(image,Ifp,image_SIFT,color =
       (0,255,0),flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    print('The number of feature points detected by SIFT:',len(Ifp))
    return Ifp, Ides ,image_SIFT

def ORB_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create()
    Ifp,Ides = orb.detectAndCompute(Igray,None)
    image_ORB = cv2.drawKeypoints(image,Ifp,None,color = (0,255,0),
       flags= cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    print('The number of feature points detected by ORB',len(Ifp))
    return Ifp, Ides ,image_ORB
#
```

```
I = cv2.imread('MATCH8.png')
Ifp_SIFT, Ides_SIFT ,I_SIFT = SIFT_detect(I)
Ifp_ORB, Ides_ORB ,I_ORB = ORB_detect(I)
I_SIFT = cv2.cvtColor(I_SIFT,cv2.COLOR_BGR2RGB)
I_ORB = cv2.cvtColor(I_ORB,cv2.COLOR_BGR2RGB)
plt.subplot(121),plt.title('SIFT detector'),plt.axis ('off')
plt.imshow(I_SIFT)
plt.subplot(122),plt.title('ORB detector'),plt.axis ('off')
plt.imshow(I_ORB)
plt.show()
plt.waitforbuttonpress
```

§A.2 Feature points matching.

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def SIFT_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    Ifp, Ides = sift.detectAndCompute(Igray,None)
    image_SIFT =image.copy()
    image_SIFT = cv2.drawKeypoints(image,Ifp,image_SIFT,color =
       (0,255,0),flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return Ifp, Ides ,image_SIFT


def ORB_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
```

```
    orb = cv2.ORB_create(10000)
    Ifp,Ides = orb.detectAndCompute(Igray,None)
    image_ORB = cv2.drawKeypoints(image,Ifp,None,color = (0,255,0),
        flags= cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return Ifp, Ides ,image_ORB

def knn_RANSAC_match(I1,I1fp,I1des,I2,I2fp,I2des):
    '''
    Imatch,Itrans = SIFT_ANNmatch(I1,I1fp,I1des,I2,I2fp,I2des)
    I1,I2 :BGR original image
    I1fp,I2fp : feature points
    I1des, I2des : feature points corresponding descriptors
    Imatch : 100 strongest matches(FLANN)
    Itrans : Inliers found by the RANSAC method
    '''
    #Algorithm Parameter Setting
    #FLANN_INDEX_KDTREE = 1
    #index_params = dict(algorithm = FLANN_INDEX_KDTREE,trees = 5)
    #matcher = cv2.FlannBasedMatcher(index_params, dict())
    matcher = cv2.BFMatcher ( crossCheck = False )
    #find KNN matches with k = 2
    matches = matcher.knnMatch(I1des, I2des, k = 2)
    # Select good matches
    knn_ratio = 0.8
    good = []
    for m in matches:
        if len(m) > 1:
            if m[0].distance < knn_ratio *m[1].distance:
                good.append(m[0])
    matches = good
    # Displaying top 50 matches
    num_matches = 50
    matches = sorted(matches,key = lambda x:x.distance)
    Imatch = cv2.drawMatches(I1,I1fp,I2,I2fp,matches[:num_matches],None
        ,flags = cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,matchColor=
        (0,255,0))
    #Executing RANSAC to calculate the transformation matrix
    MIN_MATCH_COUNT = 10
    if len(matches) <MIN_MATCH_COUNT:
        print('no enough matches.')
    else:
        #create arrays of point coordinates
        I1pts = np.float32([I1fp[m.queryIdx].pt for m in matches]).
            reshape(-1,1,2)
        I2pts = np.float32([I2fp[m.trainIdx].pt for m in matches]).
            reshape(-1,1,2)
        #Run RANSAC method
        M,mask = cv2.findHomography(I1pts,I2pts,cv2.RANSAC,5)
        mask = mask.ravel().tolist()
    h,w = I1.shape[:2]
```

```
    I1box = np.float32([[0,0],[0,h-1],[w-1,h-1],[w-1,0]]).reshape
        (-1,1,2)
    I1to2box = cv2.perspectiveTransform(I1box,M)
    #Draw a red box on the second image
    I2res = cv2.polylines(I2,[np.int32(I1to2box)],True,(0,0,255),3,cv2.
        LINE_AA)
    Itrans = cv2.drawMatches(I1,I1fp,I2res,I2fp,matches,None,
        matchesMask= mask,flags= cv2.
        DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,matchColor=(0,255,0))
    return Imatch,Itrans,I2res

#ORB
I11 = cv2.imread('match7.png')
I21 = cv2.imread('match8.png')
I12 = cv2.imread('match7.png')
I22 = cv2.imread('match8.png')

I1fp_SIFT,I1des_SIFT,I1_SIFT = SIFT_detect(I11)
I2fp_SIFT,I2des_SIFT,I2_SIFT = SIFT_detect(I21)
I1fp_ORB,I1des_ORB,I1_ORB = ORB_detect(I12)
I2fp_ORB,I2des_ORB,I2_ORB = ORB_detect(I22)

Imatch_SIFT,Itrans_SIFT,Ires_SIFT = knn_RANSAC_match(I11,I1fp_SIFT,
    I1des_SIFT,I21,I2fp_SIFT,I2des_SIFT)
Imatch_ORB,Itrans_ORB,Ires_ORB = knn_RANSAC_match(I12,I1fp_ORB,
    I1des_ORB,I22,I2fp_ORB,I2des_ORB)

Imatch_SIFT = cv2.cvtColor(Imatch_SIFT,cv2.COLOR_BGR2RGB)
Itrans_SIFT = cv2.cvtColor(Itrans_SIFT,cv2.COLOR_BGR2RGB)
Ires_SIFT = cv2.cvtColor(Ires_SIFT,cv2.COLOR_BGR2RGB)
Imatch_ORB = cv2.cvtColor(Imatch_ORB,cv2.COLOR_BGR2RGB)
Itrans_ORB = cv2.cvtColor(Itrans_ORB,cv2.COLOR_BGR2RGB)
Ires_ORB = cv2.cvtColor(Ires_ORB,cv2.COLOR_BGR2RGB)

plt.subplot(2,1,1)
plt.imshow(Imatch_SIFT),plt.title('50 strongest matches with FLANN
    method and SIFT descriptors'),plt.axis ('off')
plt.subplot(2,1,2)
plt.imshow(Imatch_ORB),plt.title('50 strongest matches with brute force
     method and ORB descriptors'),plt.axis ('off')
plt.show()
plt.waitforbuttonpress
plt.subplot(2,1,1)
plt.imshow(Itrans_SIFT),plt.title('Inliers found by the RANSAC method(
    SIFT,kd-trees,knn,RANSAC)'),plt.axis ('off')
plt.subplot(2,1,2)
plt.imshow(Itrans_ORB),plt.title('Inliers found by the RANSAC method(
    ORB,BF,knn,RANSAC)'),plt.axis ('off')
plt.show()
plt.waitforbuttonpress
```

```
plt.subplot(1,2,1)
plt.imshow(Ires_SIFT),plt.title('Detected object(SIFT,kd-trees,knn,
    RANSAC)'),plt.axis ('off')
plt.subplot(1,2,2)
plt.imshow(Ires_ORB),plt.title('Detected object(ORB,BF,knn,RANSAC)'),
    plt.axis ('off')
plt.show()
plt.waitforbuttonpress
```

§A.3 Stitching 2 images

and you can click ⋆here⋆ to return to reading the report

Problem 1

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def SIFT_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    Ifp, Ides = sift.detectAndCompute(Igray,None)
    image_SIFT =image.copy()
    image_SIFT = cv2.drawKeypoints(image,Ifp,image_SIFT,color =
        (0,255,0),flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return Ifp, Ides ,image_SIFT

def ORB_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create(1000)
    Ifp,Ides = orb.detectAndCompute(Igray,None)
    image_ORB = cv2.drawKeypoints(image,Ifp,None,color = (0,255,0),
        flags= cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

```
33      return Ifp, Ides ,image_ORB
34
35
36  def point_transfor(p1,Mr2l):
37      p1x = (Mr2l[0][0]*p1[0] + Mr2l[0][1]*p1[1] + Mr2l[0][2]) / ((Mr2l
            [2][0]*p1[0] + Mr2l[2][1]*p1[1] + Mr2l[2][2]))
38      p1y = (Mr2l[1][0]*p1[0] + Mr2l[1][1]*p1[1] + Mr2l[1][2]) / ((Mr2l
            [2][0]*p1[0] + Mr2l[2][1]*p1[1] + Mr2l[2][2]))
39      p1_after = (int(p1x), int(p1y)) # after transformation
40      return p1_after
41
42
43  def stitch(img_left,left_fp,left_des,img_right,right_fp,right_des):
44      #Algorithm Parameter Setting
45      #FLANN_INDEX_KDTREE = 1
46      #index_params = dict(algorithm = FLANN_INDEX_KDTREE,trees = 5)
47      #matcher = cv2.FlannBasedMatcher(index_params, dict())
48      #find KNN matvhes with k = 2
49      matcher = cv2.BFMatcher ( crossCheck = False )
50      matches = matcher.knnMatch(left_des, right_des, k = 2)
51      # Select good matches
52      knn_ratio = 0.75
53      good = []
54      for m in matches:
55          if len(m) > 1:
56              if m[0].distance < knn_ratio *m[1].distance:
57                  good.append(m[0])
58      matches = good
59      num_matches = 50
60      matches = sorted(matches,key = lambda x:x.distance)
61      Imatch = cv2.drawMatches(img_left,left_fp,img_right,right_fp,
            matches[:num_matches],None,flags = cv2.
            DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,matchColor= (0,255,0))
62      #Executing RANSAC to calculate the transformation matrix
63      MIN_MATCH_COUNT = 4
64      if len(matches) <MIN_MATCH_COUNT:
65          print('no enough matches.')
66      else:
67          #create arrays of point coordinates
68          I1pts = np.float32([left_fp[m.queryIdx].pt for m in matches]).
                reshape(-1,1,2)
69          I2pts = np.float32([right_fp[m.trainIdx].pt for m in matches]).
                reshape(-1,1,2)
70          #Run RANSAC method
71          Mr2l,mask = cv2.findHomography(I2pts,I1pts,cv2.RANSAC,5)
72      h_left,w_left = img_left.shape[:2]
73      h_right,w_right = img_right.shape[:2]
74      #image = np.zeros((max(h_left,h_right),w_left+w_right,3),dtype= '
            uint8')
75      #image[0:h_right,0:w_right] = img_right
```

```
    p1 = (w_right,0) # Coordinates of the upper right point
    p2 =(w_right,h_right) # Coordinates of the lower right point
    p1_after = point_transfor(p1,Mr2l)
    p2_after = point_transfor(p2,Mr2l)
    print(p1_after)
    image = cv2.warpPerspective(img_right,Mr2l,(min(p1_after[0],
       p2_after[0]),img_left.shape[0]))
    image[0:h_left,0:w_left] = img_left
    return Imatch,image


I1 = cv2.imread('SIFT1.png')
I2 = cv2.imread('SIFT2.png')
I1fp_SIFT,I1des_SIFT,I1_SIFT = SIFT_detect(I1)
I2fp_SIFT,I2des_SIFT,I2_SIFT = SIFT_detect(I2)
Imatch_SIFT,I_stitch_SIFT = stitch(I1,I1fp_SIFT,I1des_SIFT,I2,I2fp_SIFT
   ,I2des_SIFT)
I1fp_ORB,I1des_ORB,I1_ORB= ORB_detect(I1)
I2fp_ORB,I2des_ORB,I2_ORB = ORB_detect(I2)
Imatch_ORB,I_stitch_ORB = stitch(I1,I1fp_ORB,I1des_ORB,I2,I2fp_ORB,
   I2des_ORB)

I_stitch_SIFT = cv2.cvtColor(I_stitch_SIFT,cv2.COLOR_BGR2RGB)
Imatch_SIFT = cv2.cvtColor(Imatch_SIFT,cv2.COLOR_BGR2RGB)
I_stitch_ORB = cv2.cvtColor(I_stitch_ORB,cv2.COLOR_BGR2RGB)
Imatch_ORB = cv2.cvtColor(Imatch_ORB,cv2.COLOR_BGR2RGB)

plt.subplot(2,1,1)
plt.imshow(Imatch_SIFT),plt.title('match image(SIFT)'),plt.axis ('off')
plt.subplot(2,1,2)
plt.imshow(Imatch_ORB),plt.title('match image(ORB)'),plt.axis ('off')
plt.show()
plt.waitforbuttonpress
plt.subplot(2,1,1)
plt.imshow(I_stitch_SIFT),plt.title('stitch image(SIFT)'),plt.axis ('
   off')
plt.subplot(2,1,2)
plt.imshow(I_stitch_ORB),plt.title('stitch image(ORB)'),plt.axis ('off'
   )
plt.show()
plt.waitforbuttonpress
```

§A.4 Stitching 3 images

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
```

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

def ORB_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create(1000)
    Ifp,Ides = orb.detectAndCompute(Igray,None)
    image_ORB = cv2.drawKeypoints(image,Ifp,None,color = (0,255,0),
       flags= cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return Ifp, Ides ,image_ORB
def SIFT_detect(image):
    '''
    Ifp,Ides,iamge_SIFT = SIFT_detect(image)
    image : BGR image
    Ifp : feature points
    Ides : feature points corresponding descriptors
    imageSIFT :image displaying SIFT feature points
    in green color with scale and orientation
    '''
    Igray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    Ifp, Ides = sift.detectAndCompute(Igray,None)
    image_SIFT =image.copy()
    image_SIFT = cv2.drawKeypoints(image,Ifp,image_SIFT,color =
       (0,255,0),flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return Ifp, Ides ,image_SIFT

def point_transfor(p1,Mr2l):
    p1x = (Mr2l[0][0]*p1[0] + Mr2l[0][1]*p1[1] + Mr2l[0][2]) / ((Mr2l
       [2][0]*p1[0] + Mr2l[2][1]*p1[1] + Mr2l[2][2]))
    p1y = (Mr2l[1][0]*p1[0] + Mr2l[1][1]*p1[1] + Mr2l[1][2]) / ((Mr2l
       [2][0]*p1[0] + Mr2l[2][1]*p1[1] + Mr2l[2][2]))
    p1_after = (int(p1x), int(p1y)) # after transformation
    return p1_after


def stitch(img_left,left_fp,left_des,img_right,right_fp,right_des):
    #Algorithm Parameter Setting
    #FLANN_INDEX_KDTREE = 1
    #index_params = dict(algorithm = FLANN_INDEX_KDTREE,trees = 5)
    #matcher = cv2.FlannBasedMatcher(index_params, dict())
    #find KNN matvhes with k = 2
```

```
    matcher = cv2.BFMatcher ( crossCheck = False )
    matches = matcher.knnMatch(left_des, right_des, k = 2)
    # Select good matches
    knn_ratio = 0.75
    good = []
    for m in matches:
        if len(m) > 1:
            if m[0].distance < knn_ratio *m[1].distance:
                good.append(m[0])
    matches = good
    num_matches = 100
    matches = sorted(matches,key = lambda x:x.distance)
    Imatch = cv2.drawMatches(img_left,left_fp,img_right,right_fp,
       matches[:num_matches],None,flags = cv2.
       DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,matchColor= (0,255,0))
    #Executing RANSAC to calculate the transformation matrix
    MIN_MATCH_COUNT = 10
    if len(matches) <MIN_MATCH_COUNT:
        print('no enough matches.')
    else:
        #create arrays of point coordinates
        I1pts = np.float32([left_fp[m.queryIdx].pt for m in matches]).
           reshape(-1,1,2)
        I2pts = np.float32([right_fp[m.trainIdx].pt for m in matches]).
           reshape(-1,1,2)
        #Run RANSAC method
        Mr2l,mask = cv2.findHomography(I2pts,I1pts,cv2.RANSAC,5)
    h_left,w_left = img_left.shape[:2]
    h_right,w_right = img_right.shape[:2]
    #image = np.zeros((max(h_left,h_right),w_left+w_right,3),dtype= '
       uint8')
    #image[0:h_right,0:w_right] = img_right
    p1 = (w_right,0) # Coordinates of the upper right point
    p2 =(w_right,h_right) # Coordinates of the lower right point
    p1_after = point_transfor(p1,Mr2l)
    p2_after = point_transfor(p2,Mr2l)
    print(p1_after)
    image = cv2.warpPerspective(img_right,Mr2l,(min(p1_after[0],
       p2_after[0]),img_left.shape[0]))
    image[0:h_left,0:w_left] = img_left
    return Imatch,image

I1 = cv2.imread('SIFT9.png')
I2 = cv2.imread('SIFT10.png')
I3 = cv2.imread('SIFT11.png')
I1fp_ORB ,I1des_ORB ,I1_ORB = ORB_detect(I1)
I2fp_ORB ,I2des_ORB ,I2_ORB = ORB_detect(I2)
I3fp_ORB,I3des_ORB,I3_ORB = ORB_detect(I3)
I1fp_SIFT,I1des_SIFT,I1_SIFT = SIFT_detect(I1)
I2fp_SIFT,I2des_SIFT,I2_SIFT = SIFT_detect(I2)
```

```
92 I3fp_SIFT,I3des_SIFT,I3_SIFT = SIFT_detect(I3)
93
94 Imatch_ORB ,I_stitch_ORB  = stitch(I1,I1fp_ORB ,I1des_ORB ,I2,I2fp_ORB
       ,I2des_ORB )
95 Istitch1fp_ORB ,Istitch1des_ORB ,Istitch1_ORB = ORB_detect(I_stitch_ORB
       )
96 Isum_ORB ,Isum_stitch_ORB  = stitch(I_stitch_ORB ,Istitch1fp_ORB ,
       Istitch1des_ORB ,I3,I3fp_ORB,I3des_ORB)
97 Imatch_SIFT,I_stitch_SIFT = stitch(I1,I1fp_SIFT,I1des_SIFT,I2,I2fp_SIFT
       ,I2des_SIFT)
98 Istitch1fp_SIFT,Istitch1des_SIFT,Istitch1_SIFT = SIFT_detect(
       I_stitch_SIFT)
99 Isum_SIFT,Isum_stitch_SIFT = stitch(I_stitch_SIFT,Istitch1fp_SIFT,
       Istitch1des_SIFT,I3,I3fp_SIFT,I3des_SIFT)
100
101 Isum_stitch_ORB  = cv2.cvtColor(Isum_stitch_ORB ,cv2.COLOR_BGR2RGB)
102 Isum_ORB  = cv2.cvtColor(Isum_ORB ,cv2.COLOR_BGR2RGB)
103 Imatch_ORB  = cv2.cvtColor(Imatch_ORB ,cv2.COLOR_BGR2RGB)
104 I_stitch_ORB  = cv2.cvtColor(I_stitch_ORB ,cv2.COLOR_BGR2RGB)
105 Isum_stitch_SIFT = cv2.cvtColor(Isum_stitch_SIFT,cv2.COLOR_BGR2RGB)
106 Isum_SIFT = cv2.cvtColor(Isum_SIFT,cv2.COLOR_BGR2RGB)
107 Imatch_SIFT = cv2.cvtColor(Imatch_SIFT,cv2.COLOR_BGR2RGB)
108 I_stitch_SIFT = cv2.cvtColor(I_stitch_SIFT,cv2.COLOR_BGR2RGB)
109
110 plt.subplot(2,1,1)
111 plt.imshow(Imatch_SIFT),plt.title('match image(SIFT)'),plt.axis ('off')
112 plt.subplot(2,1,2)
113 plt.imshow(Imatch_ORB ),plt.title('match image(ORB)'),plt.axis ('off')
114 plt.show()
115 plt.waitforbuttonpress
116 plt.subplot(2,1,1)
117 plt.imshow(I_stitch_SIFT),plt.title('stitch image(SIFT)'),plt.axis ('
       off')
118 plt.subplot(2,1,2)
119 plt.imshow(I_stitch_ORB ),plt.title('match image(ORB)'),plt.axis ('off'
       )
120 plt.show()
121 plt.waitforbuttonpress
122 plt.subplot(2,1,1)
123 plt.imshow(Isum_SIFT),plt.title('match image(SIFT)'),plt.axis ('off')
124 plt.subplot(2,1,2)
125 plt.imshow(Isum_ORB ),plt.title('match image(ORB)'),plt.axis ('off')
126 plt.show()
127 plt.waitforbuttonpress
128 plt.subplot(2,1,1)
129 plt.imshow(Isum_stitch_SIFT),plt.title('stitch image(SIFT)'),plt.axis (
       'off')
130 plt.subplot(2,1,2)
131 plt.imshow(Isum_stitch_ORB ),plt.title('stitch image(ORB)'),plt.axis ('
       off')
```

```
132 plt.show()
133 plt.waitforbuttonpress
```