Project 2: Feature Detection and Matching

Content

1 Feature detection

Harris' corner detector takes the differential of the corner score into account with reference to direction directly.

To identify the points of interest, a window of pixels around each point should be considered. Compute the Harris matrix H for (the window around) around every point, defined as

$$H(x,y) = \sum_{(x,y)\in W} w_{x,y} \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$$

After obtaining the Harris matrix, we calculate the response of detector at each pixel depending on it.

$$f = Det(H) - \alpha(Trace(H))^2$$

A threshold is determined by the maximum value multiply a ratio value.

$$Threshold = ratio * f(H).\max$$

Finally, we compute the nonmax suppression. Select the domain space. if the neighborhood space distance is less than min, only one corner point is selected.

1.1 Harris Matrix

In matrix operation, we do not directly calculate the Harris Matrix by defining a window around every point. In the convolution operation, we implement this operation by defining a convolution kernel with the size 3*3.The calculation to get the Harris matrix could be divided into following steps.

1. Compute x and y derivatives of images

$$I_x = G_\sigma^x * I \; I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x2} = I_x * I_x \;\; I_{y2} = I_y * I_y \;\; I_{xy} = I_x * I_y \;\; I_{xy} = I_x * I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{x2} = G_{\sigma'} * I_{x2} \quad S_{y2} = G_{\sigma'} * I_{y2} \quad S_{xy} = G_{\sigma'} * I_{xy}$$

4. Define at each pixel(x,y) the matrix

$$H(x,y) = \sum_{(x,y)\in W} w_{x,y} \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix} = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix}$$

```
# Return the partial derivatives in x and y
filters.gaussian_filter(self.img, ksize, (0, 1), gradient_imx)
filters.gaussian_filter(self.img, ksize, (1, 0), gradient_imy)
# Caculate the elements in Harris Matrix
Sx = cv2.GaussianBlur(gradient_imx * gradient_imx, ksize, sigmaX=0, sig
maY=0)
Sy = cv2.GaussianBlur(gradient_imy * gradient_imy, ksize, sigmaX=0, sig
maY=0)
Sxy = cv2.GaussianBlur(gradient_imx * gradient_imy, ksize, sigmaX=0, si
gmaY=0)
```

1.2 The response of the detector at each pixel

$$f = Det(H) - \alpha(Trace(H))^2$$

$$\begin{cases} Det(H) = S_{x2}(x,y) * S_{y2}(x,y) - [S_{xy}(x,y)]^2 \\ Trace(H) = S_{x2}(x,y) + S_{y2}(x,y) \end{cases}$$

α is an empirically determined constant; $\alpha \in [0.04, 0.06]$. α is set to 0.04 in this project. Set the threshold by the response.

```
det = Hx * Hy - Hxy ** 2
trace = Hx + Hy
alpha = 0.04
# Get the corner response
resp = det - alpha * np.square(trace)
threshold = ratio * resp.max()
```

Selecting the points above the threshold. If the intensity of pixel is smaller than threshold, we set it to zero. Then get the coordinates of the non-zero point.

```
# Set the intensity of non-target points to zero
threshold_img = (threshold_img > threshold) * 1
# Get the coordinates of keypoints
coords = np.array(threshold_img.nonzero()).T
candidate_values = [self.img[c[0], c[1]] for c in coords]
index = argsort(candidate_values)
```

1.3 Non-max suppression

If the intensity of pixel point is less than the threshold, or it’s not the maximum around a 3*3 window, it's could not be selected as the point of interests.

```
neighbor = zeros(self.img.shape)
neighbor[min:-min, min:-min] = 1
filtered_coords = []
for i in index:
     if neighbor[coords[i, 0], coords[i, 1]] == 1:
        filtered_coords.append(coords[i])
        neighbor[(coords[i, 0] - min):(coords[i, 0] + min),
        (coords[i, 1] - min):(coords[i, 1] + min)] = 0
```

2 Feature description

In the last step, we have obtained the coordinates of interest points through the HarrisKeypointDetector, next we will implement two descriptors for the feature centered at each interest point. To implement this work, we define a class named FeatureDescriptor which includes three functions. The simplified structure of code for feature description is as follows.

```
class FeatureDescriptor:
    def __init__(self, image, keypoints):
        self.img = image
        self.keypoints = keypoints
    def rotation_angles(self):
        return angles
    def SimpleFeatureDescriptor(self):
        return features
    def Simp_MOPS(self, angles):
        return features
```

2.1 Simple descriptor

This is a simple method to describe the features around each interest point.Before the formal work, we should pad the picture by zero to eliminate the affect caused by the boundary.

```
img = np.pad(img, [(2, 2), (2, 2)], mode='constant')  # zero padding
```

we simply extract a 5*5 window around each interest point which contains the pixel intensity values as the simple descriptor. To implement this work, we define a function named SimpleFeatureDescriptor.

```
def SimpleFeatureDescriptor(self):
```

The output of this function is a three-dimensional list containing all 5*5 windows.

2.2 MOPS

MOPS (Multi-Scales Oriented Patches) is a more complicated descriptor. The critical process of MOPS includes prefiltering, rotating and normalizing. In order to achieve these process, we should first calculate the angles of rotating. We define a function to implement this work.

```
def rotation_angles(self):
```

The method of calculating the rotating angles is simple. We find the eigenvectors of H(the matrix in **Feature detection**) corresponding to the larger eigenvalue. As we have known, these eigenvectors are the principal axes of the quadratic form. Therefore, we can use the orientation of the principal axes to express the rotating angles.

```
angle = np.arctan(eigenvectors[1, index] / eigenvectors[0, index])
```

After getting the rotating angles of each interest point, we can begin to build the MOPS. We also define a function to implement this work.

```
def Simp_MOPS(self, angles):
```

First, as same as the simple descriptor, we should pad the pictures by zero. This time, we pad by 35*35, as the size of the small image we will extract later is 70*70.

```
img = np.pad(img, [(35, 35), (35, 35)], mode='constant')
```

Next we extract a small image (70*70) around each keypoint to package the original MOPS window (40*40) which is inclined. Then we use prefiltering to scale the small image (70*70) to 1/5 size (14*14), in the meantime the inclined MOPS window (40*40) is scaled to 1/5 size (8*8).

```
# Scale to 1/5 size (using prefiltering)
small_img_filtered = filters.gaussian_filter(small_img, 5)
small_img_scaling = cv2.resize(small_img_filtered, (0, 0), fx=0.2, fy=0
.2, interpolation=cv2.INTER_CUBIC)
```

As we have got the rotating angles of windows around each interest point in the function rotation_angles, we can now use these angles to rotate each scaled small image (14*14) to inclined, while also rotate each scaled MOPS window (8*8) to horizontal.

```
# Rotate to horizontal
rot_mtrx = cv2.getRotationMatrix2D((7, 7), -angle, 1)
```

```
small_img_scaling_rot = cv2.warpAffine(small_img_scaling, rot_mtrx, sma
ll_img_scaling.shape)
```

In the next step, we sample a 8x8 square window centered at the inclined small image (14*14) to get the standard sized MOPS window. Ultimately, we normalize the MOPS window by subtracting the mean, dividing by the standard deviation in the window. If the variance is very close to zero (less than 10^{-10} in magnitude), we just return an all-zeros window to avoid a divide by zero error.

The output of the function `Simp_MOPS` is a three-dimensional list containing all 8*8 MOPS windows.

To summarize the feature description, the main idea of each kind of descriptor is to get the pixel intensity around the interest point. The two descriptors we design are invariant to translation, 2D rotation and scale. MOPS and Simple descriptor is built based on these invariance and well perform in feature description.

3 Feature Matching

3.1 SSD

[1].Principle:

SSD (sum of square differences) is a matching method that calculates the distance between features of two interest points and selects the smallest distance value to match the two most similar points.

Suppose that the feature matrix of some two points of interest on the two pictures is shown below:

$$Feature1 = \begin{bmatrix} x_{11} & \cdots & \cdots & x_{1n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{m1} & \cdots & \cdots & x_{mn} \end{bmatrix} \quad Feature2 = \begin{bmatrix} x'_{11} & \cdots & \cdots & x'_{1n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x'_{m1} & \cdots & \cdots & x'_{mn} \end{bmatrix}$$

$$SSD\ Value = \sum_{i=0}^{m} \sum_{j=0}^{n} (x_{ij} - x'_{ij})^2$$

After the SSD values between all feature point pairs are calculated, we select the point pairs that produce the smallest SSD values and record their index values. This means that we found the closest feature points in the two images using the SSD method.

[2].Code

```
SSD = []  # Used to store the SSD distance between any two points in fe
ature list
SSDpairs = []  # Used to store the index values of the two matched elem
ents in the two feature lists
zerofeature1 = []  # Used to store index values of the elements that al
l values are 0 in the feature list
nonzeroDes1 = []  # Used to store index value of the elements that not 
all values are 0
nonzeroDes2 = []
zerofeature2 = []
```

Create list format variables, the purpose of which is indicated in the comments.

```
for i in range(len(self.ft1)):
  if np.all(self.ft1[i] != 0):
          nonzeroDes1.append(i)
  else:
          zerofeature1.append(i)
for j in range(len(self.ft2)):
  if np.all(self.ft2[j] != 0):
           nonzeroDes2.append(j)
  else:
           zerofeature2.append(j)
```

Obtain the index values of elements whose feature list values are all 0, and operate the two feature lists separately.

```
for x in nonzeroDes1:
   for y in nonzeroDes2:
        SSD.append(np.sum((self.ft1[x] - self.ft2[y]) ** 2))
```

Perform SSD operations and add the values to the preset SSDlist.

```
SSD = np.array(SSD)
SSD = np.resize(SSD, (len(nonzeroDes1), len(nonzeroDes2)))
indexMatrix = np.argsort(SSD)  # Get the sort of SSD value size
for x in range(len(nonzeroDes1)):
     SSDpairs.append([x, indexMatrix[x][0]])
```

Transform the number of rows and columns of the matrix so that the SSD value corresponding to each point of interest can be obtained.

Use the argsort function to get the sort of SSD values.

3.2.Ratiotest

[1].Principle

Ratiotest is an optimization method based on SSD FeatureMatch. It can effectively screen out mismatched point pairs.

$$Ratio = \frac{The\ minimum\ SSD\ value}{The\ second\ minimum\ SSD\ value}$$

$$Ratio = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n}(x_{ij} - x'_{ij})^2}{\sum_{i=0}^{m}\sum_{j=0}^{n}(x_{ij} - x''_{ij})^2}$$

If the ratio of SSD value of the most matched interest point to SSD value of the second matched interest point is close to 1, it indicates that the first matching point is too close to the second matching point, which may lead to mismatching.

In order to solve this problem, we set a threshold value less than 1 (the threshold value in this project is set to 0.2), and the ratio of point pairs higher than the threshold value will be discarded, ensuring that the remaining point pairs are highly differentiated and improving the identification accuracy.

[2].Code

```
for t in range((len(self.ft1) - len(zerofeature1))):
    if SSD[t][indexMatrix[t][1]] == 0:  # Determines whether the second
 smallest point of the SSD value is 0
        ratio[t] = 1
    else:
        ratio[t] = SSD[t][indexMatrix[t][0]] / SSD[t][indexMatrix[t][1]]
```

When the denominator of ratio is 0, we set ratio to 1 so that we can discard these pairs of points with the denominator of 0 by threshold adjustment.

```
 for m in range(len(SSDpairs)):
            if ratio[m] <= threshold:
                ratiopairs.append(SSDpairs[m])
```

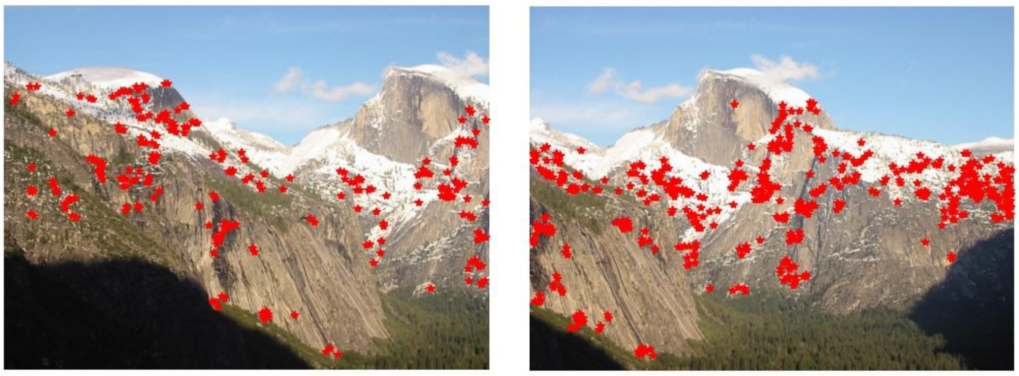
Elements in ratiopairs are a subset of the number of elements in SSDpairs.

4 Performance Analysis

4.1 Result

[1] detection result

Interesting points obtained by Harris corner detection are shown in the figure below:

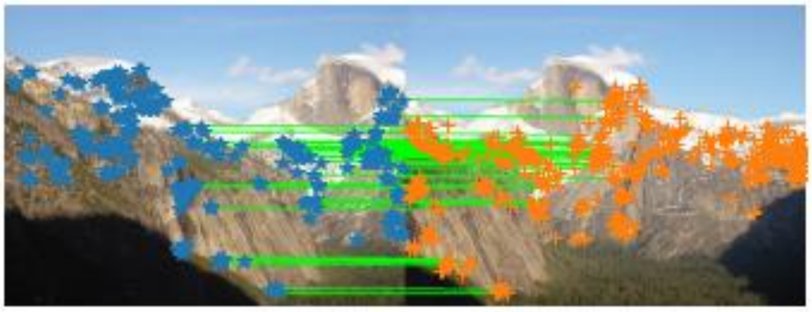


(a) yosemite1 (b) yosemite2

The red points in the figure are the feature points. In parameter setting of detection function, ratio was set as 0.05 and min as 3. Since ratio is a threshold parameter, points larger than the threshold value will be retained and points less than the threshold value will be omitted. Therefore, the larger ratio is, the fewer feature points will be left.And the parameter min is given by the project requirement.

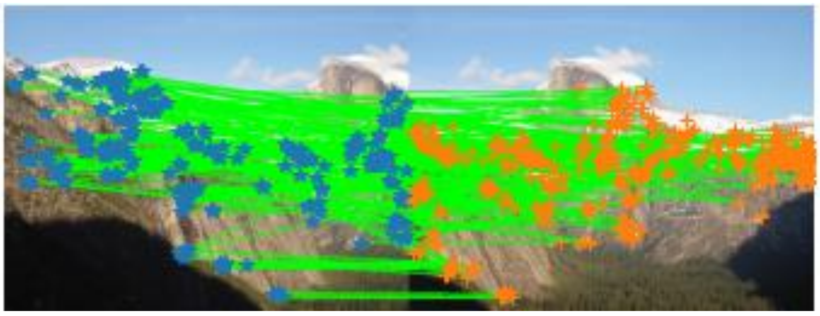
Under this parameter setting, there are 246 feature points in yosemite1 and 487 feature points in yosemite2.

[2] Matching result:



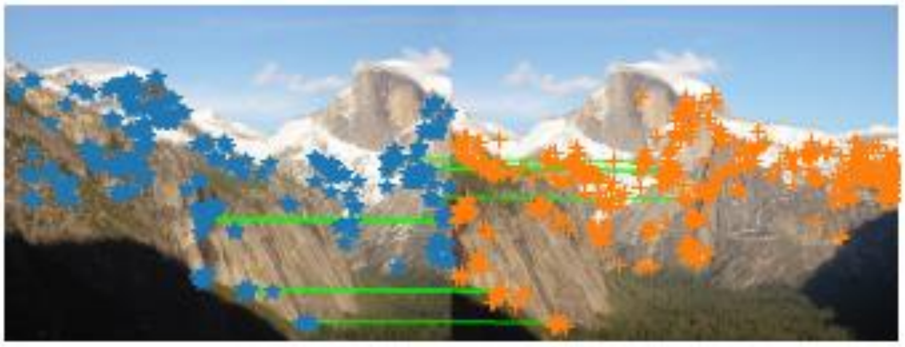
MOPS+ratio(0.2)

This picture is a line graph obtained by using the MOPS Descriptor and ratio match method, which has the highest AUC value, meaning the highest accuracy. Ratio was set to 2, and 53 pairs of feature points were finally matched.



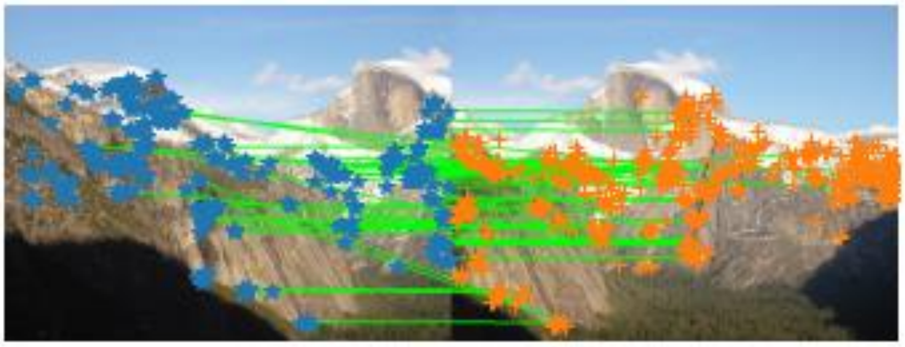
MOPS+SSD

Use the method of combining MOPS Descriptor and SSD matching to get the line graph. The SSD algorithm looks for the best match in Yosemite2 for each feature point in Yosemite1,so the number of matches is 246. This number is equal to the number of feature points in Yosemite1. The AUC value obtained by this combination method is significantly lower than the MOPS+ratio.



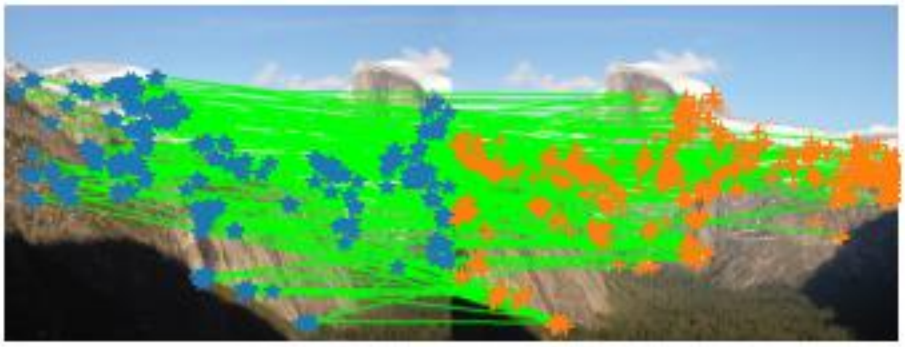
Simple+ratio(0.2)

When we keep ratio=0.2 unchanged and use the combination of SSD method and Ratio test, the generated line graph is shown above. You can obviously see that there are fewer matched points.



Simple+ratio(0.643)

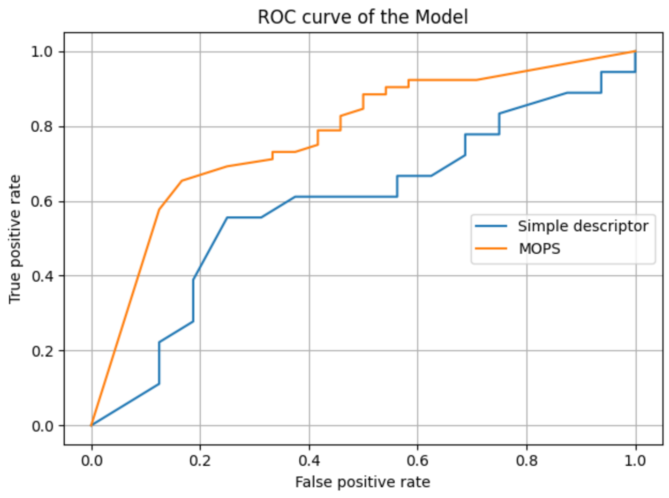
We change the parameters so that 53 pairs of feature points match. The matching accuracy is significantly reduced compared to MOPS+Ratio(0.2) because we can see a lot of slanted lines in the figure. One of the reasons is that the simple Descriptor can't distinguish the rotation correctly.



Simple+SSD

The AUC value obtained using SSD+Simple Description is smaller than that obtained using Ratio+Simple Description.

[3].ROC curve result



ROC curves

As you can see from the curve comparison, the accuracy of using MOPS Descriptors is much higher than that of using simple Descriptors

[4].AUC comparision

	AUC	
	Simple descriptor	MOPS
SSD	0.734	0.895
Ratio(0.9)	0.751	0.872
Ratio(0.8)	0.662	0.839

From the table, we can see that:

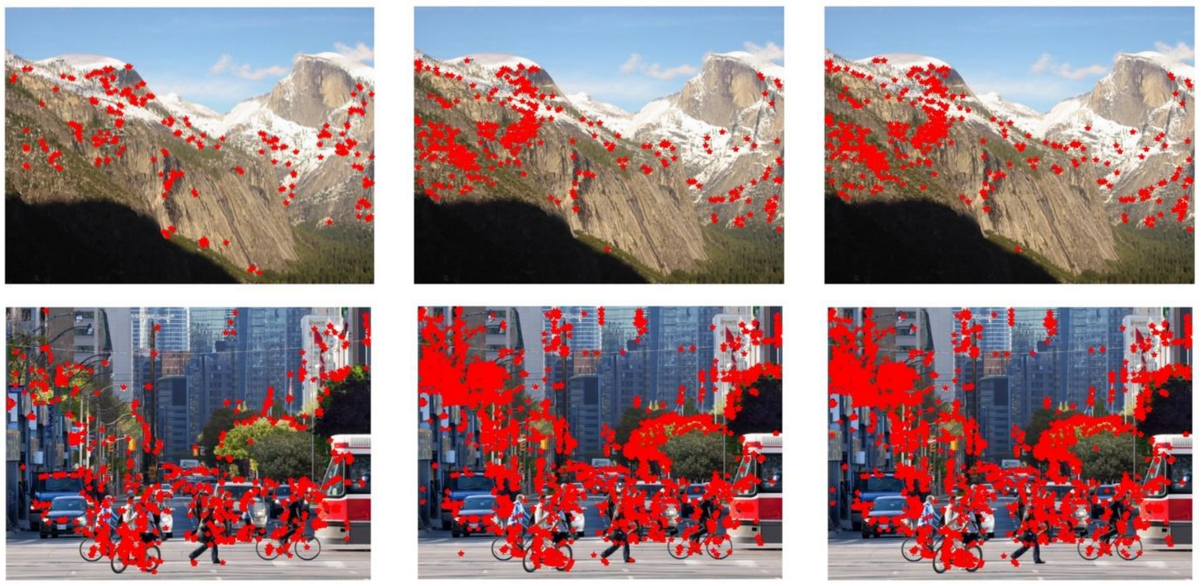
1. The AUC value of MOPS is generally larger than that of simple Descriptor, whether SSD is used or ratio of different thresholds is used.
2. When the threshold is set to 0.9, the AUC value of MOPS decreases slightly, but the AUC value of Simple Descriptor increases. This is because for MOPS, we filter out many correctly matched point pairs, resulting in lower accuracy, while for Simple Descriptor, after ratio operation, we filter out fewer correct point pairs than wrong point pairs, thus increasing the AUC value. This also means that MOPS

descriptor has excellent characterizing ability for pixel blocks.

3. As the threshold continues to rise, the AUC value of both MOPS and Simple Descriptor decreases.

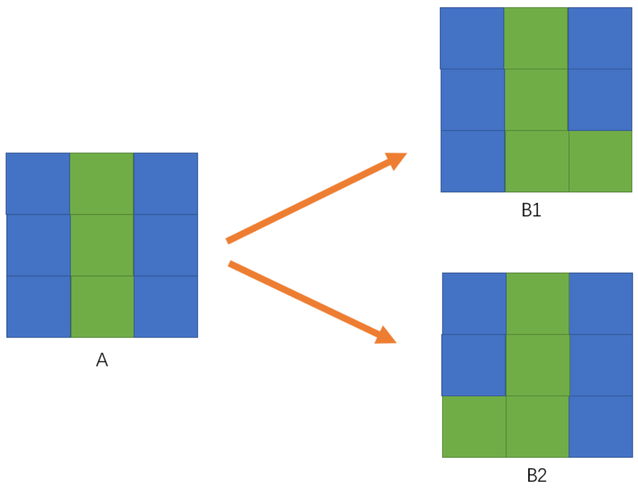
4.1 Discussion

[1]. Compared with Gaussian filter, we attempt to use sobel operator or Scharr operator to calculate the partial derivatives in x&y direction, but Gaussian filter achieves ideal corner-detection effect in practical application when it's applied in street scene.



Gaussian Filter Sobel Operator Scharr Operator

[2]. It can be seen from the results that MOPS characterizes pixel blocks better than simple Descriptors, This is also very predictable, because the simple Descriptor cannot distinguish between two blocks of pixels after the same graph has been rotated.For example:



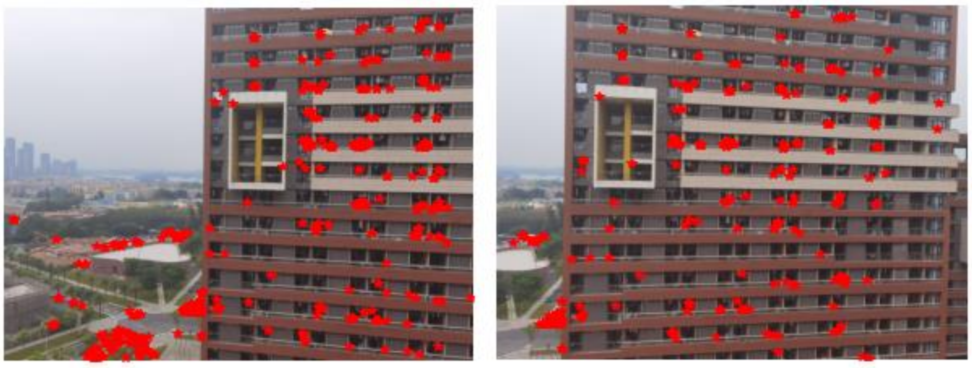
In the case above, we use the SSD algorithm, and when A is used to match another image, the recognition effect of B1 and B2 is the same, while in fact A can only correspond to one descriptor. The disadvantages of the simple descriptor are clearly exposed. When we use MOPS, this problem is solved nicely due to the "Angle" part.

[3]: The advantage of ratio over SSD is that we avoid false matches and improve the AUC value by setting thresholds to eliminate matches where the first match is too close to the second match.

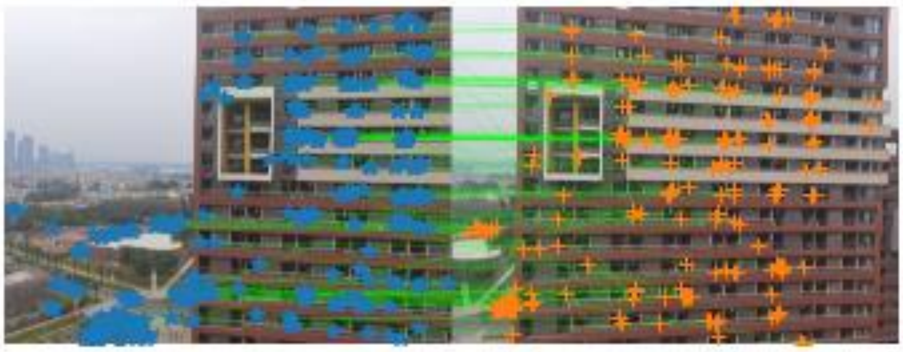
5 My images

The following two photos were taken in the school dormitory.

After Harris corner detection, the feature points are shown in the figure. In this case, ratio is set to 0.13 in detection Function. The first image has 423 points of interest and the second image has 264 points of interest.



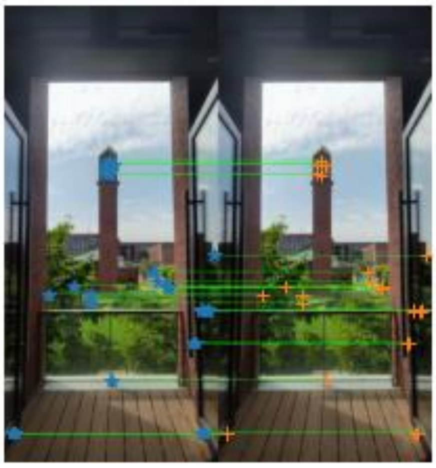
After MOPS and Ratio test, the generated line graph is shown as follows:

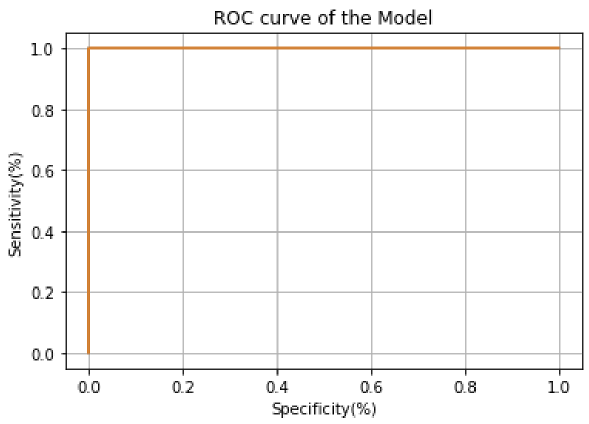


In Ratiotest, ratio is set to 0.7, and 191 pairs of feature points are matched. The final rendering shows that most of the connecting points are the same, and the result is roughly accurate. This justifies the algorithm provided above.

After the above test, we are going to use two identical images for the test:

The above two pictures are exactly the same. If we set H matrix as the identity matrix, the following results can be obtained:





We can see that all the points match up correctly, which means that our algorithm is successful. ROC curve and AUC value =1 also confirm this conclusion.

6 References

[1]Project 2 reading. *MOPS paper*[EB/OL]. [2021/10/25]. 2021 Machine vision and sensing system Project2.

[2]Project 2 reading. *MOPS simplified*[EB/OL]. [2021/10]. 2021 Machine vision and sensing system Project2.

[3] W. F¨orstner. A feature-based correspondence algorithm for image matching. *Int.l Arch. Photogrammetry & Remote Sensing*, 26(3):150–166, 1986

[4]K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *Proceedings of the Interational Conference on Computer Vision and Pattern Recognition* (CVPR03), 2003.

[5]M. Brown and D. Lowe. Recognising panoramas. *In Proceedings of the 9th International Conference on Computer Vision* (ICCV03), volume 2, pages 1218–1225, Nice, October 2003.