Lab Report
AIC - Data Knowledge - Image Mining
Lab 1: Feature Detection, Description and Matching

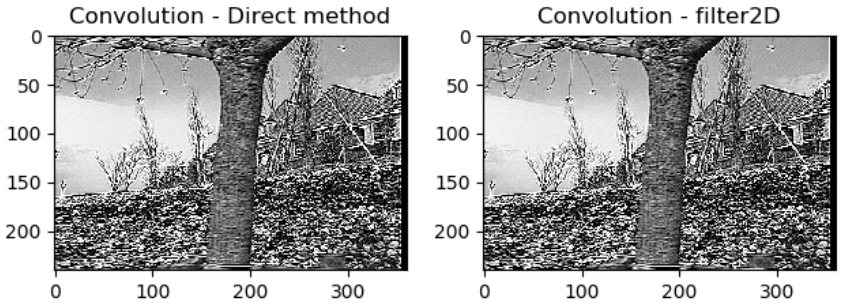
ZHANG Xin

January 2020

2 Image Format and Convolutions

Q1 Experiment the convolution code given as example in *volutions.py*.

Firstly, it is noted that the calculation using *filter2d* (0.05s) is way faster than the direct calculation (0.19s). One difference is that the *filter2d* computes the correlation instead of convolution, but as the kernel here is symmetric, the results obtained are the same except at the borders, as shown below. (the filtering of negative values in filter2d is later performed by the `vmin` parameter of `plt.show()`)



The differences in efficiency lies in the optimization of calculations of *filter2d*, but as the kernel size here is 3×3, which is samller than 11×11, DFT and convolution theorem is not yet applied, the efficiency comes probably from SIMD.

As for the `OpenCV` and `matplotlib` functions, in `OpenCV`, `cv2.imread` is used to read the image, where `flag=0` indicates that the image will be read in greyscale format. It returns a NumPy array representing the image, and in the example the values in the array are converted into `float64` in order to facilitate the calculation. `cv2.copyMakeBorder()` is used to copy the image, the function will add additional borders to the image in order to enable us to calculate the convolution for the borders, and the `BORDER_REPLICATE` option means the row or column at the very edge of the original is replicated to the extra border.

Here, for `matplotlib`, `plt.imshow()` is used to display the image, what's worth noticing is that `camp="grey"` is used to replace the default *viridis color palettes*, and `vmin = 0.0,vmax = 255.0` is used for `filter2d` method in order to filter out the values not in range.

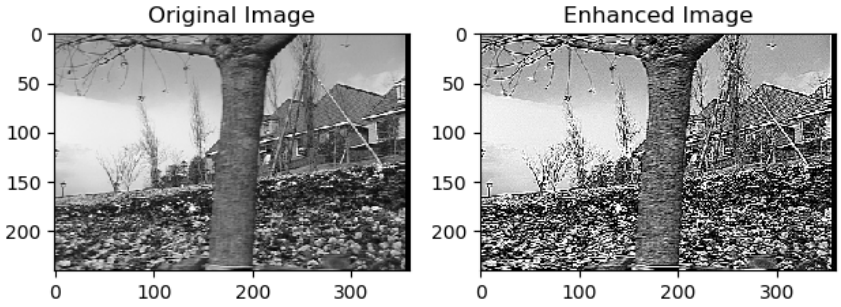
Q2 Explain why the convolution kernel provided as example realises a contrast enhacement with respect to the original image.

The convolution kernel can be interpreted as:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

where $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ is an approximation of the Laplacian $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$. As the Laplacian is the second derivative, which will detect the edges of an image as the value will be 0 where there is no change of adjacent pixels but will be relatively

negative and positive for the brighter and darker side of an edge, therefore detect the edge. So after having abstracted the Laplacian, the bright side of the edge will become brighter, and the darker side darker, therefore the edges have higher contrast, the image is enhanced, as shown below



Q3 Modify the code to compute the convolution that approximates the partial derivative and the gradient modulus

To approximate the partial derivative $I_x = \frac{\partial I}{\partial x}$, we can do the convolution with Sobel's mask $h_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, therefore we modify the code as follows:

```
val = -img[y-1,x-1] + img[y-1,x+1] - 2*img[y,x-1] + 2*img[y,x+1] - img[y+1,x-1] + img[y+1,x-1]
```

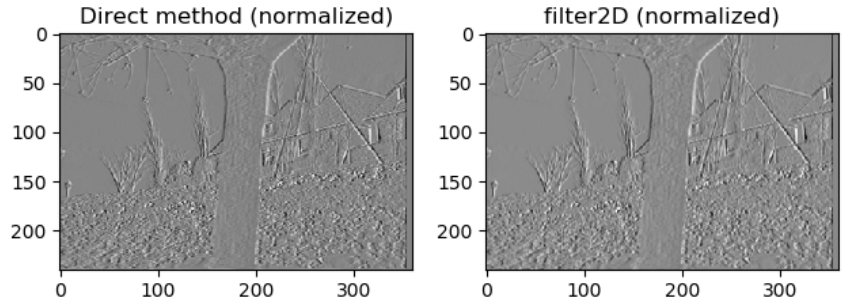
for direct method and

```
kernel = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
```

for `filter2D`.
For the sake of **getting a correct display**, I have chosen to normalize (rescale) the results in the range from 0 to 255 as follows:

```
img = 255*(img-np.amin(img))/(np.amax(img)-np.amin(img))
```

The images for x-deriative is shown below:



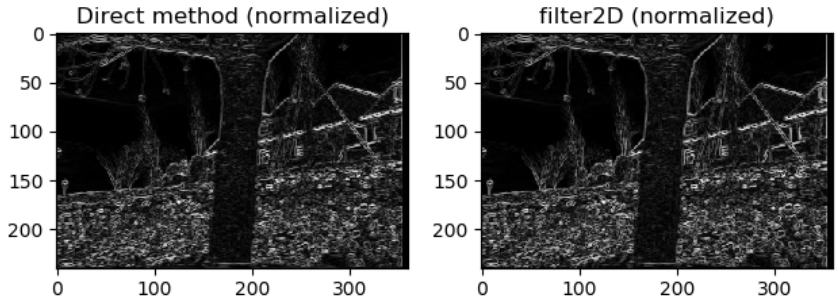
Similarly, for $I_y = \frac{\partial f}{\partial y}$, we have $h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$, therefore we can compute $\|\nabla I\| = \sqrt{I_x^2 + I_y^2}$ as I_y can be calculated with the same procedure.
Direct:

```
    valx = -img[y-1,x-1] + img[y-1,x+1] - 2*img[y,x-1] + 2*img[y,x+1] - img[y+1,x-1] + img[y+1,x-1]
    valy = img[y+1,x-1]-img[y-1,x-1]+2*img[y+1,x]-2*img[y-1,x]+img[y+1,x+1]-img[y-1,x+1]
    img2[y,x] = np.sqrt(valx**2+valy**2)
```

filter2d:

```
kernelx = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
imgx = cv2.filter2D(img,-1,kernelx)
kernely = np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])
imgy = cv2.filter2D(img,-1,kernely)
img3 = np.sqrt(np.square(imgx)+np.square(imgy)
```

The result images is shown below



3 Detectors

Q4 Compute the interest function of Harris and the corresponding interest points.

We know that the derivative of a Gaussain kernel is as follows:

$$G_x(x,y,\sigma) = -\frac{x}{\sigma^2}G(x,y,\sigma), \quad G_y(x,y,\sigma) = -\frac{y}{\sigma^2}G(x,y,\sigma)$$

and for a 3×3 Gaussian kernel,

$$\frac{x}{\sigma^2} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \dot{\sigma}^2, \quad \frac{y}{\sigma^2} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \dot{\sigma}^2$$

So the code completed to compute the interest function of Harris is as follows:

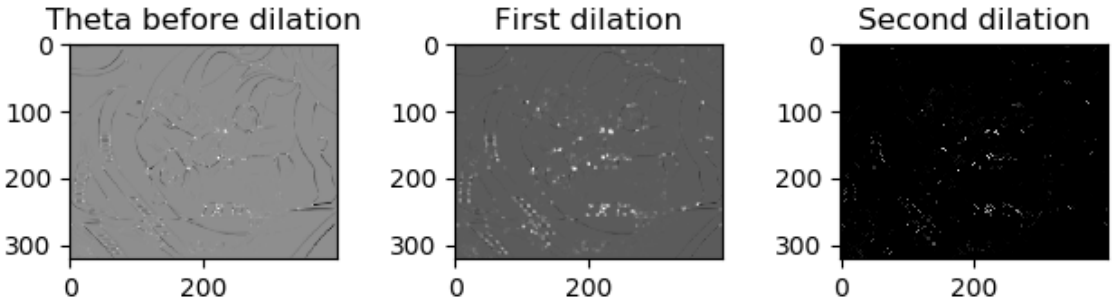
```
alpha = 0.06
scale1 = 3
scale2 = 2*scale1
#Compute the first Gaussian derivatives
dir_gauss = cv2.getGaussianKernel(3, scale1)
gauss = np.multiply(dir_gauss.T, dir_gauss)
gauss_dev_x = np.multiply(-np.array([[-1, 0, 1],[-1, 0, 1],[-1, 0, 1]])/scale1**2,gauss)
gauss_dev_y = np.multiply(-np.array([[1, 1, 1],[0, 0, 0],[-1, -1, -1]])/scale1**2,gauss)
img_x = cv2.filter2D(img, -1, gauss_dev_x)
img_y = cv2.filter2D(img, -1, gauss_dev_y)
#Compute the autocorrelation matirx and the result
I_xx = cv2.GaussianBlur(img_x**2, (3,3), sigmaX=scale2, sigmaY=scale2)
I_xy = cv2.GaussianBlur(img_x*img_y, (3,3), sigmaX=scale2, sigmaY=scale2)
I_yy = cv2.GaussianBlur(img_y**2, (3,3), sigmaX=scale2, sigmaY=scale2)

det_H = I_xx*I_yy - I_xy**2
trace_H = I_xx + I_yy
Theta = det_H - alpha*trace_H**2
Theta_norm = cv2.normalize(Theta, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX)
```

As for the effect of morphological dilation, it is performed two times by `Theta_dil = cv2.dilate(Theta,se)` and `cv2.dilate(Theta_maxloc,se_croix)`, where $se = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ and $\bigoplus = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$.

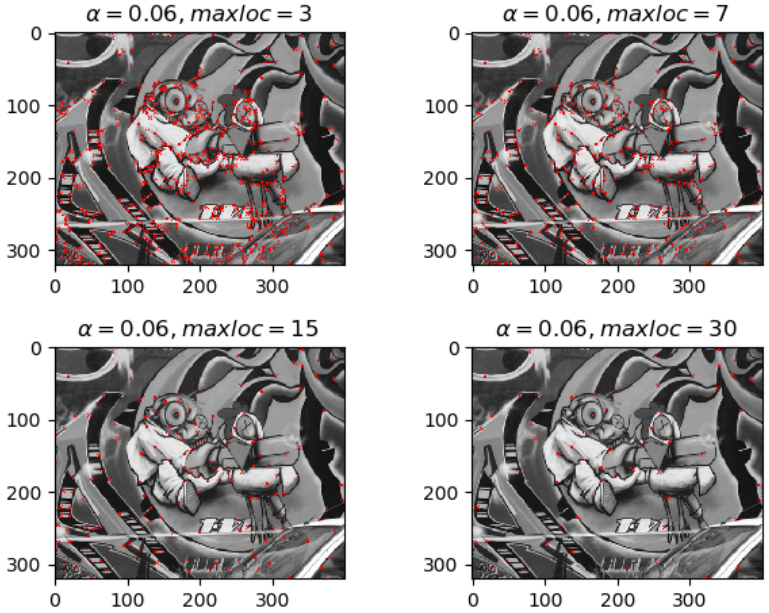
We know that the dilation of greyscale images is defined as $(f \oplus b)(x) = \sup_{y \in E}[f(y) + b(x-y)]$, where $b(x)$ is the kernel. In the first dilation, With the 3×3 square kernel, the interest points of the interest map will be expanded in width as the

differences of values between interest points and non-interest points are huge. The purpose of the first dilation is to prevent loss of real corner points in the second dilation (as some interest points is only one pixel, it can be eliminated by if not expanded). For the second dilation, as we are only interested in corners and we are not interested in edges, $\oplus$ will filter out the points (those points will be set to 0) vertically and horizontally connected to the local maxima, thus the corners points will be found. The effects of the two dilations are shown below:



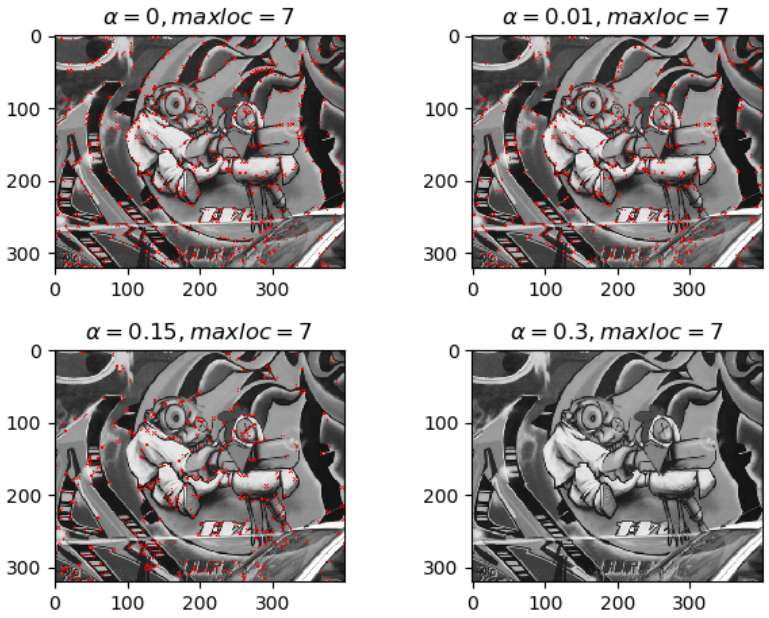
Q5 the effect of parameters of the Harris detector

The size of the summing window effects the result directly, for it means the size of the region we will find a local minimum, the result with varied `d_maxloc` is shown below:



It can be observed that `d_maxloc` has dramatically changed the number of points detected. But when `d_maxloc` is too big (e.p. 30), we can also lose disired corner points, So we need to choose a propre `d_maxloc` for the tradeoff of precision and recall.

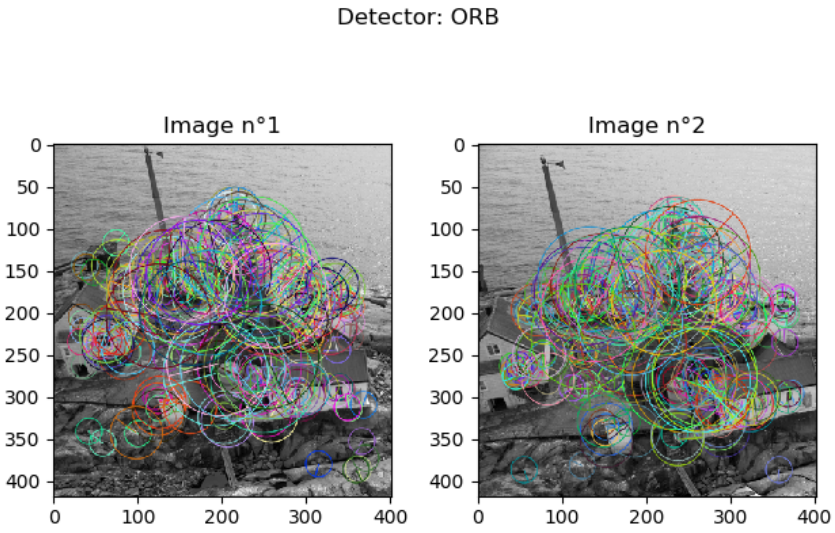
As for the value of α, it is the parameter in $\Theta(x, y) = det\,\Xi - \alpha\, trace\,\Xi^2$. with a bigger α, less false corners will be detected but some true corners are missed (high precision), with a smaller α a lot more corners are detected, less true corners are missed, but with a lot of false ones (high recall). But if α is too big, the result will always be negative and no points can be found, there effects are illustrated below:

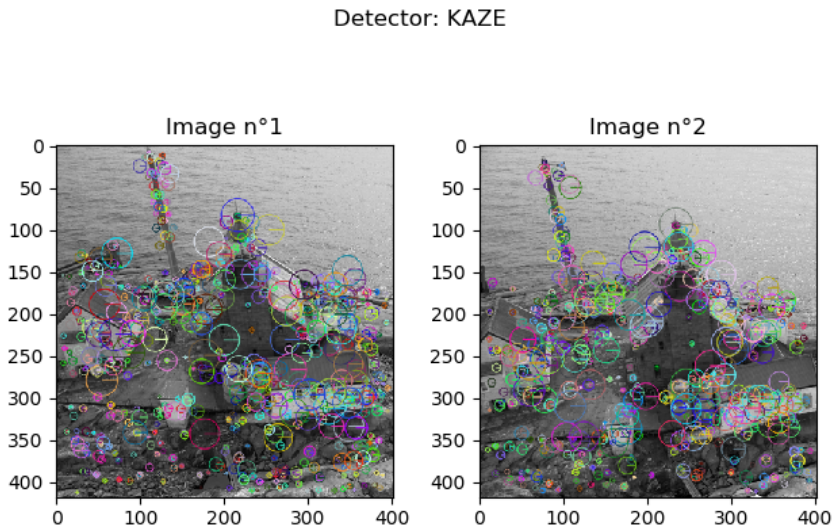


In order to extend this computation on several scales, we can change the standard deviation σ. By increasing the σ, we can get a Gaussian smoothed image, thus with the smoothed image, the corner points will be calculated in a larger scale, as minor contours will be smoothed.

Q6 Compare the two detectors ORB and KAZE

Firstly, we run the two detectors with their given parameters, the results are as follows:





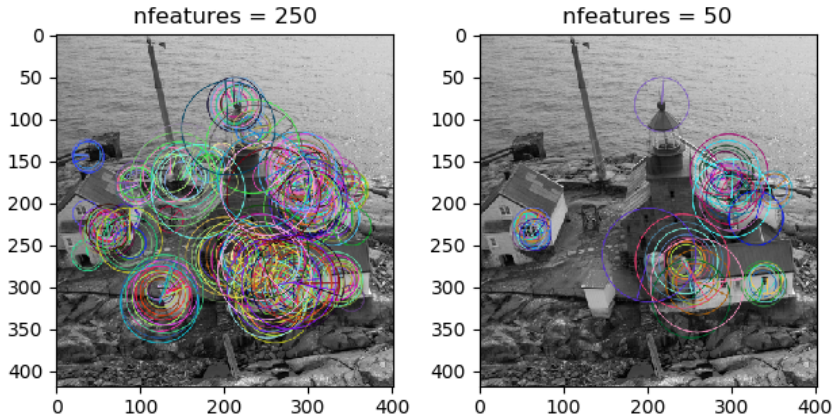
We can see that the results are very different. For ORB, less feature points are detected and most of the points are found in larger scales, which has formed big circles in the result, but for most of the feature points, an correct orientation of the keypoint is found. In the contrary, KAZE has found more keypoints, and the points are found in smaller "scales", but few correct orientations are given by KAZE.

The principle of **ORB**:
Firstly, the detector FAST is performed on a pyramid of multiscale image rely on Gaussian scale space. That is, different levels of Gaussian blurring is used on the image in order to get smaller images with more general features to form the pyramid. And the FAST detector will select selects points p whose circular neighbourhood shows long contiguous runs with values significantly brighter (resp. darker) than p. Thanks to the pyramid, the feature points selected in several scales will be scale invariant. Then, the orientation is computed by intensity centroid, which assumes that a corner's intensity is offset from its center, and this vector may be used to compute an orientation.

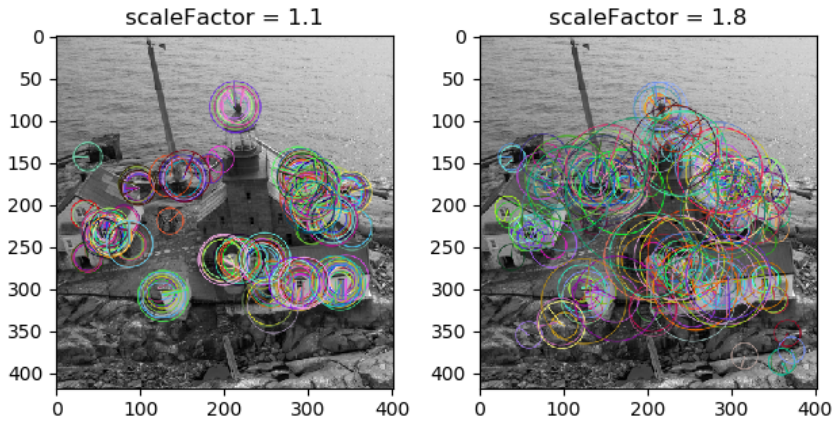
The principle of **KAZE**:
Given an input image, KAZE firstly build the nonlinear scale space up to a maximum evolution time using AOS (Additive Operator Splitting) techniques and variable conductance diffusion. The idea is to represent the change of the lightness L of the image in different scales with the divergence of a flow function, which is $\frac{\partial L}{\partial t} = div(c(x, y, t)\nabla L)$, and then in order to converge at a solution for this nonlinear differential equation, the AOS methos is used. Then, on top of the scale space, 2D features of interest that exhibit a maxima of the scale-normalized determinant of the Hessian response through the nonlinear scale space are detected. The idea is with the Hessian determinant, we can serach for the maxima in a window size, which makes it possible to locate (in a sub-pixel manner) the points of interest at each scale of the image

parameters of **ORB**:

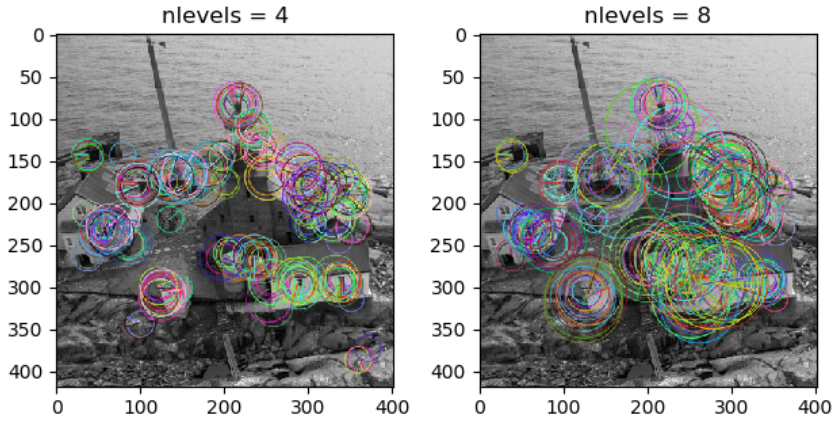
- `nfeatures` The maximum number of features to retain. It dicides the number of feature points we will get, If it is smaller, feature points found will be limited in certain areas.



- `scaleFactor` Pyramid decimation ratio. This parameters decides how the image will be blurred in each level. If it is bigger, with the same number of levels, it can help us find more general features.

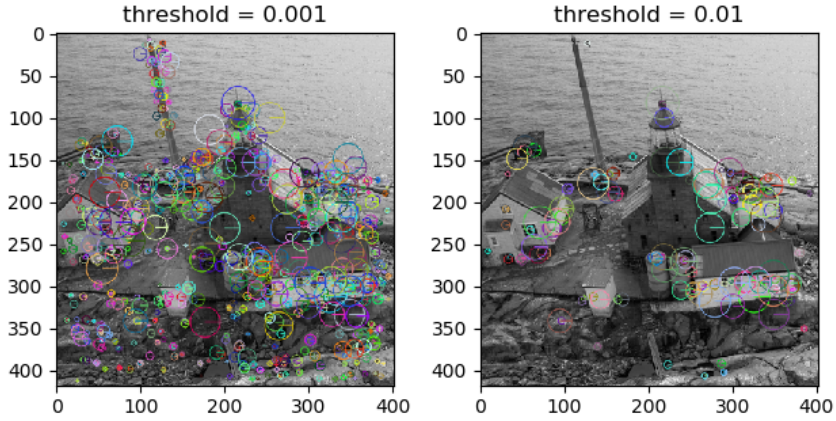


- `nlevels` The number of pyramid levels. This parameter has similar effect as `scaleFactor`, as it decides how many levels we will blur our image, with the same decimation ratio, a larger number will help us identify more general features.

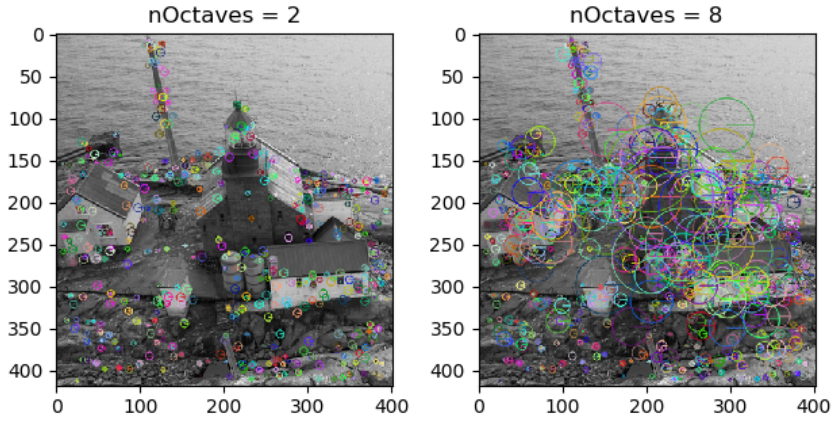


parameters of **KAZE**:

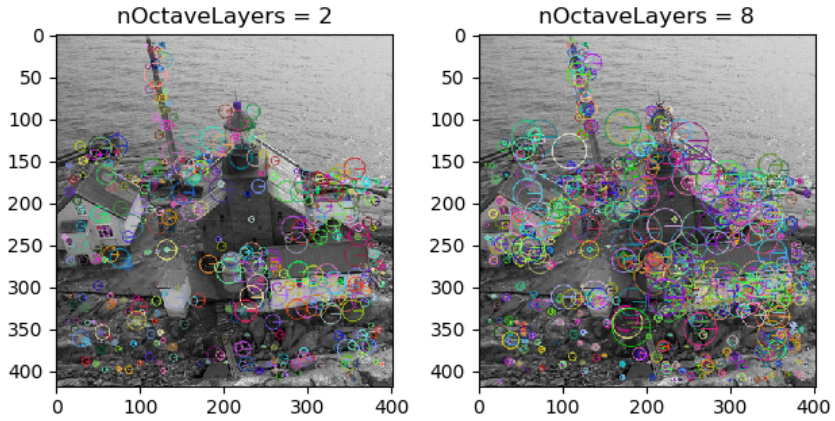
- `threshold` Detector response threshold to accept point. This is the threshold for the value of the Hessian determinant of the image. A point will be accepted as a keypoint if its Hessian determinant surpasses the threshold. The bigger the threshold, fewer points will be accepted.



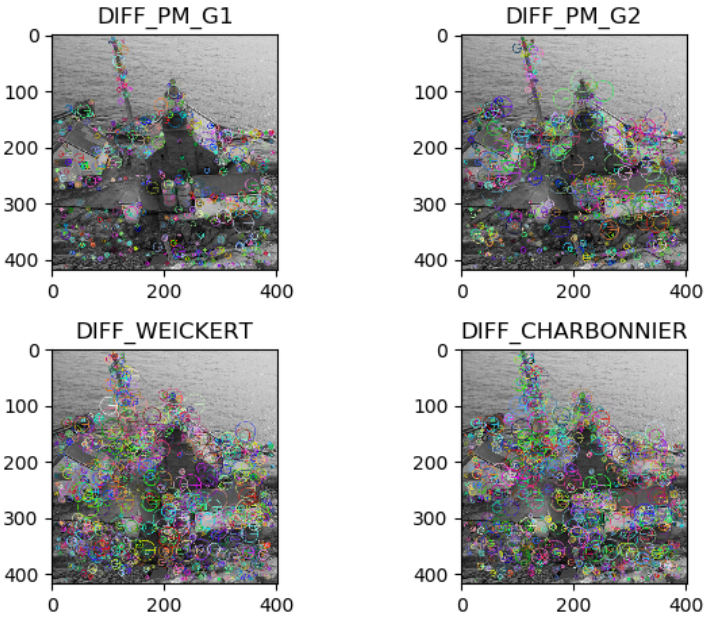
- `nOctaves` Maximum octave evolution of the image. It determines the number of scale levels of our nonlinear scale space, A larger number of octaves permets us to find more general features.



- `nOctaveLayers` Default number of sublevels per scale level. It determines the number of sublevels of each octave. Increasing it will let us to find more features with little scale differences for each octave.



- `diffusivity` Diffusivity type. DIFF_PM_G1, DIFF_PM_G2, DIFF_WEICKERT or DIFF_CHARBONNIER. This parameter determines which nonlinaer diffusion technique is used to reduce the noise of the image. From the result, we can observe that, with the DIFF_WEICKERT and DIFF_CHARBONNIER technique, more keypoints are detected as the two approaches reduced the diffusion at the location of edges [4], which allows more edge information to be conserved.



As to visually evaluate the repeatability of each detector applied on a pair of images, we can display an image of the overlapping (intersection) region of the two images, and we marked the repeated detection in green, and the other points in the region of the object (or subject) points in red. Thus, We can visually represent the repeatability of a detector.

4 Description and Matching

Q7 the principle of the descriptors attached to ORB and KAZE

The descriptor of **ORB**:
ORB used an algorithm called rBRIEF(Rotation-aware BRIEF), which is an approach to steer BRIEF(Binary robust independent elementary feature) according to the orientation of the keypoints to make the descriptor rotation invariant. As for the BRIEF, the principle is to construct a binary vector by selecting a random pair of pixels in a defined neighborhood (called patch) around that keypoint. The first one is selected by a Gaussian distribution centered around the keypoint with a stranded deviation and the second one by a Gaussian distribution centered around the first pixel. if the first pixel is brighter than the second, it assigns 1 to bit else 0. For a 128-bit to 512-bit vector, brief repeat this process for 128-512 times for a keypoint, repectively. Then, rBRIEF uses the patch orientation θ and the corresponding rotation matrix to construct a steered version of the points selected by BRIEF. Thus a look-up table is constructed for us to find the precomputed BRIEF patterns, which makes the descriptor orientation invariant.

The descriptor of **KAZE**:
Different from ORB, the orientation of the keypoint is not calculated in detection phase. So, in KAZE, firstly, the dominant orientation is calculated at first for the descriptor. The orientation is found by summing the first order derivative responses, which are weighted by a Gaussian centered at the interest point, within a sliding circle segment. And the longest vector is selected as the dominant orientation. Then, with the orientation, a M-SURF descriptor is adapted. The descriptor $d_v = (\sum L_x, \sum L_y, \sum |L_x|, \sum |L_y|)$ is computed by summing up derivative responses through overlapped divisions of a rectangular grid around the interest point, and the sum of each subregion and the whole grid are all weighted by Gaussian and according to the dominant orientation. Finally, the descriptor vector is normalized into a unit vector to achieve invariance to contrast.

Scale and rotation invariant:
The properties of detectors and/or descriptors which allow to make the matching scale and rotation invariant are listed in the tables below:

ORB

	detector: oriented FAST	**descriptor: rBRIEF**
scale invariance	multiscale detection achieved by the pyramid	multiscale sampling achieved by the pyramid
rotation invariance	orientation computed by intensity centroid	look-up table to the precomputed BRIEF patterns calculated with the orientation

KAZE

	detector: KAZE	**descriptor: M-SURF**
scale invariance	multiscale detection achieved by nonlinear diffusion	descriptors calculated in the scale the feature was found
rotation invariance	the differentiel invariance of Hilbert space (Hessian determinant)	a dominant orientation is considered in the sum calculation of derivative responses

Q8 Explain and compare qualitatively the effects of the three different point matching strategies

The *cross-check* method's matching strategy is a brute-force matching. For each descriptor in the query descriptors (from image1), it will be compared to every descriptor in the train set (from image2). Only the train descriptor with the closet distance (hamming distance for rBRIEF, L2 distance for M-SURF) to the query descriptor will be considered a match. And the checking strategy of *cross-check* is to rerun the process, find a match in the query set for the found train descriptor, if and only if the match for the train descriptor is the query descriptor earlier, the match will be considered valid.

The *ratio-check* method use the same brutal-force matching strategy as *cross-check*, What is different is that we will keep the 2 best matches in the train descriptor set. Then, for the checking, we will compare the distance of the two matches. In

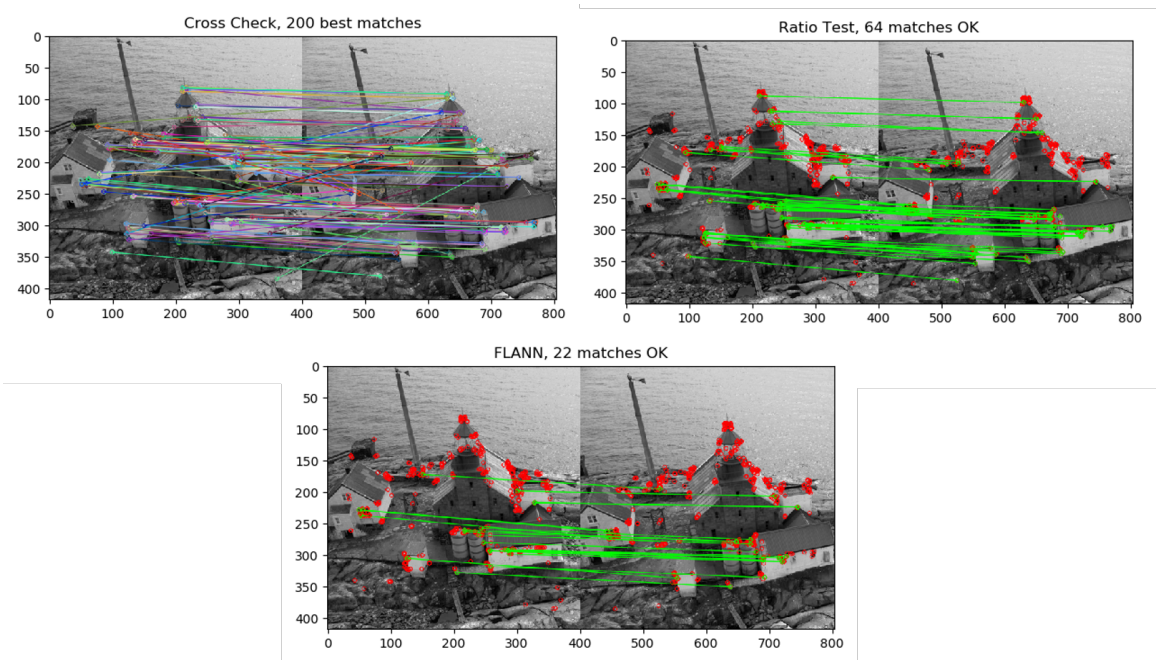
the given example, the better one of the two matches will be considered a valid match if and only if its distance is smaller than 0.7 times the distance of the additional (worse) one. This method is apparently more **efficient** than *cross-check* as we don't need to do a second brutal-force. And it is reasonable as for an exact real repetition (match), the distance should be considerably smaller than other pairs.

The *FLANN* method uses the library FLANN (Fast Library for Approximate Nearest Neighbor)in which we can choose a variety of nearest neighbor searching algorithms for the matching process. In the example given, the k-d tree algorithm is applied. The principle of k-d tree algorithm is to construct the query descriptors into a indexed binary tree. The binary tree is constructed by firstly decide the axis to divide by finding the axis with biggest variance, and then divide with the median in this axis, these two steps will be until there is only one point in one division, thus we have the binary tree. A binary search will be performed for the matching to find a "current best", and then decides the "real best" by unwinding the recursion of the tree by drawing a hypersphere around the search point that has a radius equal to the current nearest distance to find if there is other points inside.

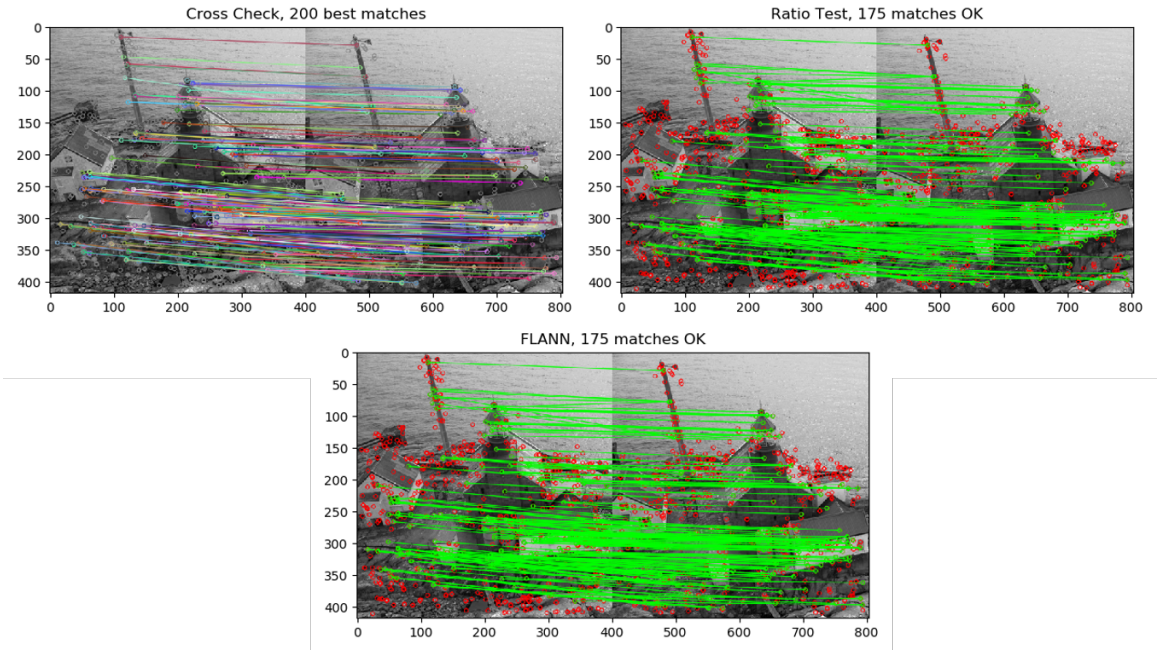
In the given example of *FLANN*, two best matches are found, and a verification procedure same as the one of *ratio-check* is performed to check the validity of the match.

The results of the three strategies on the ORB and KAZE are illustrated as follows:

ORB



KAZE



ORB

Strategy	valid matches	Computation Time (ms)
Cross-check	190	5.11
Ratio-test	64	2.35
FLANN	22	4.99

KAZE

Strategy	valid matches	Computation Time (ms)
Cross-check	351	20.46
Ratio-test	175	5.69
FLANN	175	46.98

From the results, there are several things to notice:

Firstly, we can see that *cross-check* for **ORB** has provided some apparently wrong matches, which are not noticed in other results. The reason is probably that rBRIEF descriptors are rotated binary vectors and the sampling is random with a Gaussian distribution, it is possible for different interest points to have similar patterns but as the pattern is similar, it will become a match. So there are some visible errors in this result.

Secondly, we can see that *FLANN* (and ratio-test in a lesser extent) doesn't work well with **ORB** points, this is because for (steered) binary vectors, there can be many points with the same Hamming distance between each other. In the checking procedure where we compare the distance of the two matches for a query descriptor, it is probable for the two matches to have the same distance even if the first one is correct. Thus this match will be marked invalid though it is actually valid. Additionally, for FLANN, when performing the recursion in the searching, there will also be unwanted points in the drawn hypersphere as the binary vectors can easily have the same distance, and FLANN will be unable to find a match, thus FLANN has matched even less points than ratio-test.

Q9 Propose a strategy to evaluate quantitatively the matching by using an image with a known geometric transformation

In order to quantitatively evaluate the matching, we can define the matching rate by

$$matching\ rate = \frac{2 \times \#\ matched\ points}{\#\ features\ in\ original\ +\ \#\ features\ in\ transformed}$$

So, to evaluate quantitatively the matching by using an image with a known geometric transformation, we can firstly get a transformed image with the geometrical transformation function of `OpenCV`, then we will run the detection on all of the two images. After that, we can evaluate the matching quantitatively with the matching rate and the computation time.

References

[1] Harris, Christopher G., and Mike Stephens. "A combined corner and edge detector." Alvey vision conference. Vol. 15. No. 50. 1988.

[2] Rublee, Ethan, et al. "ORB: An efficient alternative to SIFT or SURF." ICCV. Vol. 11. No. 1. 2011.

[3] Alcantarilla, Pablo Fernández, Adrien Bartoli, and Andrew J. Davison. "KAZE features." European Conference on Computer Vision. Springer, Berlin, Heidelberg, 2012.

[4] Perona, P., Malik, J.: Scale-space and edge detection using annisotropic diffusion. IEEE Trans. Pattern Anal. Machine Intell. 12, 1651–1686 (1990)

[5] Muja, M., and D. Lowe. "Fast library for approximate nearest neighbors (FLANN),"." git://github. com/mariusmuja/flann. git. url: http://www. cs. ubc. ca/research/flann (2013)