# Problem1-Texture Analysis and Segmentation

## a. Texture Classification

### Abstract and Motivation

Image texture is a large area without sharp edges where the content can be characterized as a consistent structure. Many scientists tried to give qualitative definitions of texture, but the definitions they gave us are not simple. However, it is easy for people to do texture classification and segmentation by visualization. For example, in an image of grass, the grass is the texture and in an image of sand, the sand is the texture. The difference between sand and grass is quite obvious. In this problem, we want to implement some mathematical method to classify different textures.

**Task:** Design and implement a method based on Laws Filter and K-means to classify textures of 12 images..

### Approaches and Procedures

**Theoretical Approach:**

The given 1-D kernel of 5*5 Laws Filters are:

$$E5 = \frac{1}{6}[-1 \;\; -2 \;\; 0 \;\; 2 \;\; 1], \qquad S5 = \frac{1}{4}[-1 \;\; 0 \;\; 2 \;\; 0 \;\; -1], \qquad W5 = \frac{1}{6}[-1 \;\; 2 \;\; 0 \;\; -2 \;\; 1]$$

First, we have to do 9 tensor products in order to get 9 5*5 Filter. Take $E5^T E5$ as an example to illustrate how to do tensor product:

$$E5^T E5 = \frac{1}{6*6}\begin{pmatrix} -1*(-1) & -1*(-2) & -1*0 & -1*2 & -1*1 \\ -2*(-1) & -2*(-2) & -2*0 & -2*2 & -2*1 \\ 0*(-1) & 0*(-2) & 0*0 & 0*2 & 0*1 \\ 2*(-1) & 2*(-2) & 2*0 & 2*2 & 2*1 \\ 1*(-1) & 1*(-2) & 1*0 & 1*2 & 1*1 \end{pmatrix}$$

In this way, we can get 9 Laws Filters.

Second, we have to do a pre-processing task. Given that the images have DC components that will give us very high energy but less information, we have to remove them before classification. Here is the formula to do this manipulation:

$$Globalmean = \sum Input(i, j) / sizeof(image)$$

$$Nimg(i, j) = Input(i, j) - Globalmean$$

Third, we can generate our feature vector by calculate the convolution of each Laws Filter and the image without DC component. Because the Laws Filter are symmetric, the convolution of center pixel is equal to the summation of the product of corresponding position in a local window. Assume that the size of the input image is N*M, We can calculate the energy of each image and each filter. The formula of energy calculation is:

$$E = \frac{1}{N*M}\sum\sum(I(i, j))^2$$

Therefore, we can get a 9-D energy vector for each image.

Last, we have to do the K-means to classify each image. First, we have to choose the initial centroids. Because

there is no convergence guarantee, I used 2 different strategies to initialize it. One is based on my observation, we can define it as supervised learning. Another is choosing randomly. I select the supervise learning. The decision rule of K-means is based on the Euclidean distance between the energy vector of each image and the centroids of each category. After calculating the Euclidean distance, the image belongs to the category with minimal distance. After an iteration, we have to update the centroids by this formula:

$$C_{\mathrm{k}} = \frac{1}{\#(\text{images})} \sum T(i, j) .$$

We iterate this algorithm until the centroids vector do not change.

**Implementation by C++**

Step1: Read input 12 images and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height and width and number of channels.

Step2: Pre-processing: remove DC component. Use a 2-nested loop to subtract the global mean of each input image.

Step3: Convolution. For each image, convolve it with 9 Law filters, then we can get 9 processed images for each input image. Then calculate the global averaged energy for each processed image. Therefore, we can get a 9*12 feature matrix.

Step4: Initial centroids for K-means, in order to get better results, I record the maximal value and minimal value for each feature. For unsupervised learning, we can generate our initial centroids within this special range. For supervised learning, we can choose the mean vector of each class as our initial centroids.

Step5: Calculate the Euclidean distance of each feature vector (1*9) to each centroids vector. The index of minimal value implies the class of the image.

Step6: After an iteration, update the centroids by the previous formula and check the difference between the old centroids and new centroids. If they remain unchanged, stop iteration. Otherwise, go back to step5 and clear the classification results. The index 1 was marked was rock, index 2 was marked as grass, index 3 was marked as weave and index 4 was marked as sand.

## Results:

| Filter Type | Texture1 | Texture2 | Texture3 | Texture4 |
|---|---|---|---|---|
| $E5^{T}E5$ | 344.02 | 46.36 | 25.24 | 2.59 |
| $E5^{T}S5$ | 188.6 | 31.25 | 10.69 | 2.32 |
| $E5^{T}W5$ | 61.3 | 9.42 | 2.89 | 1.19 |
| $S5^{T}S5$ | 120.67 | 17.47 | 5.89 | 2.33 |
| $S5^{T}E5$ | 186.06 | 23.55 | 13.25 | 2.37 |
| $S5^{T}W5$ | 44.24 | 5.96 | 1.69 | 1.25 |
| $W5^{T}W5$ | 15.49 | 1.86 | 0.56 | 0.70 |
| $W5^{T}E5$ | 53.74 | 5.92 | 4.15 | 1.21 |
| $W5^{T}S5$ | 39.00 | 4.79 | 1.89 | 1.26 |

Energy Values

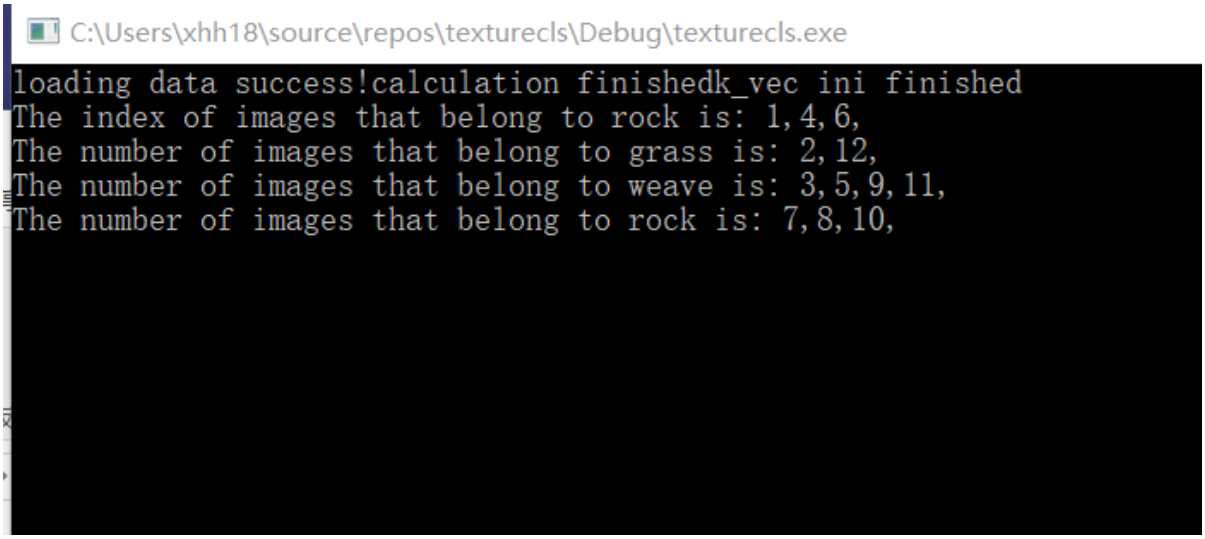



Figure1 Results

### Discussion:

The strongest discriminant power is from $E5^{T}E5$ and the weakest energy is from $W5^{T}W5$.

The category of ninth image should be grass. However, K-means puts it in weave. Therefore, this method is unstable. Because it will easily affected by many factors. For example, the distribution of the image, the shape of the content. Therefore, we should be careful when implementing this method.

# b. Texture Segmentation

### Abstract and Motivation

In problem a, we can divide serval images into certain groups based on their texture information. We can conclude that the texture of an image represents the information of the image. Different texture has different energy and feature vector, which can be used in image segmentation. In this problem, we will design a method to solve segmentation problem for a single image with several textures.

**Task:** Perform texture segmentation over a given image and discuss on the outputs.

### Approaches and Procedures

**Theoretical Approach:** The approach of solving this kind of problems is quite similar to the way we implemented in problem (a). However, this problem is much complex since we have to classify each pixel rather than a whole image. Here are the theoretical steps:

Step1: Laws feature extraction. First, we have to calculate total 9 Laws Filters, in this problem we use 3*3 Laws Filter rather than 5*5. The given 1-D kernel of 3*3 Laws Filter is:

$$L3 = \frac{1}{6}\begin{pmatrix}1 & 2 & 1\end{pmatrix}$$

$$W3 = \frac{1}{2}\begin{pmatrix}-1 & 0 & 1\end{pmatrix}$$

$$S3 = \frac{1}{2}\begin{pmatrix}-1 & 2 & -1\end{pmatrix}$$

We can implement tensor product to this 1-D vector to get target filter mask. Then, we have to do the pre-processing manipulation: remove the DC component, which is that each pixel minus the global mean of the whole image.

Step2: Convolution. For each image, convolve it with 9 Law filters, then we can get 9 processed images for each input image. Now, we can get a 9*N*M feature matrix (image after convolution and assume the size of input image is N*M).

Step3: Local averaged energy calculation. Because the input image contains several different textures, each texture has different energy. Therefore, we have to calculate the averaged energy for each pixel instead of calculating global averaged energy. Those pixels does not have same energy though they belong to the same texture. Therefore, we have to take their neighbor into consideration. We have to set up a local window whose size is 13*13, 15*15 or bigger to calculate the averaged energy. We can use the mean of normalized energy in this local window to approximate the energy of central pixel in this local window.

Step4: Now, we get a 9-D vector for each pixel. Therefore, the size to energy matrix is 9*N*M. Notice that the mean of $L3^{T}L3$ is not equal to zero. Therefore, for each pixel, the normalized energy vector is the ratio of other 8 entries with the first entry ($L3^{T}L3$). For each pixel, the first entry of the normalized energy is 1 so that we can reduce our energy matrix size by removing the first column of the matrix.

Step5: K-mean training and segmentation. Because there is no convergence guarantee and there are 6 six textures in the image, I'd like to choose 6 pixel points with 6 different textures in the image and choosing their energy vector as the initial centroids. Then, for each pixel, implement K-means clustering as the problem (a). Calculate the Euclidean distance of each feature vector (1*9) to each centroids vector. The index of minimal value implies the class of the pixel. After each iteration, update the centroids.

## Results:

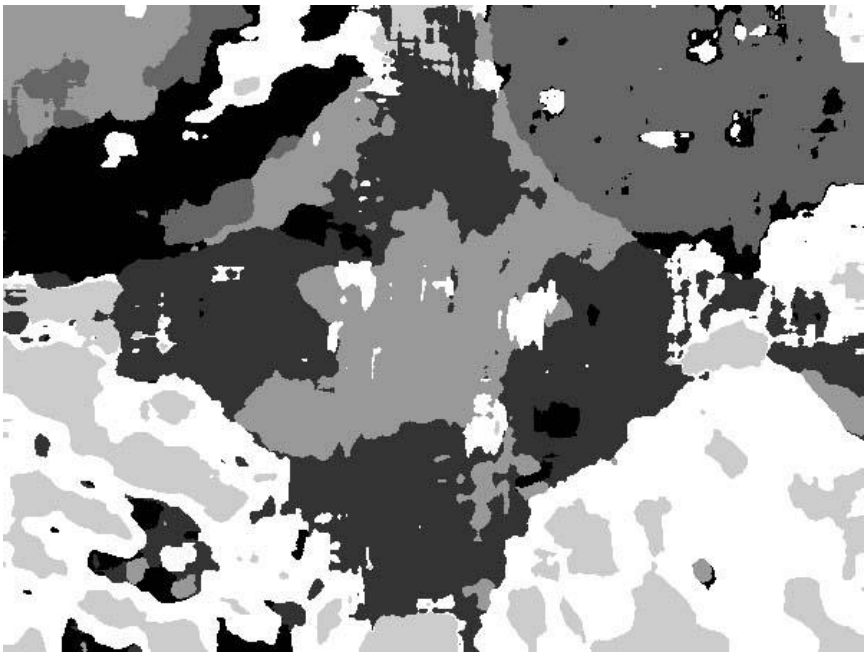


Figure2 local window size is 13*13.

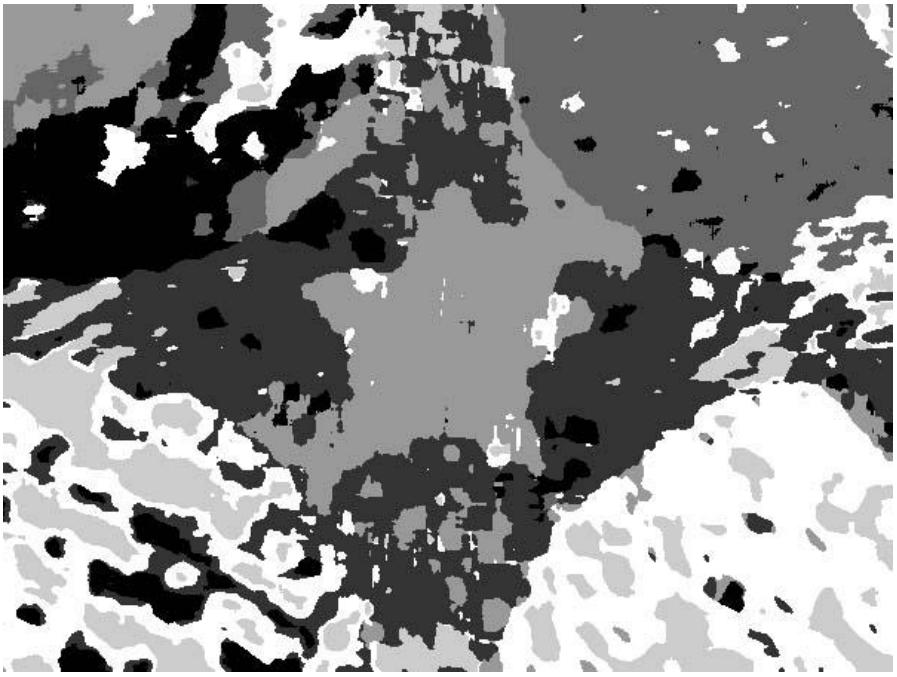


Figure3 local window size is 17*17

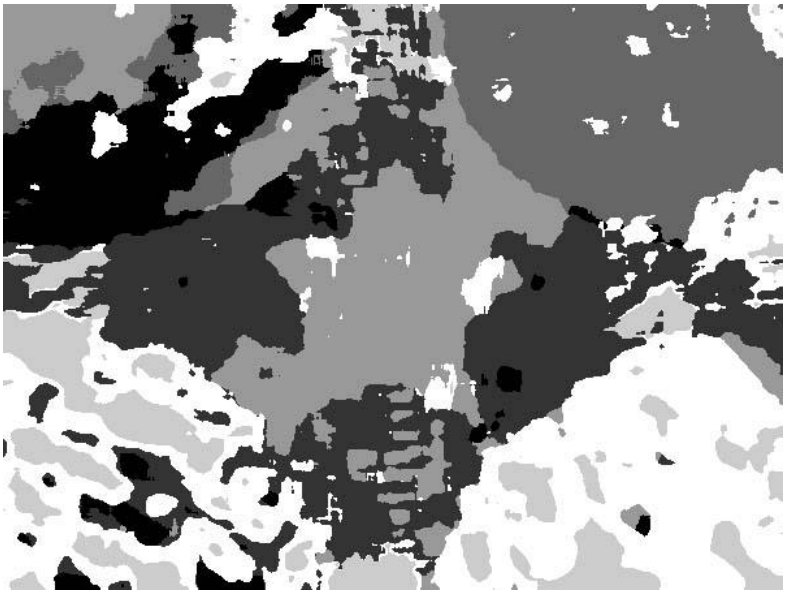


Figure4 local window size is 21*21

### Discussion:

Because this image is made up of 6 textures. Each texture is distributed in a certain area. Therefore, for pixels in each area, they are highly correlated so that when the local window size is bigger, we can get better results. Also, because we initialize the centroids by choosing the value of 6 pixels in 6 textures. It cannot represent the characteristic of the region will. That is why there are some holes in the final result.

## c. Improvement of segmentation by PCA

### Abstract and Motivation

The results in (b) are not very good. We have to come up with a method to improve it. PCA is a good method. Because in problem(b), we only use 9 Laws Filters to do the segmentation. The total number of Laws Filter is 25. Therefore, in this part we have to use all Laws Filters and use PCA to reduce the dimension of feature vectors.

### Approaches and Procedures

Step1: Laws feature extraction. First, we have to calculate total 25 Laws Filters, in this problem we use 5*5 Laws We can implement tensor product to this 1-D vector to get target filter mask. Then, we have to do the pre-processing manipulation: remove the DC component, which is that each pixel minus the global mean of the whole image.

Step2: Convolution. For each image, convolve it with 9 Law filters, then we can get 9 processed images for each input image. Now, we can get a 25*N*M feature matrix (image after convolution and assume the size of input image is N*M).

Step3: Local averaged energy calculation. Because the input image contains several different textures, each texture has different energy. Therefore, we have to calculate the averaged energy for each pixel instead of calculating global averaged energy. Those pixels does not have same energy though they belong to the same texture. Therefore, we have to take their neighbor into consideration. We have to set up a local window whose size is 13*13, 15*15 or bigger to calculate the averaged energy. We can use the mean of normalized energy in this local window to approximate the energy of central pixel in this local window.

Step4: Now, we get a 25-D vector for each pixel. Therefore, the size to energy matrix is 25*N*M. Notice that the mean of $L3^TL3$ is not equal to zero. Therefore, for each pixel, the normalized energy vector is the ratio of other 8 entries with the first entry ($L3^TL3$). For each pixel, the first entry of the normalized energy is 1 so that we can reduce our energy matrix size by removing the first column of the matrix.

Step5: Output the 24-D feature vector data in a txt file. Input it in MATLAB and implement the built-in PCA function to reduce it to 6-D feature vector for each pixel.

Step6: Load the reduced dimension feature vector and do K-means training and segmentation. Because there is no convergence guarantee and there are 6 six textures in the image, I'd like to choose 6 pixel points with 6 different textures in the image and choosing their energy vector as the initial centroids. Then, for each pixel, implement K-means clustering as the problem (a). Calculate the Euclidean distance of each feature vector (1*9) to each centroids vector. The index of minimal value implies the class of the pixel. After each iteration, update the centroids.

### Results:

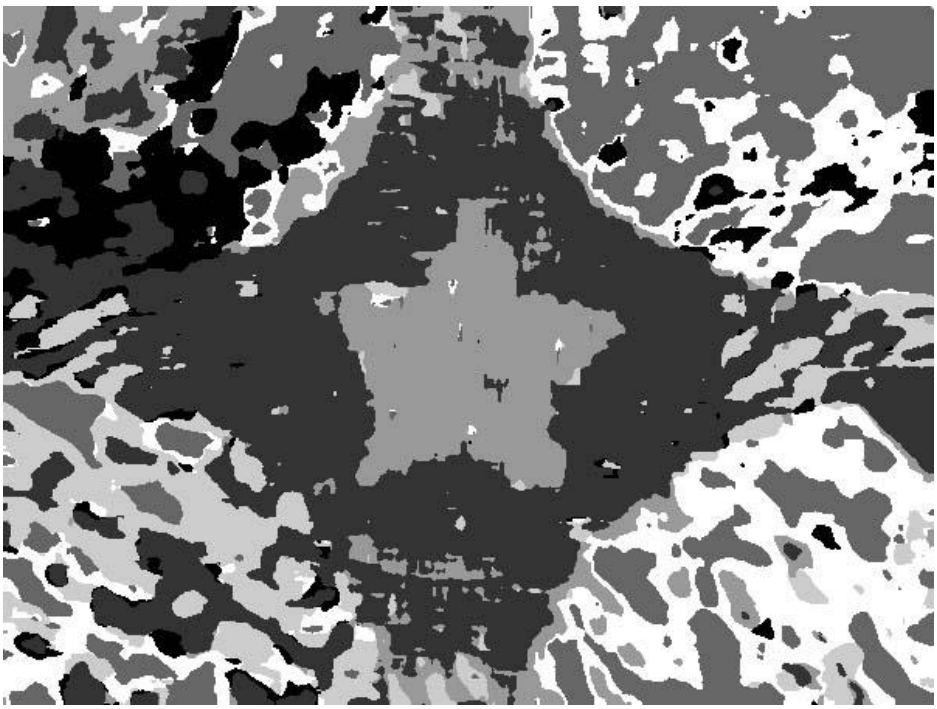


Figure5 Result from PCA and K-Means

# Problem2-Edge Detection

## a. Basic Edge Detector

### Abstract and Motivation:

Edge is very important to image processing and image analysis. The edge is characterized by its height, slope angle and horizontal coordinate of the slope midpoint. An edge exists if the edge height is greater than a specified value [1]. There are many methods to extract the edge information such as Sobel, Canny, Laplacian and LOG etc… If we get the edge information, we can do some post-processing such as image reconstruction and some applications in computer vision. Therefore, it is very important to know how to extract edge of an image.

**Task:** Perform Sobel and LOG Edge Detection over a given set of images.

## Approaches and Procedures

**Theoretical Approach:**

**Sobel Edge detection:**

Because pixel values are discrete in spatial domain, it is common to use difference between pixels to approximate the differential manipulation in continuous domain. Sobel detector is the one that cares the first-order derivative of the input image. There are 2 masks for Sobel detector. One is calculating the first-order derivative with respect to x. The other is calculating it with respect to y. Here are the 2 masks:

$$\frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

In order to calculate the 2 first-order derivatives, we have to do the convolution of these 2 masks and the input image. In image processing, we can regard the convolution manipulation as the summation of template products. Here are the formula:

$$\frac{\partial I(i,j)}{\partial x} = \frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \otimes I(i,j) \qquad \frac{\partial I(i,j)}{\partial y} = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \otimes I(i,j)$$

Because the derivative can be either positive or negative, we only take its magnitude into consideration. Therefore, we can calculate the final magnitude of the input image by this formula:

$$F(i,j) = \sqrt{\left(\frac{\partial I(i,j)}{\partial x}\right)^2 + \left(\frac{\partial I(i,j)}{\partial y}\right)^2}$$

The last step is thresholding. Because the edge content is the pixels that have big magnitude of its first-derivative, we can set a threshold like 90% or 85%, which means those pixels whose pixel value is greater than or equal to the maximal value of the total pixel can be regarded as the edge pixels. Otherwise, the pixels is the background.

**Implementation by C++:**

Step1: Read input image “Boat.raw” and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height, width, number of channels.

Step2: Run two nested loop for getting access to every pixel value. Since the input image is a RGB colorful image, we have to convert it to a grayscale image first.

Step3: Do convolution with 2 Sobel templates. We can get the first-order derivative with respect to x and with respect to y. Calculate its magnitude and normalize it to 0-255(in order to output the x-gradient and y-gradient images).

Step4: Calculate the total magnitude F(i,j) of the processed image by previous formula. Sort it in an ascend order, set threshold and get the final results.

**LOG edge detection**

If we use Sobel detector to detect the edges of image, we will lose much information. For example, because we only take the first-derivative into consideration, if a pixel value increases slowly but it reaches a high value in the end, it is still an edge pixel. However, its first-order derivative is small since it increases slowly and it would not be regarded as edge in Sobel detection. Also, Sobel detector is vulnerable to all kinds of noise. Therefore, we can do some research on the second-order derivative of image, which is the LOG edge detection.

LOG means Laplacian of Gaussian. The Laplacian descriptor and Gaussian kernel are:

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{(-\frac{x^2+y^2}{2\sigma^2})}$$

$$\nabla^2=\frac{\partial^2 G(x,y)}{\partial x^2}+\frac{\partial^2 G(x,y)}{\partial y^2}$$

LOG is equivalent to do Gaussian filtering first and do the Laplacian convolution second. The previous formula implies that we can use a template to create the effect of Gaussian+Laplacian. The LOG kernel is :

$$LoG \triangleq \triangle G_\sigma(x,y)=\frac{\partial^2}{\partial x^2}G_\sigma(x,y)+\frac{\partial^2}{\partial y^2}G_\sigma(x,y)=\frac{x^2+y^2-2\sigma^2}{\sigma^4}e^{-(x^2+y^2)/2\sigma^2}$$

We can generate any size LOG mask by discrete sampling that kernel function with a parameter sigma(standard deviation of Gaussian kernel).

After convolution, we get the result of the second-order derivative of this image. We have to normalize it to 0-255 in order to get its histogram and plot it. We can choose the double knee points in the histogram as our left and right threshold. Then we can map the image into a 3-valued array whose values are -1, 0 and 1.

Finally, we have to do zero crossing for each pixel. The purpose to do this is that check if there is a pair of -1 and 1 in the 3*3 local window of the center pixel. If it exists, regard it as edge.

**Results:** (orange points are knee points)

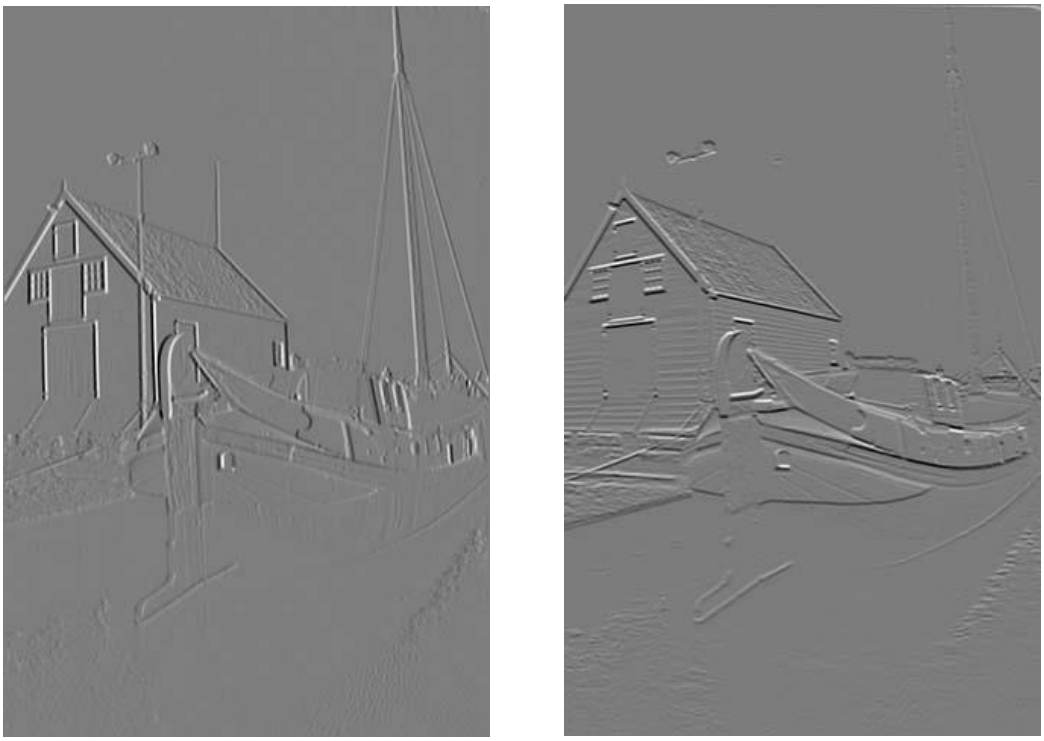


Figure5 Normalized x-gradient and y-gradient image(Boat.raw)

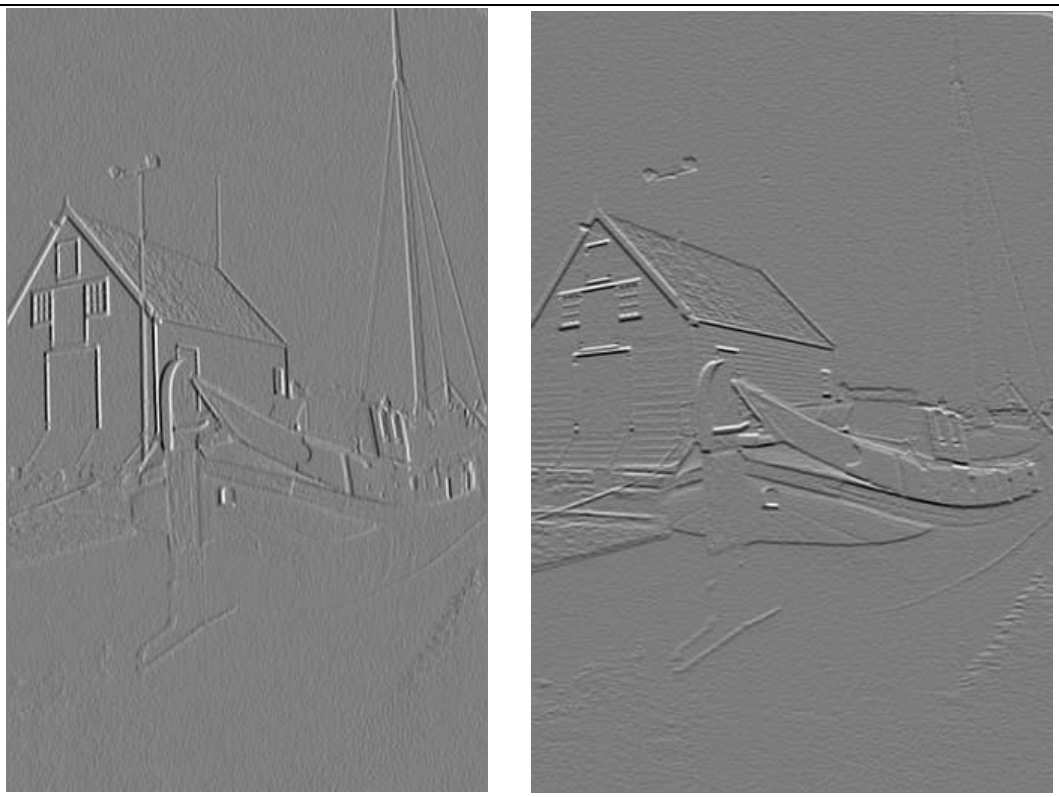


Figure6 Normalized x-gradient and y-gradient image(Boat_noisy.raw)

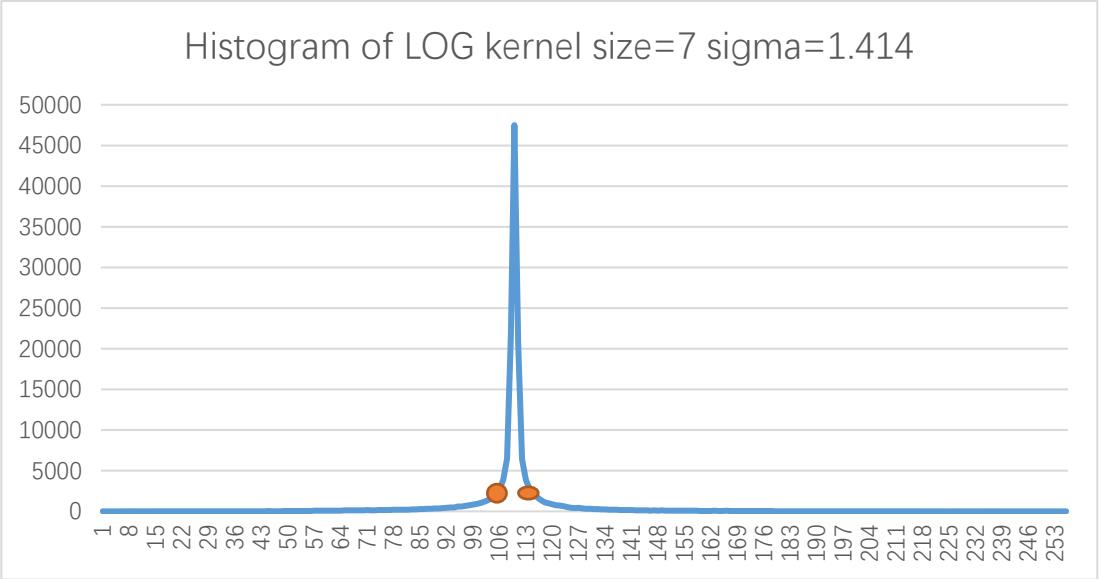



Figure7 Histogram of LOG kernel size=7 sigma=1.414 (boat.raw)

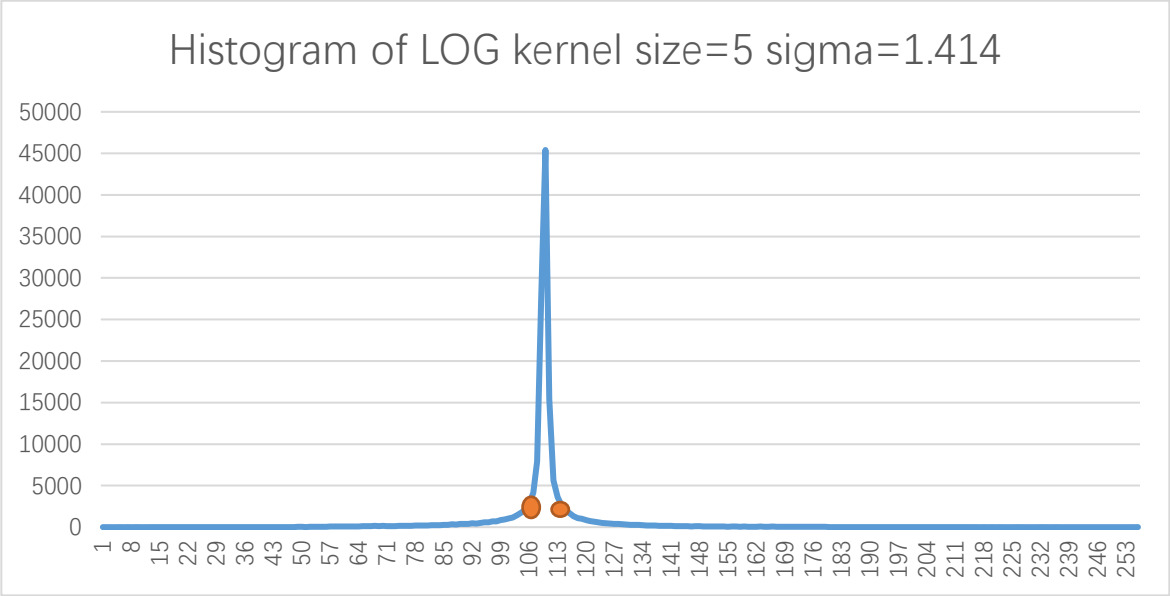



Figure8 Histogram of LOG kernel size=5 sigma=1.414 (boat.raw)

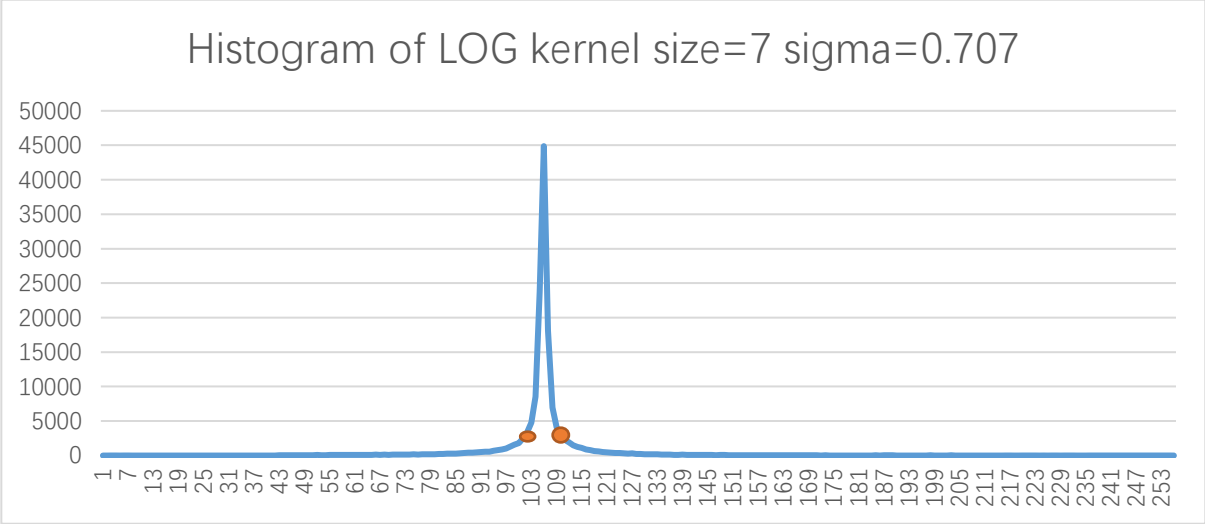



Figure9 Histogram of LOG kernel size=7 sigma=0.707 (boat.raw)

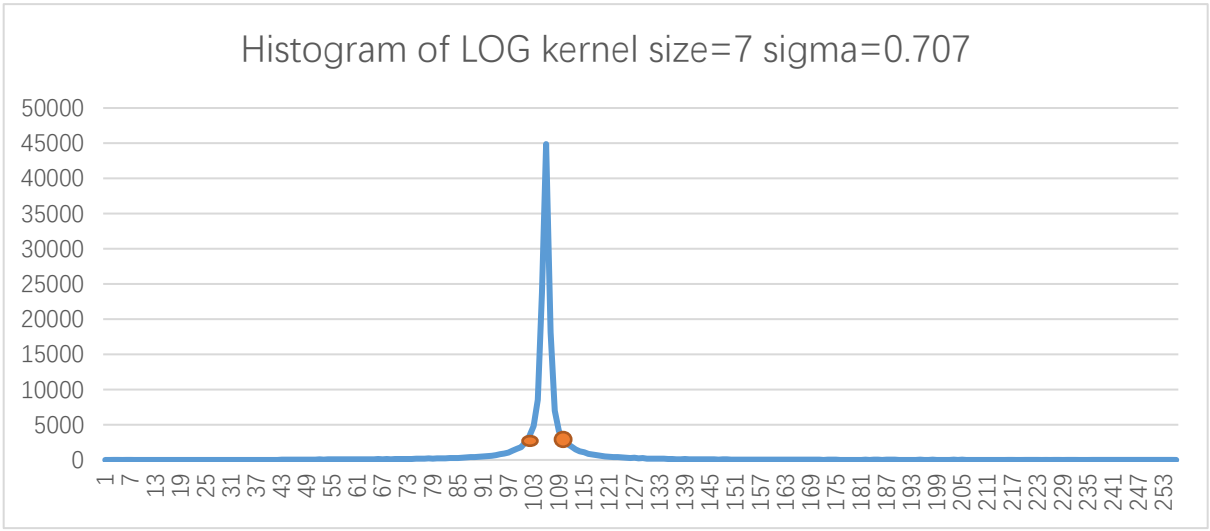



Figure10 Histogram of LOG kernel size=7 sigma=0.707 (boat.raw)

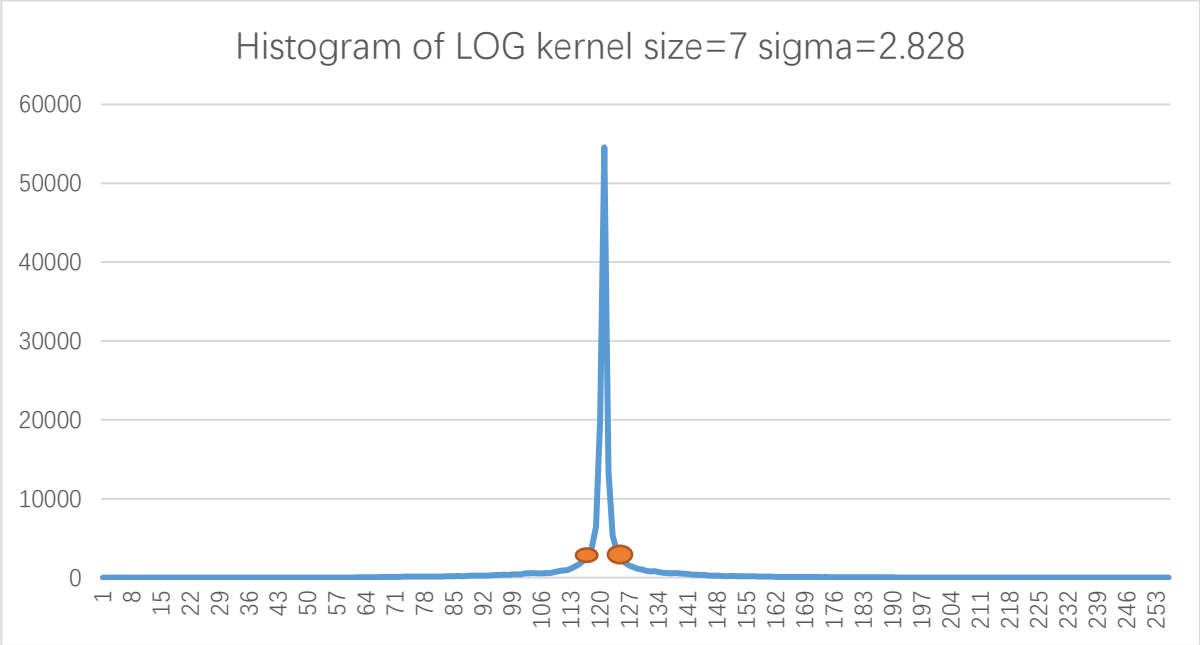



Figure11 Histogram of LOG kernel size=7 sigma=2.828 (boat.raw)

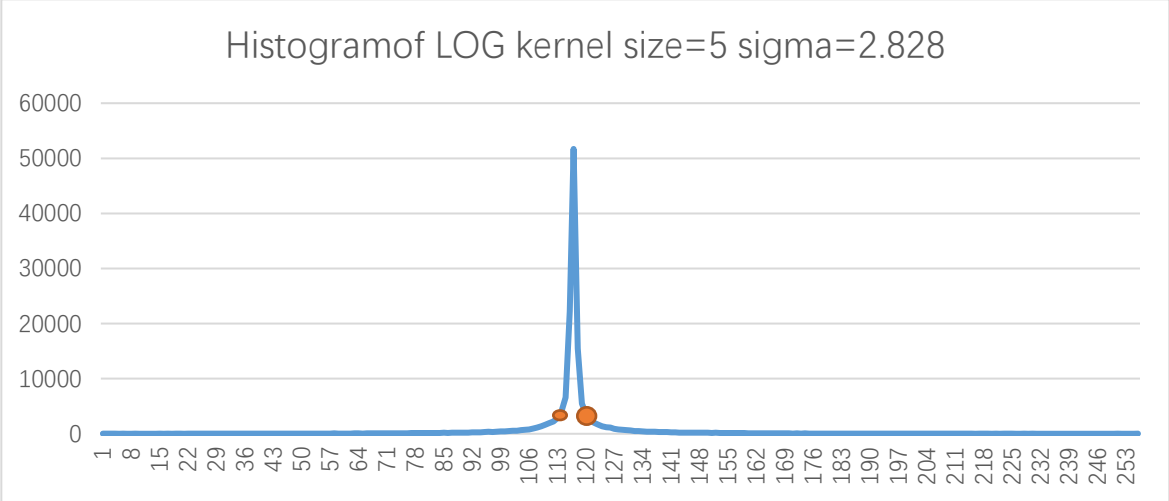



Figure12 Histogram of LOG kernel size=5 sigma=2.828 (boat.raw)

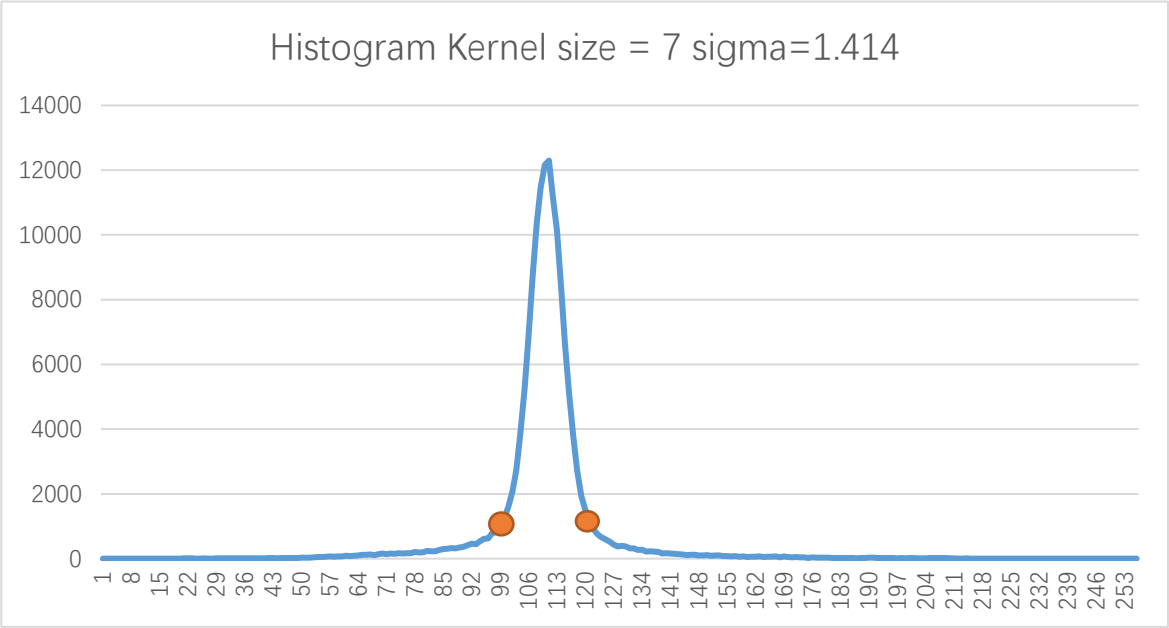



Figure13 Histogram of LOG kernel size=7 sigma=1.414 (boat_noisy.raw)

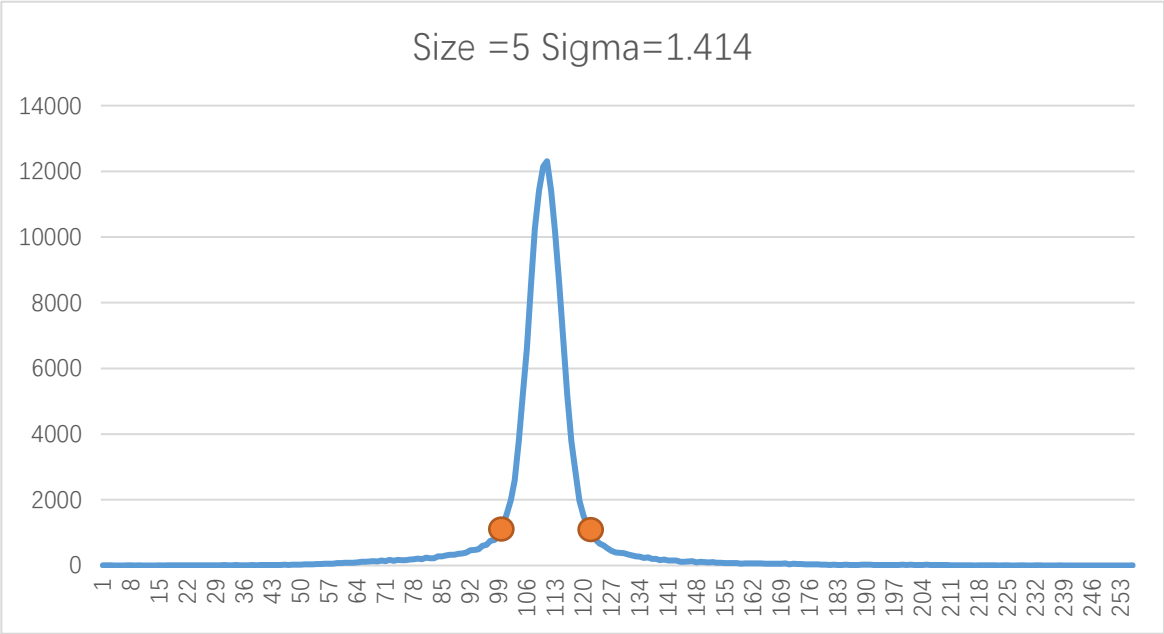



Figure14 Histogram of LOG kernel size=5 sigma=1.414 (boat_noisy.raw)

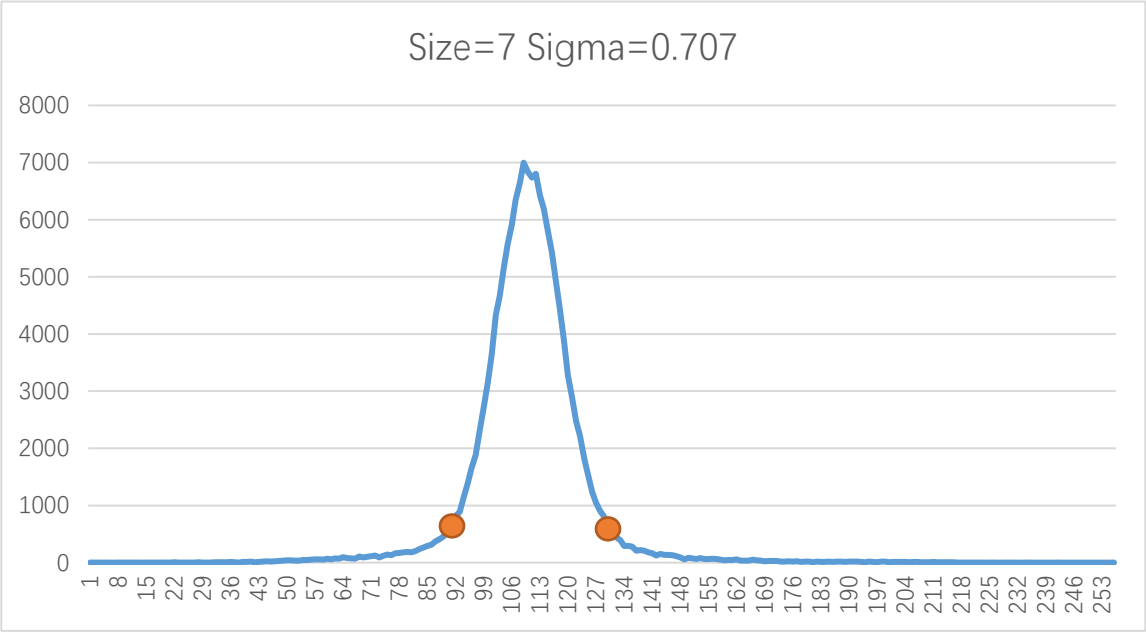



Figure15 Histogram of LOG kernel size=7 sigma=0.707 (boat_noisy.raw)

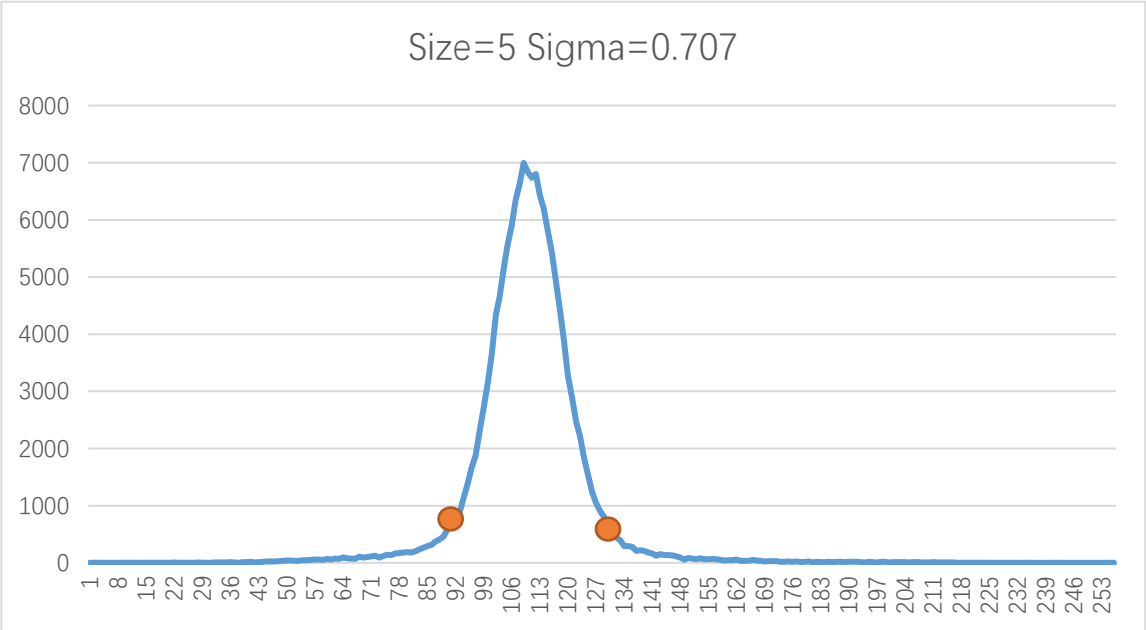



Figure16 Histogram of LOG kernel size=5 sigma=0.707 (boat_noisy.raw)

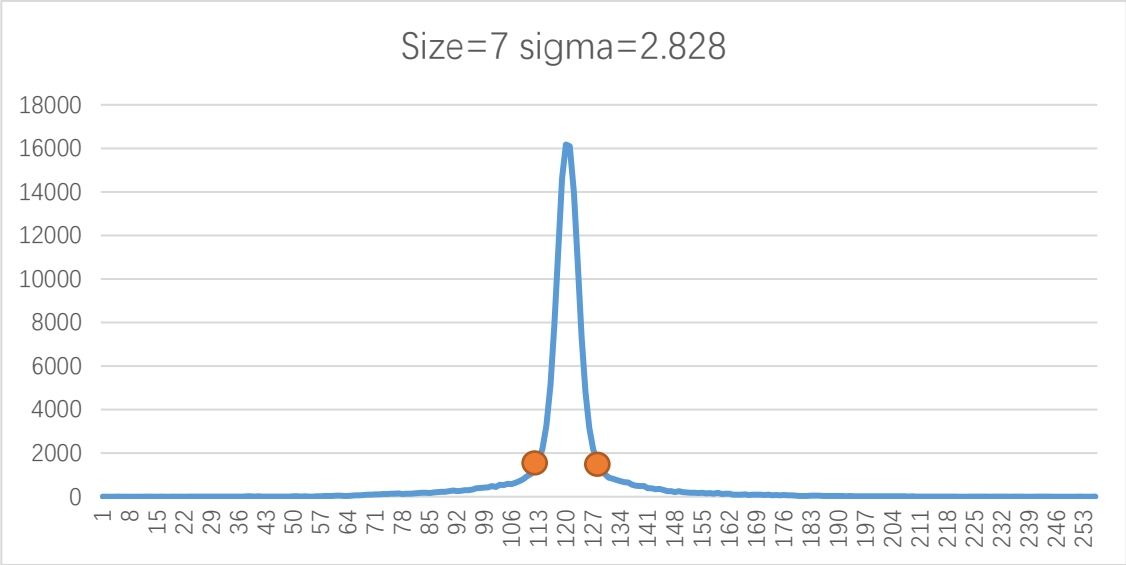



Figure17 Histogram of LOG kernel size=7 sigma=2.828 (boat_noisy.raw)

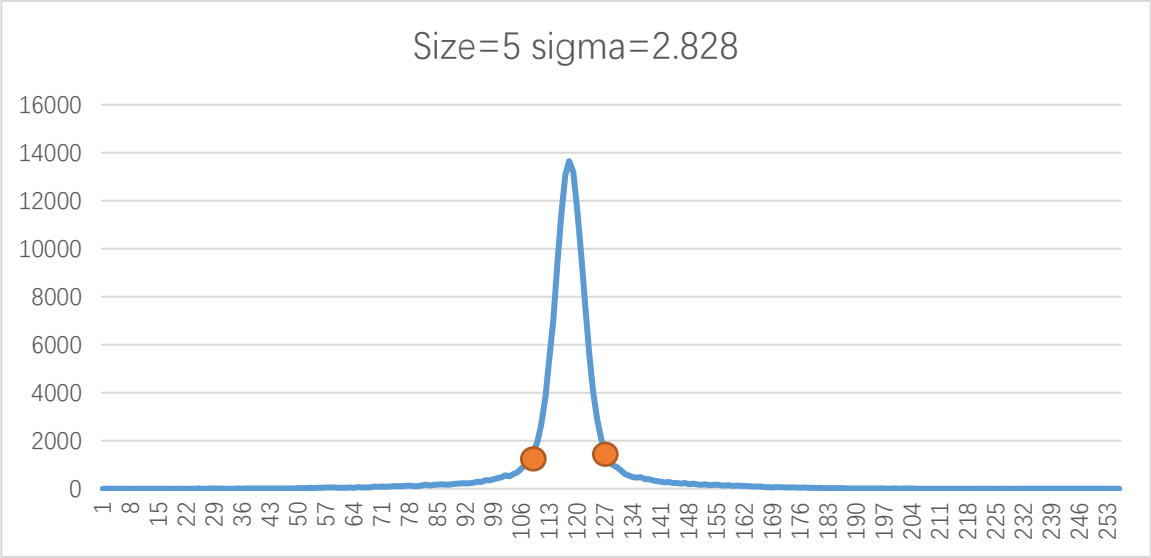



Figure18 Histogram of LOG kernel size=5 sigma=2.828 (boat_noisy.raw)

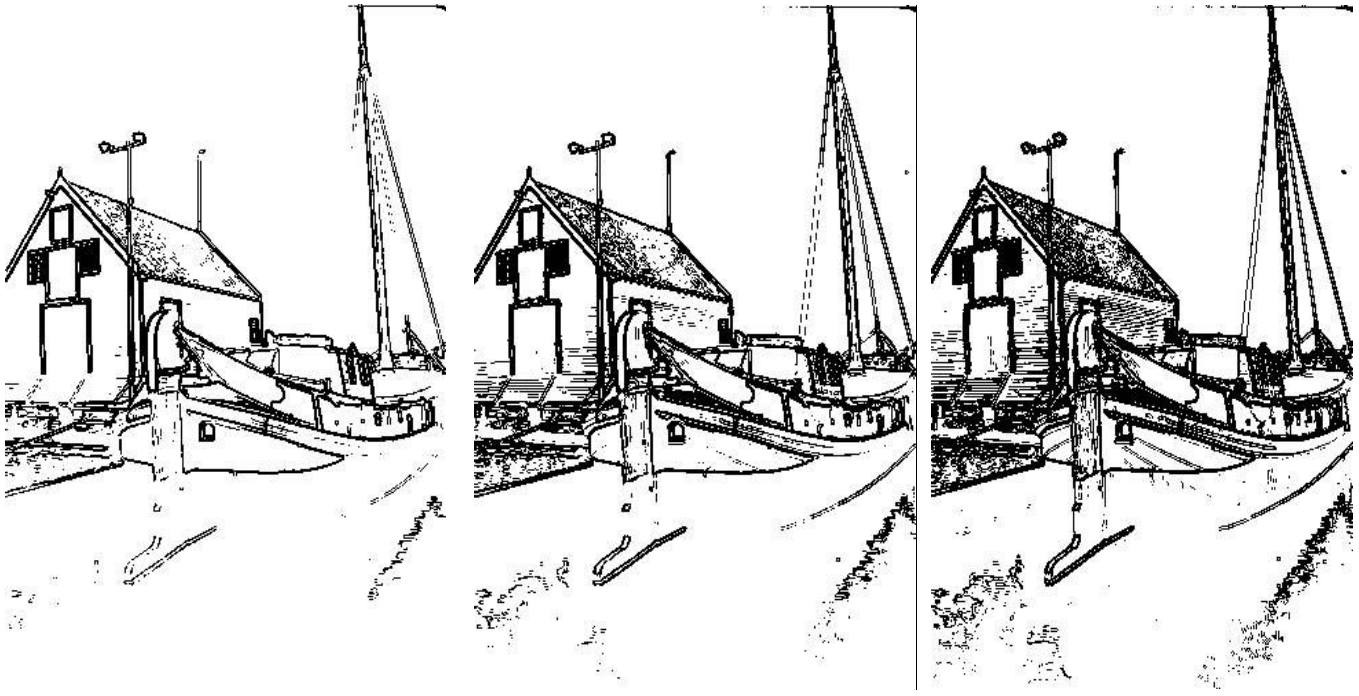


Figure19 Sobel Edge Detection Results for Boat.raw(Threshold= 0.9, 0.85, 0.8)

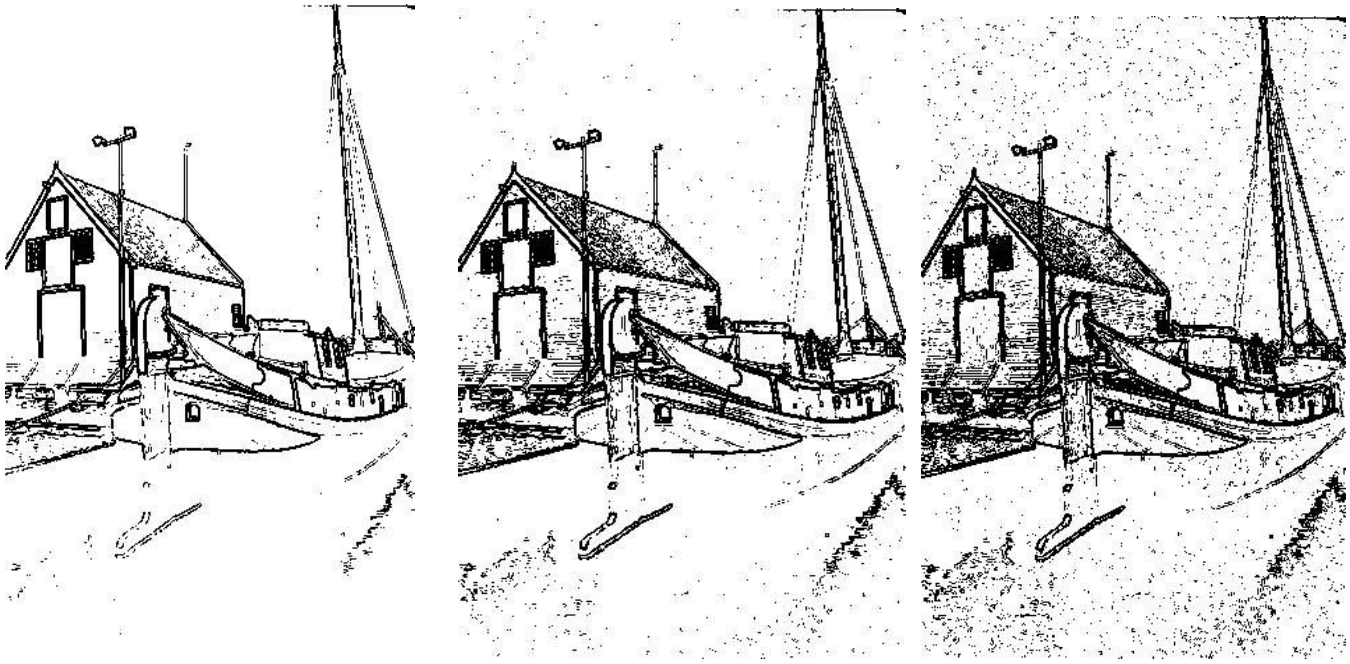


Figure20 Sobel Edge Detection Results for Boat_noisy.raw (Threshold= 0.9, 0.85, 0.8)

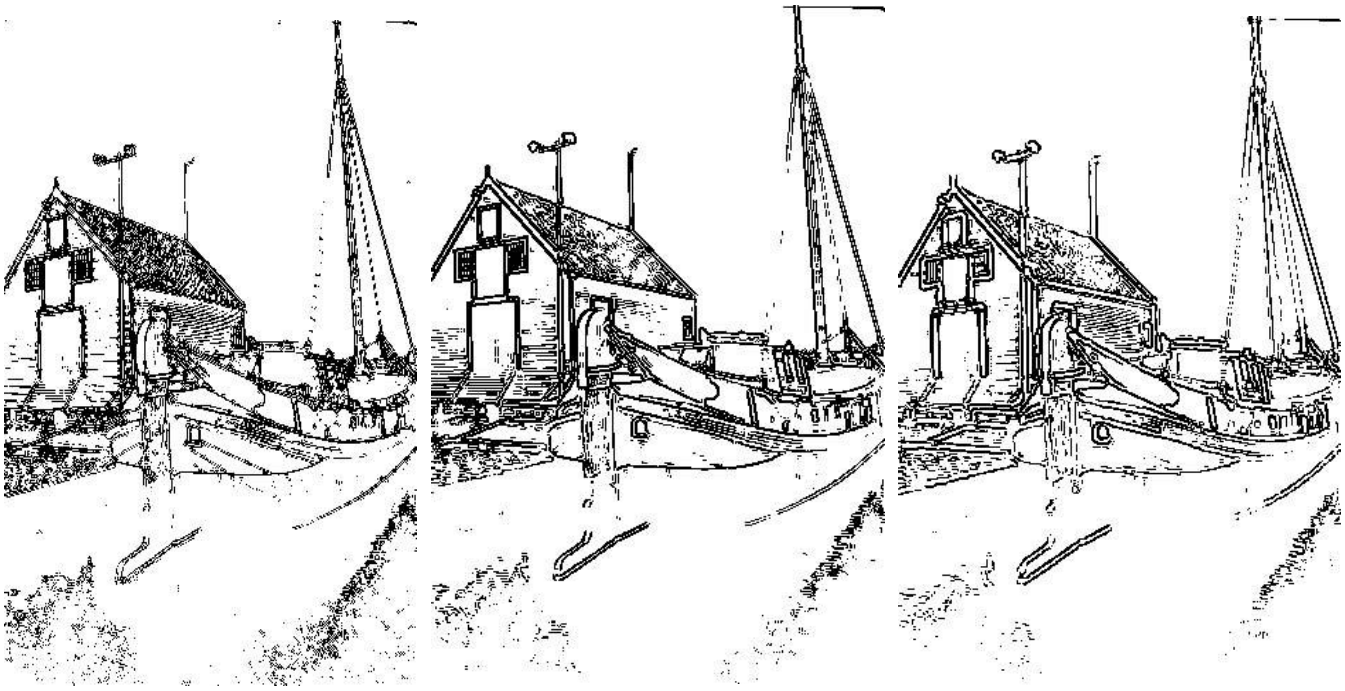


Figure21 LOG Edge Detection Results for Boat.raw (Gaussian kernel size =5,sigma= 0.707, 1.414, 2.828)

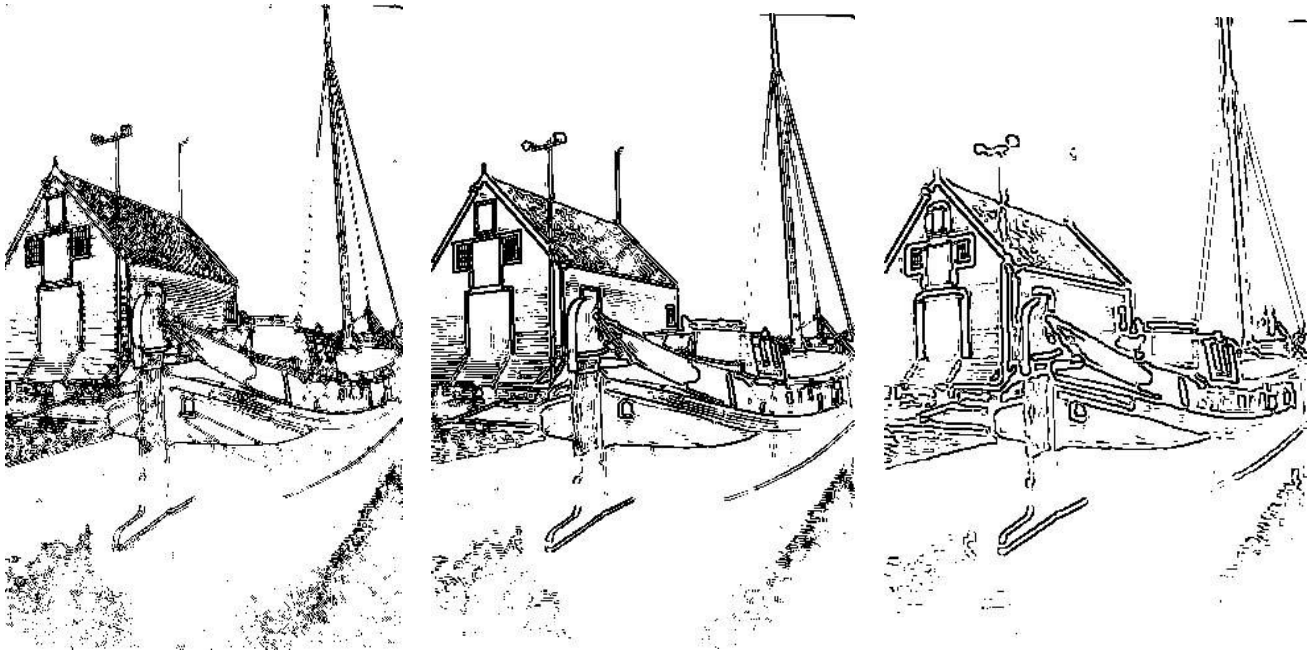


Figure22 LOG Edge Detection Results for Boat.raw (Gaussian kernel size =7,sigma= 0.707, 1.414, 2.828)

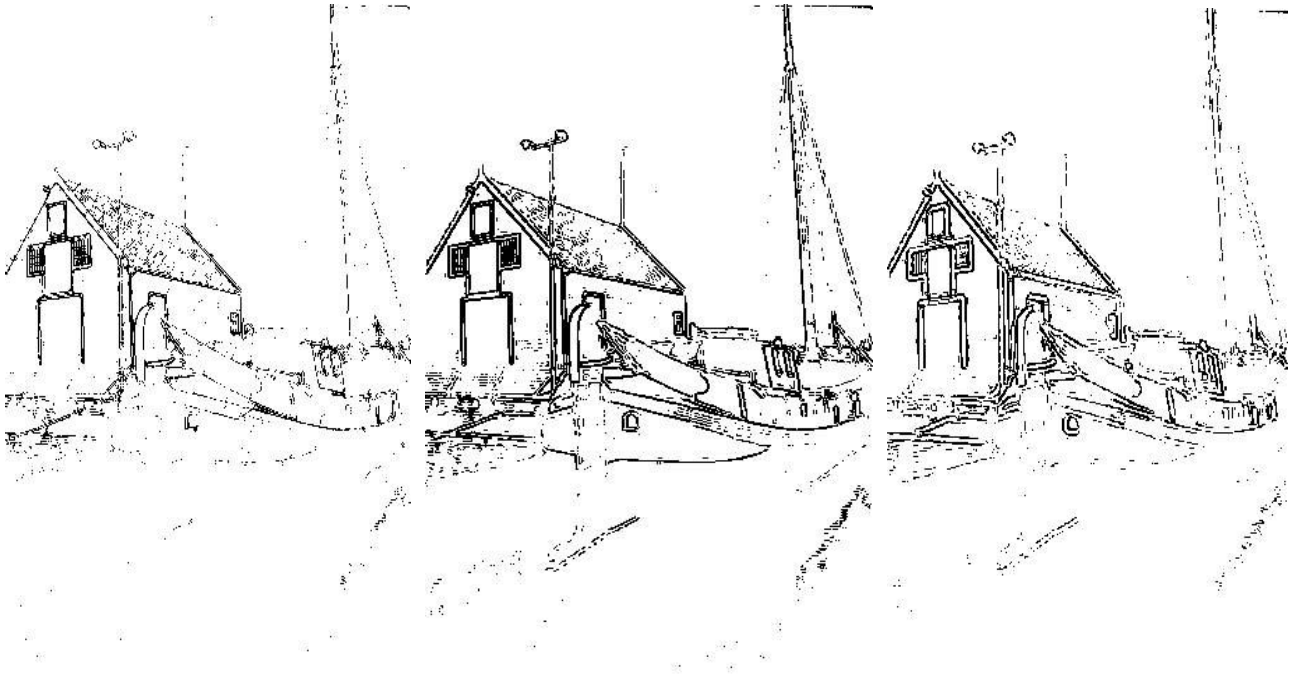


Figure23 LOG Edge Detection Results for Boat_noisy.raw (Gaussian kernel size =5,sigma= 0.707, 1.414, 2.828)

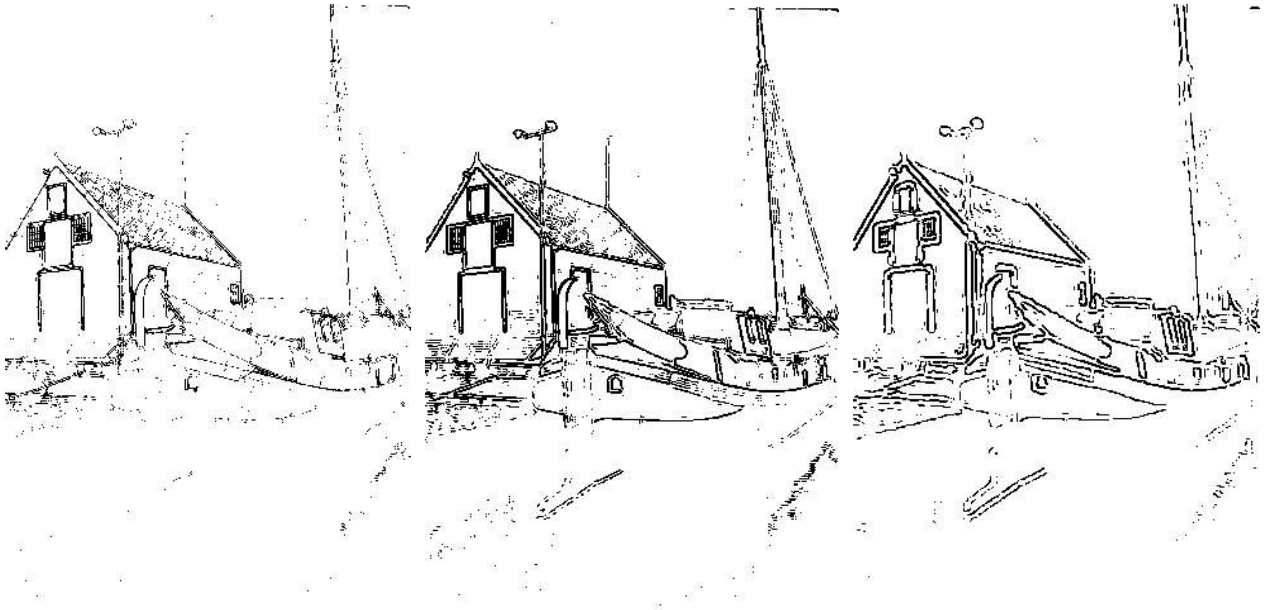


Figure24 LOG Edge Detection Results for Boat_noisy.raw (Gaussian kernel size =7,sigma= 0.707, 1.414, 2.828)

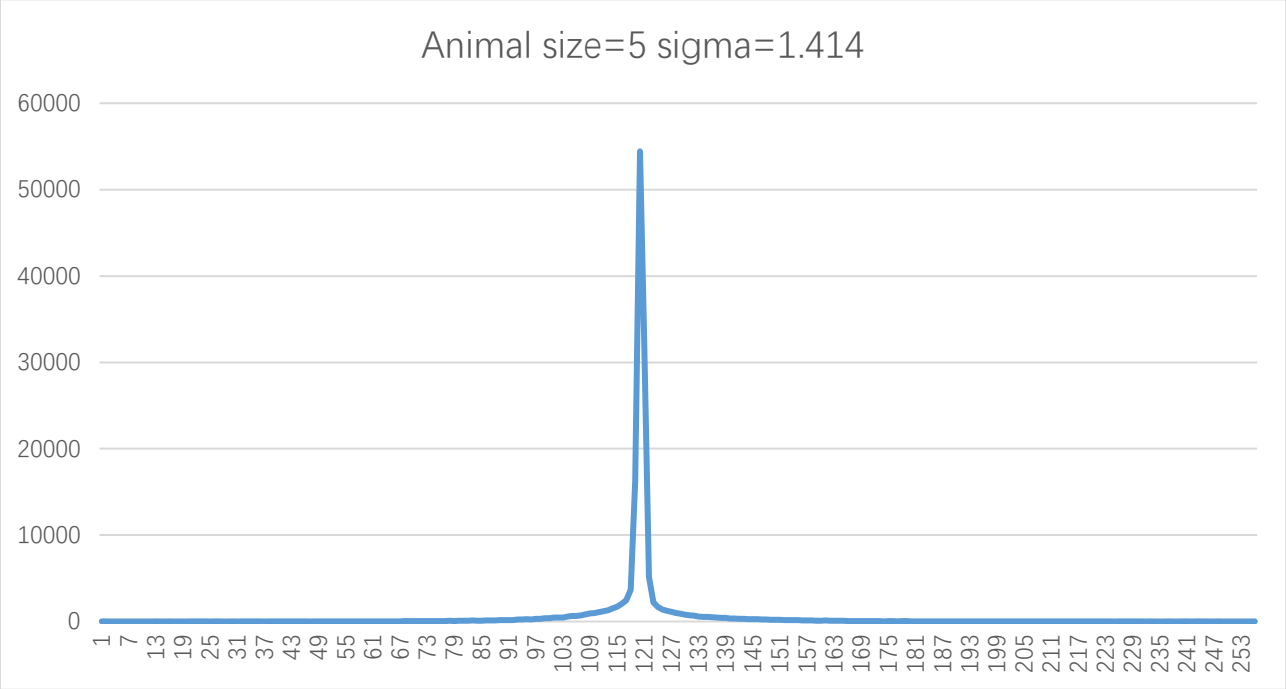



Figure25 Histogram of LOG Filter size=5 sigma=1.414(Animal.raw)

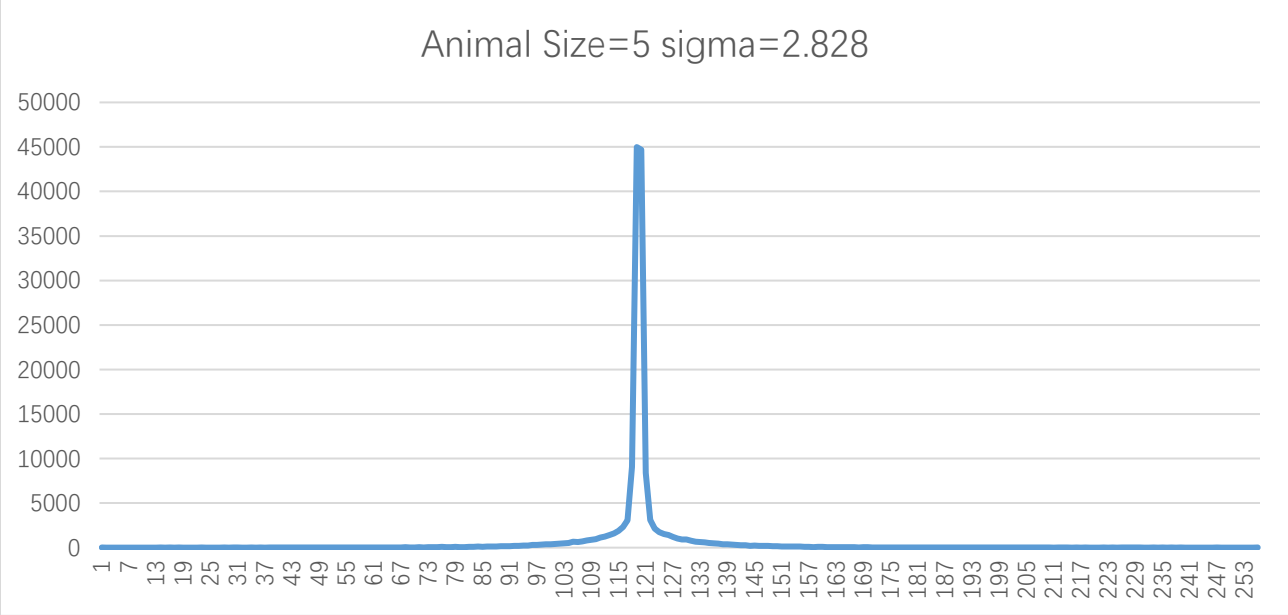



Figure26 Histogram of LOG Filter size=5 sigma=2.828(Animal.raw)

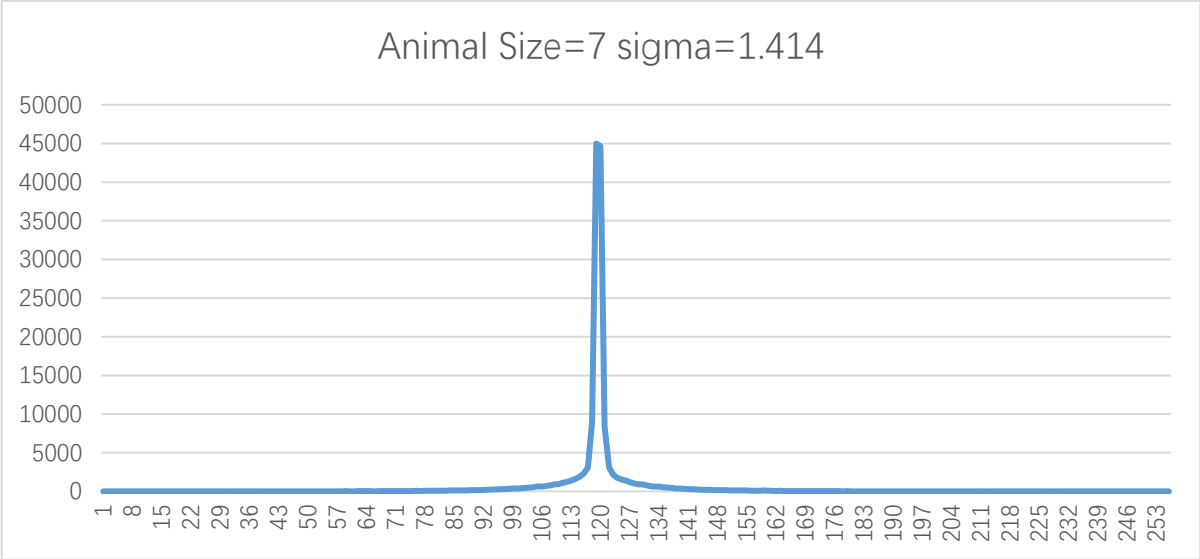



Figure27 Histogram of LOG Filter size=7 sigma=1.414(Animal.raw)

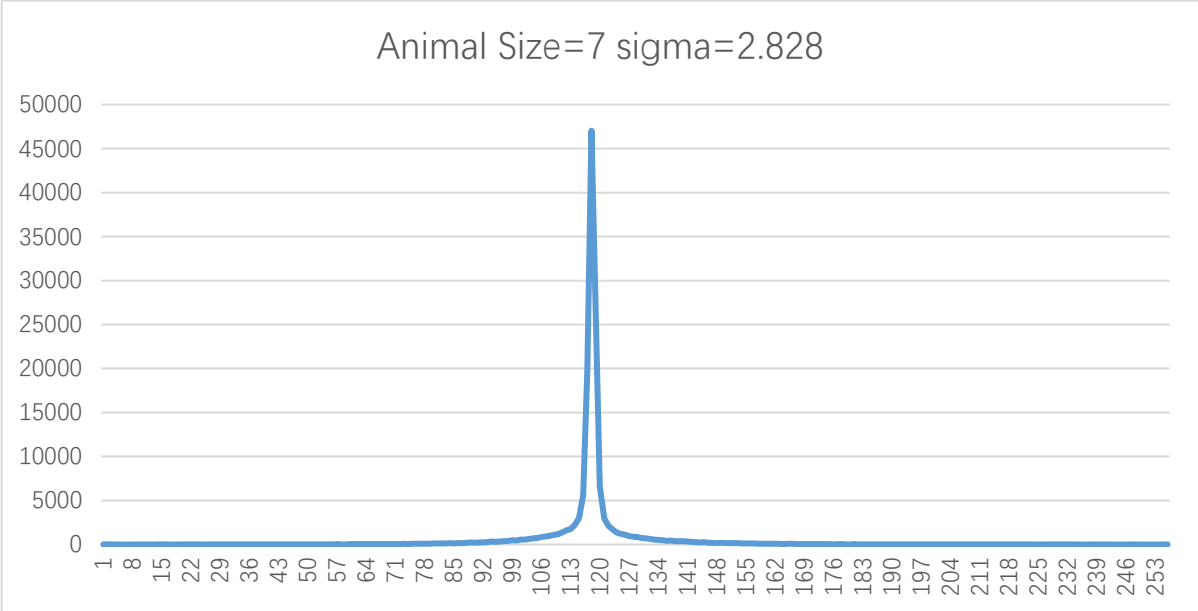



Figure28 Histogram of LOG Filter size=7 sigma=2.828(Animal.raw)

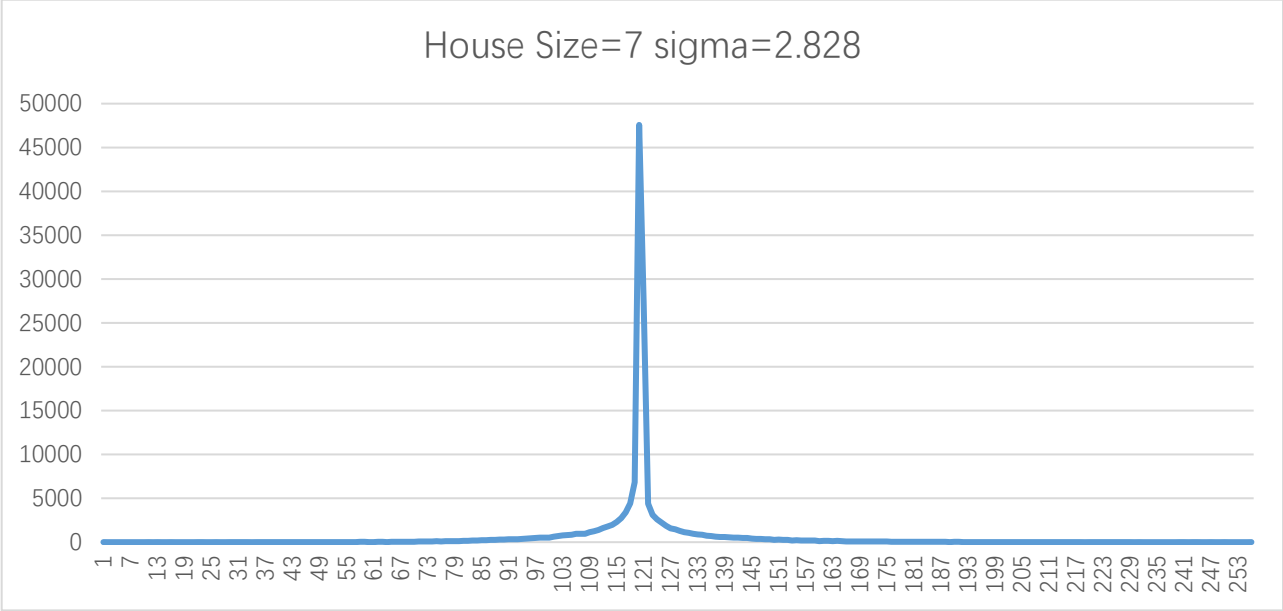



Figure29 Histogram of LOG Filter size=7 sigma=2.828(House.raw)

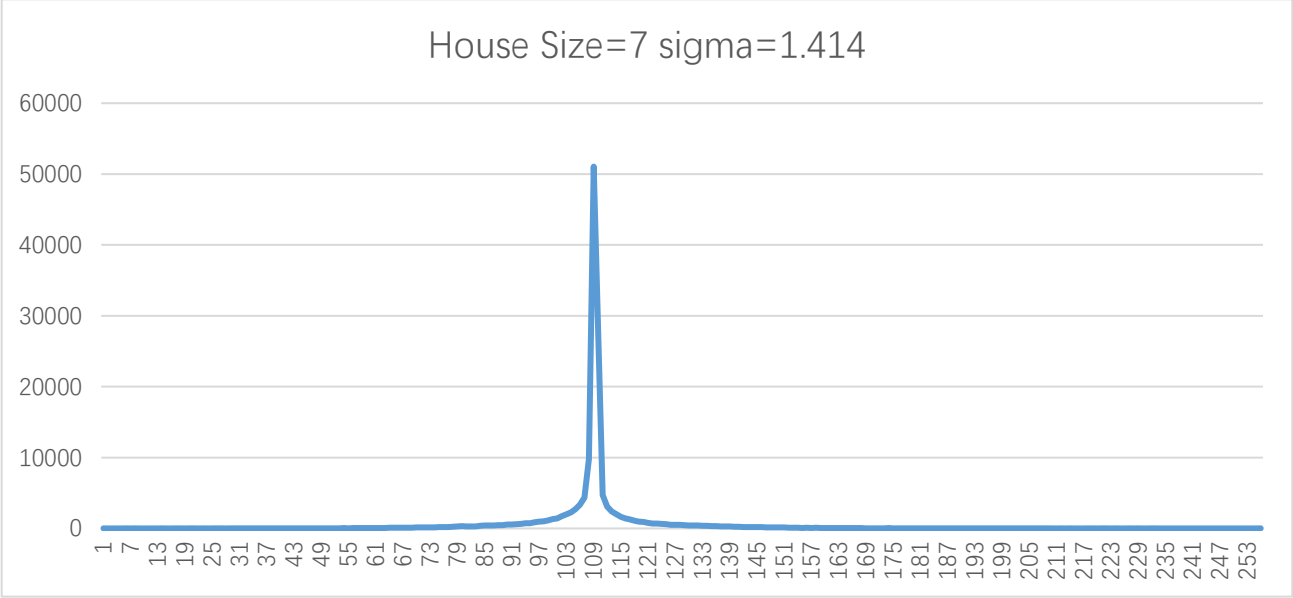



Figure30 Histogram of LOG Filter size=7 sigma=1.414(House.raw)

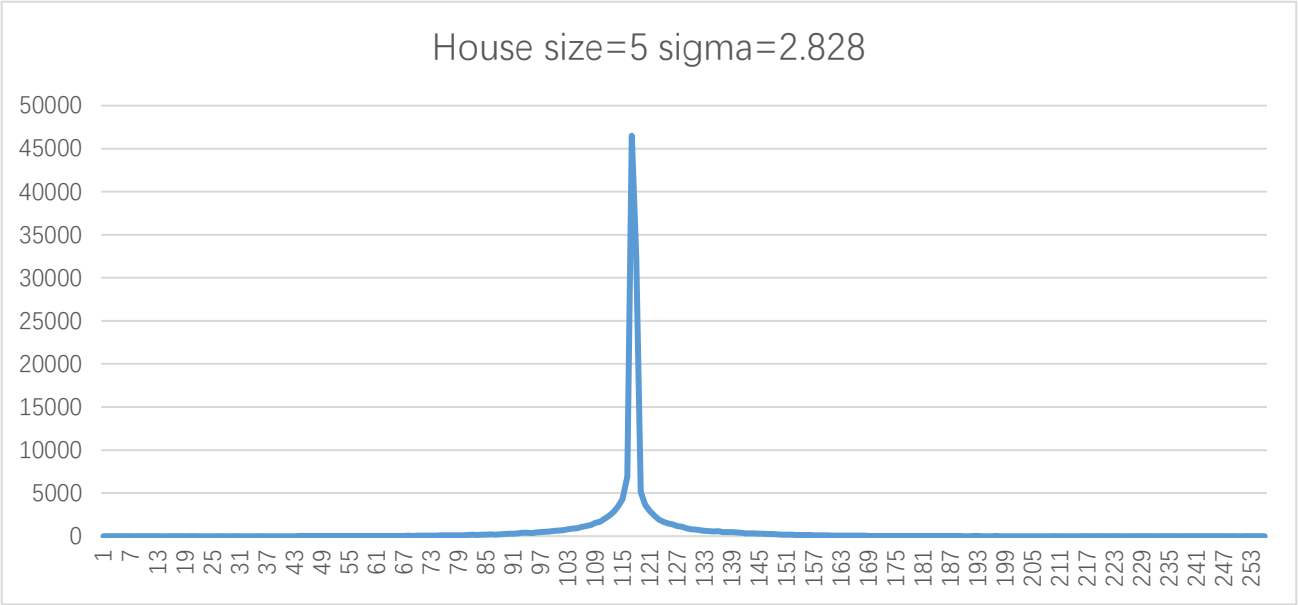



Figure31 Histogram of LOG Filter size=5 sigma=2.828(House.raw)

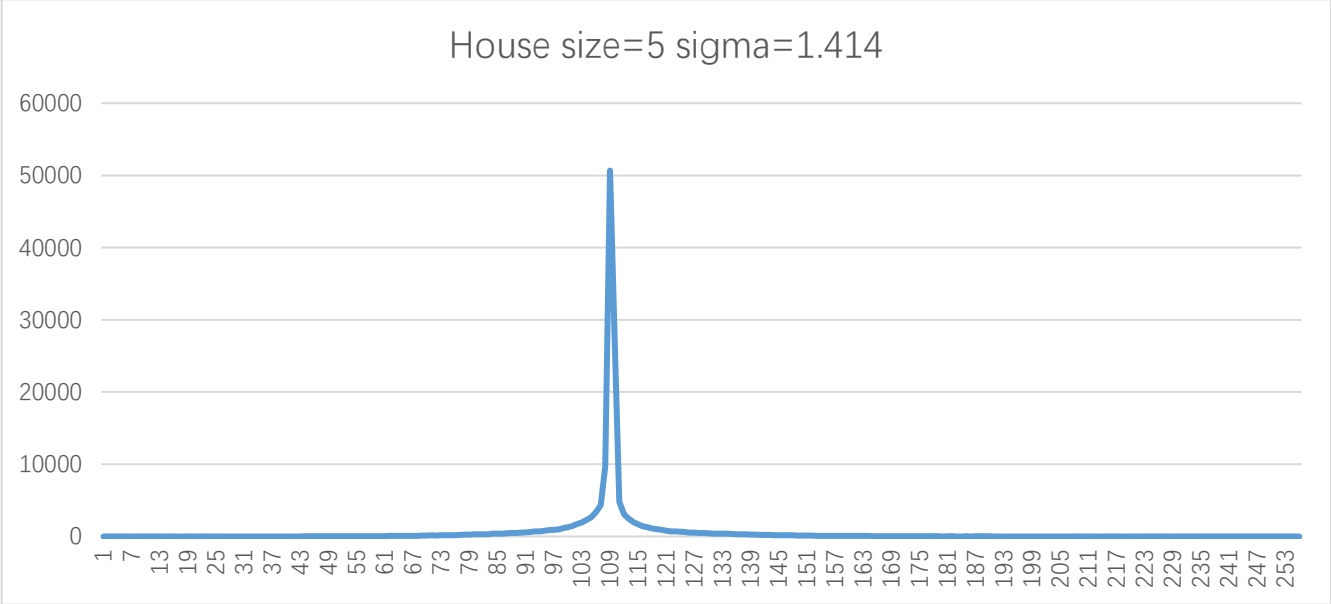



Figure32 Histogram of LOG Filter size=5 sigma=1.414(House.raw)

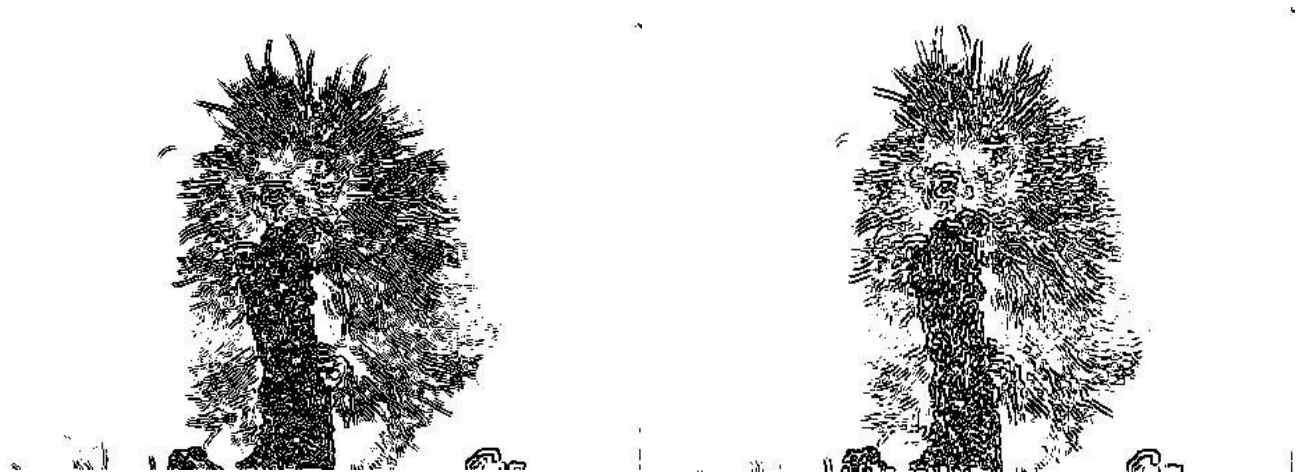


Figure33 LOG Edge Detection Results for Animal.raw (Gaussian kernel size =5,sigma= 1.414, 2.828)

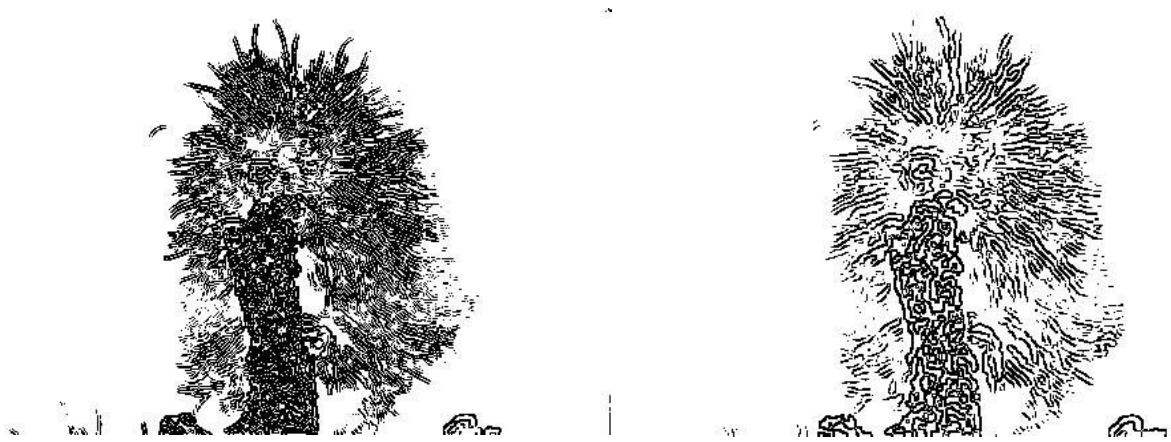


Figure34 LOG Edge Detection Results for Animal.raw (Gaussian kernel size =7,sigma= 1.414, 2.828)

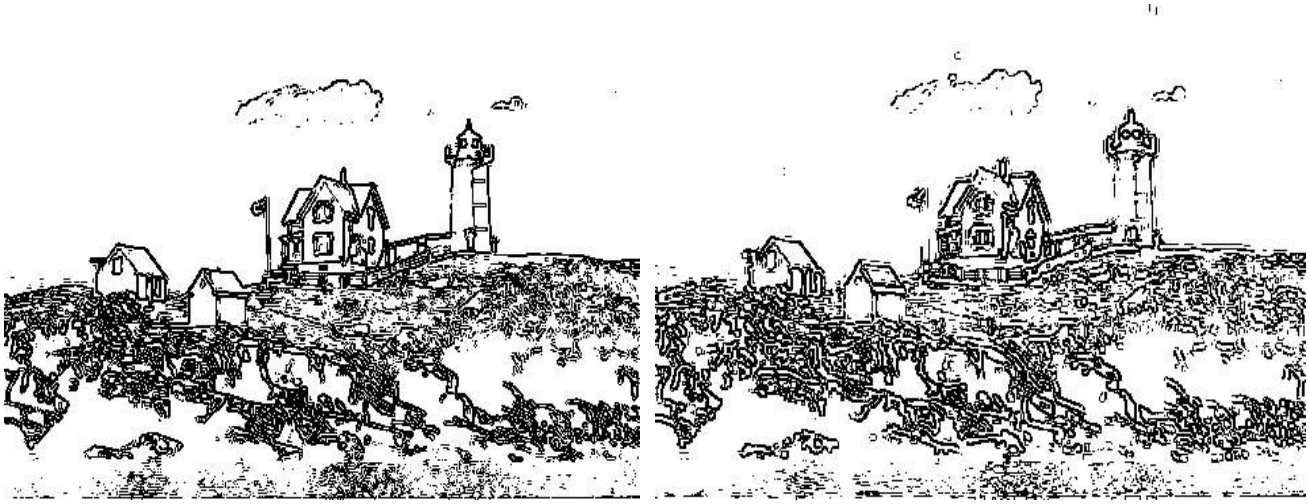


Figure35 LOG Edge Detection Results for House.raw (Gaussian kernel size =5,sigma= 1.414, 2.828)

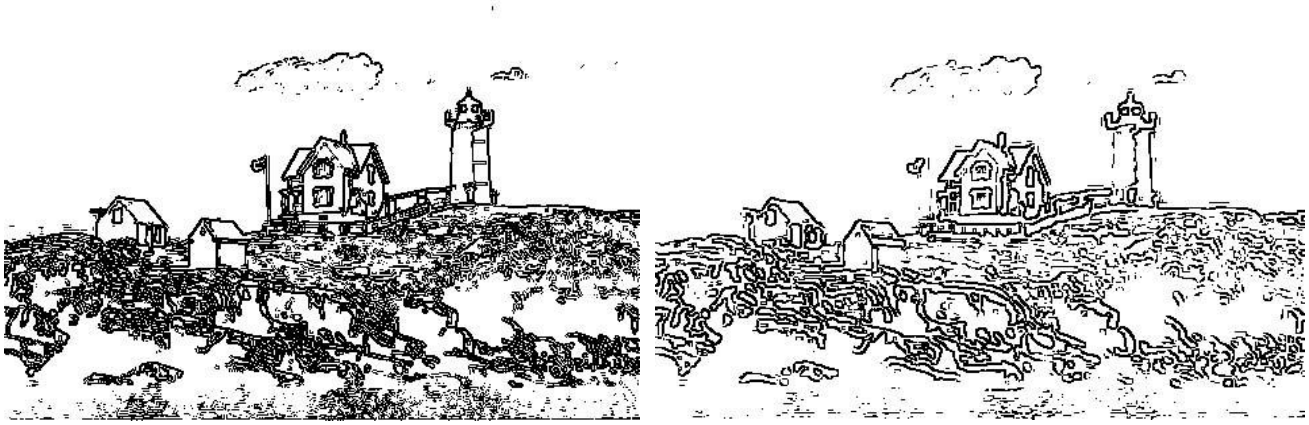


Figure36 LOG Edge Detection Results for House.raw (Gaussian kernel size =5,sigma= 1.414, 2.828)

### Discussion:

It is obvious that the results from Sobel detector is worse than the results from LOG detection. We can see that there are still some pixels that are not edges in the results from Sobel detector such as the window part of the house even though the threshold is 0.9. Also, the Sobel detector is vulnerable to the noise. We can conclude that when the threshold decrease there are many noise in the image. Because the noise is Gaussian, when we choose low threshold, the noise will come up. On the contrary, the results from LOG is much better. Because the LOG consists of Laplacian and Gaussian kernel, it can remove the Gaussian noise before edge detection. Moreover, different size and different sigma will bring different results. Based on my visualization, the results from kernel size 7 is much better that from kernel size 5. Also, the results from large sigma is better than results from small sigma. Also, if the histogram distributes in a small range, the results are better.

## b. Structured Edge

### Abstract and Motivation

After finishing problem a, we can easily find that the edge detection methods such as Sobel Edge Detection and LOG Edge Detection are not very good. There are some limitations of them including the precision. In this problem, we can have access to a creative method called “Structured Edge Detection”. It is based on random forest classifier.

**Task:** To apply Structured Edge Detection by using open source code. Compare and comment on the visual results of the Sobel detector and the SE detector.

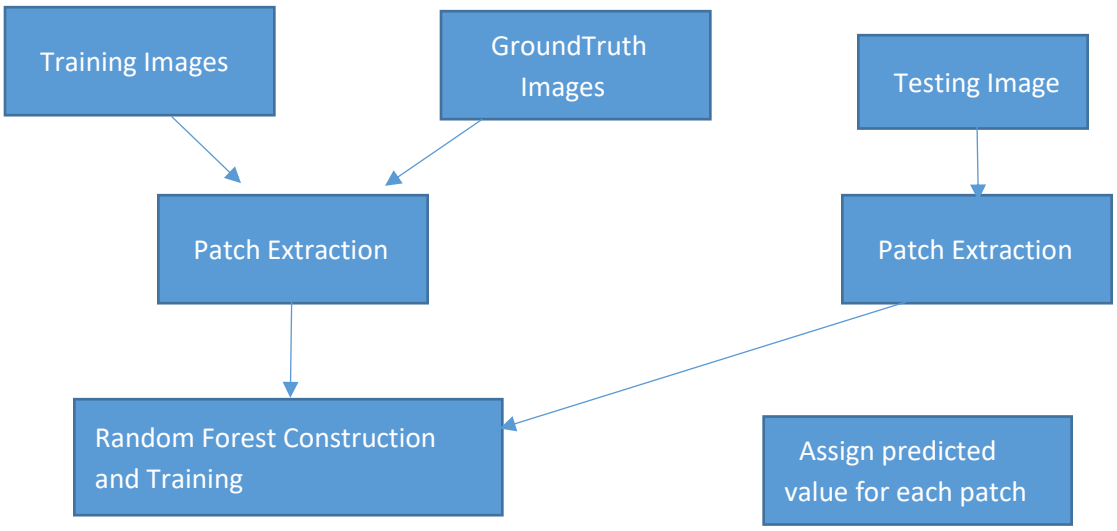


Figure 37 Flowchart of structured Edge Detection algorithm

## Approaches and Procedures

**Theoretical Approach:**

First, we have to construct and train our random forest. In the open source code, there is a training data set in the directory so that we start from here. The algorithm is patch-based so that the algorithm will divide the input into several patches(32*32*K). K is the number of channels. The feature used in this algorithms is based on CIE-LUV color space and gradient magnitude. After that the channels of image patch increased to 13. Then, downsampling it to a resolution of 5*5. We can get over 7000 candidate features per patch.

Second, we have to train the Random Forest algorithm to mapping the features. It is important to find a good mapping function in this algorithm. Also, the structure labels y are 16*16 segmentation masks. This is a binary image and the edge pixel value is 255. Using Euclidean distance is computational complex. According to thesis, we can sample a pair of pixels with different index and check if their values are equal. We can achieve mapping Y to Z by this decision rule and indicator function.

Third, we have to find an ensemble model to merge multiple segmentation masks. In this algorithms, the model we choose is to average all segmentation masks. Therefore, we can get a probability map image in the end.

Fourth, after finishing the random forest training, we can input our test image to extract the edge information. Initially, we have to extract patch-based features. Then, using trained RF to generate segmentation masks. Last, combining these results together and output the probability map image.

Random Forest:

A random forest consists of several decision trees. Therefore, if we want know the principle of Random forest, we can start from the principle of a single decision tree. A single decision tree can classify any input x by branching left or right until finding the final leaf node. In order to express this kind of rule, we can use the split function:

$$h(x, \theta_j) \in \{0, 1\}$$

For a certain $\theta_j$, if the value of h is 0, then the tree branches to left. We can use Y to denote the set of the output of x.

Decision forest consists of several independent decision trees. In Structured Edge Detection, The goal of training the decision forest is to find a good split function. The most common way to solve it is calculating the information gain. For the training of the decision tree we use bagging technique. A set of such trees are trained independently to find the parameters in the split function that result in a good split of data. Splitting parameters are chosen to maximize the information gain.

## Results and Discussion

The source code of this program is from[2]. Default Parameter choice:

```
%% set detection parameters (can set after training)
model.opts.multiscale=0;          % for top accuracy set multiscale=1
model.opts.sharpen=2;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=4;            % max number threads for evaluation
model.opts.nms=0;                 % set to true to enable nms
```

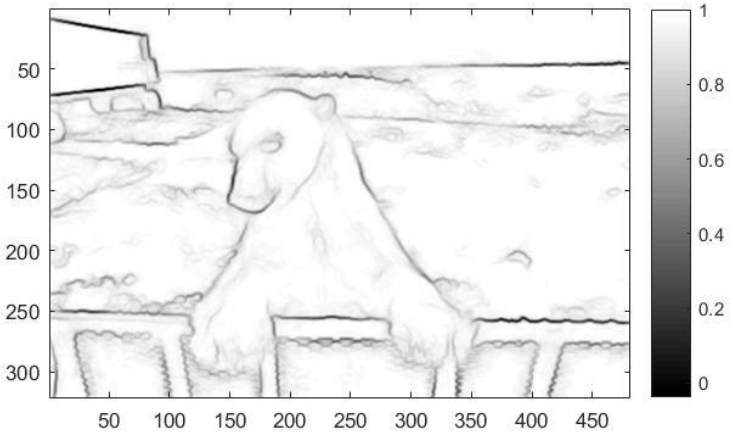
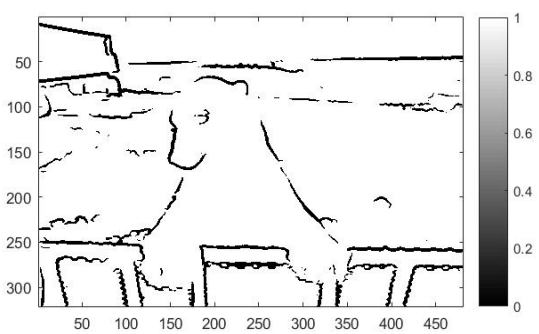
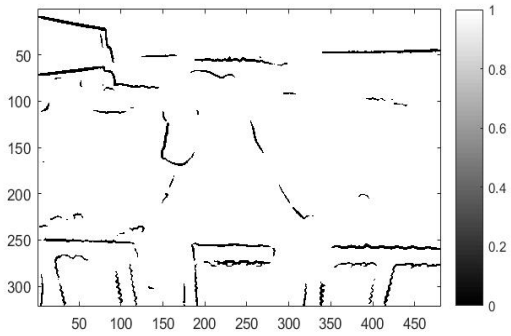



Figure 38 Probability Map image for animal.raw using SE detector

Parameter changes:

(1) Threshold: 0.2,0.3,0.4:



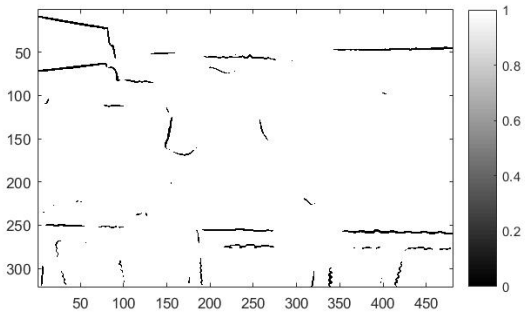



Figure39 Threshold changes(0.2, 0.3, 0.4)

We can see that if the probability threshold is large, some useful edge information is filtered. However, if it is small, some pixels that are not edges show up. Therefore, we should choose threshold carefully.

(2) Multiscale:0 or 1:

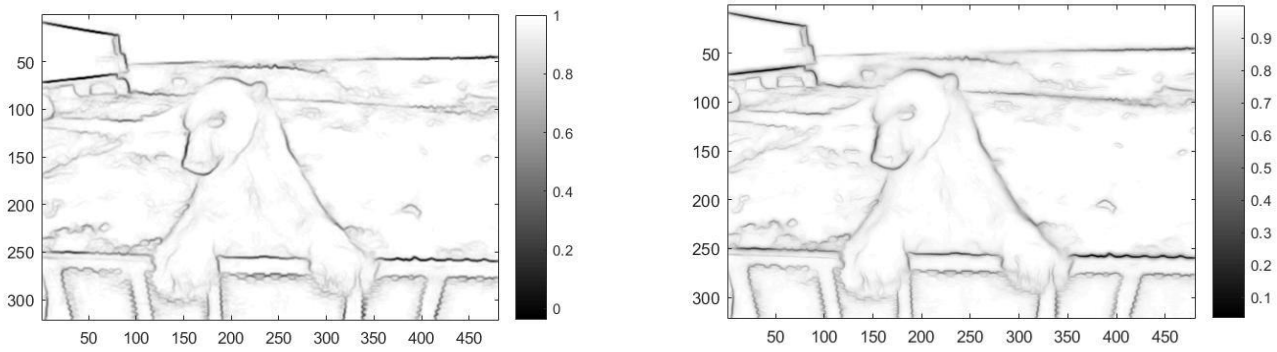



Figure40 multiscale changes (0,1)

We can see that the difference between 2 images are not very obvious. There are some darker lines in the second image. According to official documentation, if we want to get more accuracy we would like to choose multiscale equal to 1. However, the efficiency of choosing multiscale=1 is lower than the situation that the multiscale=0. The running time for default parameter choosing is 0.089s. The running time for the second choosing is 0.048s.

(3) Sharpen (0,1,2):

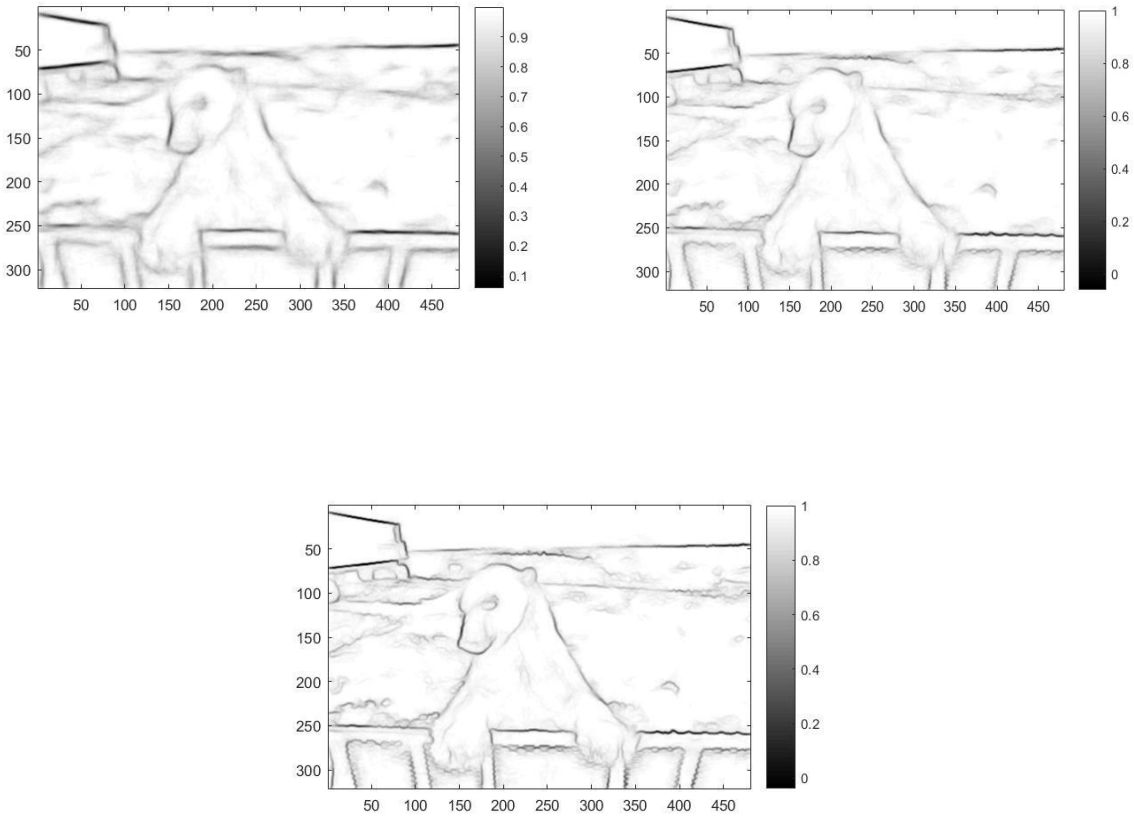


Figure41 sharpen changes (0,1,2)

We can see clearly that if we choose sharpen=2, the effect is the best. Small sharpen parameter means the image will be blur. Although the running time for small sharpen will be reduced(0.047s, 0.071s), in order to get good results I'd like to choose sharpen=2.

(4) Number of TreesEval(1,2,4)

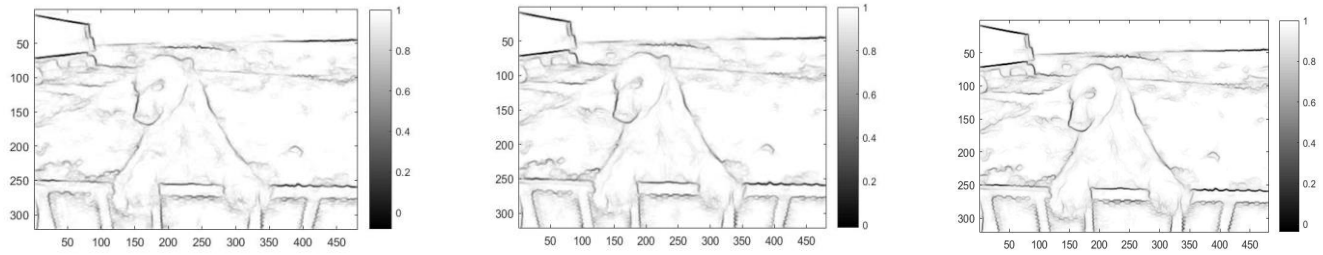


Figure42 # of TreesEval(1,2,4)

The smaller number of evaluation tree, the higher efficiency we can get. The running time is 0.047s, 0.053s and 0.089s. However, if we want to get better results, we have to choose the big number of evaluation trees.

(5) Number of thread(1,4)

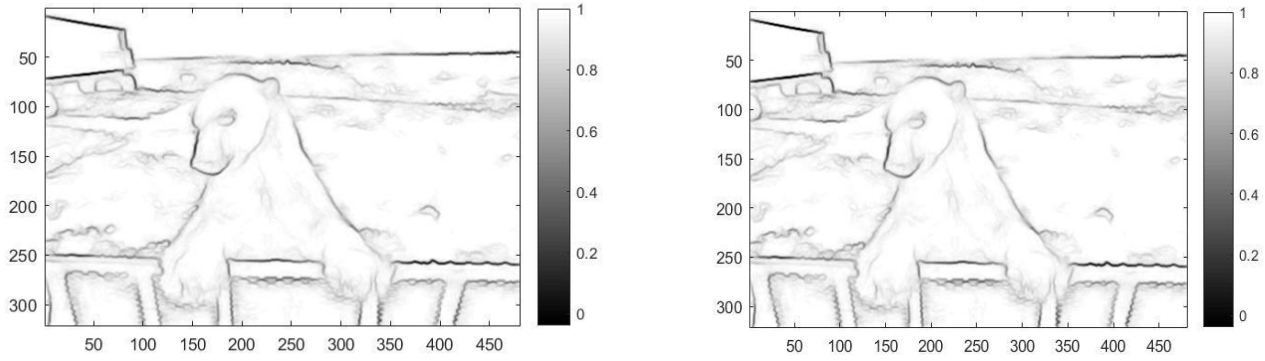



Figure43 # of Threads (1,4)

This parameter is only associated with the running time. It has no contribution to the quality of image. The running time for choose it equal to one is 0.184s. The other is 0.89s.

(6) Nms(0,1)

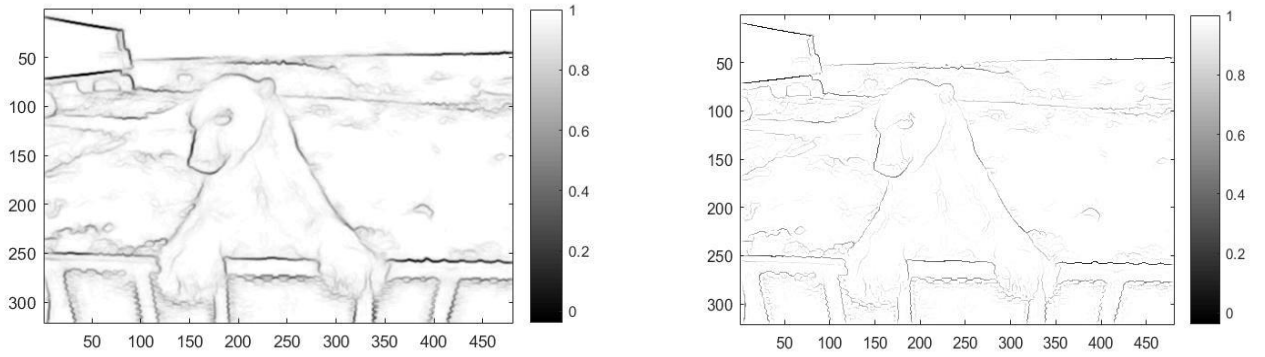



Figure44 output when choosing nms(0,1)

We can conclude that if we choose NMS, the edge contour of the image will be thinner. Therefore, in order to get better results, we should set nms equal to 0.

The results of Animal.raw and House.raw using SE detector and default parameters.

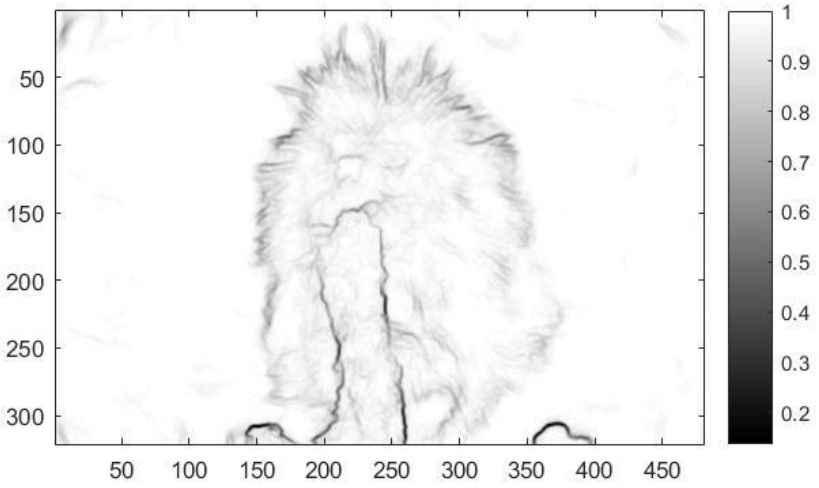



Figure45 Probability Map image for animal.raw using SE detector

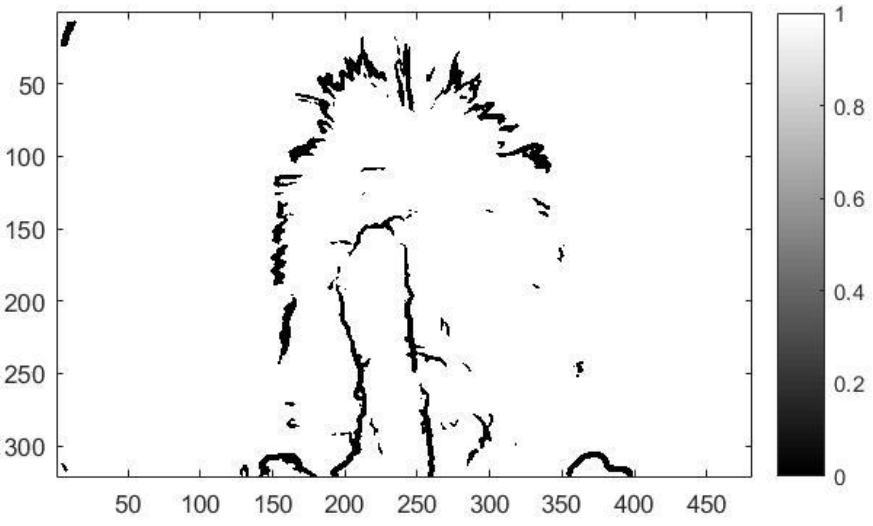



Figure46 Binary Image for animal.raw (choosing threshold=0.15 for probability map image)

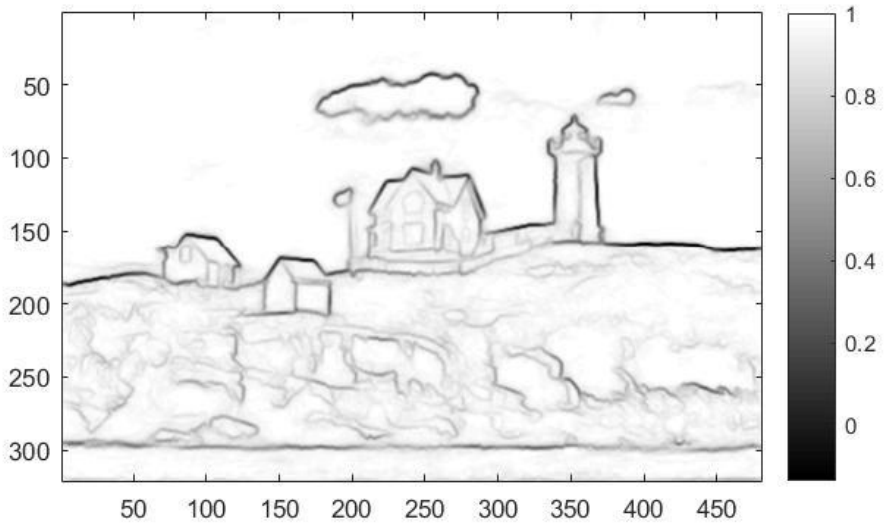



Figure 47 Probability Map image for House.raw using SE detector

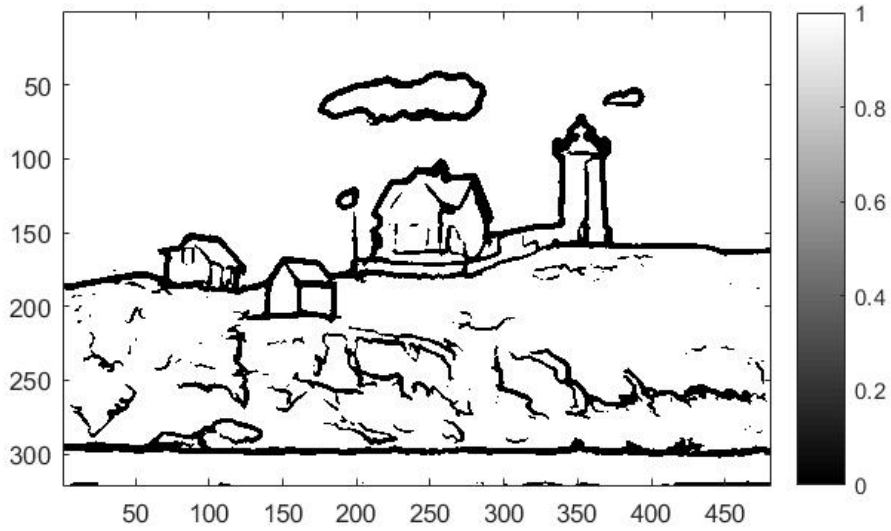



Figure 48 Binary Image for house.raw (choosing threshold=0.15 for probability map image)

Figure49 Results from LOG filter

According to previous results, we can see that the edges from the Sobel and LOG detector are much more than the results from SE. We can compare it to the original RGB image. It is obvious that there are much redundant information in the image from LOG and Sobel detector.

# c. Performance Evaluation

## Abstract and Motivation

We have implemented 3 edge detection algorithms. It is important to know how to evaluate them. It is not accurate to judge them only by our visualization. In this problem, we can calculate some critical statistics that can tell us which algorithms is better.

**Task:** Implement F-measure method to evaluate the performance of 3 edge detectors.

## Approaches and Procedures

In this problem, we use the data from BSDS500 as our ground truth. We select the ground truth file of animal.raw and house.raw. We can regard the ground truth as a template. The only thing we will do is to compare the results from all kinds of edge detectors with the ground truth. The bigger F measure, the better edge detector.

There are some notions of terms:

True positive: Edge pixels in the results from all kinds of edge detectors correspond to edge pixels in the ground truth.

True Negative: Non-edge pixels in the results from all kinds of edge detectors correspond to non-edge pixels in the ground truth.

False positive: Edge pixels in the results from all kinds of edge detectors correspond to the non-edge pixels in the ground truth.

False negative: Non-edge pixels in the results from all kinds of edge detectors correspond to the true edge pixels in the ground truth.

Here are the F-measure calculation formula:

$$\text{Precision}: P = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}$$

$$\text{Recall}: R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}$$

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

**Implementation by calling the SE toolbox function edgesEvalImg:**

[thrs,cntR,sumR,cntP,sumP,V] = edgesEvalImg(E, groundTruth(1));

The input E is the probability map image from the edge detector.

This function will return 5 vectors. The size of thrs, cntR, sumR, cntP,sump is 99*1. Thrs is the threshold vector. Each entry of cntR, sumR, cntP, sumP corresponds to a certain threshold in thrs. Therefore, we can get a 99*1 F measure vector by implementing previous formula and we choose the maximal F measure as our final result.

## Results and Discussion:

Before evaluation, my intuition of which image would get bigger F measure is the House.raw. We can find the edge detection results in problem (a) and (b). For the Sobel and LOG, there are many irrelevant information in the final result such as the main part of the trunk. Also, there are less differences between the results of SE detector and the Sobel, LOG detector. Therefore, I think the House.raw will get larger F measure.

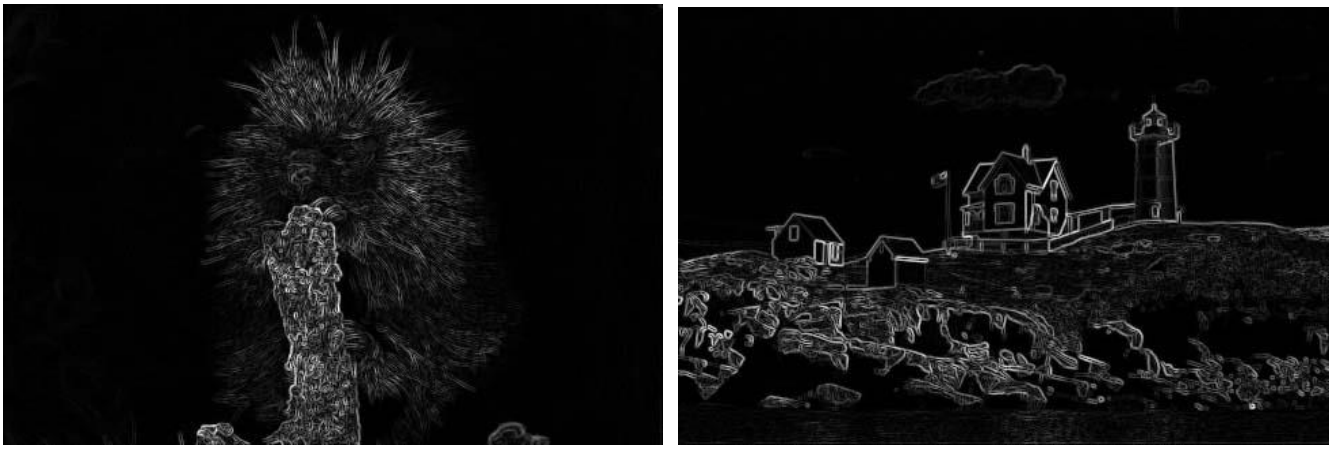


Figure50 Normalized images from Sobel Filter

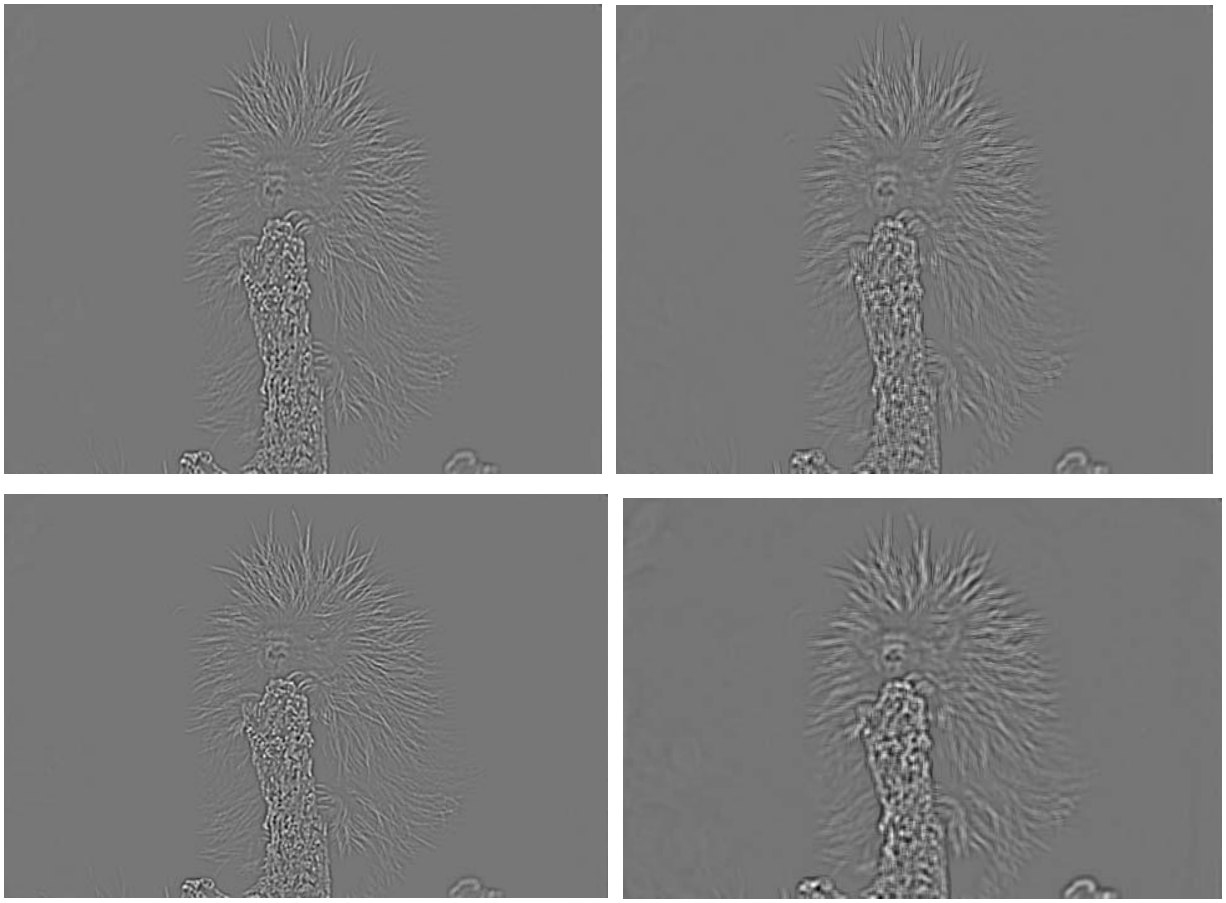


Figure51 Normalized images from LOG Filter

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.7477 | 0.5889 | 0.6589 |
| Ground Truth 2 | 0.7727 | 0.5214 | 0.6226 |
| Ground Truth 3 | 0.4479 | 0.4266 | 0.4370 |
| Ground Truth 4 | 0.7767 | 0.6191 | 0.6890 |
| Ground Truth 5 | 0.6936 | 0.5518 | 0.6146 |
| Mean | 0.6877 | 0.5416 | 0.6044 |

Table 1 F measure for SE detector(Animal.raw)

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.7367 | 0.7623 | 0.7493 |
| Ground Truth 2 | 0.7959 | 0.6465 | 0.7135 |
| Ground Truth 3 | 0.7557 | 0.6688 | 0.7096 |
| Ground Truth 4 | 0.5573 | 0.7111 | 0.6249 |
| Ground Truth 5 | 0.5259 | 0.7006 | 0.6008 |
| Mean | 0.6743 | 0.6979 | 0.6796 |

Table 2 F measure for SE detector(House.raw)

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Ground Truth 1 | 0.2234 | 0.2820 | 0.2493 |
| Ground Truth 2 | 0.2213 | 0.3180 | 0.2610 |
| Ground Truth 3 | 0.2719 | 0.2367 | 0.2531 |
| Ground Truth 4 | 0.2365 | 0.3783 | 0.2910 |
| Ground Truth 5 | 0.2150 | 0.3091 | 0.2536 |
| Mean | 0.2336 | 0.3048 | 0.2616 |

Table 3 F measure for Sobel detector(Animal.raw)

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.2153 | 0.4817 | 0.2976 |
| Ground Truth 2 | 0.2732 | 0.4575 | 0.3421 |
| Ground Truth 3 | 0.2559 | 0.4448 | 0.3249 |
| Ground Truth 4 | 0.3013 | 0.5355 | 0.3856 |
| Ground Truth 5 | 0.3803 | 0.7058 | 0.4943 |
| Mean | 0.2852 | 0.5251 | 0.3689 |

Table 4 F measure for Sobel detector(House.raw)

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.1849 | 0.6062 | 0.2834 |
| Ground Truth 2 | 0.2008 | 0.6804 | 0.3100 |
| Ground Truth 3 | 0.2026 | 0.2468 | 0.2225 |
| Ground Truth 4 | 0.2428 | 0.7128 | 0.3622 |
| Ground Truth 5 | 0.1690 | 0.8803 | 0.2835 |
| Mean | 0.2002 | 0.6253 | 0.2932 |

Table 5 F measure for LOG detector (Animal.raw) Size=5 sigma =1.414

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.1792 | 0.7597 | 0.2919 |
| Ground Truth 2 | 0.1936 | 0.8170 | 0.3130 |
| Ground Truth 3 | 0.1550 | 0.4122 | 0.2253 |
| Ground Truth 4 | 0.2399 | 0.7331 | 0.3615 |
| Ground Truth 5 | 0.1765 | 0.8100 | 0.2899 |
| Mean | 0.1888 | 0.7064 | 0.2963 |

Table 5 F measure for LOG detector (Animal.raw) Size=5 sigma =2.828

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.1729 | 0.8049 | 0.2847 |
| Ground Truth 2 | 0.1875 | 0.8578 | 0.3077 |
| Ground Truth 3 | 0.1878 | 0.2691 | 0.2212 |
| Ground Truth 4 | 0.2458 | 0.7001 | 0.3639 |
| Ground Truth 5 | 0.1735 | 0.8626 | 0.2888 |

| | | | |
|---|---|---|---|
| Mean | 0.1935 | 0.6989 | 0.2932 |

Table 6 F measure for LOG detector (Animal.raw) Size=7 sigma =1.414

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.2106 | 0.7198 | 0.3259 |
| Ground Truth 2 | 0.2306 | 0.6753 | 0.3438 |
| Ground Truth 3 | 0.1261 | 0.7079 | 0.2141 |
| Ground Truth 4 | 0.2621 | 0.8023 | 0.3952 |
| Ground Truth 5 | 0.1946 | 0.8410 | 0.3160 |
| Mean | 0.2048 | 0.7493 | 0.319 |

Table 7 F measure for LOG detector (Animal.raw) Size=7 sigma =2.828

We can find that the result of size = 7 is much better than the results from size =5. For House.raw, I only list the consequences of size = 7.

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.1975 | 0.5793 | 0.2945 |
| Ground Truth 2 | 0.2765 | 0.6337 | 0.3850 |
| Ground Truth 3 | 0.2542 | 0.6664 | 0.3680 |
| Ground Truth 4 | 0.2812 | 0.6484 | 0.3923 |
| Ground Truth 5 | 0.3722 | 0.7084 | 0.4880 |
| Mean | 0.2763 | 0.6472 | 0.3856 |

Table 7 F measure for LOG detector (House.raw) Size=7 sigma =1.414

| Ground Truth Image | Precision(P) | Recall(R) | F measure(F) |
|---|---|---|---|
| Ground Truth 1 | 0.2603 | 0.4509 | 0.3301 |
| Ground Truth 2 | 0.2958 | 0.6126 | 0.3990 |
| Ground Truth 3 | 0.3068 | 0.5540 | 0.3949 |
| Ground Truth 4 | 0.3122 | 0.5765 | 0.4051 |
| Ground Truth 5 | 0.3697 | 0.7128 | 0.4869 |
| Mean | 0.3090 | 0.5814 | 0.4032 |

Table 7 F measure for LOG detector (House.raw) Size=7 sigma = 2.828

The results justifies our intuition. From problem(b), we conclude that the LOG is much better than the Sobel based our visualization. In problem(c), we calculate the F measure for both Sobel and LOG. The F-measure of LOG whose kernel size is 7 and sigma is 2.828 is much bigger than the F-measure of Sobel detector in 2 images. However, the best edge detector is SE detector because its F-measure is greater than 0.6 in 2 images so that the performance of SE detector is much better than other 2 detectors.

The reason why the F-measure of House image is bigger is that there are much more texture in the image and the edge of each texture is much easier to detect. Also, the color of the animal is quite similar to the color of the trunk, which means it is more difficult to detect it perfectly.

About the relationship between P, R and F, because both P and R are in the range of (0,1) and F is calculated from R, we can draw the Figure of P,R and F:

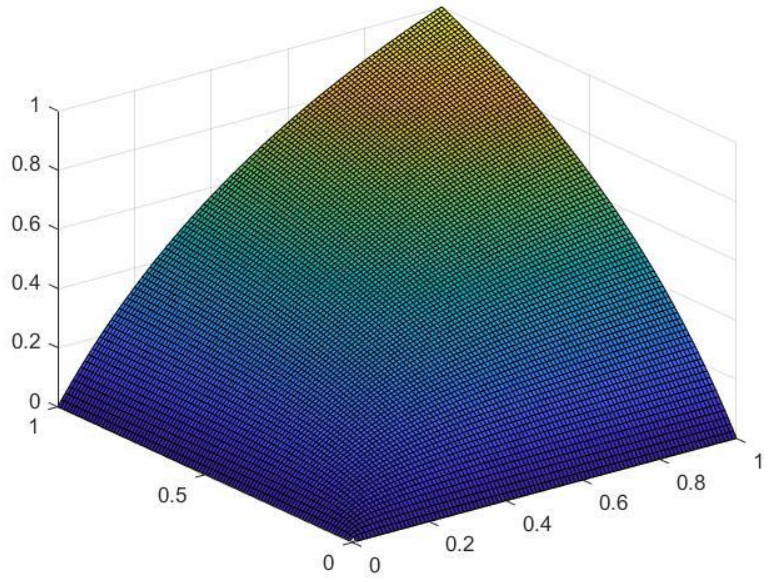



Figure 52 The relationship between P, R and F.

The only vertical axis is F measure. We can conclude from the picture even though P or R is big enough, if the other one is small, the value of F measure is still small. Therefore, it is impossible to get a high F measure if P is much bigger than R. The proof is in this picture:

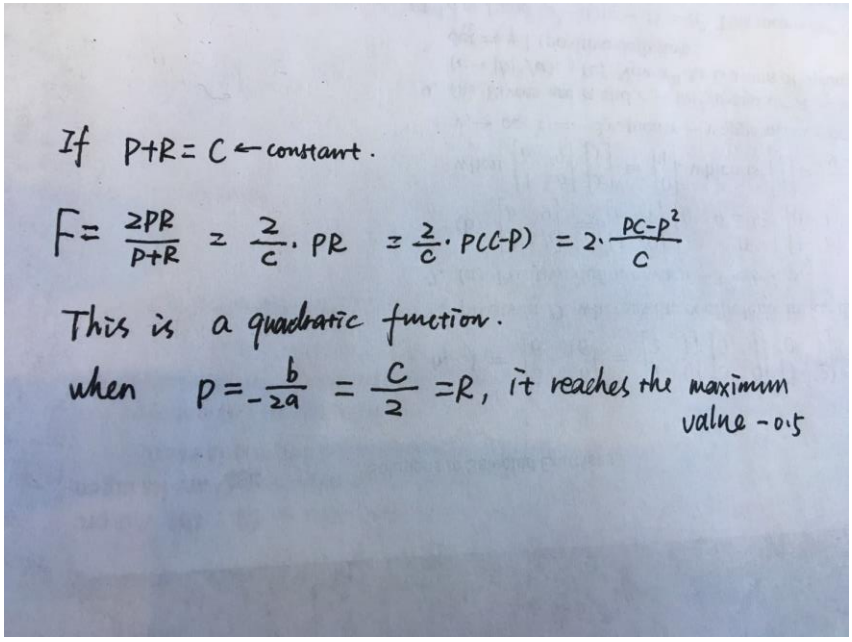



Figure 53 Proof.

# Problem3-Salient Point Descriptors and Image Matching

## a. Extraction and Description of Salient Points

### Abstract and motivation

In previous problems, we know that the edge information is quite important in the image processing. Here comes another important definition in contour detection and image matching and registration: Salient points. Salient points are the blob pixels or corner points. In this part, we can use openCV to implement some salient points detection methods such as SIFT and SURF.

**Task:** Use openCV to implement SIFT and SURF for extracting salient points.

## Approaches and Procedures

**Theoretical Approach:**

**SIFT:**

Step1: Build the LOG octave. In problem 2, we know that the LOG is the summation of second-order partial derivative with respect to x and y. According to the proof of Linderberg[3], we can use the difference of Gaussian kernel to approximate the LOG kernel, which will be convenient for the generation of LOG octave. Assume that the Input image is $I(x,y)$. G is the Gaussian kernel with sigma standard deviation. We denote the output blurred image $L(x,y,\sigma)$. It can be expressed as this formula:

$$L(x,y,\sigma) = G(x,y,\sigma) \otimes I(x,y)$$

For each floor in a octave, we can use Gaussian kernel with different parameter to convolve with the input image. Therefore, we can get several Gaussian-blurred images.

Step2: Use the difference of Gaussian-blurred image to approximate image filtered by LOG. This can be expressed as this formula:

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) \otimes I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma)$$

Now, we can get the subtraction of these images. Then we can do downsampling to this image in order to get more octaves. This can be described by a flow chart:

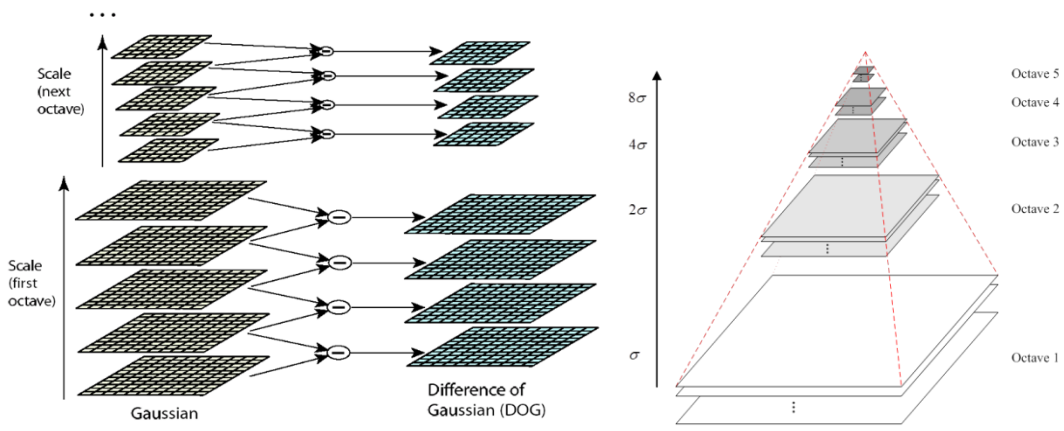



Figure54 Gaussian pyramid construction

Step3: Now, we got the Gaussian Octave. We have to find the salient points. In this step, we select the local maximum and minimum points as our target points. The way we choose it is compare the pixel value with its 26 neighbors like this figure:

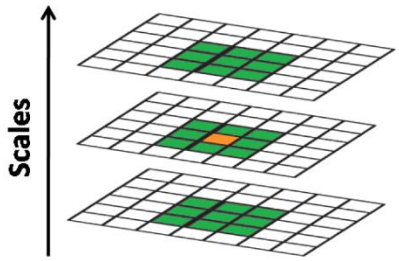



Figure55 Extrema selection

Step4: The points we get from step 3 contain lots of unstable edge points. We have to remove them. According to David[4], he used the Taylor Series expansion in DOG space:

$$D(X) = D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D^T}{\partial X^2} X$$

Then, we can calculate the low contrast points by thresholding. Then we use the 2*2 Hessian Matrix to justify the key points. Next, we can calculate the trace and determinant of this Hessian matrix. Alpha and Beta is the 2 eigenvalues of this Hessian Matrix.

$$\mathrm{H}=\begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix},\ Tr(\mathrm{H}) = D_{xx} + D_{xy} = \alpha + \beta,\quad Det(\mathrm{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

Step5: Find the ratio of the square of trace and determinant. If it is bigger than a threshold, delete it.

Step6: Assign orientation. For those remaining points, we can calculate its magnitude and angle by this formula:

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

$$\theta(x,y) = \tan^{-1}((L(x+1,y) - L(x-1,y)) / (L(x,y+1) - L(x,y-1)))$$

Then, divide the whole region into several sub-region. Calculate the histogram for each direction(total 8 directions, 45 degree*8) like this:

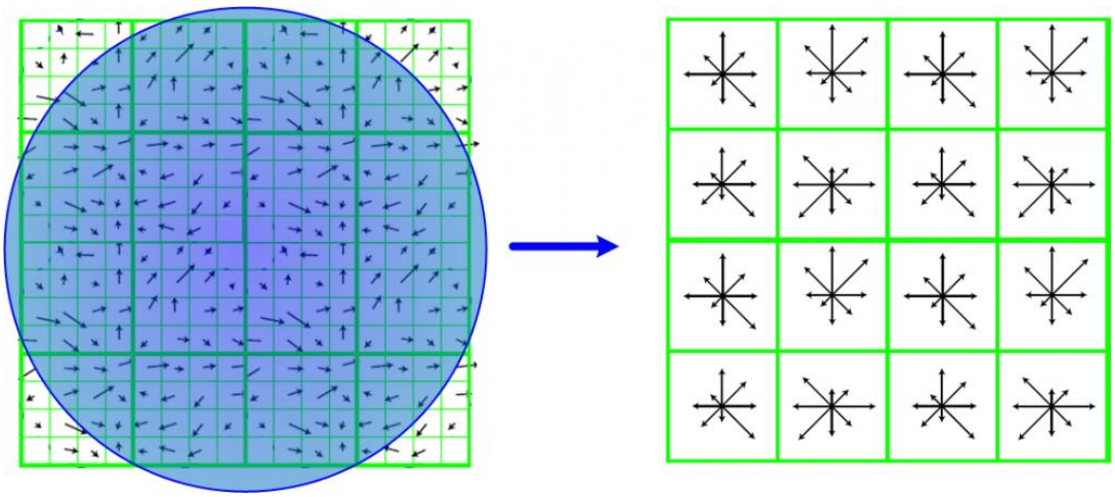


Figure56 Descriptor Orientation

Step7: Using the closet distance rule to decide the orientation from the Gaussian image. Histograms contain 8 bins, and descriptors contains array of 4 histograms around the key point and this leads to a SIFT feature vector with (4 × 4 × 8 =) 128 dimensions

**SURF:**

Because SIFT is computationally complex and very sophisticated, many scientist want to improve it. Therefore, Herbert Bay invented the SURF detector in 2006. It is much more efficient than SIFT and widely used in computer vision.

Step1: We have to calculate the integral image. Because the pixel value is discrete in spatial domain, we can use summation to approximate integration:

$$\sum I = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j)$$

Step2: After we get the integral image, we can compute the determinant of Hessian Matrix:

$$\mathrm{H}(X, \sigma) = \begin{pmatrix} L_{xx}(X, \sigma) & L_{xy}(X, \sigma) \\ L_{xy}(X, \sigma) & L_{yy}(X, \sigma) \end{pmatrix}$$

We use LoG (second order derivative filter) as the derivative filter in the Hessian matrix. SURF uses a 9*9 box filter to approximate Gaussian, which reduce the time of computation.

Step3: Octave construction. Instead of changing the size of image, SURF change the size of filter to create different scale. Then we can select those extrema as the candidates of salient points.

Step4: Localization. Unlike SIFT, the orientation of pixels in SURF is determined by the features of Harr wavelets. We have to do the Harr Transform. Calculate the horizontal and vertical Harr wavelet feature sums of all points in a 60-degree fan. After the fan rotates at an interval of 0.2 radians and again counts the harr wavelet eigenvalues in the region, the orientation of the fan with the largest value is taken as the feature.

Step5: Generating descriptor vectors. This is a 64-D feature vector including the derivative with respect to x and y and their magnitude. They are generated in a local window. The derivative is the Harr response in horizontal and vertical orientation.

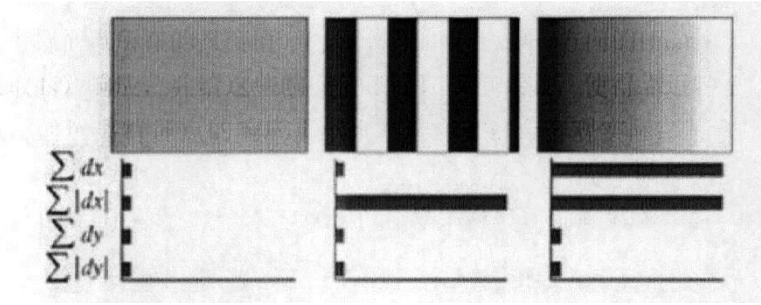



Figure57 SURF descriptor

**Implementing in C++(use openCV):**

**SIFT:**

Step1: Read the images of Truck and Bumblebee Car (jpg file) using imread function and store them in the Mat structure.

Step2: Construct the SIFT detector provided by the package xfeatures2d.

Step3: Call the detect method of the detector and store the keypoints in the Data structure in a vector composed by a openCV called Keypoint.

Step4: Using the drawpoints() function in OpenCV to draw the keypoints and show the results.

**SURF:**

Step1: Read the images of Truck and Bumblebee Car (jpg file) using imread function and store them in the Mat structure.

Step2: Construct the SURF detector provided by the package xfeatures2d.

Step3: Call the detect method of the detector and store the keypoints in the Data structure in a vector composed

by a openCV called Keypoint.

Step4: Using the drawpoints() function in OpenCV to draw the keypoints and show the results.

### Results(default parameter)

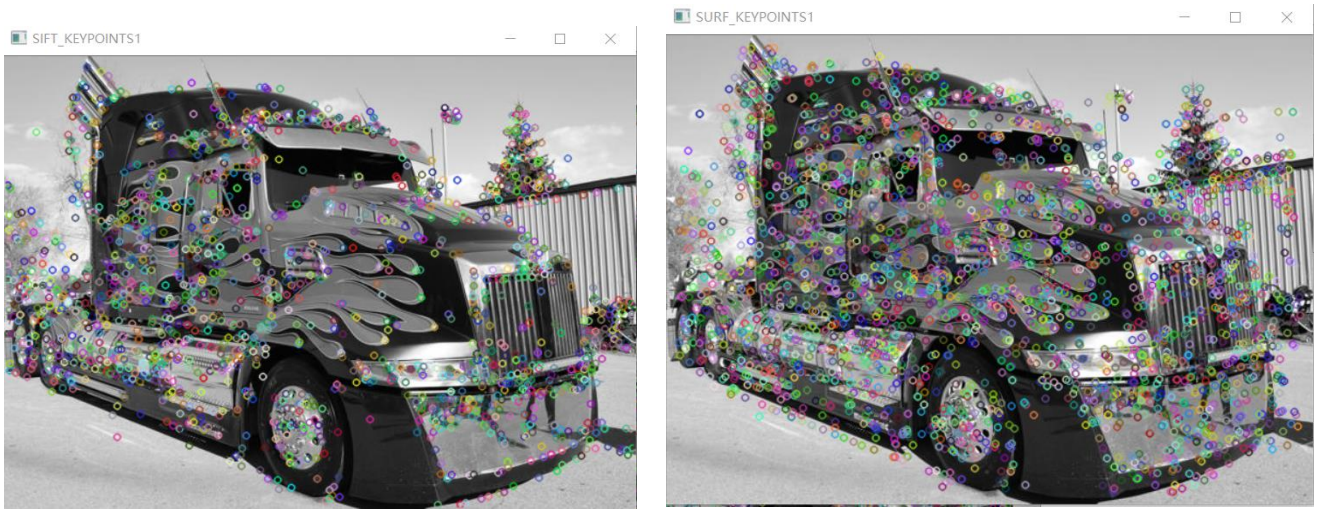



Figure58 SIFT and SURF features of truck.jpg

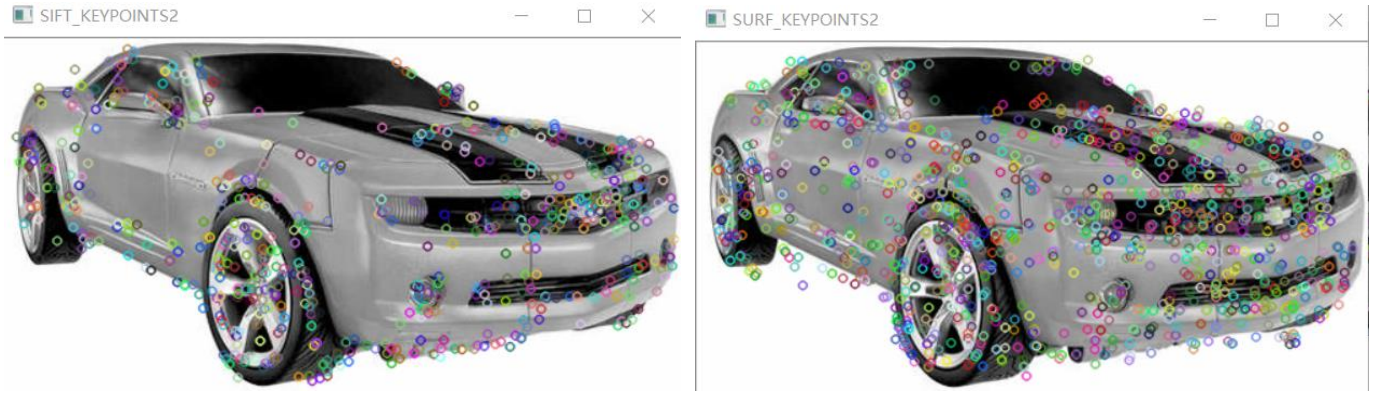



Figure59 SIFT and SURF features of bumblebee.jpg

### Discussion:

Actually, the speed of SURF is faster than SIFT(1.2679s VS 1.389s) because it simplifies the procedure of Gaussian octave and use Harr response instead of histogram to assign orientation, which can reduce the computation. However, because the number of datasets is small so that the difference is not obvious. In this situation, SIFT has more advantages. One of the biggest advantages of SIFT is that it is invariant to scaling, rotation, change in illumination, noise and small changes in the point of view. We can conclude from results that there are many unqualified points in the image from SURF. First, the descriptor of SIFT is 128-D. The other one is 64-D. Second, some low contrast points are removed in SIFT.

## b. Image Matching

### Approaches and Procedures

**Theoretical Approach:**

In problem 3(a), we get a lots of keypoints. Now, we have to calculate their corresponding descriptor in order to do image matching. For SIFT, we can get a group of 128D descriptor. For SURF, we can get a group of 64D descriptor. The descriptor contains the magnitude and orientation. We can use these 2 values to compare those descriptor and find the nearest one to match.

**SIFT:**

Step1: Read the images of Truck, Ferrari_1, Ferrari_2 and Bumblebee Car(jpg file) using imread function and store them in the Mat structure.

Step2: Construct the SIFT detector provided by the package xfeatures2d. Set the number of features parameters to 20.

Step3: Call the detect method of the detector and store the keypoints in the Data structure in a vector composed by a openCV called Keypoint.

Step4: Compute the descriptor by calling the method in SIFT detector->compute. Store those descriptors in Mat.

Step5: Using the Brute Force matcher provided by OpenCV to do the image matching.

Step6: Draw the matching pairs.

**SURF:**

Step1: Read the images of Truck, Ferrari_1, Ferrari_2 and Bumblebee Car(jpg file) using imread function and store them in the Mat structure.

Step2: Construct the SURF detector provided by the package xfeatures2d. Set the number of features parameters to 20.

Step3: Call the detect method of the detector and store the keypoints in the Data structure in a vector composed by a openCV called Keypoint.

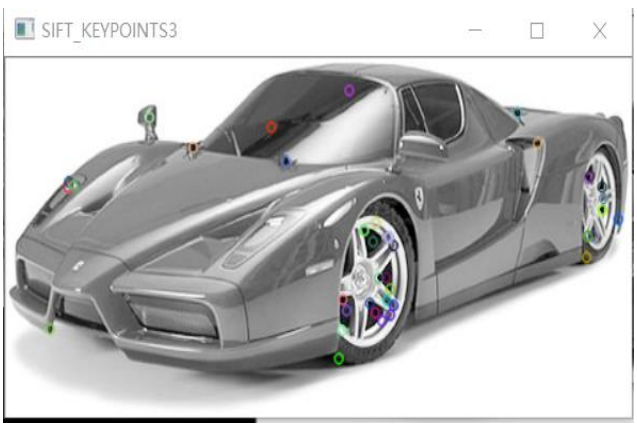
Step4: Compute the descriptor by calling the method in SURF detector->compute. Store those descriptors in Mat.

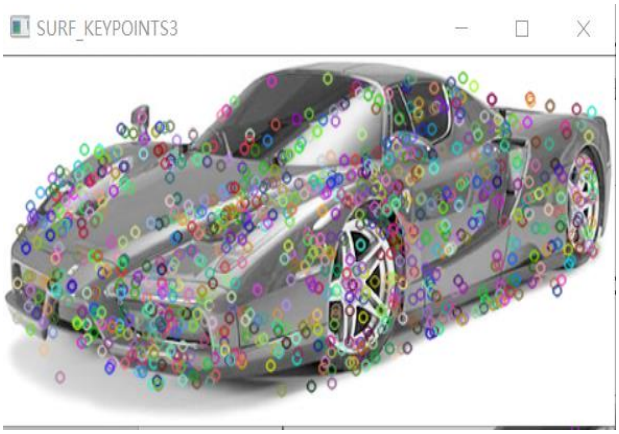
Step5: Using the Brute Force matcher provided by OpenCV to do the image matching.

Step6: Draw the matching pairs.

In order to reduce the matching pairs, I change the nFeature parameter of SIFT to 50. Hessian parameter in SURF to 50. So it will reduce many keypoints.

## Results(nFeature parameter of SIFT is 50. Hessian parameter in SURF is 50)



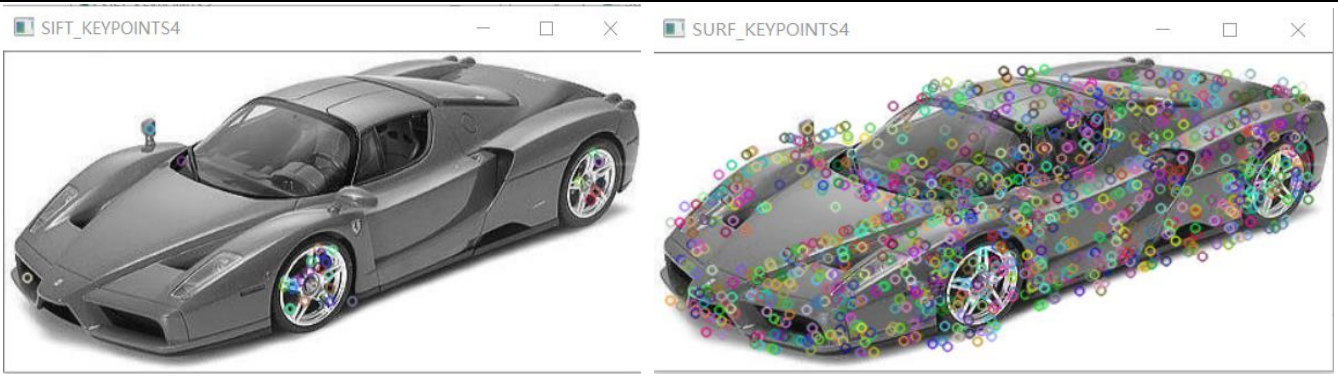



Figure 60 SIFT and SURF features for Ferrari_1 and Ferrari_2

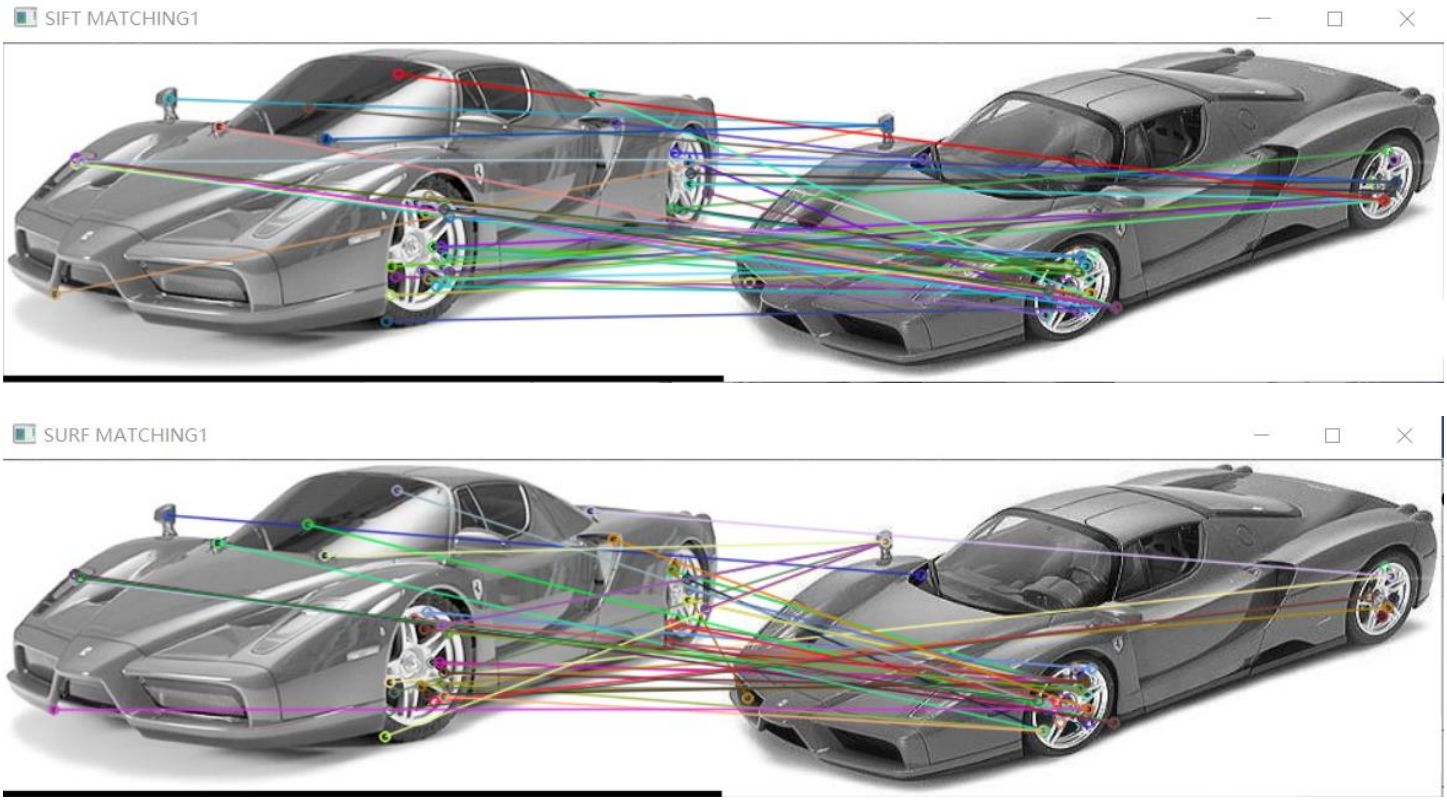
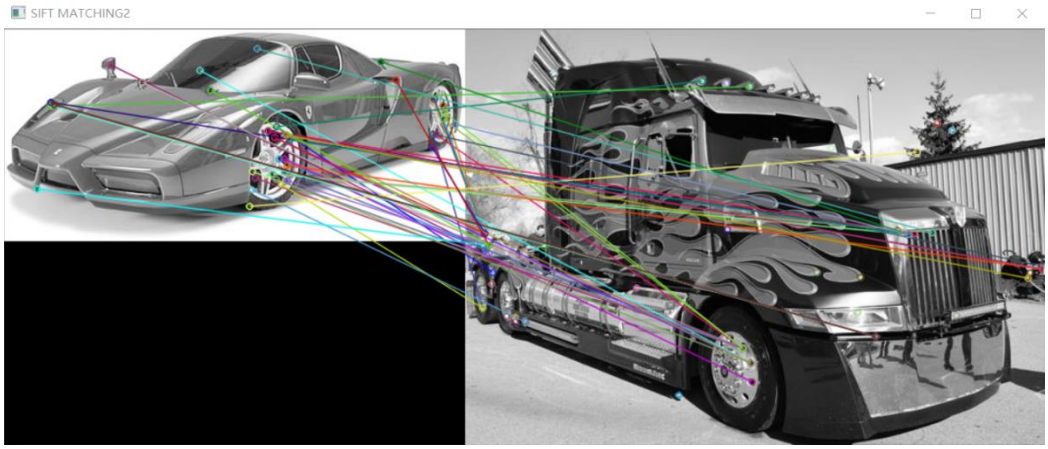



Figure 61 SIFT and SURF matching pairs for Ferrari_1 and Ferrari_2



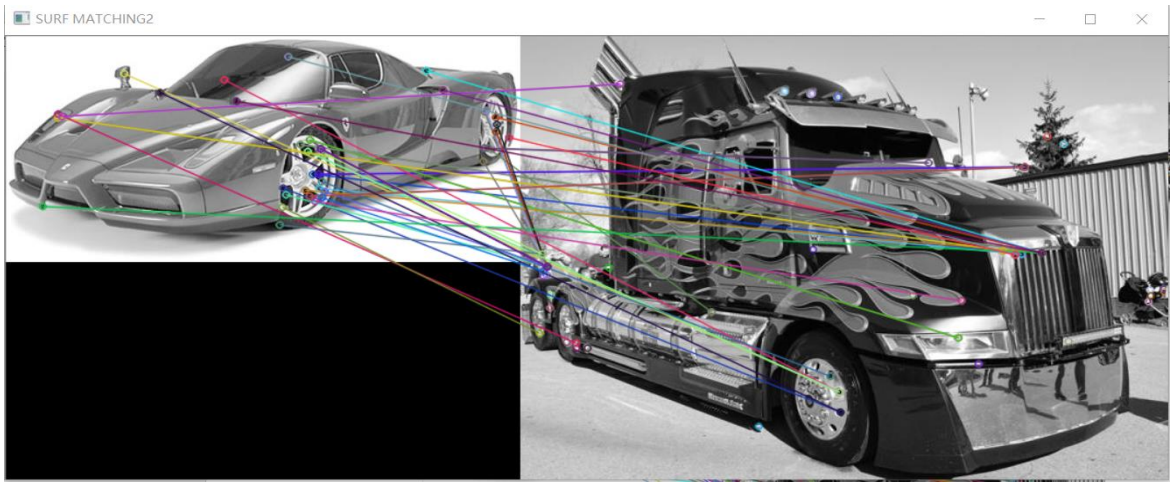



Figure 62 SIFT and SURF matching pairs for Ferrari_1 and Truck

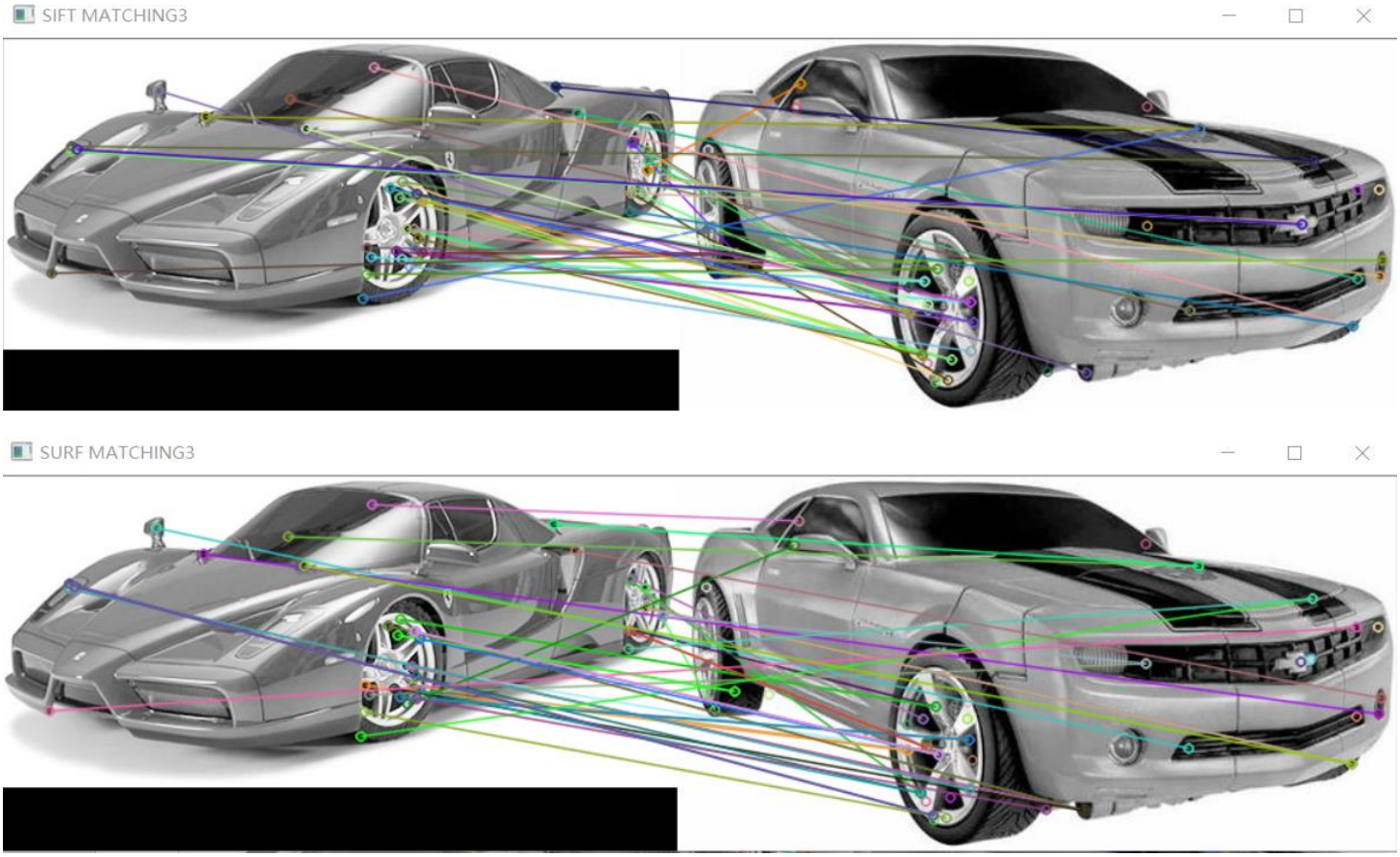



Figure 63 SIFT and SURF matching pairs for Ferrari_1 and Bumblebee

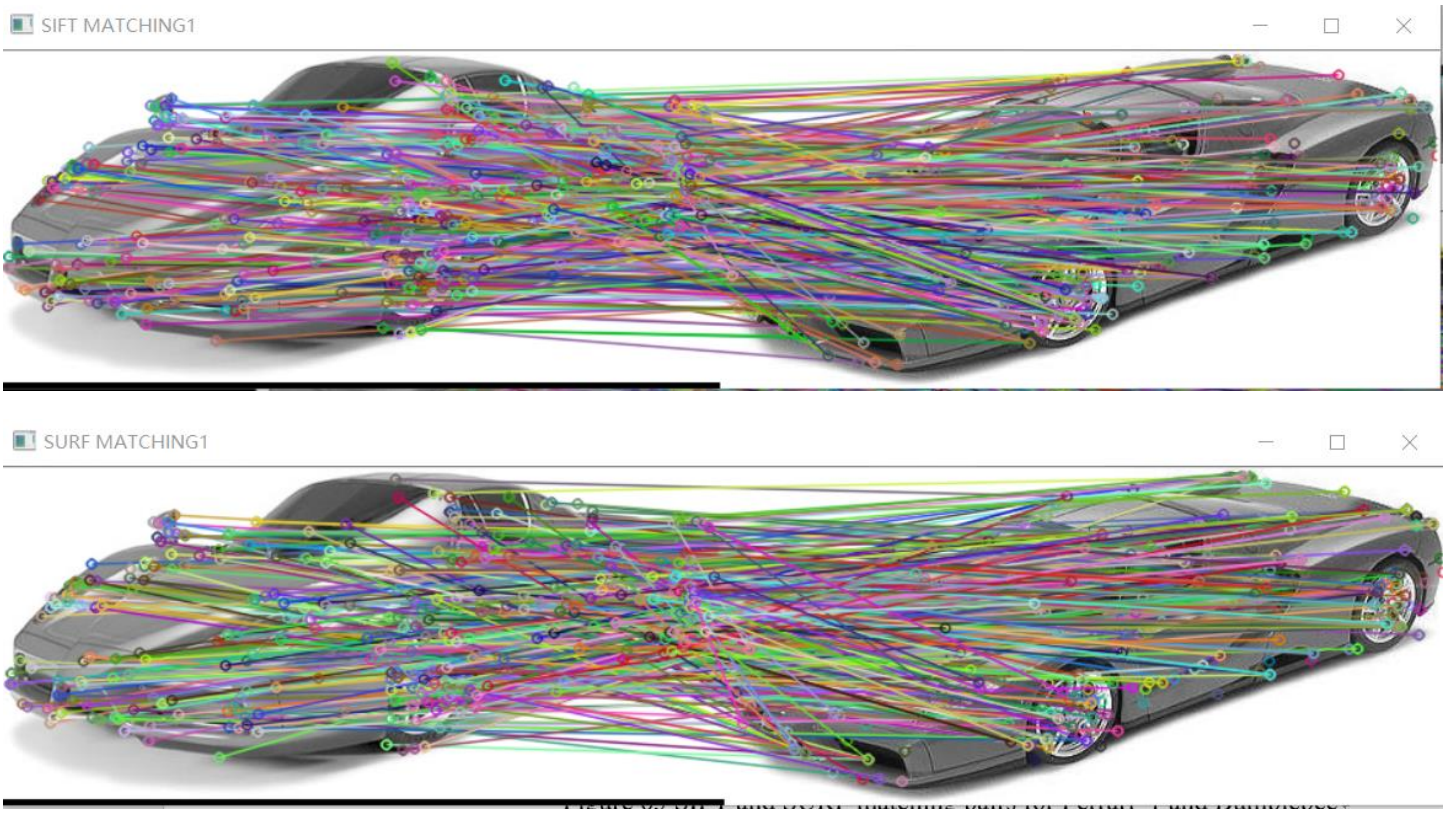



Figure 64 SIFT and SURF matching pairs for Ferrari_1 and Ferrari_2(default parameter)

### Discussion:

If we choose the first parameter of SIFT method that is for the number of features properly, we can get great results in matching. For example, we set it to 20, it will show the top 20 key points of the image. As we can see, the top 20 salient points are distributed in the wheel, window shield and the mirror, which can be correctly matching to those points in Ferrari(2). For different images, the matching is difficult. You can see the failure in the Figure 62. Last but not least, if we do not set the first parameter of SIFT small. Although we have the images from same class, the matching result will be very bad because there are many irrelevant points recorded in SURF.

## c. Bag of Words

### Approaches and Procedures

**Theoretical Approach:**

The descriptor can be used not only in image matching but also in image classification. If 2 images belong to a certain category, the difference between their descriptors is small. We can load many images' descriptors in a bag and make a "vocabulary". Therefore, we can do classification based on the descriptors in the vocabulary. Moreover, k-means is a good way to train the model.

**Implementing in C++:**

Step1: Read the images of Truck, Ferrari_1, Ferrari_2 and Bumblebee Car(jpg file) using imread function and store them in the Mat structure.

Step2: Construct the SIFT detector provided by the package xfeatures2d. Set the number of features parameters to 20.

Step3: Call the detect method of the detector and store the keypoints in the Data structure in a vector composed by a openCV called Keypoint.

Step4: Compute the descriptor by calling the method in SURF detector->compute. Store those descriptors in Mat.

Step5: Calling the variable provided by OpenCV: BOWKMeansTainer. Create a variable called BGWDS(8).

Step6: Calling the method in the BGWDS: add to put the descriptors of Ferarri_1, Truck and Bumblebee in the "vocabulary". Then, call the method cluster(). The store the "vocabulary" in a Mat.

Step7:Decision rule is the Euclidean distance. Get the classification result.

## Results and Discussion

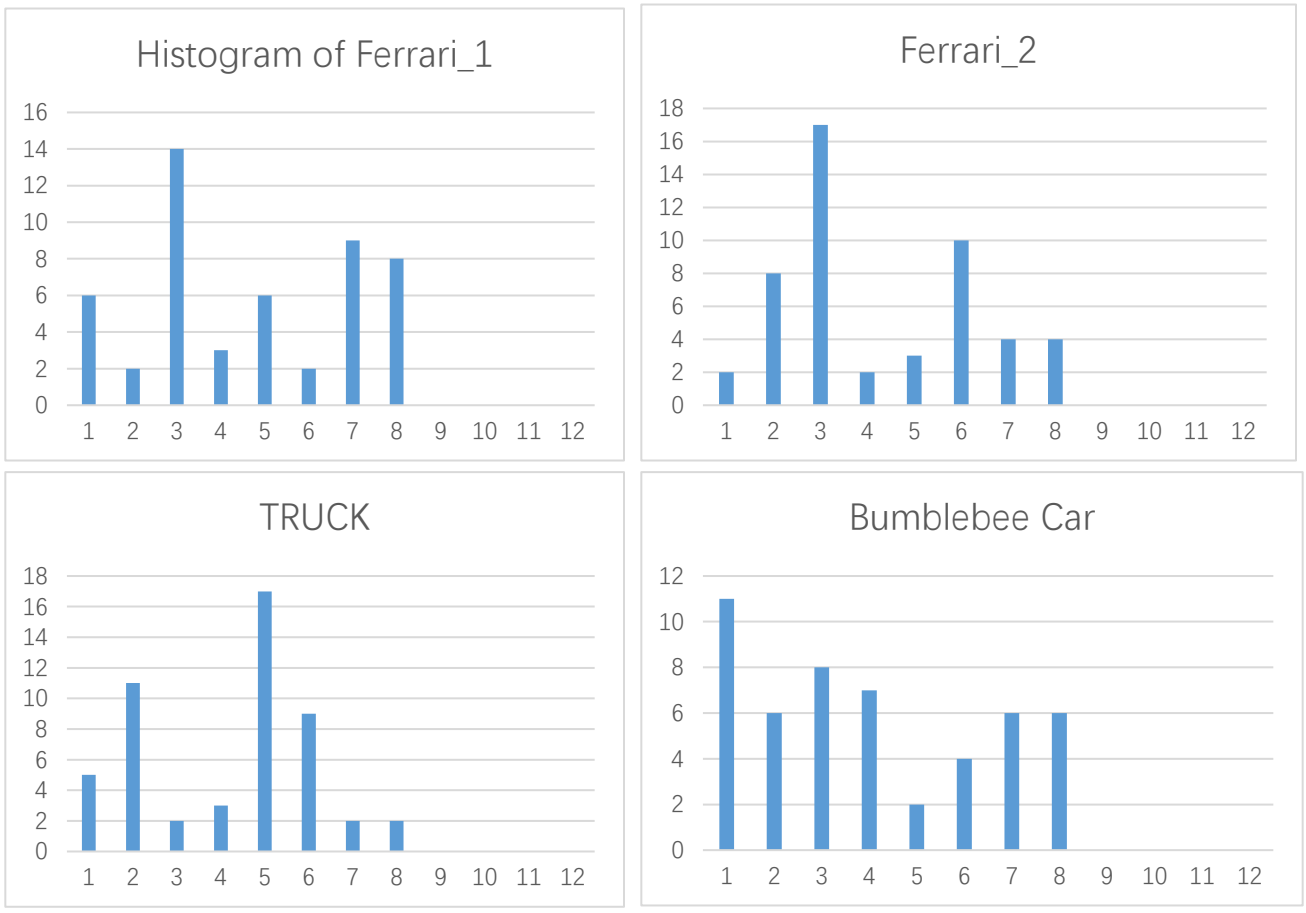



Figure 65 Histogram of 4 images.

It is obvious that the shape of Ferrari_2 is quite similar to the shape of Ferrari_1. We can conclude that the Bag of Word can correctly classify the image based on SIFT descriptor.

## Reference:

[1] Digital Image Processing, Fourth Edition, William K. Pratt, Wiley-Interscience Publication

[2] https://github.com/pdollar/edges

[3] Lindeberg T. Scale-space for discrete signals. IEEE Trans. on Pattern Analysis and Machine Intelligence. 207. 1980: 187-217.

[4] David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60(2), 91-110, 2004