EE569: Homework #3
Report

ISHAN MOHANTY
USC ID: 4461-3447-18
Email: imohanty@usc.edu

Problem 1: Texture Analysis and Segmentation

1(a) Texture Classification

I. Abstract and Motivation

Image Texture is qualitatively defined by the amount of its coarseness and hence, due to this property give the spatial arrangement of pixel intensity or color intensity in a selected region of an image. These textures can be natural or artificially created. The image texture helps in texture analysis, classification and segmentation in the field of image processing. There are two approaches to studying or analysis image textures namely, structured approach and the other statistical approach. The structure approach is more useful in studying artificial textures as it see image textures as primitive pixel elements repeated in patterns whereas statistical approach uses the idea of the quantitative measure of the spatial arrangement of the pixel color intensities in a selected area of the image. Classification is the process of assigning an object to a set of predefined categories. In texture classification we do the exact same thing in a two-stage process: learning phase and then the recognition phase. In the learning phase we understand features and build a model to extract feature content and after this we classify the different sample images into the classes defined earlier. In this section we are motivated to apply the texture classification algorithm to the 4 categories of images namely: rock, grass, weave and sand.

II. Approach and Procedures

The following approach accurately describes the procedure/algorithm used for texture classification:

1. Read the twelve texture images which are of categories: rock, grass, weave and sand.
2. Construct nine 5x5 Laws filters using the 3 laws filters: E5, S5 & W5.

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

3. Then Perform Feature Extraction:
   I. Normalize the Each of the 12 images so that you get a zero-mean image.
   II. Perform convolution using the nine 5x5 Laws filter and hence obtain 9 features per image.
   III. Perform feature averaging for each of the 9 laws filters to obtain 9 averaged features per image. Therefore, now we obtain a 12*9 feature matrix.

4. Now execute the K-means clustering algorithm which is describe below:
   I. In the K-means algorithm, has 3 steps namely: Initialization, Assignment and update.

   II. In the Initialization step we use the **K-means++ algorithm [1]** described below:
      - Choose one centroid randomly from among the 12 feature data points.
      - For each data point *x*, calculate the distance D(x) between *x* and the nearest centroid that has already been chosen.

- Select one new feature data point as the new centroid, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.
- Repeat Steps 2 and 3 until 4 centroids have been chosen.
- Now that the 4 initial centroids have been chosen, proceed to K-means clustering.

III. In the Assignment Step:
- Assign each feature data point to the Cluster group based on least Euclidean distance to centroids.

IV. In the Update Step:
- Calculate new means by averaging the feature data points in each cluster group and then update the new centroid values with the respective new means that were calculated.

III. Experimental Results

```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -std=c++11 TextureClassify.cpp TextureClassifyMain.cpp -lm
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./a.out
####################  K-means++ Intialization  ######################


centroid index 1: 6
centroid index 2: 10
centroid index 3: 12
centroid index 4: 3


#########################################################################


################## Texture Classification  ######################


 Cluster-Group ID:1

 Feature ID:

 1   4   6


 Cluster-Group ID:2

 Feature ID:

 7   8   10


 Cluster-Group ID:3

 Feature ID:

 2   12


 Cluster-Group ID:4

 Feature ID:

 3   5   9   11

#################################################################
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$
```

Figure 1: Texture Classification Results

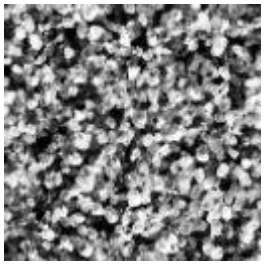


Image 1

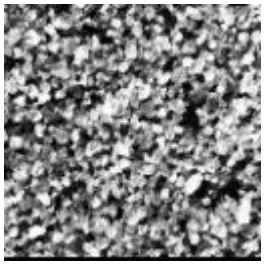


Image 4

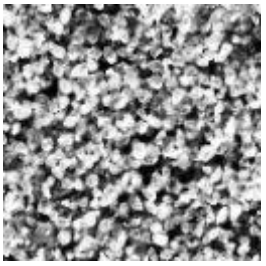


Image 6

Rock

Image 7

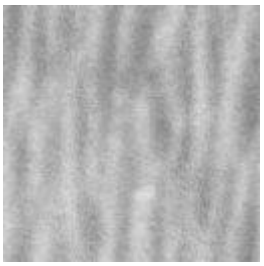


Image 8

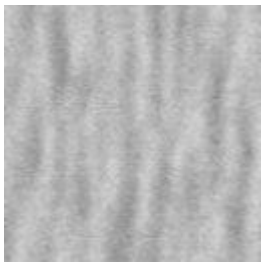


Image 10

Sand

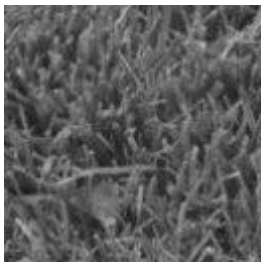


Image 2

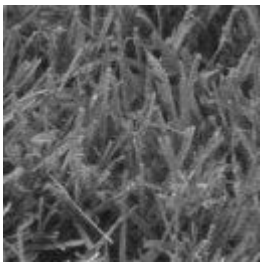


Image 12

Grass

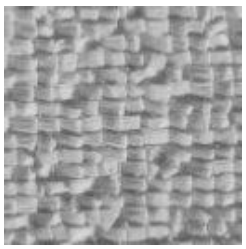


Image 3

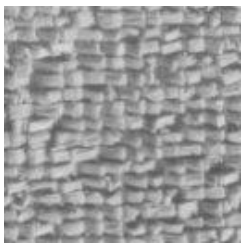


Image 5

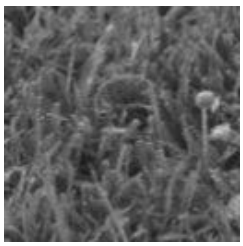


Image 9

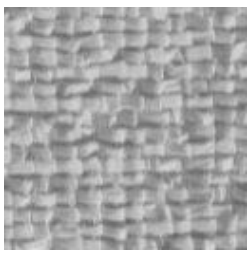


Image 11

Weave + Grass

Figure 2: Texture Images seen after classification

	1	2	3	4	5	6	7	8	9
1	3.6600e+04	9.6528e+03	7.4585e+03	1.8189e+04	1.5264e+04	1.8129e+04	8.5454e+03	1.4340e+04	8.0849e+03
2	9.6528e+03	2.5795e+03	2.0187e+03	4.8368e+03	4.0834e+03	4.7991e+03	2.2986e+03	3.8121e+03	2.1717e+03
3	7.4585e+03	2.0187e+03	1.6033e+03	3.7647e+03	3.2007e+03	3.7219e+03	1.8131e+03	2.9722e+03	1.7101e+03
4	1.8189e+04	4.8368e+03	3.7647e+03	9.0966e+03	7.6608e+03	9.0196e+03	4.2997e+03	7.1455e+03	4.0597e+03
5	1.5264e+04	4.0834e+03	3.2007e+03	7.6608e+03	6.4756e+03	7.5821e+03	3.6449e+03	6.0197e+03	3.4364e+03
6	1.8129e+04	4.7991e+03	3.7219e+03	9.0196e+03	7.5821e+03	9.0038e+03	4.2540e+03	7.1374e+03	4.0294e+03
7	8.5454e+03	2.2986e+03	1.8131e+03	4.2997e+03	3.6449e+03	4.2540e+03	2.0580e+03	3.3869e+03	1.9405e+03
8	1.4340e+04	3.8121e+03	2.9722e+03	7.1455e+03	6.0197e+03	7.1374e+03	3.3869e+03	5.6739e+03	3.2105e+03
9	8.0849e+03	2.1717e+03	1.7101e+03	4.0597e+03	3.4364e+03	4.0294e+03	1.9405e+03	3.2105e+03	1.8346e+03

Figure 3: Covariance Matrix of the 9 feature dimensions

IV. Discussion

From the results we see that after performing texture classification we get features: 1, 4 and 6 in one cluster group which belongs to the class of rock, we then get features: 7, 8 and 10 in one cluster group which belongs to the class of Sand, then we get features: 2 and 12 which belong to the class of grass images and finally we get the features: 3,5,9 and 11 which belong to the Last cluster group composed of images 3,5 & 11 from class weave and feature 9 that belongs to the class of grass images.

There is a misclassification of the feature corresponding to texture 9 of class grass. It should ideally be classified in Cluster-Group 3 mentioned above in the figure 1 under experimental results. This discrepancy is observed since the feature information content in texture 9 is identical or very similar to the information present in the feature content in cluster group of Weave. Hence, Texture image 9 cannot be classified properly

Now, let us look at figure 3 describing the covariance matrix of the 9 feature dimensions. The feature dimension that has the strong discriminant powers is either feature dimension 4 or feature dimension 6. These 2 feature dimensions are very close to having the strongest feature dimension. Let us analyse the figure3 data, for strong discriminant power the variance should be very large and the covariance between the feature and other features should be low. We can see that feature dimension 4 has the largest variance of **9096.613** followed by feature dimension 6 with variance **9003.848.** The covariance between 4th feature dimension and other dimensions is comparable to covariance between 6th dimension and other dimensions. Lower the covariance then lower is the similarity between the feature dimension content. **Hence, if we have to choose between feature dimension 4 and feature dimension 6, then we would consider feature dimension 4 for having the strongest discriminant power**. **Therefore, feature dimension 4 has the strongest discriminant power.**

The feature dimension with the weakest discriminant power has to be chosen between 3 and 9. Let us analyse this, dimension 3 has variance **1603.281** and dimension 9 has variance **1834.602,** hence here dimension 9 has an edge over dimension 3. Also if we look at the covariances among the 3rd dimension with other features and 9th dimension with other features, we see that they are almost equal to each other. But the covariances is lower in the case of dimension 3 than dimension 9. Overall, if we consider this situation we pick the 3rd feature dimension to have the weakest discriminant power.

1(b) Texture Segmentation

I. Abstract and Motivation

Texture segmentation is the process by which we segment an image into different regions based on image texture. There are basically two types of Texture segmentation namely, region based and boundary based. In the region based method we group image pixels based on texture properties whereas in the boundary based method we group pixels based on edges between pixels that come from different textures. In this section we are motivated to use the region based approach by using the techniques from the previous task of texture classification and with that information segment the image into 6 different regions with 6 different gray levels.

II. Approach and Procedures

The following approach is used to perform texture segmentation:

1. Firstly, subtract the global mean from the Image buffer to make it zero-mean and then apply all 3x3=9 Laws filters to this zero-mean buffer and get 9 gray–scale images. This step is known as Laws Feature Extraction. Below, the 3 Laws filters are mentioned:
2. $$E3 = [-1 \quad 0 \quad 1]$$ $$S3 = [-1 \quad 2 \quad -1]$$ $$L3 = [1 \quad 2 \quad 1]$$
3. Use a windowed approach to computer the energy measure for each pixel based on the results from Step 1. After this step, we will obtain 9-D energy feature vector for each pixel. This step is known as energy computation.
4. All kernels have a zero-mean except for $L3^{T}L3$. $L3^{T}\,L3$ is typically not a useful feature, and can be used for the normalization of all other features. This step is known as Normalization.
5. After this use the k–means algorithm to perform segmentation on the composite texture image. There are six textures in the image, we use six gray levels (0, 51, 102, 153, 204, 255) to denote six segmented regions in the output image.

III. Experimental Results:

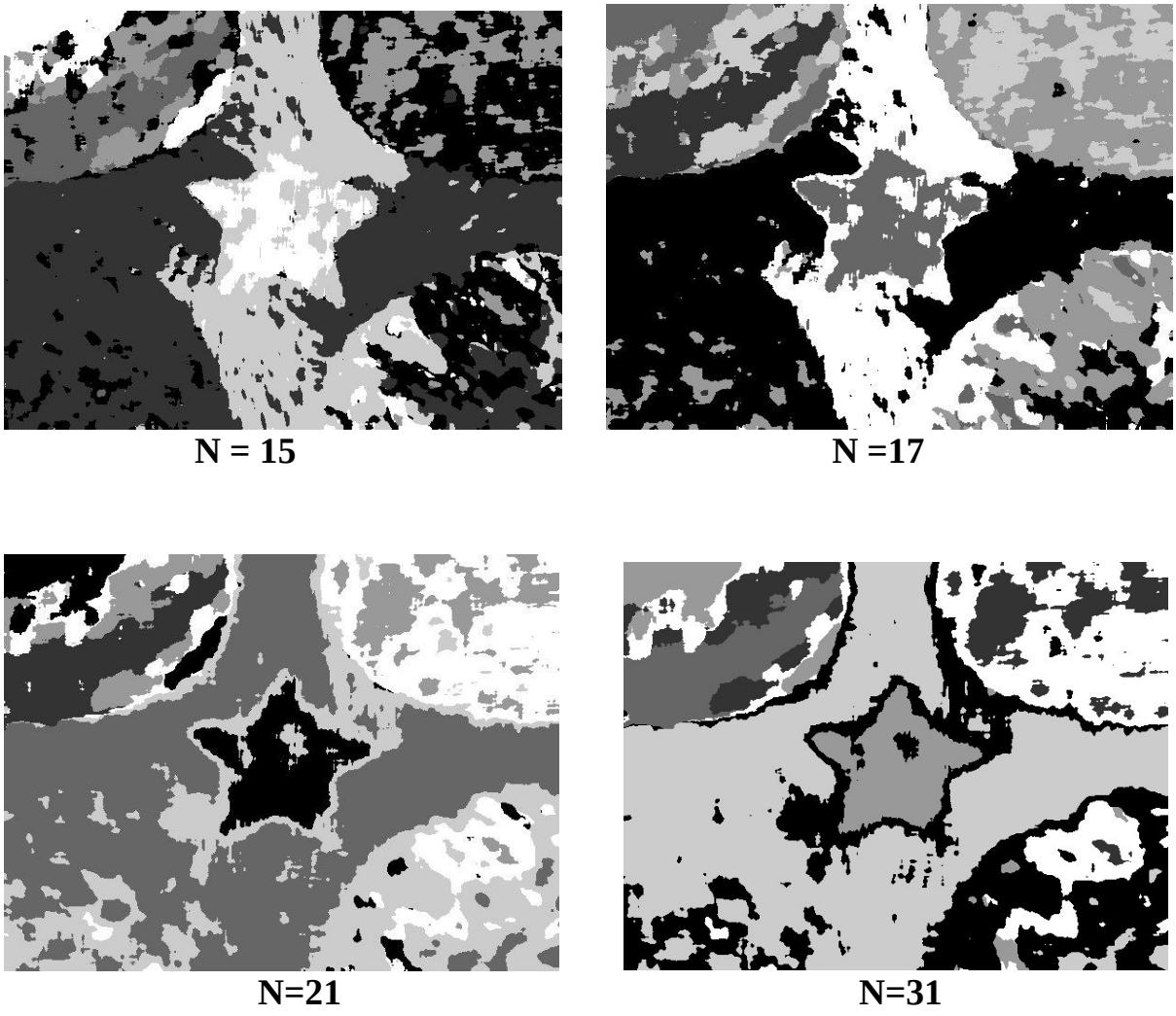


Figure 4: Texture Segmentation for different window sizes N

IV. Discussion

According to the algorithm mentioned we perform texture segmentation for different window sizes for energy computation to obtain different 9-D feature vectors for each image pixel. In this case the window sizes taken were N= 15, 17, 21 & 31. We observe that after normalizing the L3*L3 filter and subtracting the global mean from the image, we get much better results in terms of clarity in the texture segmented image. Also, if we were to compare the texture segmented results among different window sizes, we observe that by increasing the window sizes N, we get better segmentation in terms of boundary separation and uniform coloration of the segmented regions. Therefore, in our case we see great results for the images segmented with window size N=21 and window size N=25. To get better results than this we need to perform PCA for feature dimensionality reduction, this is a pre-processing step. Next, we perform post-processing using an appropriate algorithm.

1(c) Advanced

I. Abstract and Motivation

In this section we are motivated to perform PCA for feature reduction also known as Pre-processing. We use twenty-five 5x5 Laws filters and reduce the 25-dimensional feature vector to a smaller number using PCA and observe the results. After this we feed in the Image to the Post-processing algorithm to improve results. These two techniques help us get better results in certain scenarios. We attempt to study these improvements in texture segmentation to better our understanding.

II. Approach and Procedures

The following procedure was adopted to perform better processing of segmentation:

1. Construct the 25 different 5x5 Laws filters.

Name	Kernel
L5 (Level)	[1 4 6 4 1]
E5 (Edge)	[-1 -2 0 2 1]
S5 (Spot)	[-1 0 2 0 -1]
W5(Wave)	[-1 2 0 -2 1]
R5 (Ripple)	[1 -4 6 -4 1]

2. Feature extraction: In this step we find the global mean of the image and subtract it from the image so that we can take off the DC components of the image. After this we perform convolution with the 25 different 5x5 filters and compute the 25-D energy feature vectors and store the 25-D feature vectors for each pixel in the image.

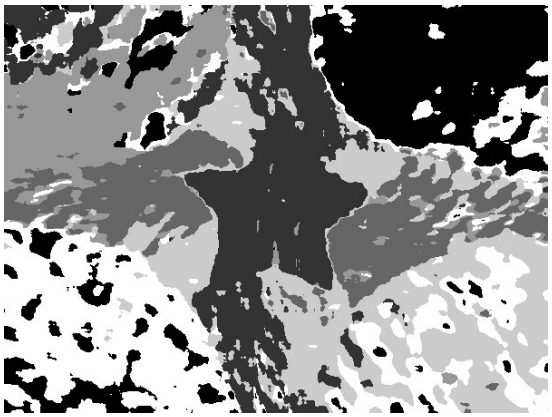
3. All kernels have a zero-mean except for $L5^{T}L5$. $L5^{T}L5$ is typically not a useful feature, and can be used for the normalization of all other features. This step is known as Normalization.

4. After obtaining the feature vectors which are normalized, we perform PCA to reduce the dimensions of the feature vector from D=25 to D=3,5,7,11,15,20,25 etc. This results of PCA leads to dimensionality reduction.

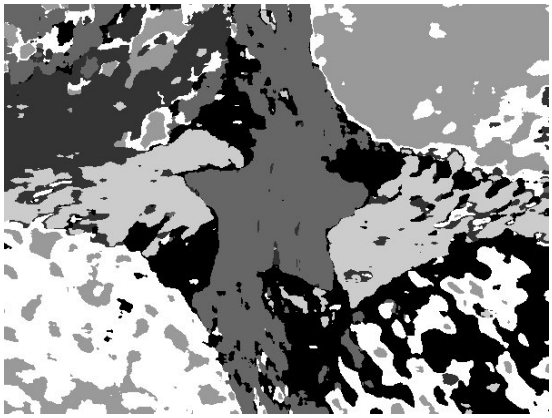
5. After reducing the 25-D feature vector to different-sizes, we feed this feature vector to our K-means clustering algorithm to perform segmentation on the composite texture image. There are six textures in the image, we use six gray-levels (0, 51, 102, 153, 204, 255) to denote six segmented regions in the output image.

6. **Post processing**: In the Post Processing step we use the image obtained from step 5 and use "**Most-occurring element in window filter**" approach where in for a given window size N, we find the most occurring pixel intensity in the window and fill the centre pixel with that pixel intensity. This ensures that holes and other breaks in the image are filled by the most-occurring pixel intensity in the image and hence, we see that we get a smoothened image where there are less holes and distortions. Therefore, we see a nice uniform image which is segmented in a better fashion.

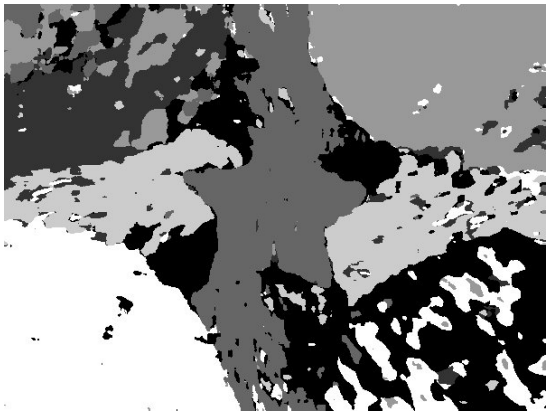
III. Experimental Results



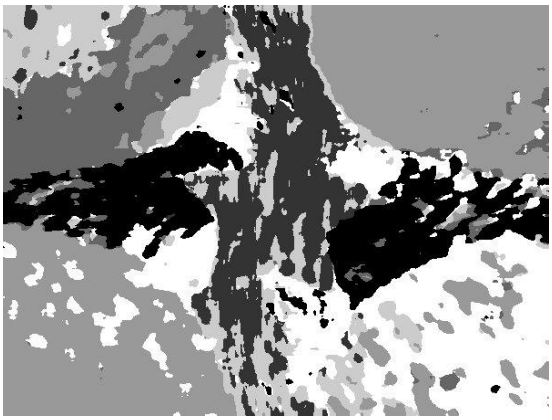
PCA, D =3 , WINDOW = 15



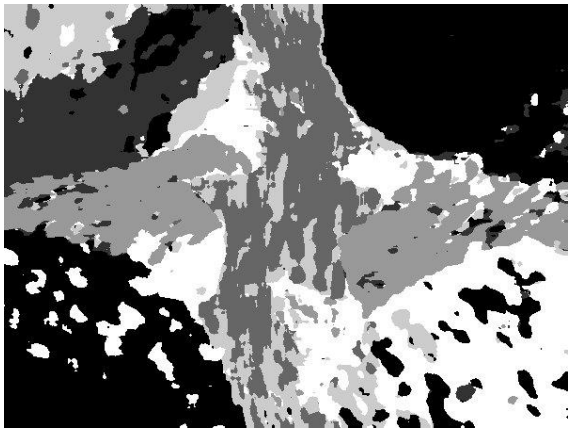
PCA , D =3 , WINDOW = 31



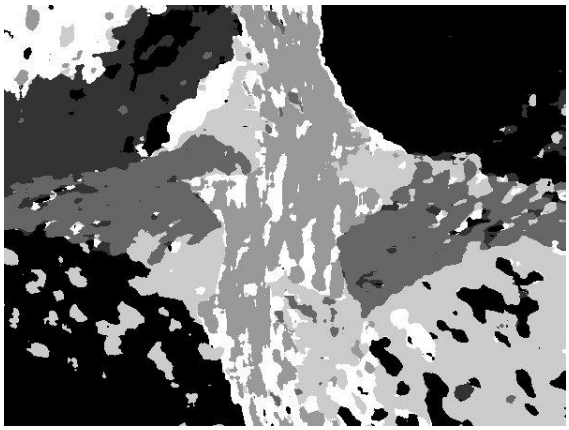
PCA, D =7 , WINDOW = 15



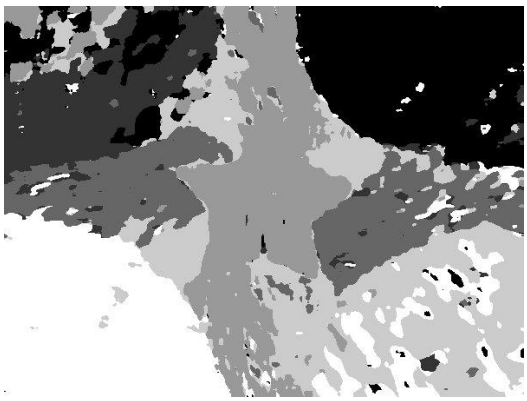
PCA , D =7 , WINDOW = 31



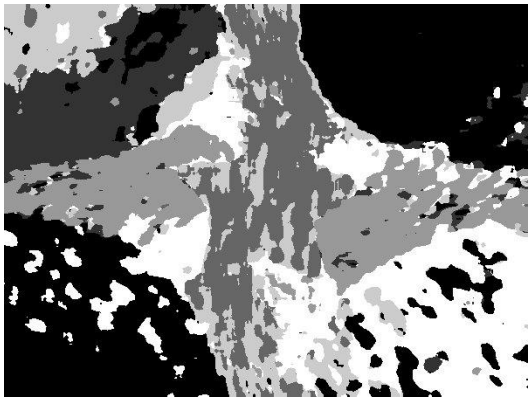
PCA, D =15 , WINDOW = 15



PCA , D =15 , WINDOW = 31

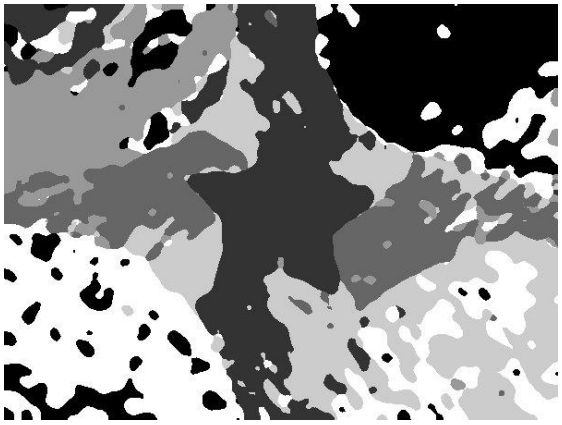


PCA, D =20 , WINDOW = 15

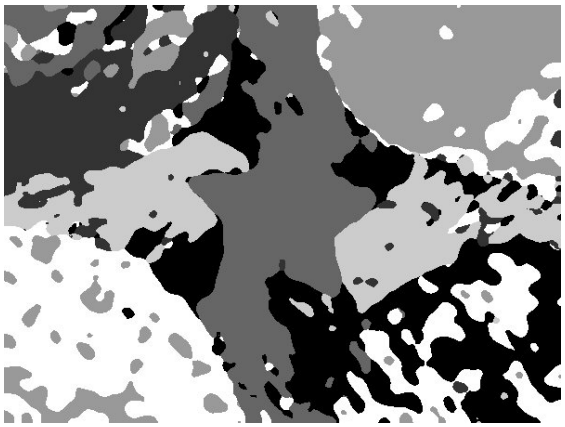


PCA, D =20 , WINDOW = 31

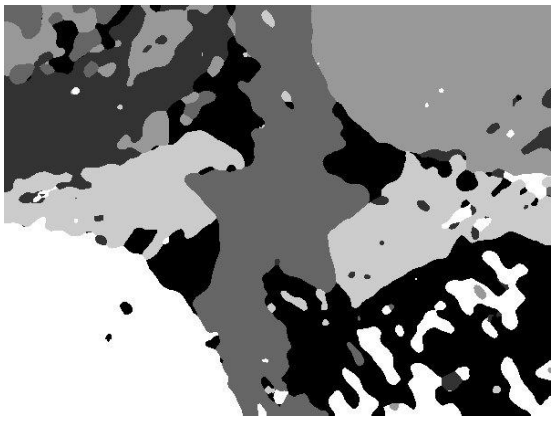
Figure 5: Perform PCA with different reduced Dimensions D & Segment for Different Window Sizes N



PCA +POST, D =3, WIN= 15

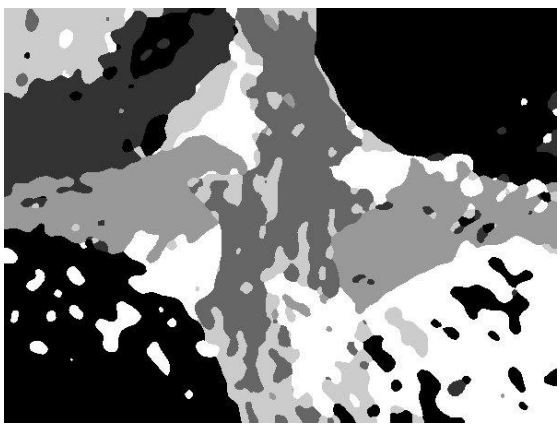


PCA +POST , D =3 , WIN= 31

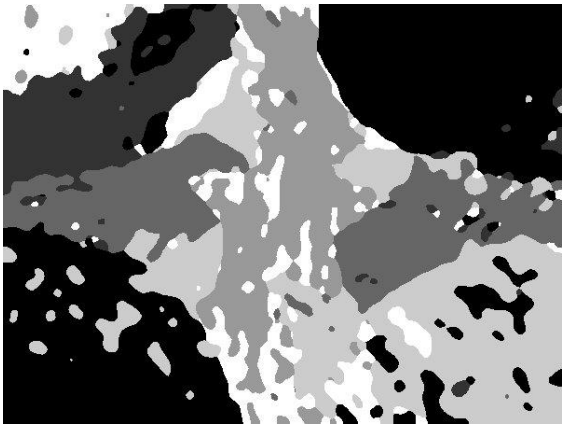


PCA +POST, D =7 , WIN= 15

PCA +POST , D =7 , WIN= 31

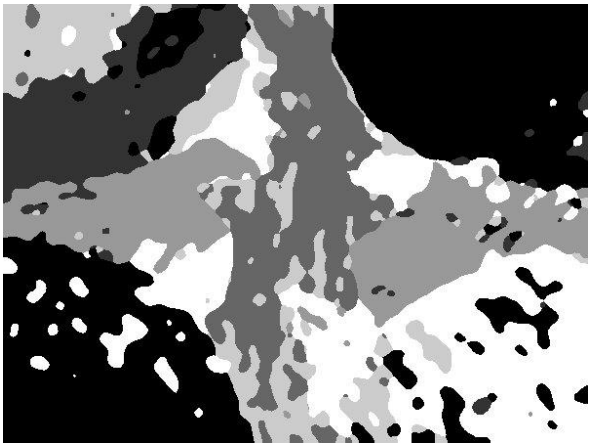


PCA +POST, D =15 , WIN= 15



PCA +POST , D =15 , WIN= 31

PCA +POST, D =20, WIN= 15



PCA +POST, D =20 , WIN= 31

Figure 6: PCA with different reduced Dimensions D & Segmentation for Different Window sizes N plus Post-Processing with Most occurring element filter.

IV. Discussion

In this section we perform PCA to reduce dimensions for getting better results. This is done due to the curse of dimensionality, which poses a problem in classifying data properly. More dimensions results in overfitting and hence, poor classification. To avoid this, we reduce dimensions of the 25-D feature vectors and try segmenting the image in order to get better results. We perform PCA to get different Dimensions for the 25-D feature vector like D=3,7,15 & 20 with two different window sizes namely, N=15 & 31. The results for pre-processing feature vectors with PCA is shown in figure 5.

We observe that PCA pre-processing gives us better results and hence, proves that reducing dimensions helps in texture segmentation. The only problem is if have very less dimensions, we cannot get enough texture information in the feature vector and if we have too many dimensions, there is a problem of overfitting due to curse of dimensionality. Therefore, to get optimum results we should have dimensions somewhere between too few and too many. We

can see good results particularly with D=3 & 7. The figure 5, shows PCA without post-processing.

After Performing PCA, we need to perform post-processing to get the best results. For post-processing, we use the "**Most-occurring element in window filter**" approach where in for a given window size N, we find the most occurring pixel intensity in the window and fill the centre pixel with that pixel intensity. This ensures that holes and other breaks in the image are filled by the most-occurring pixel intensity in the image and hence, we see that we get a smoothened image where there are less holes and distortions. Therefore, we see a nice uniform image which is segmented in a better fashion.

We use different kernels for convolution for post-processing and the results are shown as per figure 6. We see there is a visual difference in the images between figures 5 and figures 6. The breaks and holes in the images in figure 5 are filled and made smoother. We see that for D=3,7 and 20 after post-processing they get the best results in terms of segmentation and uniform coloration.

Problem 2: Edge Detection

a) Basic Edge Detector

I. Abstract and Motivation

In this section we attempt to study two edge detectors namely, Sobel detector and zero crossing detector. The Sobel detector uses two 3x3 convolution kernel which when applied performs a 2-D spatial gradient measurement on an image and highlights the high spatial frequency regions corresponding to the edges in the image. It is a first derivative edge detector.It is used to detect edges in the horizonal and vertical direction using two 3x3 convolution kernels. The zero-crossing detector uses the Laplacian of Gaussian operator to detect edges by considering second order derivatives. The Laplacian of Gaussian or LoG operator is rotationally invariant or isotropic and after application of the Laplacian of Gaussian operator, the zero crossings in the image represent the edges. We are motivated to learn about the performance and edge detecting capabilities of the two detectors.

II. Approach and Procedures

The following approach best describes the Sobel edge detection algorithm:

1. Convert the RGB image to gray scale.
2. Apply the gaussian kernel to denoise the noisy image but do not apply the gaussian kernel to the original image.
3. Apply the sobel operator Gx and Gy to calculate the horizontal gradient and vertical gradient across the image and find the magnitude of the gradient value G by taking the square root of the sum of the squares of Gx and Gy.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

4. Store this gradient value G in the buffer and iterate through to get the maximum gradient value of G. Let's call this max G.
5. Using the max G value, we find the threshold value required to get the edge map.
6. Using this threshold-value we make a calculated decision. If the gradient buffer image pixel intensity is greater than the threshold, we assign the edge pixel intensity else we output the background pixel intensity.
7. Store the output sobel edge map into an appropriate file location.

The following approach best describes the Zero-crossing detection algorithm:

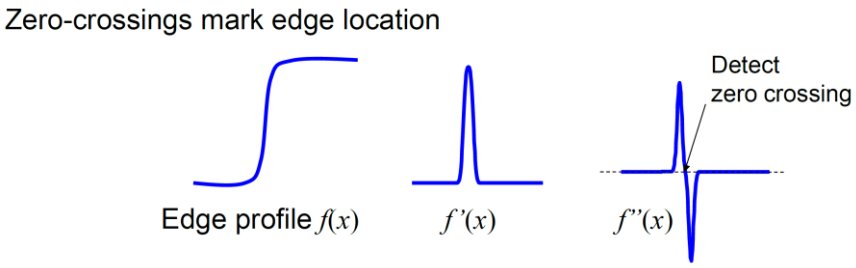
1. Convert the RGB image to gray-scale format.
2. Apply the gaussian filter to generate a blur in the image.
3. Apply the Laplacian of Gaussian filter and store the convolved values into a buffer. Find the minimum and maximum values in this buffer and scale the LoG buffer from 0-255 by performing normalization using the min-max values.

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & [-4] & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

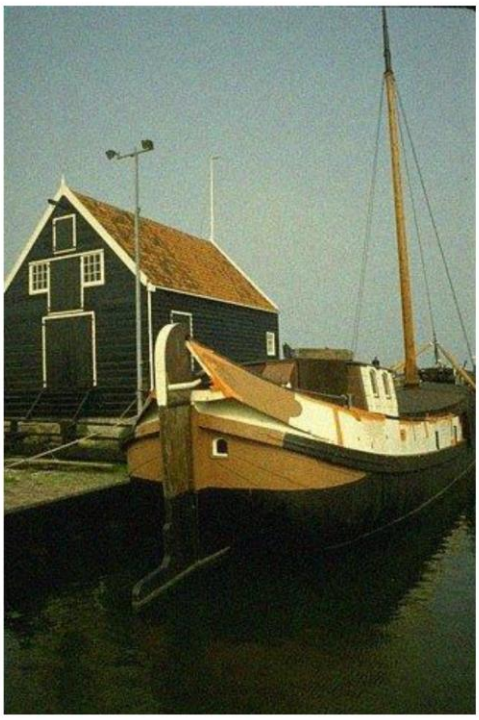
4. Output the normalized buffer and display in the results.

5. Compute the histogram and calculate the knee values of the LoG buffer image. The first knee value is calculated by taking the cumulative histogram at a point where it is 7.5% of the area under the curve and by multiplying it by -1, we get the second knee value.
6. Using these Knee values, we threshold the LoG buffer. The method we use is if the pixel intensity is between the two knee values, we assign 0 to a ternary map buffer and assign 128 gray value to another gray-level buffer which has 3 gray levels. If the pixel intensity of the LoG buffer is less than the first knee value we assign -1 to the ternary map and 64 gray-level to the gray-level buffer. If the pixel intensity is greater than the second knee threshold then assign 1 to the ternary map buffer and assign 192 to the 3 gray level buffer.
7. Using the ternary map buffer we threshold to get the best LoG edge map. We see if there is a zero crossing we assign the edge pixel intensity value else we assign the pixel intensity as that of the background.

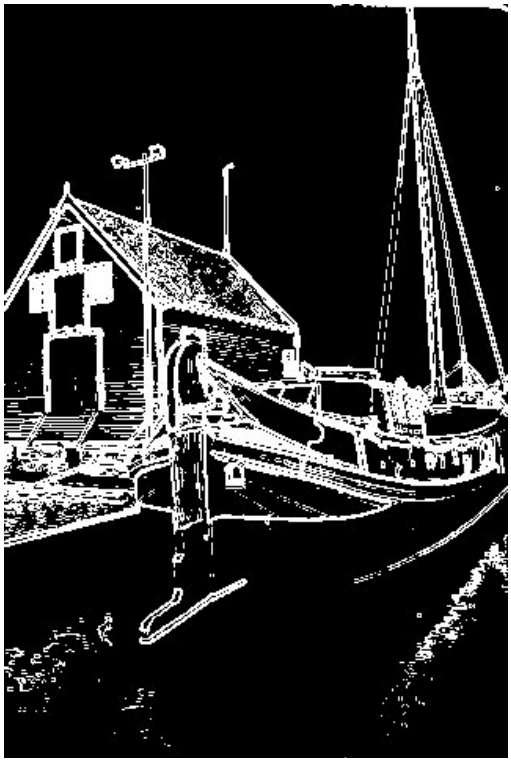


III. Experimental Results

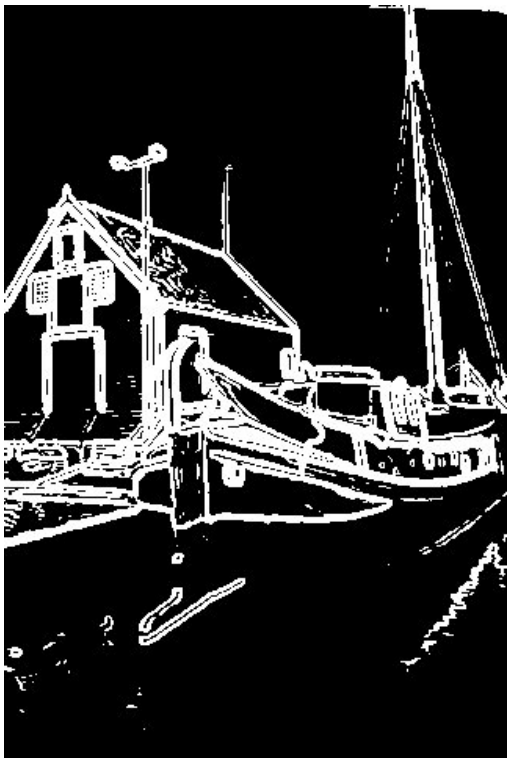
Original Boat Image



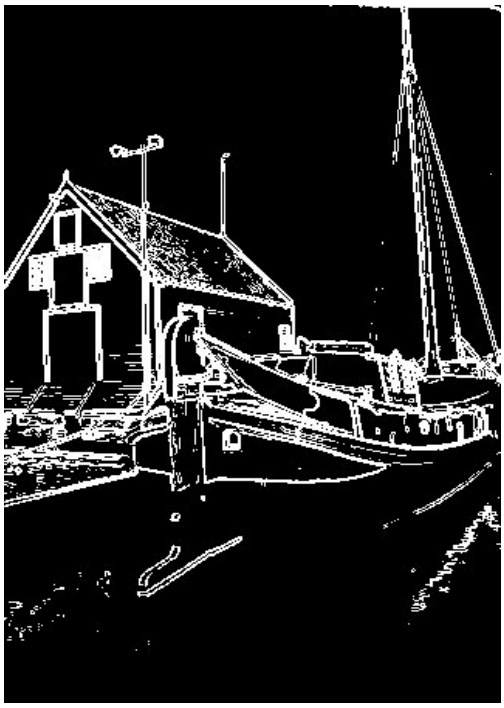
Noisy Boat Image



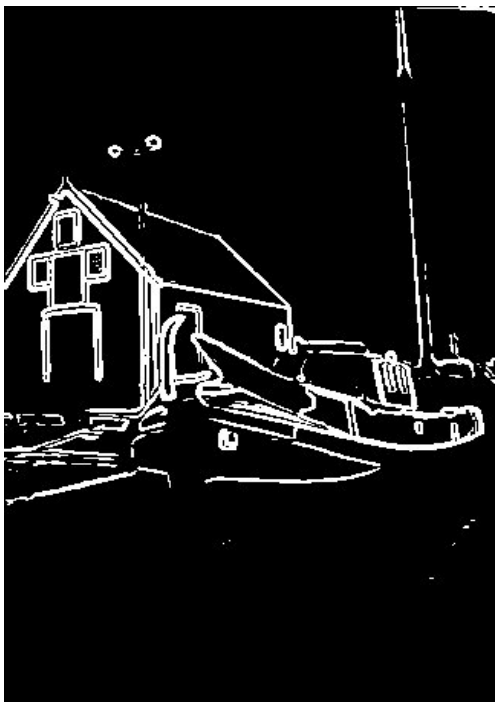
T = 77.5



T = 47.4



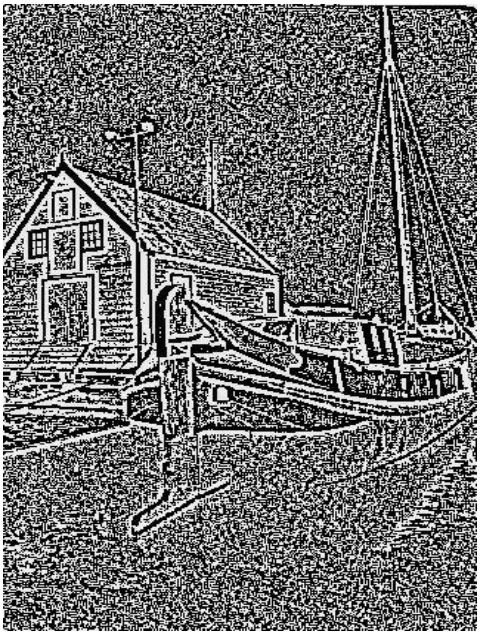
T=108.5



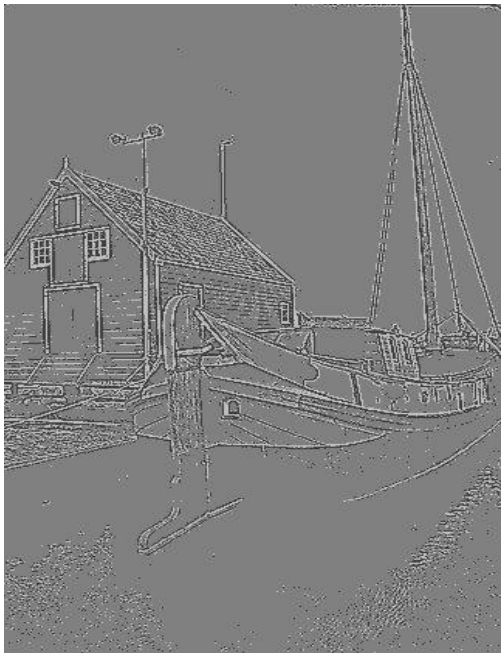
T= 118.5

Figure 7: Sobel Edge Maps with different threshold values T for Original Image on left side and Noisy Image on Right side.

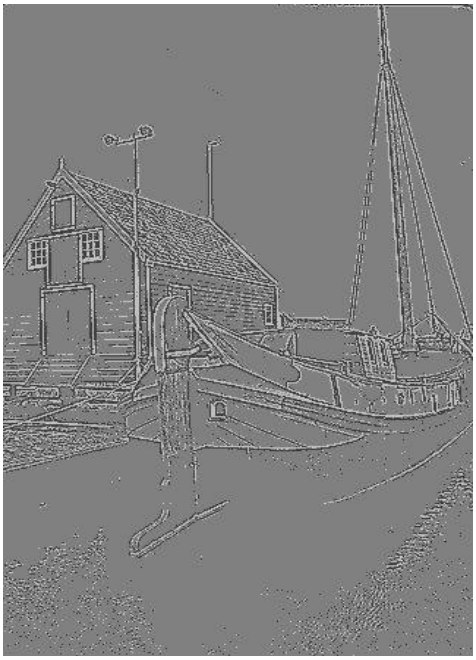
Normalized Image Original



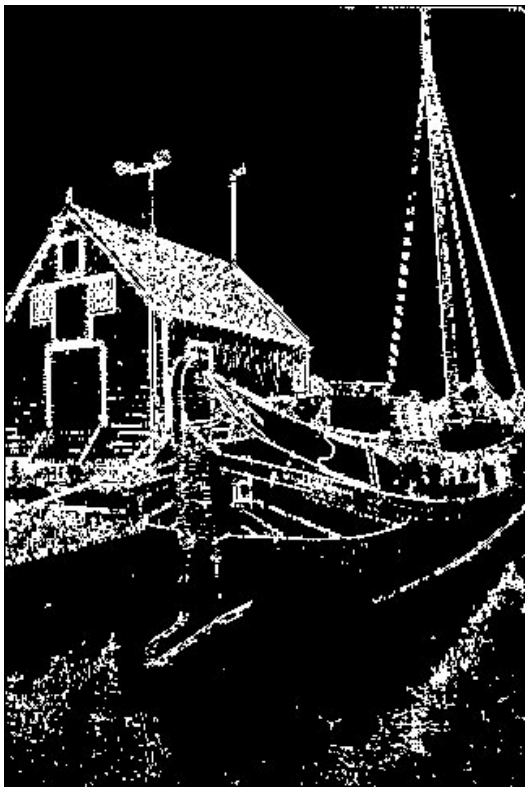
Normalized Image Noisy



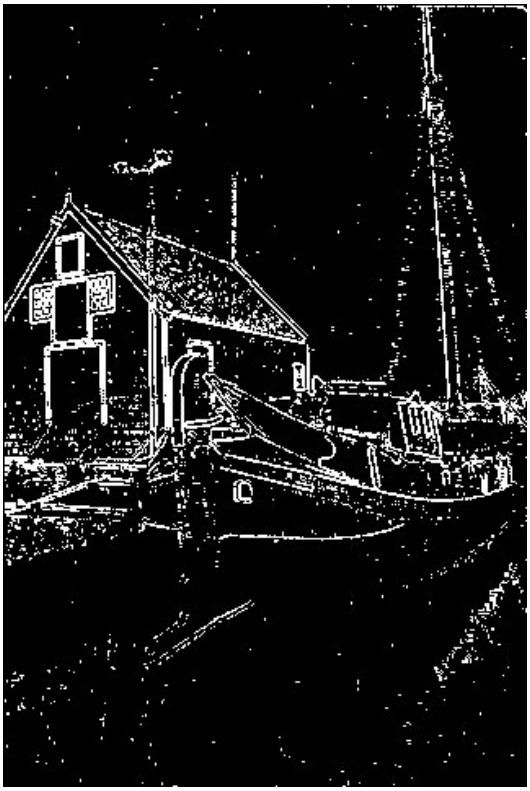
Ternary Map Original



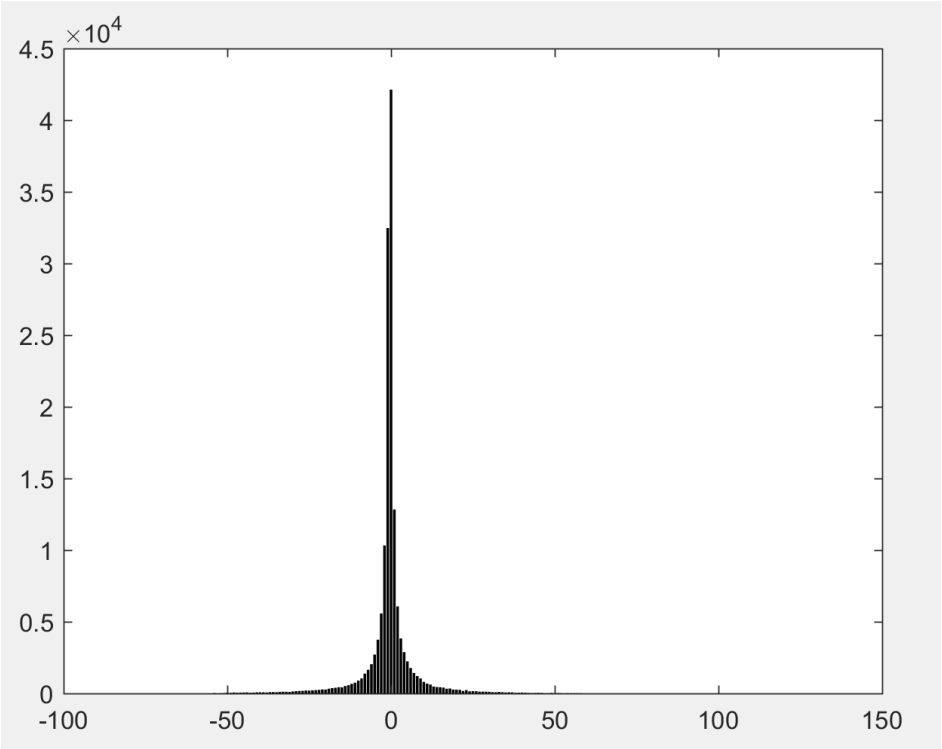
Ternary Map Noisy



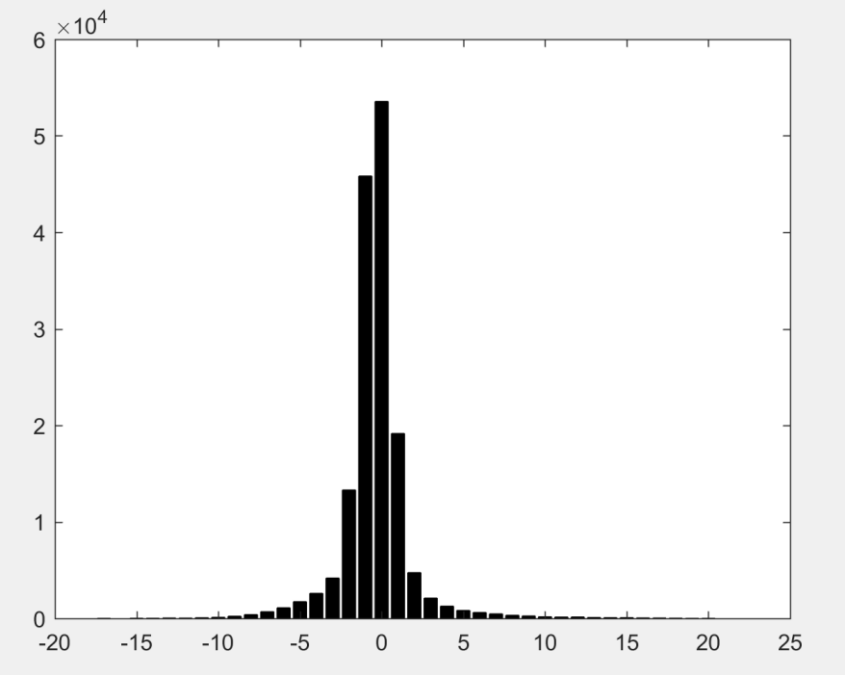
Edge Map Original



Edge Map Noisy



Original--- Histogram: Knee value1 = -6, Knee value 2 = +6



Noisy---Histogram: Knee value 1: -1 & knee value 2: +1

Figure 8: Zero-Crossing detector results : Normalized Images, Ternary Maps, Edge Maps and Corresponding Histograms for the images after convolution with the LoG Kernel.

IV. Discussion

The results of Sobel detector and the zero-crossing detector are shown above. The Sobel detector is a first derivative gradient based detector whereas the zero-crossing detector which uses the Laplacian of Gaussian is a second derivative detector. If we see the time-complexity and efficiency of the algorithm the Sobel detection algorithm is much faster than that of the zero-crossing algorithm.

If we compare figure (x) with figure(y), we see that the Zero-crossing detector is much more sensitive to noise whereas the Sobel detector is comparatively less sensitive to noise. Hence, we see better images in the results of Sobel detector compared to the Zero-crossing detector.

We can also observe by looking at the images, due to the zero-crossing algorithm of detecting edges the Zero-crossing detector detects strong as well as weak edges in a much better way than the Sobel detector. The Sobel detector detects more false edges than the Zero-crossing detector using the LoG operator.

Hence, we can say that in scenarios where we need fast computation and more immunity to noise, we can use the Sobel detector and in places of better edge map requirement for denoised images we can use the Zero-crossing detector with more confidence.

b) Structured Edge
c) Performance Evaluation

I. Abstract and Motivation

Edge detection is an important process by which we segment objects and also use of it for object detection. In this section we learn more about the structured edge detection algorithm that utilizes the structure information present in the local patches of the image and helps in very efficient real-time edge detection using random forest classifiers. We also try to measure the performance of this edge detection technique with the help of parameters like Precision, Recall and F measure.

II. Approach and Procedures

Approach for Structured Edge Detection Algorithm:

The SE detector Algorithm is discussed below with an explanation and a flow chart:

Structured edge detector is a highly efficient edge detector that is used for constructing edges in an image. It takes a local patch around the image pixel and deduces the validity of the pixel to be an edge point or not.

The Edge patches are converted into something called "**Sketch Tokens**" using the random forest classifier. Sketch tokens are a collection of local image edge structures like T junctions, Y junctions, Straight lines, Parallel lines etc.
The function of structured edge learning is geared towards addressing difficulties in training a mapping function where the input or the output space is complex.

In the beginning, the random forest model is trained by considering a local patch from the input image and ground-truth images. By doing so, the model forms a structured output which is subjected to classification as an edge or not an edge depending on various combinations used in the model.

After training the random forest model by certain deterministic number of samples, it is then given for testing. During the testing cycle, the test image is given and the random forest model quickly generates the segmented output.

The random forest classifier uses an ensemble model of approach. The ensemble model of the random forest runs on the divide and conquer algorithm that speeds up performance. The main jist of this approach is to feed weak learners to give rise to a stronger learner. The primary element of the random forest is a decision tree which are weak learners and our collected together to form a Random forest.

The decision tree classifies the input that belongs to one space to an output that belongs to another space. In a decision tree, the input is assigned as the root node and the input then recursively traverses down the tree till it reaches a leaf node. Every node in the decision tree

contains a binary split function with a number of parameters. This split function then makes a decision on whether the testing sample should move to the right child node direction or the left child node direction. The split function is determined by training a bunch of trees independently to gain the parameters for the function. These parameters are chosen in such a fashion so as to maximize the gain in information for the split function such that the model works efficiently and correctly.

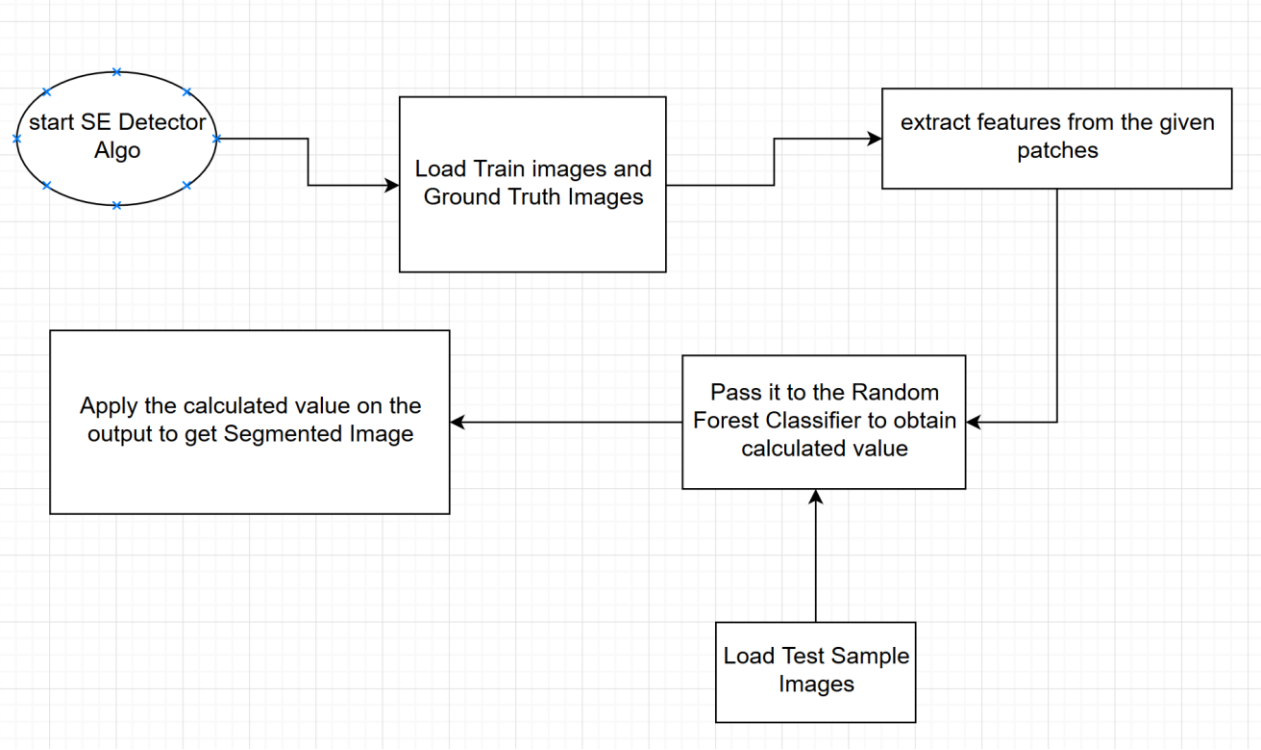


Figure 9: Flowchart for the Description of SE detection algorithm

Approach for Performance Evaluation:

To evaluate the performance of an edge map, we need to identify the error. All pixels in an edge map belong to one of the following four classes:

(1) True positive: Edge pixels in the edge map coincide with edge pixels in the ground truth. These are edge pixels the algorithm successfully identifies.

(2) True negative: Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth. These are non-edge pixels the algorithm successfully identifies.

(3) False positive: Edge pixels in the edge map correspond to the non-edge pixels in the ground truth. These are fake edge pixels the algorithm wrongly identifies.

(4) False negative: Non-edge pixels in the edge map correspond to the true edge pixels in the ground truth. These are edge pixels the algorithm misses.

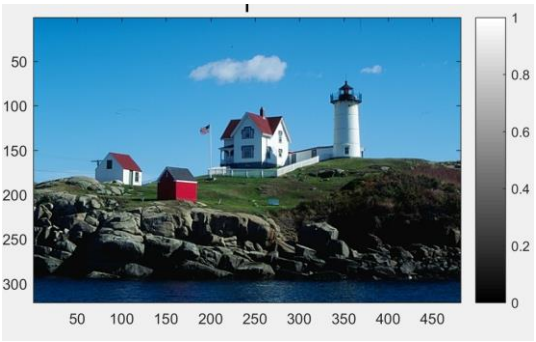
Clearly, pixels in (1) and (2) are correct ones while those in (3) and (4) are error pixels of two different types to be evaluated. The performance of an edge detection algorithm can be measured using the F measure, which is a function of the precision and the recall.

$$\text{Precision}: P = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}$$
$$\text{Recall}: R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}} \qquad (1)$$

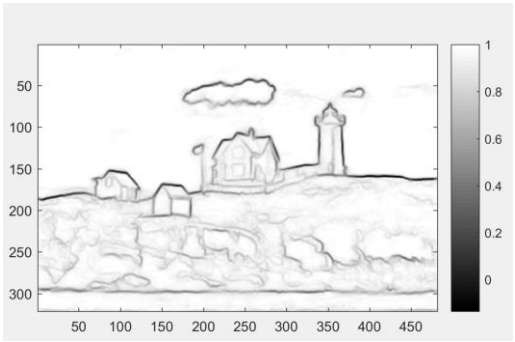
$$F = 2*P*R/(P+R)$$

One can make the precision higher by decreasing the threshold in deriving the binary edge map. However, this will result in a lower recall. Generally, we need to consider both precision and recall at the same time and a metric called the F measure is developed for this purpose. A higher F measure implies a better edge detector.

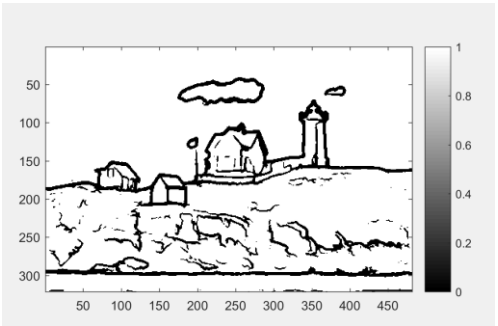
III. Experimental Results



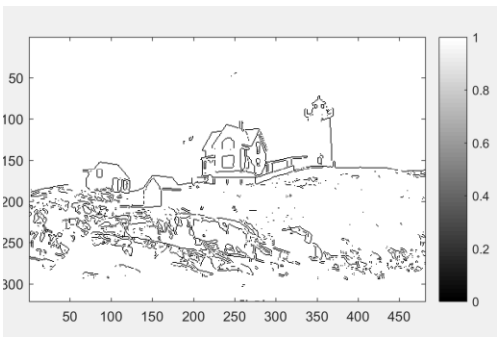
House.raw



Probability edge map for SE



Binary Edge Map for SE

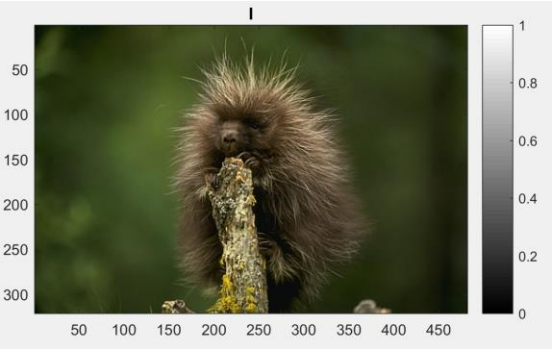


Edge Map for Sobel

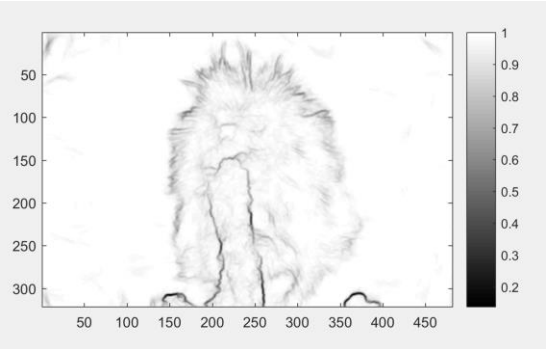
Parameter	GT1	GT2	GT3	GT4	GT5
P	12.40	14.87	7.37	14.52	13.12
R	14.62	14.57	12.75	14.85	14.59
F measure	13.42	14.72	9.34	14.68	13.82

F measure mean: 13.196 (All in terms of percentage)

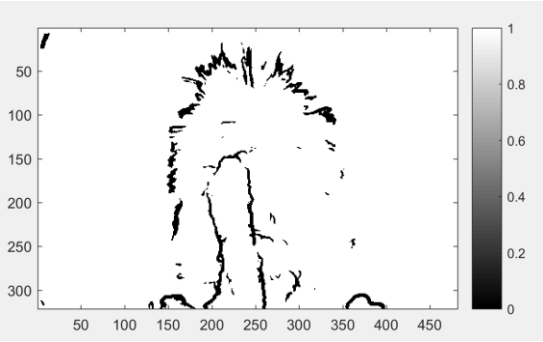
Figure 10: Edge Maps for the House.raw image for SE and Sobel Detector and Performance Evaluation Table.



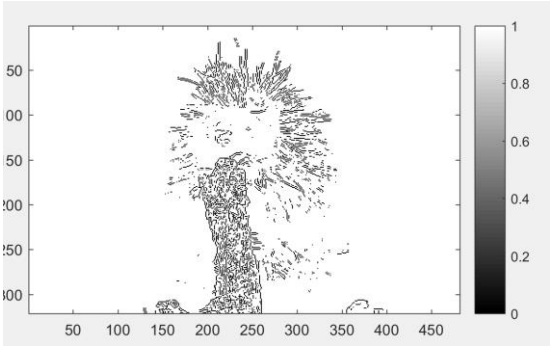
Animal.raw



Probability edge map for SE



Binary Edge Map for SE



Edge Map for Sobel

Parameter	GT1	GT2	GT3	GT4	GT5
P	8.24	9.09	11.37	8.76	9.01
R	21.44	19.24	16.05	21.20	19.47
F measure	12.33	12.72	12.45	12.56	12.29

F measure mean: 12.47 (All in terms of percentage)

Figure 11: Edge Maps for the Animal.raw image for SE and Sobel Detector and Performance Evaluation Table.

IV. Discussion

Discussion for SE- Detection Algorithm:

1. The Structured Edge Detection algorithm explanation along with the flowchart has been written and explained in the Approach and Procedure section under "Approach for Structured Edge Detection Algorithm**"**.

2. The process of decision tree construction and the principle of the RF classifier is also explained in the Approach and Procedure section under "Approach for Structured Edge Detection Algorithm".

3. From matlab these are the parameters that were chosen,
   Multiscale=0;
   Sharpen=2
   nTressEval=8
   nThreads = 4
   NMS = Yes
   Binary Threshold = 0.2

   The structured binary edge map was gives thicker outline of the edges and also covers less details of the images whereas Sobel gives lighter edges and covers more unnecessary details.

Discussion for Performance Evaluaion:

1. The experimental results above give the required tables for the various values of Precision, Recall and F-Score at Threshold 0.2. In F-measure, we evaluate the performance of edge maps based on ground truth (GT) developed by people. Based on the F-measure values for the Sobel and SE detector we see that SE detector has higher

F-measure values than Sobel. Therefore, the SE detector is better in performance than sobel due to High F-measure value which indicates it detects edges in a better manner than Sobel detector. The sobel detector detects a lot of false edges as well, this is also a big drawback and the reason for it's low F-measure.

2. Yes, The F-measure is image dependent. The House image will have more F-Measure than the Animal image because of less background obstruction. In the animal image there is a lot of background that hides the image and makes it unclear to view whereas in the House image the objects are clearly separated from any obstructive background.
3. The F measure is a measure of test accuracy in statistics. Its value is directly affected by both the parameters: precision and recall. If any of these parameters becomes low then this will result in a bad F-measure. An inverse relationship exists between precision and recall i.e if recall increases, then precision decreases and if precision increases, recall decreases. In order to maximize the product of the two parameters which follows an inverse relation, we need to have make them equal.

Problem 3: Salient Point Descriptors and Image Matching

(a) Extraction and Description of Salient Points
(b) Image Matching

I. Abstract and Motivation

In this section we are motivated to understand the sift and the surf algorithm for extraction and description of Salient points or features. The **scale-invariant feature transform** (**SIFT**) was designed by David Lowe to detect and describe local features in images. It is widely used in the area of computer vision for object recognition, image stitching, robotic applications and also for navigation. The **speeded up robust features** (**SURF**) algorithm is very similar to SIFT and is inspired by it. It is also used for detection and description of local features in images and is used in almost the same applications that SIFT encompasses. Using these feature descriptors we perform Image Matching which is also an application of these two algorithms. Hence, we take a look at both these algorithms and implement them using OpenCV for feature extraction and feature description.

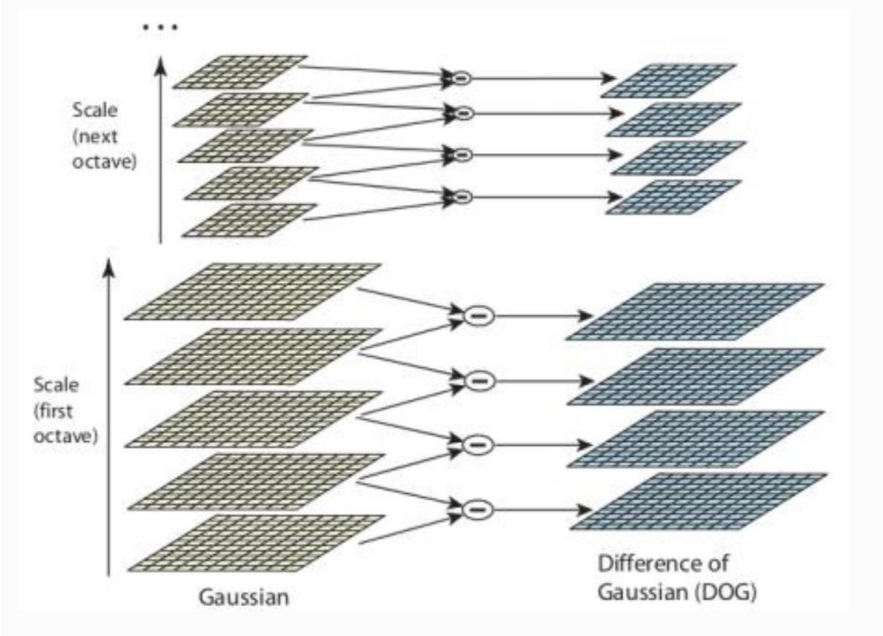
II. Approach and Procedures

First let's discuss the two algorithms and their approaches and procedures for feature extraction, Description and Object Matching.

SIFT Algorithm:

1. **Scale-space Extrema Detection**: In this step, Laplacian of Gaussian is found for the image with different sigma values. LoG behaves like blob detector which detects blobs in various sizes due to changes in sigma. Sigma here is a scaling parameter. With this method we find local extrema spanning scale and space, Hence following this we get a good keypoint (x,y) at sigma value. Since, LoG proves to be computationally expensive, therefore SIFT uses Difference of Gaussion which is performed on a pyramid like structure for a stack of images. Once, this is done images are searched

over scale and space to find local extrema. For instance, for a given pixel its 8 neighbouring pixels as well as its 9 pixel window neighbours in the corresponding upper stack and lower stack are searched for finding the local extrema. These are illustrated in the figures below.



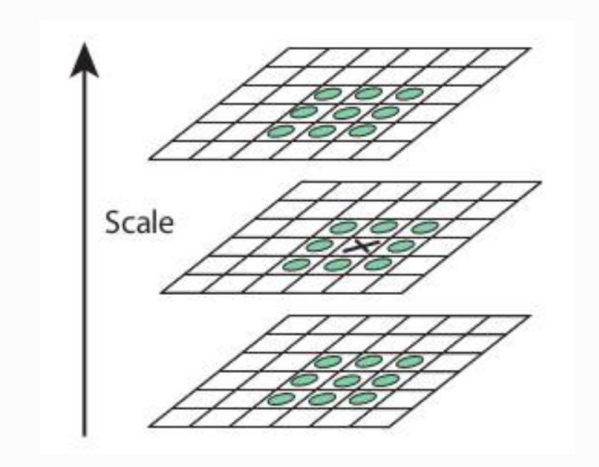


Figure 12: Scale-Space Extrema

2. **Keypoint Localization:** After finding good keypoints, they are subjected to some thresholding and also subjected to taylor series expansion of scale space to get better keypoints. In this stage bad keypoints or unsuitable keypoints are rejected.

3. **Orientation Assignment:** After the second step, an orientation is assigned in order to make the ketpoints invariant to rotation. A neigbourhood is selected around the keypoint depending on the scale, and the gradient magnitude and direction is calculated for that window. An orientation histogram with 36 bins covering 360 degrees is created.

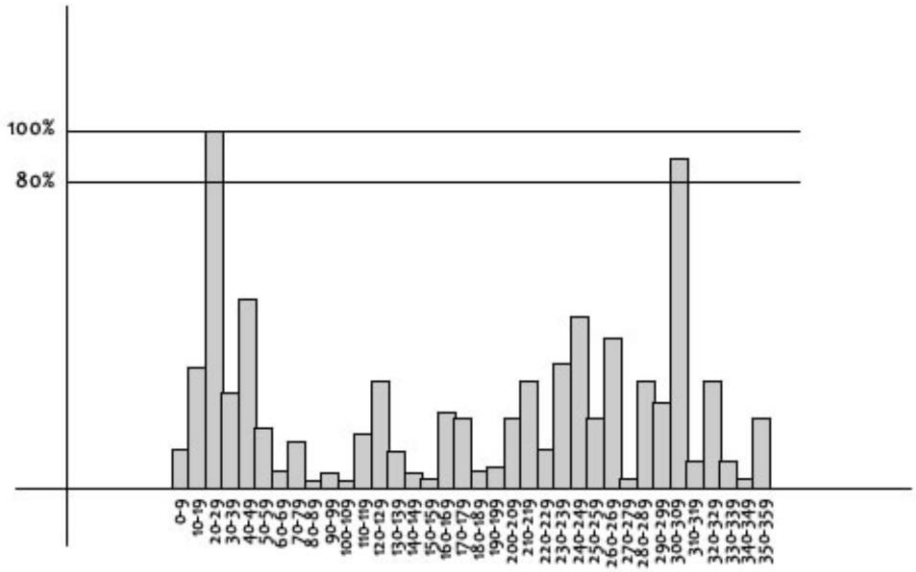
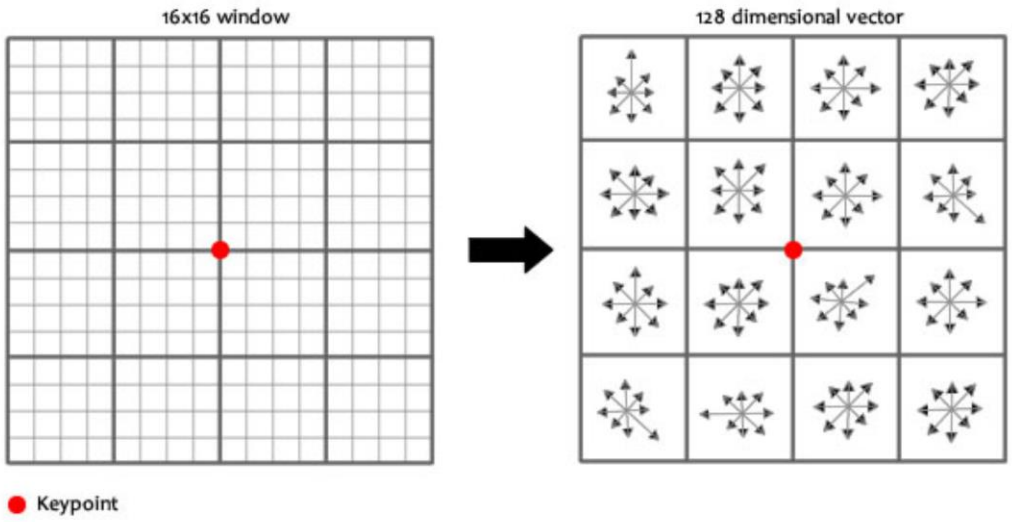


Figure 13: Orientation Histogram

4. **Keypoint Descriptor:** A keypoint descriptor is created. A 16x16 neighbourhood is taken around the keypoint. It is divided into 16 sub-blocks of size 4x4. For each sub-block, 8 bin orientation histogram is created. Therefore, a total of 128 bin values are present. This is represented in vector form to give rise to a keypoint descriptor.



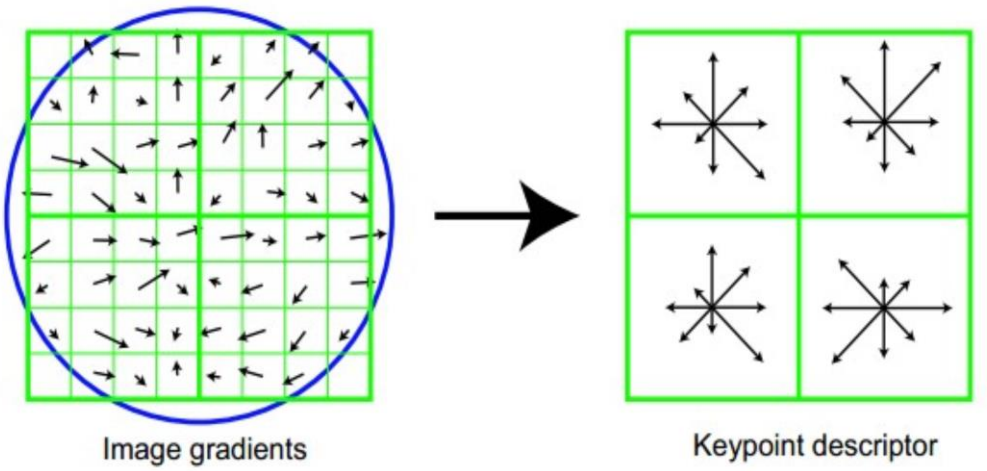


Figure 14: Keypoint Descriptor

5. Keypoint Matching: Keypoints among the two images are matched by identifying their nearest neighbours.

SURF Algorithm:

The Algorithm for SURF is very similar to SIFT hence, we shall cover it briefly:

1. **Scale-space Extrema Detection:** SURF uses LoG in square-shaped box filters for Gaussian smoothing. SURF uses a blob detector derived from the Hessian matrix to find keypoints. the Hessian matrix H(p, σ) at point p and scale σ, is given below:

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

It follows almost the same procedure as SIFT to find local extrema except that the LoG is applied in the form of box filters and influenced by the Hessian matrix to find keypoints.

2. **Descriptor:** SURF uses Wavelet responses in horizontal and vertical direction for feature description. A neighbourhood of size 20sX20s is taken around the keypoints where s is the size. To differentiate it from SIFT, SURF feature descriptor has an enlarged 128-dimension version.

3. **Orientation assignment:** The Haar wavelet responses in both x- and y-directions within a circular neighbourhood of radius 6's are computed in order to achieve rotational invariance.

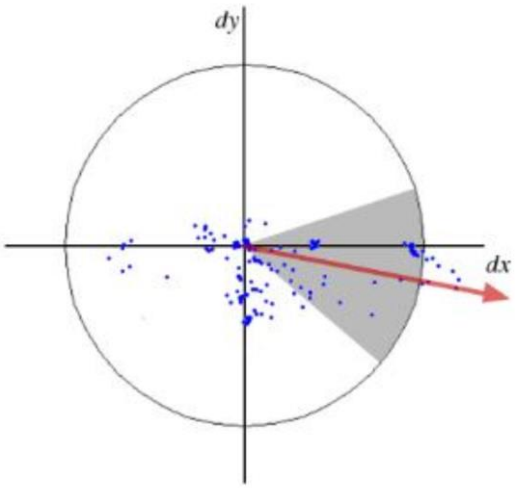
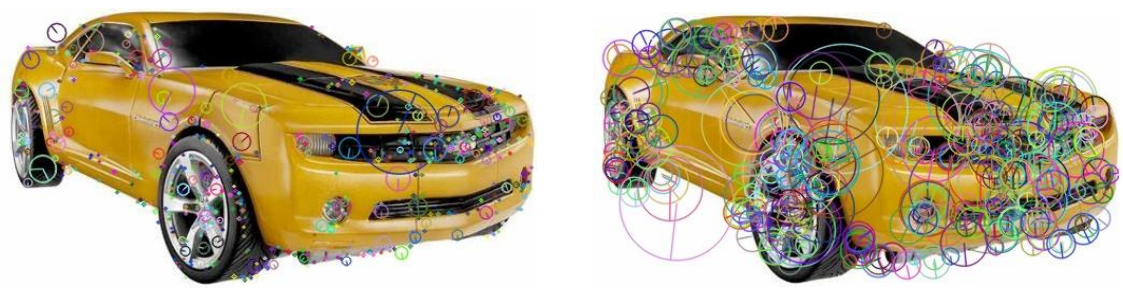


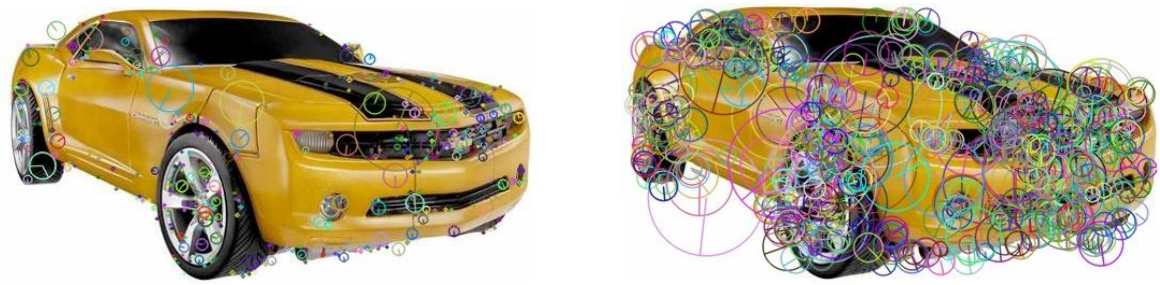
Figure 15: Orientation Assignment

4. **Matching:** Comparison between the descriptors obtained from different images yields matching pair of points.

III. Experimental Results:



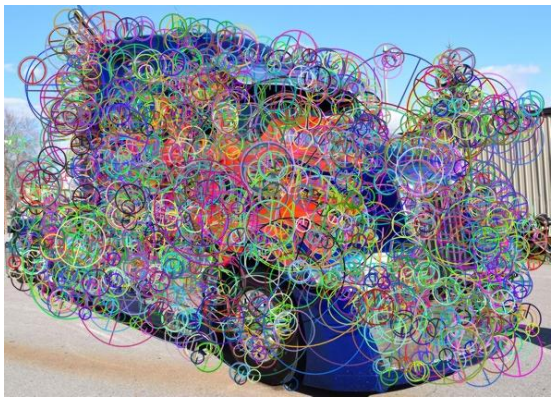
Sift-Surf Threshold = 1000



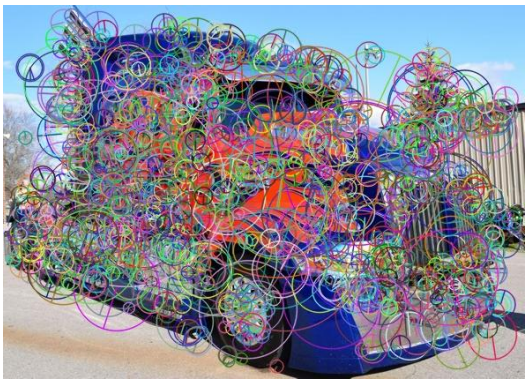
Sift-Surf Threshold=500


Sift-Surf Threshold=10,000


Sift-Surf Threshold=10,000

Sift-Surf Threshold=500



Sift – Surf Threshold=1000

```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -o Sift Sift.cpp `pkg-config opencv --cflags --libs`
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./Sift
 No of Keypoints Img1 -- Bumblebee : 500
 No of Keypoints Img2 -- Optimus Prime : 500
 No of Keypoints Img1 -- Ferrari 1 : 383
 No of Keypoints Img2 -- Ferrari 2 : 335
Sift runs for 555392 clicks 0.555392 seconds
Good Matches : 4
Good Matches : 4
Good Matches : 17
```

```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -o Sift Sift.cpp `pkg-config opencv --cflags --libs`
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./Sift
 No of Keypoints Img1 -- Bumblebee : 611
 No of Keypoints Img2 -- Optimus Prime : 700
 No of Keypoints Img1 -- Ferrari 1 : 383
 No of Keypoints Img2 -- Ferrari 2 : 335
Sift runs for 698443 clicks 0.698443 seconds
Good Matches : 5
Good Matches : 10
Good Matches : 17
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$
```

```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -o Sift Sift.cpp `pkg-config opencv --cflags --libs`
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./Sift
 No of Keypoints Img1 -- Bumblebee : 611
 No of Keypoints Img2 -- Optimus Prime : 1000
 No of Keypoints Img1 -- Ferrari 1 : 383
 No of Keypoints Img2 -- Ferrari 2 : 335
Sift runs for 619852 clicks 0.619852 seconds
Good Matches : 5
Good Matches : 11
Good Matches : 17
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$
```

Execution and Performance Results for SIFT

```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -o Surf Surf.cpp `pkg-config opencv --cflags --libs`
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./Surf
 No of Keypoints Img1 -- Bumblebee : 633
 No of Keypoints Img2 -- Optimus Prime : 1748
 No of Keypoints Img1 -- Ferrari 1 : 465
 No of Keypoints Img2 -- Ferrari 2 : 405

Surf runs for 754599 clicks 0.754599 seconds

Good Matches : 7
Good Matches : 7
Good Matches : 72
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$
```

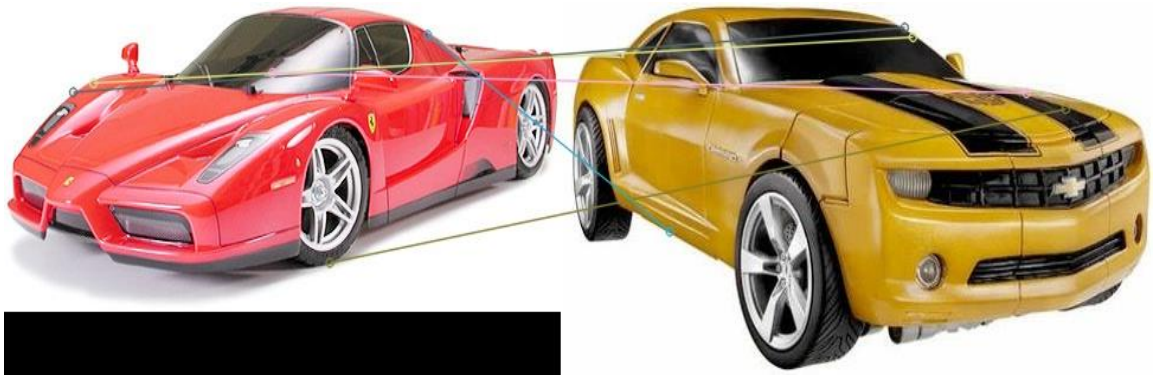
```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -o Surf Surf.cpp `pkg-config opencv --cflags --libs`
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./Surf
 No of Keypoints Img1 -- Bumblebee : 521
 No of Keypoints Img2 -- Optimus Prime : 1546
 No of Keypoints Img1 -- Ferrari 1 : 381
 No of Keypoints Img2 -- Ferrari 2 : 337

Surf runs for 652299 clicks 0.652299 seconds

Good Matches : 6
Good Matches : 7
Good Matches : 96
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$
```

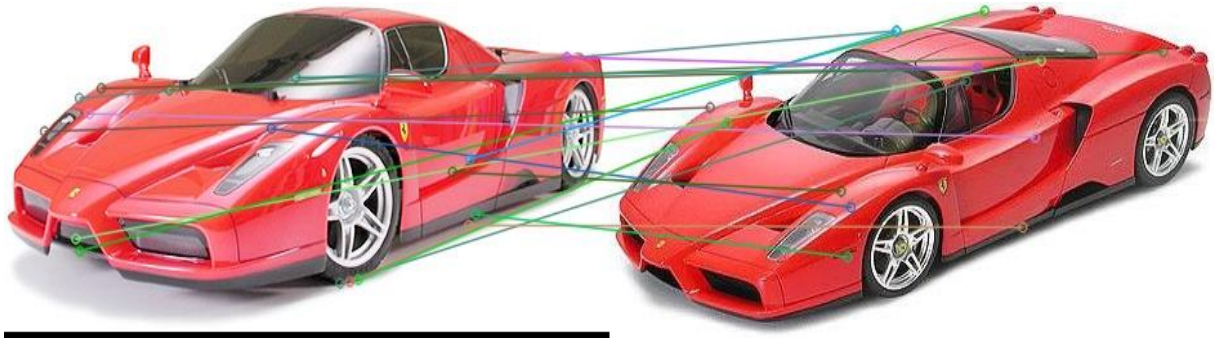
```
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ g++ -o Surf Surf.cpp `pkg-config opencv --cflags --libs`
ishan@ishan-VirtualBox:~/Desktop/DIP/HW3Programs$ ./Surf
 No of Keypoints Img1 -- Bumblebee : 421
 No of Keypoints Img2 -- Optimus Prime : 1302
 No of Keypoints Img1 -- Ferrari 1 : 294
 No of Keypoints Img2 -- Ferrari 2 : 279

Surf runs for 578928 clicks 0.578928 seconds

Good Matches : 39
Good Matches : 2
Good Matches : 75
```

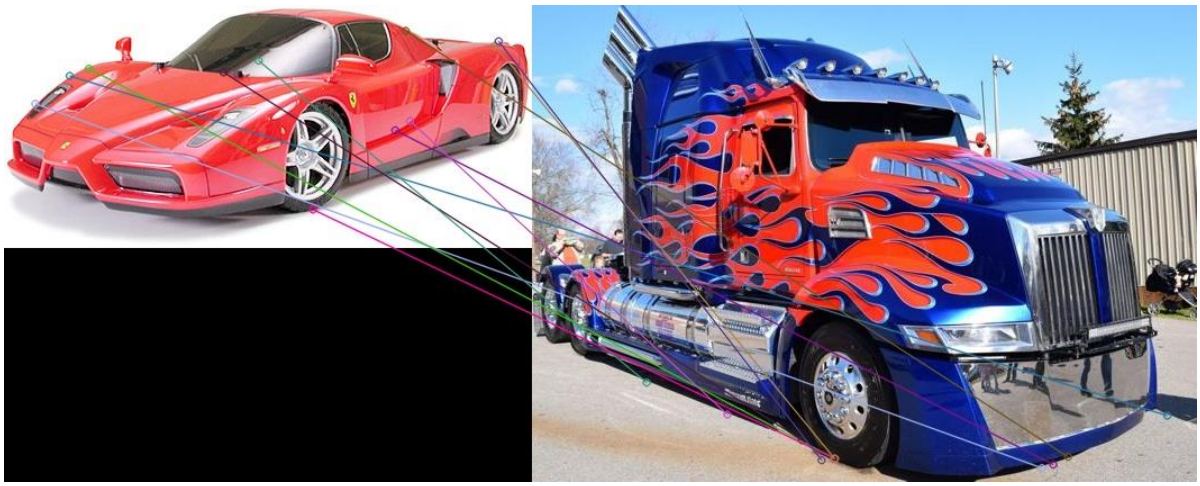
Execution and Performance Results for SURF

Figure 15: SIFT and SURF results for Extraction

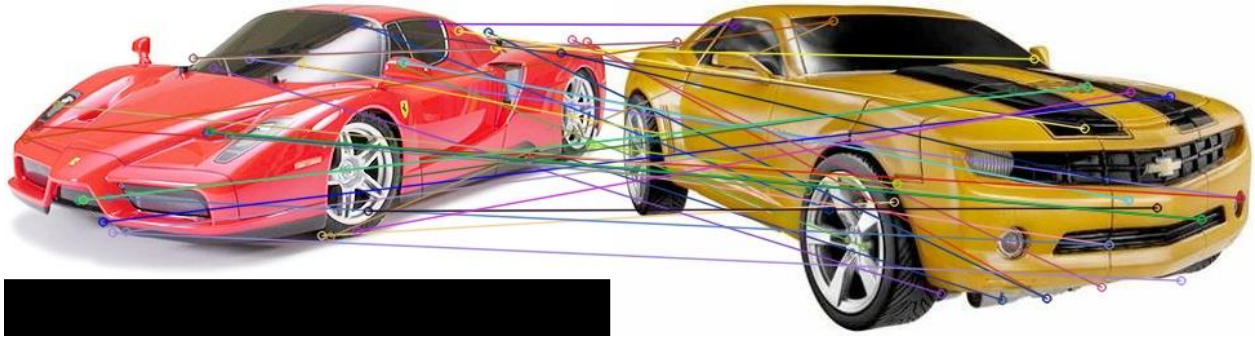
Image Matching

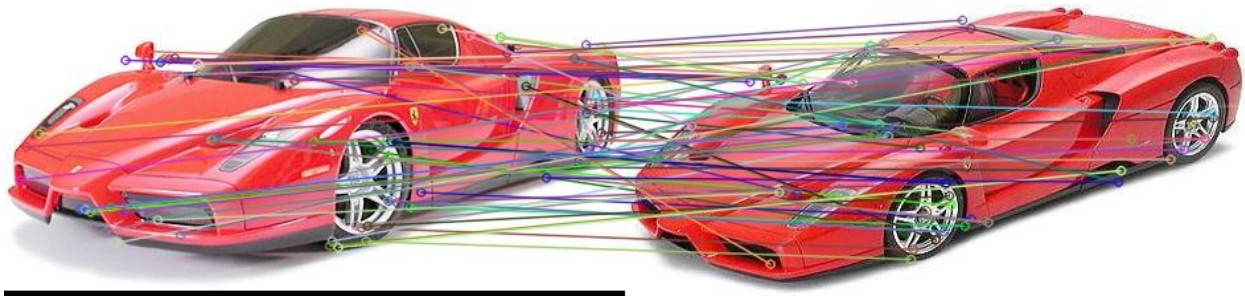


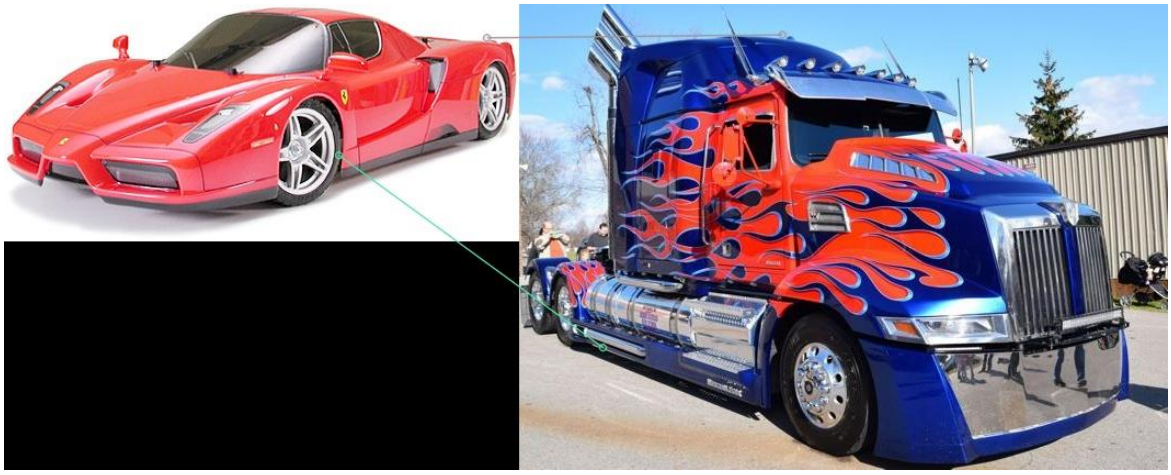




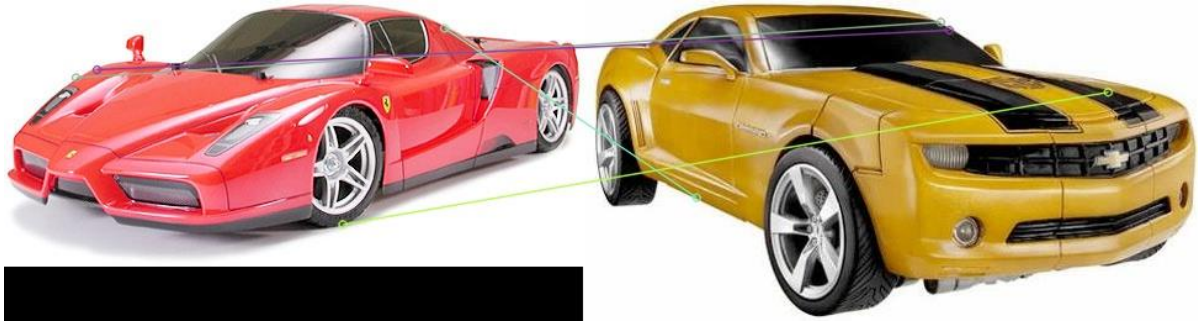
Object Matching Sift Threshold = 1000

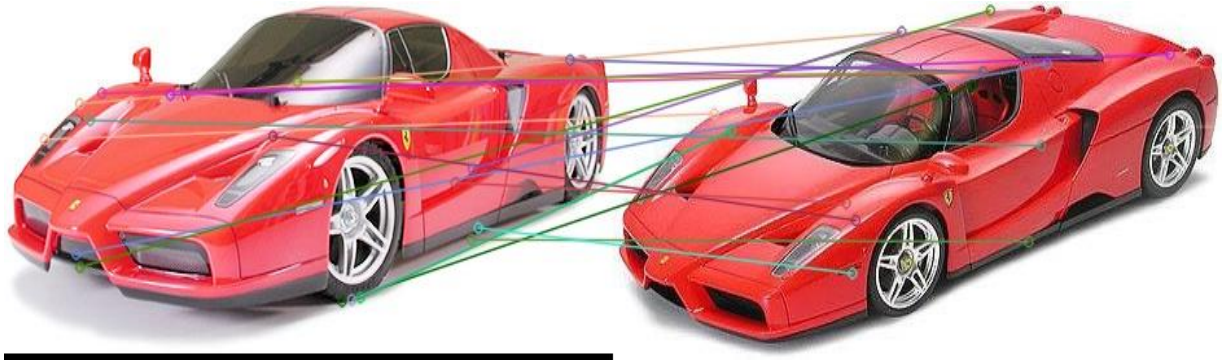


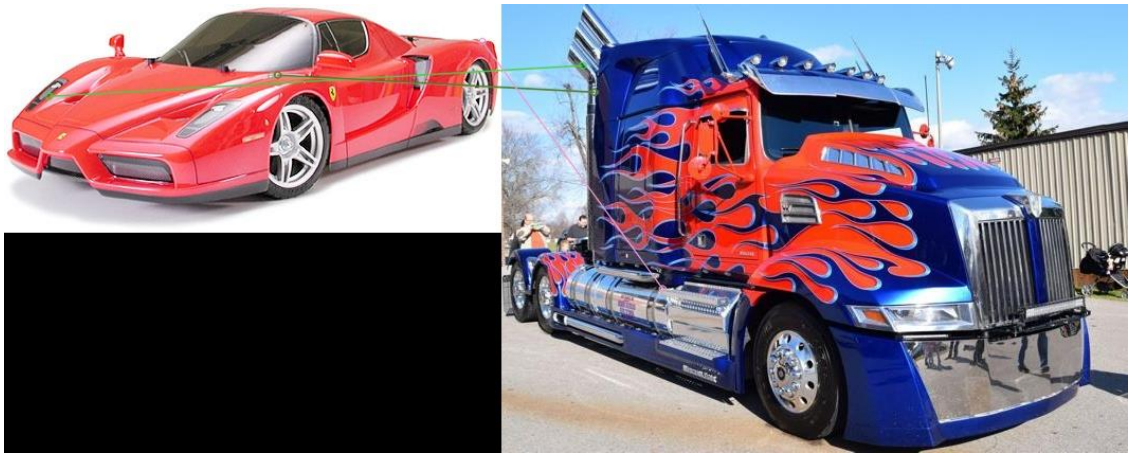




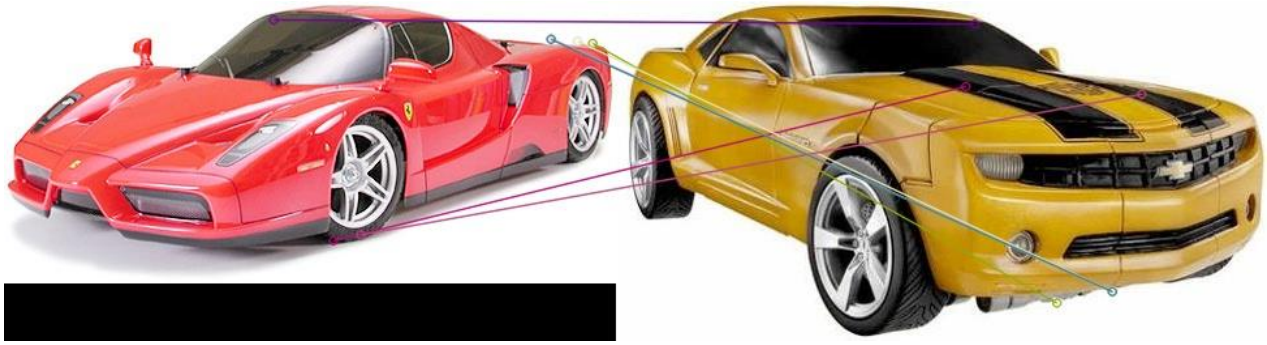
Object Matching Surf Hessian Threshold - 1000

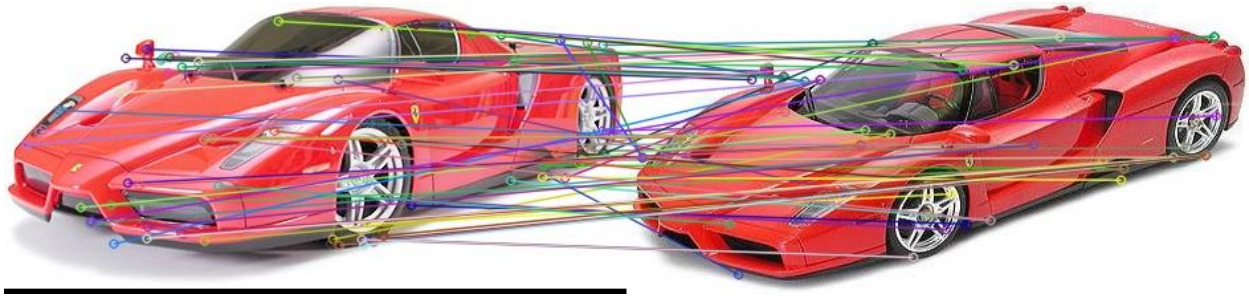


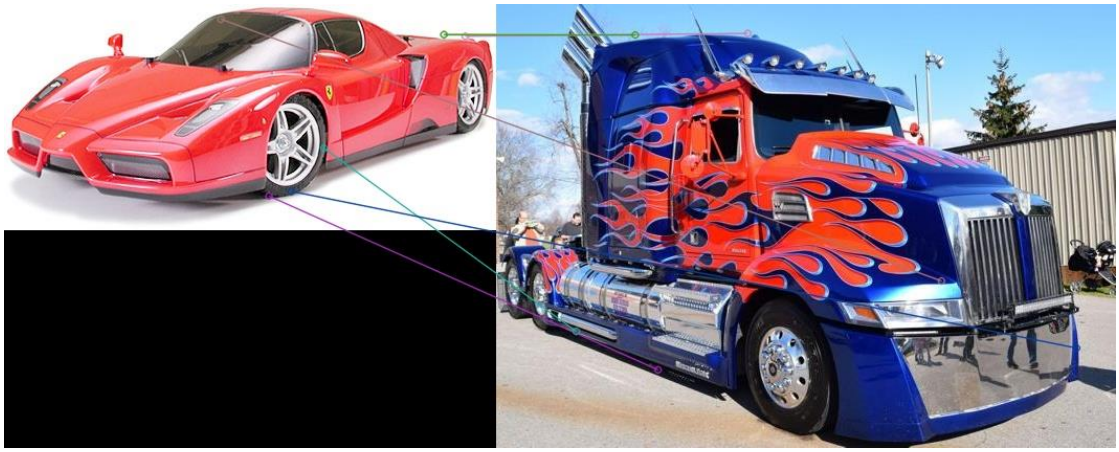




Object Matching Sift Threshold = 500







Object Matching Surf Hessian Threshold – 500

Figure 16: Object Matching SIFT and SURF

IV. Discussion

1. The keypoints and features have been extracted and described. The results have been posted in the experimental section. Let us analyse these results, for SIFT when we give a threshold of 500 & 1000 we observe that there are lesser keypoints extracted and displayed compared to when we set the Hessian Threshold of the SURF algorithm to 500 & 1000. We see that after applying SURF with these hessian thresholds, we get a huge number of salient keypoints. But when we increase this threshold for SIFT and SURF to 10,000 the number of SIFT keypoints increases dramatically whereas for SURF the number of keypoints decreases. Therefore, we see that for SIFT the number of keypoints is directly proportional to the Threshold and for SURF the number of keypoints is inversely proportional to the Hessian Threshold.

2. Performance and efficiency: Observing the figure which contains all the Execution results for SIFT and SURF we observe that both SIFT and SURF are almost identical in terms of performance. We see in the figure that in most cases SURF runs faster than SIFT but the only thing to consider here is that in those cases SURF extracts lesser number of keypoints compared to SIFT. SURF is generally a bit faster than SIFT but the OpenCV libraries for SURF is not as efficient as the real algorithm. In our Implementation case we can say that SIFT and SURF are almost equal to each other in terms of performance and efficiency, noting that SURF is faster but compromises the number of Keypoints whereas SIFT being a little slower generates more keypoints. When both the algorithms generate equal number of keypoints, they are almost equal to each other. The efficiency in SURF is due to optimization. Surf constructs its scale space and keypoint detector using integral images which is much faster than SIFT way of generating its scale space. SURF also extracts lesser but meaningful points which helps bring down execution time.

3. In the Object Matching case we see that the Matching points are lesser when you compare the Ferrari_1 with the bumble bee car and the Optimus prime image, whereas there are more matching points in case of Ferrari_1 and Ferrari_2 due to the same car being positioned with an angle with respect to the original Ferrari_1 image. The object matching may not give the best results using SIFT and SURF if we used Brute Force

technique as this technique every point is matched and this becomes irrelevant and difficult in filtering noisy points. The brute force method matches unessential points by setting up a minimum distance value. We could solve this issue by using the FLANN based matcher that utilizes the nearest neighbour technique for getting better object matching results.

(b) Bag of Words

I. Abstract and Motivation

Bag of words model is used for visual data classification in the field of computer vision by modelling image features as words in a dictionary. “In computer vision, a bag of visual words is a vector of occurrence counts of a vocabulary of local image features.”-wikipedia. The general idea regarding the bag of words is to represent “documents” as a collection of crucial keypoints while ignoring the order in which they appear. In this section we are motivated to implement the algorithm using OpenCV and to plot the histogram of the 4 images as code-words.

II. Approach and Procedures

The following describes the procedure to the bag of words algorithm:

1. **Feature extraction**: The first step in constructing a bag of visual words is to do feature extraction by collecting descriptors from each image in the set. In our approach we use SIFT to take care of detection and feature extraction.

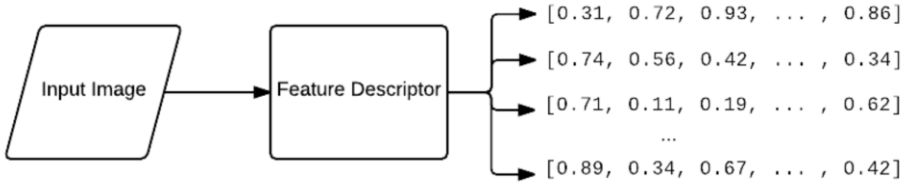


Figure 17: Feature Extraction

2. **Dictionary/Vocabulary construction:** After extracting feature vectors from each image in the set, we construct our vocabulary of possible visual words. Vocabulary construction is achieved via the **k-means clustering algorithm** where we cluster the feature vectors that we obtained from feature extraction step. The Cluster Centroids are given the meaning of dictionary of visual words.

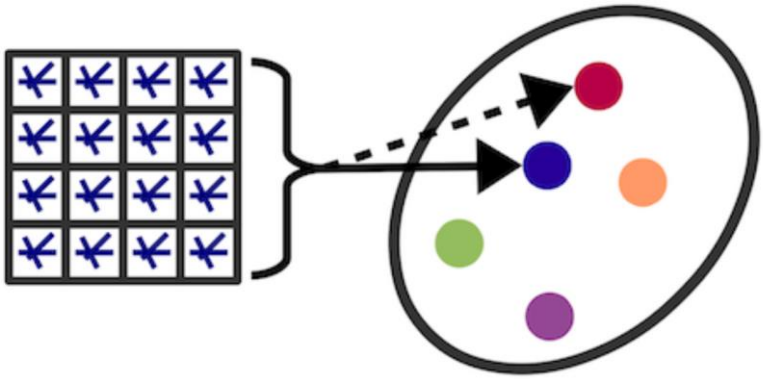


Figure 18: Dictionary Construction

3. **Vector quantization**: Now if we have to test an unknown image we can follow steps1 and 2 and then perform vector quantization. This is done by taking the set of nearest neighbour labels and building an appropriate histogram of length *k* (K classes from k-means algorithm), where the *j*'th value in the histogram is the frequency of the *j*'th visual word. This process of modeling an object by its arrangement of vectors is commonly called vector quantization.

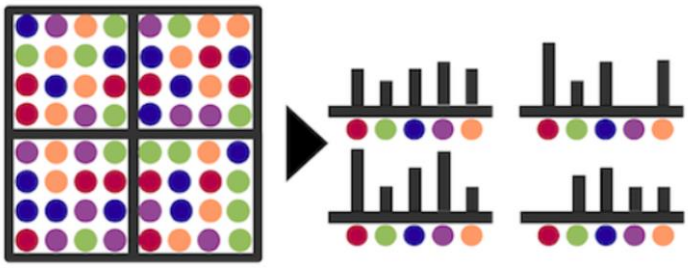


Figure 19: Vector Quantization

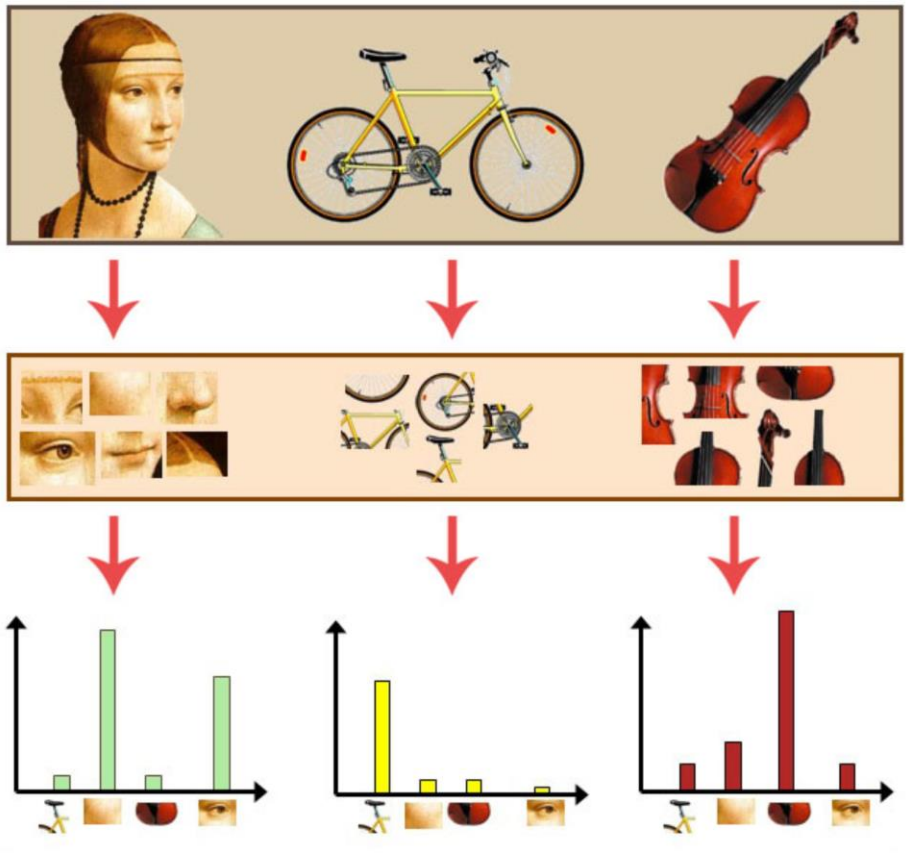


Figure20: Bag of words Entire procedure

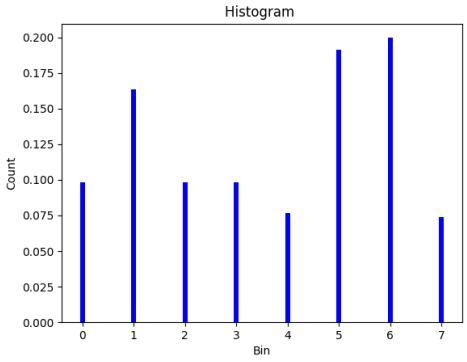
III. Experimental Results:

bumble bee

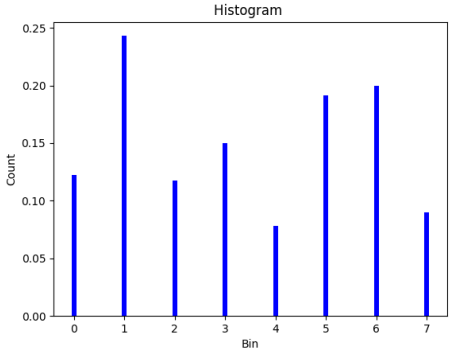
Ferrari 1

Ferrari 2

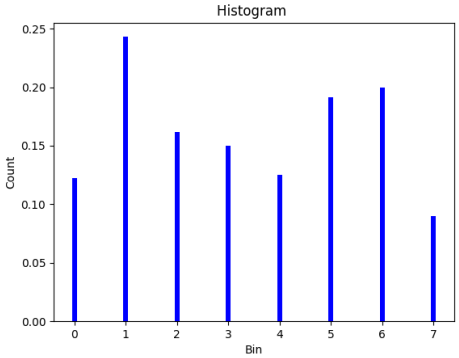
Optimus Prime



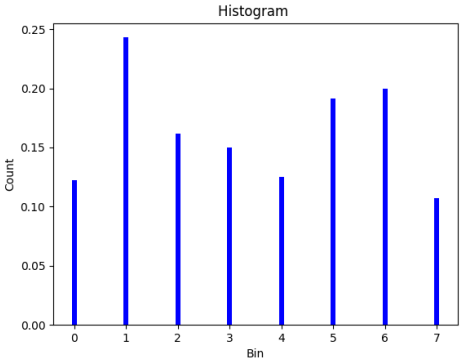
1 – bumble bee



2 – optimus prime



3 – Ferrari 1



4 – Ferrari 2

Figure 21: Histogram construction of BoW for Different Images

IV. Discussion

After running the bag of words algorithm using OpenCV, we observe the following histograms above. We see that both the Ferrari_1 image and the Ferrari_2 image have the exact same histogram, whereas the other histograms of Bumble bee and Optimus prime are different from each other and both the Ferrari cars image. This is a visual confirmation that the bag of words works according to its design. Hence, even if the same image orientation changes it will have the same histogram and therefore the same vocabulary in the dictionary of the bag of words.

References:

1. SURF: https://en.wikipedia.org/wiki/Speeded_up_robust_features.
2. Bag of words: https://gurus.pyimagesearch.com/the-bag-of-visual-words-model/
   http://clic.cimec.unitn.it/vsem/content/bovw.html
3. Fast Edge Detection Using Structured Forests - Piotr Doll´ar and C. Lawrence Zitnick