EE 569: Homework #3

Table of Contents:

Problem 1: Texture Analysis and segmentation

a) Texture Classification

I. Abstract and Motivation

Texture Analysis is a very important in many computer engineering applications like remote sensing, ground classification and segmentation of a satellite, biomedical applications etc. While analyzing the textures in an image, the local spatial variation of intensity in the image is taken into consideration. This local variation of intensity helps in identifying the spatial structure, contrast, roughness, orientation, etc. of the image. Texture Analysis mainly comprises of texture segmentation, texture classification, texture synthesis and shape analysis of textures. We are going to implement the texture classification and texture segmentation methods in this assignment.

Texture classification involves classifying the textures in an image into categories using some classification algorithm like k-means clustering. To extract the features from the image, 5x5 or 3x3 Laws filter are used and the localized energy of these laws filters is used as feature vectors for a particular image. Depending on these feature vectors, each image is then clustered into a cluster having similar properties. Each cluster is represented by the centroid for that cluster.

II. Approach and Procedure

Task: To cluster the 12 texture images into 4 clusters by using the K-means algorithm on the feature vectors generated by applying Laws filters on the images and finding the localized energy values for each image.

The texture classification algorithm is implemented using the following steps:

a) Pre-processing

In the pre-processing step, the 12 texture images obtained are first converted to double data format for ease of calculation in the further steps. The DC component of the image is then removed by finding the average of all pixel values in the image and subtracting this average from each pixel of the image. The removal of DC component in the image is required as it contains redundant information which can be neglected. Also, removal of DC components prevents any high energy value to dominate the feature vector generation. The formula for DC removal is as given below:

$$Image(DC\ removed) = Input\ image - (Mean\ of\ all\ pixels\ in\ each\ of\ the\ input\ image)$$

This DC removal is done for each of the 12 texture input images.

b) *Filtering of the image using Laws filters*

Laws filters are used as feature extraction tools wherein the given input images are convolved with each of the nine 5x5 Laws filters generated using Tensor products of 1D Kernels.

The five 1D Kernels are given as follows:

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]

The 5 filters have the following significance:
L5- To average the values of pixels
E5- To detect edges
S5- To detect spots
W5- To detect waves
R5- To detect ripples

Out of the 5 1D kernels, E5, S5 and W5 are used for generation of filter bank consisting of 9 Laws filters. The other 2 filters i.e. L5 and R5 are discarded as L5 is a low pass filter and the 12 input images have low and medium frequencies in them. The 3 filters used finally are given as follows:

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

Each filter mentioned above is convolved with another filter to get the 9 possible tensor products. To convolve the filters with the pre-processed input images, pixel replication is used, and the boundaries of the image are extended. These 9 filtered images obtained from Laws filters are then used to obtain the local energies.

c) *Calculation of energy features*

The 9 filtered outputs generated in the previous step are used to obtain the localized energy values which will in turn generate the energy vectors. This is called as multi-channel feature extraction method. To calculate the feature vectors, the energy is computed using the following formula:

$$\frac{1}{Image\ Height * ImageWidth} * [summation\ of\ all\ pixel\ values\ in\ each\ input\ image]$$

This localized energy is computed for each input image and each Laws filter. Hence, the size of the feature vector is 12x9.

d) K-means algorithm

K-means algorithm is used to cluster the 12 input images using the 12x9 feature vectors. As the data has to be clustered in 4 clusters, 4 centroids are required which are initialized in an arbitrary manner.
The K-means algorithm can be explained as follows:

Step 1: Place the 12x9 feature vectors representing 12 input images in the feature space and initialize the K centroids required to cluster the images.
Step 2: Assigning each image to a particular cluster by finding the Euclidean distance of each image with the K centroids.
Step 3: When all images have been assigned to certain clusters, recalculate the centroids by considering the images in that cluster.
Step 4: Run this algorithm iteratively until K centroids do not change over one iteration.

The entire flow of the algorithm can be explained using the following figure:

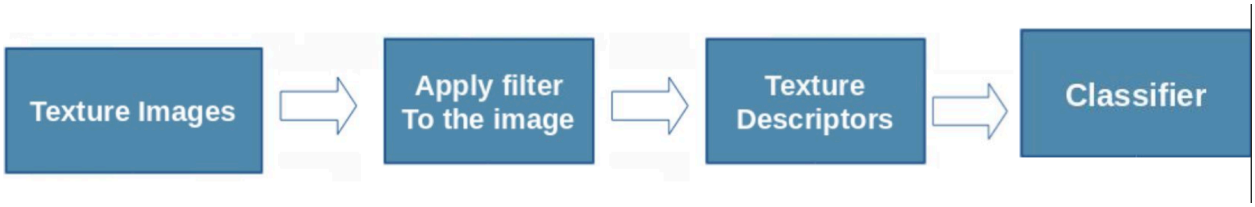


Figure 1: Flow chart for Texture Classification algorithm (Source: Discussion 8- Lecture notes)

Algorithm for Texture Classification:

Step 1: Read the contents of the 12 input images "Texture 1 to Texture 12" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert 1D input images into a 1D images of double data format.
Step 3: Convert 1D double data format images into 2D arrays for R channel as the image is comprised of a single channel.
Step 4: Calculate the tensor product of the 3 Laws filters to get nine 5x5 Laws filters.
Step 5: Remove the DC component of each image by subtracting the average from each pixel of the image. Refer to the equation for DC removal of image.
Step 6: Convolve the DC removed input images with 9 Laws filters to obtain 9 filtered outputs for each image. Before convolving the images, pixel replication is performed to extend the boundaries.
Step 7: Calculate the energy for each of the input images for each Laws filter. This step will generate the 12x9 feature vector required for K-means clustering.
Step 8: K- means algorithm
Step 8a: Initialize the 4 centroids arbitrarily.

Step 8b: Calculate the Euclidean distance of each image with the 4 centroids and assign the label for that image depending on the minimum Euclidean distance.

Step 8c: Update the centroids by finding the average of the feature Values in each of the cluster.

Step 8d: Repeat the steps iteratively until the centroids don't change for one iteration.

Step 9: Display the final label obtained from Step 8 to see which input image is classified into which cluster.

III. Experimental Results

The Nine 5x5 Laws filters are as follows:

```
 E5E5 Laws filter:
0.0277778 0.0555556 -0 -0.0555556 -0.0277778
0.0555556 0.111111 -0 -0.111111 -0.0555556
-0 -0 0 0 0
-0.0555556 -0.111111 0 0.111111 0.0555556
-0.0277778 -0.0555556 0 0.0555556 0.0277778

 E5S5 Laws filter:
0.0416667 -0 -0.0833333 -0 0.0416667
0.0833333 -0 -0.166667 -0 0.0833333
-0 0 0 0 -0
-0.0833333 0 0.166667 0 -0.0833333
-0.0416667 0 0.0833333 0 -0.0416667

 E5W5 Laws filter:
0.0277778 -0.0555556 -0 0.0555556 -0.0277778
0.0555556 -0.111111 -0 0.111111 -0.0555556
-0 0 0 -0 0
-0.0555556 0.111111 0 -0.111111 0.0555556
-0.0277778 0.0555556 0 -0.0555556 0.0277778

 S5E5 Laws filter:
0.0416667 0.0833333 -0 -0.0833333 -0.0416667
-0 -0 0 0 0
-0.0833333 -0.166667 0 0.166667 0.0833333
-0 -0 0 0 0
0.0416667 0.0833333 -0 -0.0833333 -0.0416667

 S5S5 Laws filter:
0.0625 -0 -0.125 -0 0.0625
-0 0 0 0 -0
-0.125 0 0.25 0 -0.125
-0 0 0 0 -0
0.0625 -0 -0.125 -0 0.0625

 S5W5 Laws filter:
0.0416667 -0.0833333 -0 0.0833333 -0.0416667
-0 0 0 -0 0
-0.0833333 0.166667 0 -0.166667 0.0833333
-0 0 0 -0 0
0.0416667 -0.0833333 -0 0.0833333 -0.0416667

 W5E5 Laws filter:
0.0277778 0.0555556 -0 -0.0555556 -0.0277778
-0.0555556 -0.111111 0 0.111111 0.0555556
-0 -0 0 0 0
0.0555556 0.111111 -0 -0.111111 -0.0555556
-0.0277778 -0.0555556 0 0.0555556 0.0277778
```

```
 W5S5 Laws filter:
0.0416667 -0 -0.0833333 -0 0.0416667
-0.0833333 0 0.166667 0 -0.0833333
-0 0 0 0 -0
0.0833333 -0 -0.166667 -0 0.0833333
-0.0416667 0 0.0833333 0 -0.0416667

 W5W5 Laws filter:
0.0277778 -0.0555556 -0 0.0555556 -0.0277778
-0.0555556 0.111111 0 -0.111111 0.0555556
-0 0 0 -0 0
0.0555556 -0.111111 -0 0.111111 -0.0555556
-0.0277778 0.0555556 0 -0.0555556 0.0277778
```

Figure 2: Nine 5x5 Laws filters

```
12x9 Feature vector:

348.429 186.763 61.3237 185.493 118.002 43.5267 53.9798 38.377 15.3386

47.5558 31.1735 9.43407 23.504 16.9658 5.81954 5.97848 4.7019 1.85115

25.6384 10.6018 2.89253 13.0281 5.70674 1.65945 4.12969 1.85916 0.554682

399.888 216.666 67.5494 231.678 139.945 47.3686 67.6661 45.7684 16.7502

33.8796 12.6823 2.89708 16.9947 6.64002 1.57867 4.9563 2.03836 0.487559

374.351 210.394 68.9585 207.306 133.021 49.0818 62.6901 45.0478 17.9895

2.65409 2.32582 1.20472 2.38193 2.27593 1.22917 1.22167 1.24336 0.697996

2.88115 2.35499 1.24216 2.39445 2.31956 1.29019 1.19406 1.25079 0.722932

28.3581 14.1254 3.2764 13.2958 7.47142 1.92072 3.65611 2.26212 0.636587

2.42936 1.54457 0.484627 1.97534 1.42324 0.475001 0.920923 0.706593 0.251835

17.7414 7.58545 2.23867 7.62397 3.24359 1.01368 1.71569 0.750639 0.245106

53.2846 36.6017 11.9383 28.484 23.5744 9.01584 8.81258 8.14492 3.61774
```

Figure 3: 12x9 Feature Vectors for Texture 1-12 images

Image	CLass Label	Class Name
0	1	Rock
1	2	Grass
2	3	Weave
3	1	Rock
4	3	Weave
5	1	Rock
6	4	Sand
7	4	Sand
8	3	Weave
9	4	Sand
10	3	Weave
11	2	Grass

Figure 4: Final Classification of each texture

```
Count for each label is:
1    3
2    2
3    4
4    3
```

Figure 5: Count of labels for 12 images

The following is the cluster numbering:

Rock-Cluster 1
Grass-Cluster 2
Weave- Cluster 3
Sand- Cluster 4

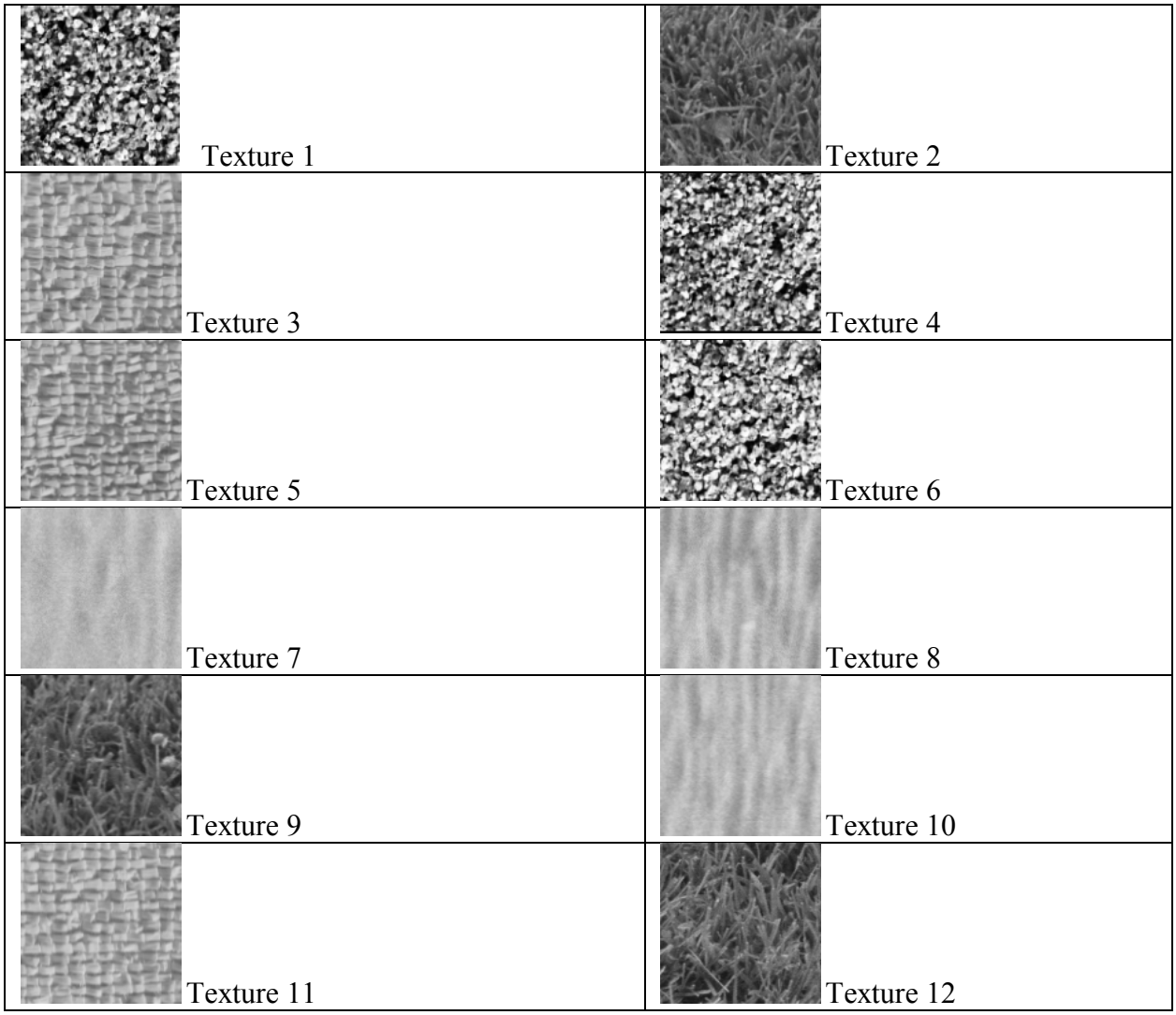


Figure 6: Twelve Input Texture Images

IV. Discussion

Texture classification on the input texture images was done by using K-means clustering algorithm. The obtained results have been shown above. The following conclusions were made by performing clustering:

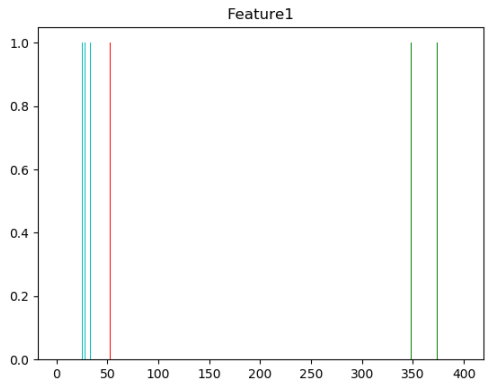
1) Pixel replication method was used to extend the boundary of the image to be classified.
2) The 12 input images can be visually inspected to make a conclusion about the clusters they belong to. The following table is prepared to compare the visually inspected and actually obtained results:

Image Number	Classification by visual Inspection	Classification according to algorithm
1	Rock	Rock
2	Grass	Grass
3	Weave	Weave
4	Rock	Rock
5	Weave	Weave
6	Rock	Rock
7	Sand	Sand
8	Sand	Sand
9	**Grass**	**Weave**
10	Sand	Sand
11	Weave	Weave
12	Grass	Grass

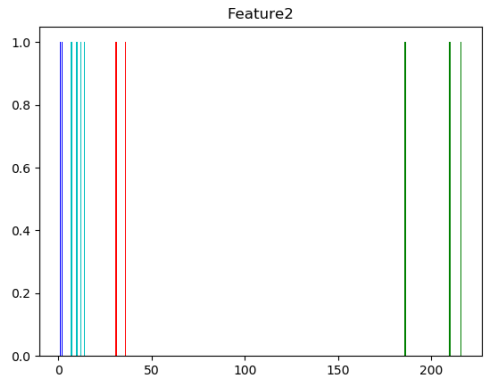
As seen from the table above, the visual inspection of the images coincides with the actual clusters obtained by running the texture classification algorithm, except that Image 9 gets misclassified as Weave instead of Grass. The actual outputs obtained from the algorithm are displayed in figure 4, where the indexing for images starts with 0.
The count for each cluster is reflected in Figure 5, which coincides with the fact that one image has been misclassified as the count for Cluster 2 which is Grass is 2 instead of 3, whereas the count for cluster 3 which is Weave is 4. Ideally, each cluster should have 3 images.

3) The discriminant power of a feature

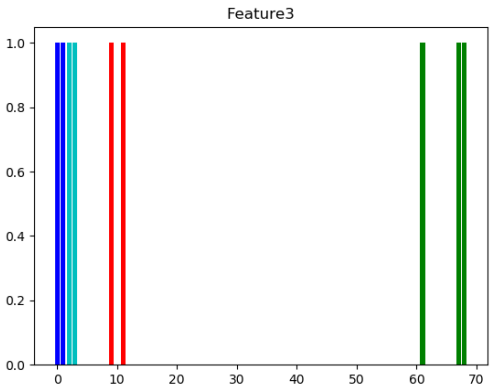
The discriminant Analysis is used to decide the discrimination power of each feature. If a feature has good discriminant power, it is able to cluster the inputs properly. As we have 12 input images and 9 features, I have plotted bar graphs for all the 9 features. The features who have a large number of overlapping points will have less discriminant power.



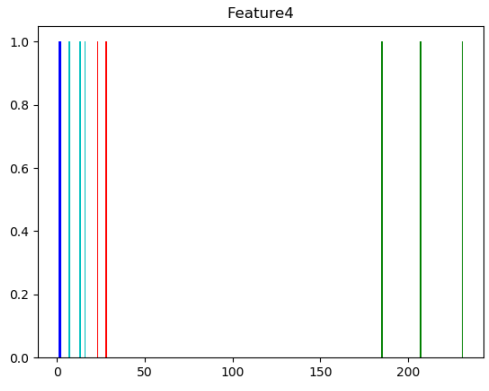
E5E5 Filter



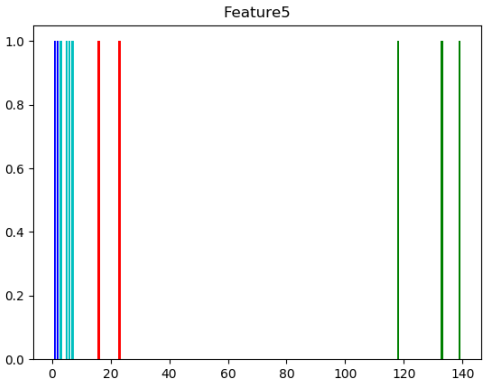
E5S5 Filter



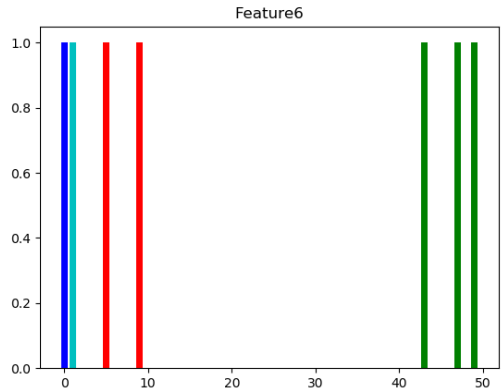
E5W5 Filter



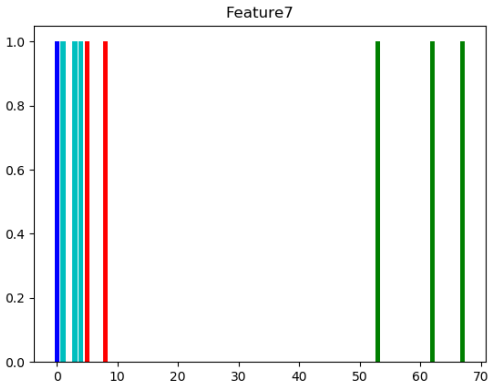
S5E5 Filter



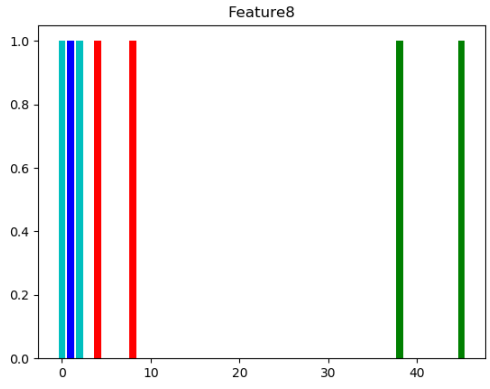
S5S5 Filter



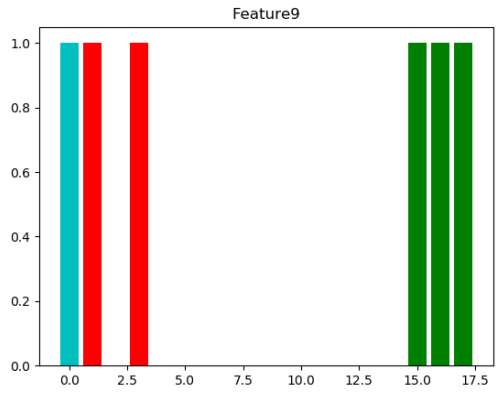
S5W5 Filter



W5E5 Filter



W5S5 Filter



W5W5 Filter

According the plots obtained,
Figure 2: **E5S5 has the strongest discriminant power** as it has 4 separate clusters for all the data points. None of the data points overlap.
Figure 8: **W5S5 has the weakest discriminant power** as it has 2 clusters very close and one cluster is inside another cluster which is not useful while clustering the inputs.

I could also conclude that a single feature is not useful while clustering the input and hence a combination of different features is required to get the desired output.

b) Texture Segmentation

I. Abstract and Motivation

Texture segmentation is an important part of texture analysis wherein the textures in the image are separated into partitions of disjoint regions based on texture properties, such that each region has some specific texture characteristics. Texture segmentation is similar to texture classification as it also involves feature extraction and derivation of metrics to segregate textures. However, it is difficult than texture classification as in texture segmentation the boundaries for different textures have to be recognized in addition to identifying the different type of textures in each region. Texture segmentation is mainly used in object recognition.

Texture segmentation involves segmenting the textures in the image into regions using some classification algorithm like k-means clustering. To extract the features from the image, 3x3 Laws filter are used and the localized energy of these laws filters for each pixel is used as feature vectors.

II. Approach and Procedure

Task: To apply texture segmentation to an image using 3x3 Laws filters.

The texture segmentation algorithm is implemented as follows:

a) Pre-processing

The preprocessing step is performed in texture segmentation as well ,the only difference is that the localized means of the pixels is calculated for each pixel using windowing approach. This localized mean is then subtracted from the respective pixel to reduce illumination effects present in the pixel. To find the localized mean of each pixel, a 21x21 filter is being used. The formula for preprocessing can be given as below:

$$Input\ image = Input\ image - (Mean\ of\ all\ pixels\ in\ each\ window\ of\ the\ input\ image)$$

As the input image is updated by subtracting the mean, the resultant input image has the DC components reduced from it.

b) Filtering of the image using Laws filters

Laws filters are used as feature extraction tools wherein the given input image is convolved with each of the nine 3x3 Laws filters generated using Tensor products of 1D Kernels.

The 3 1D kernels used are given as follows:

$$\boldsymbol{E3} = \frac{\mathbf{1}}{\mathbf{2}} * [\mathbf{-1\ 0\ 1}] \qquad \boldsymbol{S5} = \frac{\mathbf{1}}{\mathbf{2}} * [\mathbf{-1\ \ 2\ -1}] \qquad \boldsymbol{W5} = \frac{\mathbf{1}}{\mathbf{6}} * [\mathbf{1\ 2\ 1}]$$

Each filter mentioned above is convolved with another filter to get the 9 possible tensor products. To convolve the filters with the pre-processed input image, pixel replication is used, and the boundaries of the image are extended. These 9 filtered images obtained from Laws filters are then used to obtain the local energies.

c) Calculation of energy features

The 9 filtered outputs generated in the previous step are used to obtain the localized energy values which will in turn generate the energy vectors. To calculate the localized energy for each pixel for each filtered output, different window sizes are used. To calculate the feature vectors, the energy is computed using the following formula:

$$\frac{1}{Window\ size * Window\ size} * [summation\ of\ all\ pixel\ values\ in\ each\ window\ for\ each\ filtered\ image]$$

This localized energy is computed for each filtered image. Hence, 9 energy arrays are formed after the energy localization procedure.

d) *Energy feature normalization:*

All kernels have a zero-mean except for *L3TL3*. The feature extracted by the filter *L3TL3* is not a useful feature for texture segmentation. Hence, it's energy is used to normalize all other features at each pixel.

e) K-means algorithm for texture segmentation

K-means algorithm is used to segment the input image using the 9-dimension energy vector for each pixel. As the data has to be clustered in 6 clusters, 6 centroids are required which are initialized in an arbitrary manner.

The K-means algorithm can be explained as follows:

Step 1: Place the 9-dimensional feature vector for each pixel in the feature space and initialize the K centroids required to cluster the images.
Step 2: Assigning each image to a particular cluster by finding the Euclidean distance of each image with the K centroids.
Step 3: When all the pixels have been assigned to a cluster, recalculate the centroids by considering the images in that cluster.
Step 4: Run this algorithm iteratively until K centroids do not change over one iteration.

After segmenting the image into textures, each texture is assigned a gray value. As there are six textures in the given image, six gray levels 0, 51, 102, 153, 204, 255 are used to denote six segmented regions in the output image.

Algorithm for Texture Segmentation:

Step 1: Read the contents of the input image "comb" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert 1D input image into a 1D image of double data format.
Step 3: Convert 1D double data format image into a 2D array as the image is comprised of a single channel.
Step 4: Calculate the tensor product of the 3 Laws filters to get nine 3x3 Laws filters.
Step 5: Remove the DC component of each image by subtracting the average of all pixels in the window from each center pixel. The window size used here is 21x21. Refer to the equation for DC removal of image.
Step 6: Convolve the DC removed input images with 9 Laws filters to obtain 9 filtered outputs. Before convolving the images, pixel replication is performed to extend the boundaries.

Step 7: Calculate the localized energy for each pixel of the Laws filtered output images. The neighborhood taken for calculation of localized energy is changed every time. This step will generate the nine energy arrays required for K-means clustering.
Step 8: Localize the energy of each pixel by dividing the value of each pixel in the 9 energy arrays by the L3L3 pixel energy value.
Step 9: Convert the 9 energy arrays into a format such that each pixel will have 9D feature values. Hence, the total number of feature vectors is (imageHeight*imageWidth) *9. These feature vectors are then used to implement the k-means clustering algorithm.
Step 8: K- means algorithm

Step 8a: Initialize the 6 centroids arbitrarily.

Step 8b: Calculate the Euclidean distance of each pixel with the 6 centroids and assign the label for that pixel depending on the minimum Euclidean distance.

Step 8c: Update the centroids by finding the average of the feature Values in each of the cluster.

Step 8d: Repeat the steps iteratively until the centroids don't change for one iteration.

Step 9: Assign gray levels to each cluster after the k- means algorithm and display the output image using a 1D array.

III. Experimental Results

```
E3E3 Laws filter:
0.25 -0 -0.25
-0 0 0
-0.25 0 0.25

E3S3 Laws filter:
0.25 -0.5 0.25
-0 0 -0
-0.25 0.5 -0.25

E3L3 Laws filter:
-0.0833333 -0.166667 -0.0833333
0 0 0
0.0833333 0.166667 0.0833333

S3E3 Laws filter:
0.25 -0 -0.25
-0.5 0 0.5
0.25 -0 -0.25

S3S3 Laws filter:
0.25 -0.5 0.25
-0.5 1 -0.5
0.25 -0.5 0.25

S3L3 Laws filter:
-0.0833333 -0.166667 -0.0833333
0.166667 0.333333 0.166667
-0.0833333 -0.166667 -0.0833333

L3E3 Laws filter:
-0.0833333 0 0.0833333
-0.166667 0 0.166667
-0.0833333 0 0.0833333

L3S3 Laws filter:
-0.0833333 0.166667 -0.0833333
-0.166667 0.333333 -0.166667
-0.0833333 0.166667 -0.0833333

L3L3 Laws filter:
0.0277778 0.0555556 0.0277778
0.0555556 0.111111 0.0555556
0.0277778 0.0555556 0.0277778
```

Figure 7: Nine 3x3 Laws filters

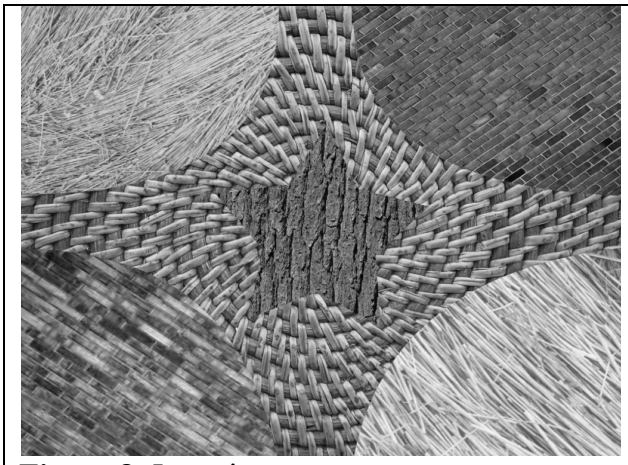


Figure 8: Input image

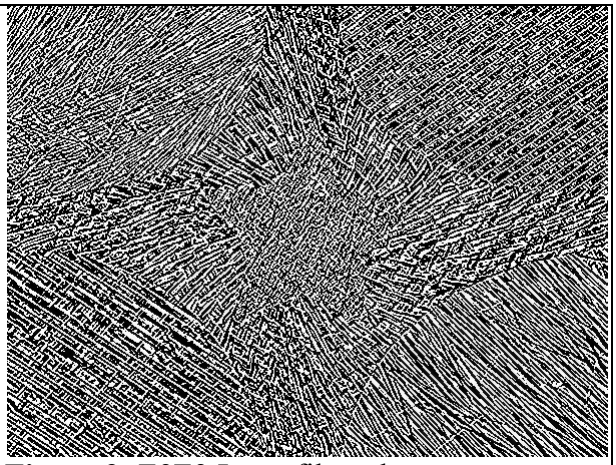


Figure 9: E3E3 Laws filtered output

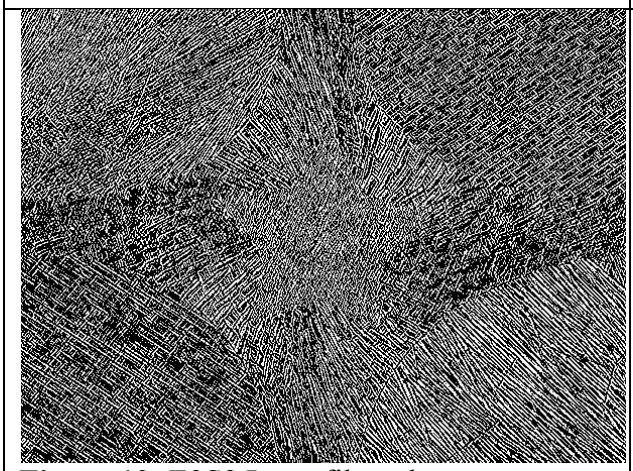


Figure 10: E3S3 Laws filtered output

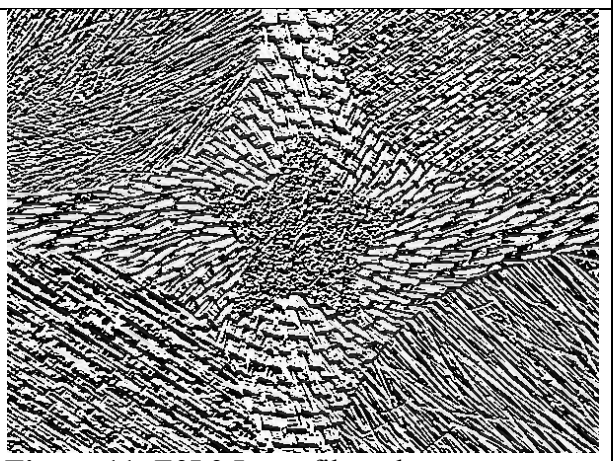


Figure 11: E3L3 Laws filtered output

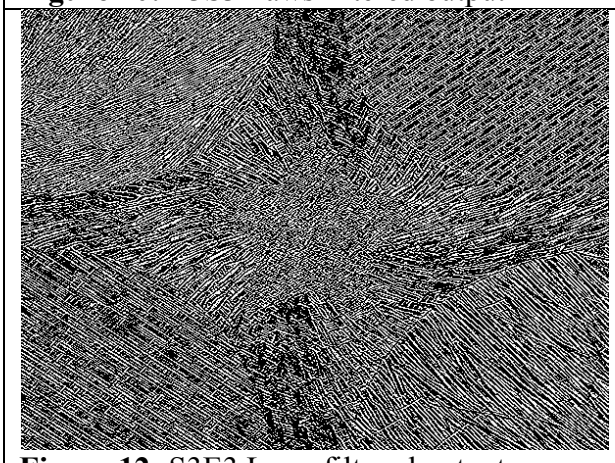


Figure 12: S3E3 Laws filtered output

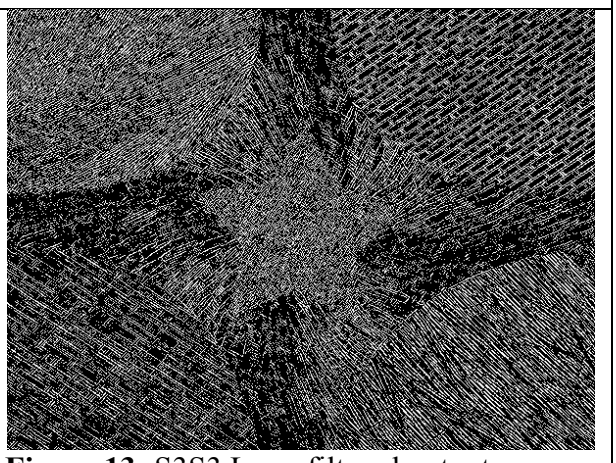


Figure 13: S3S3 Laws filtered output

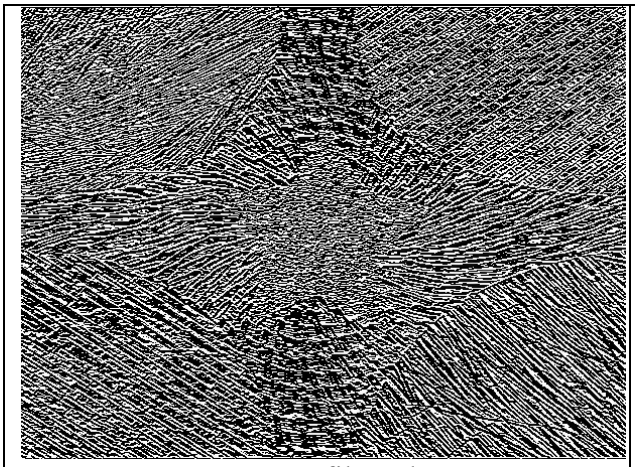


Figure 14: S3L3 Laws filtered output

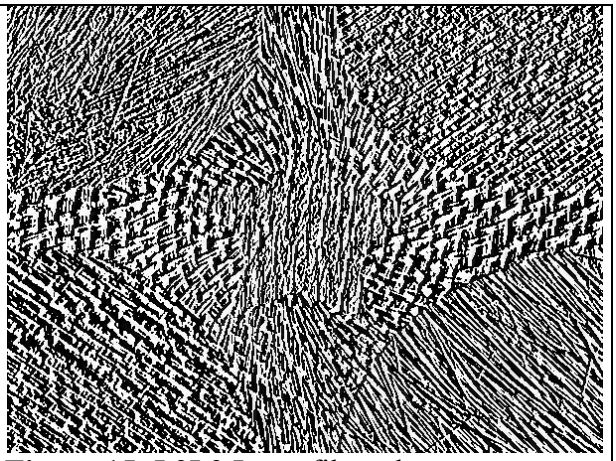


Figure 15: L3L3 Laws filtered output

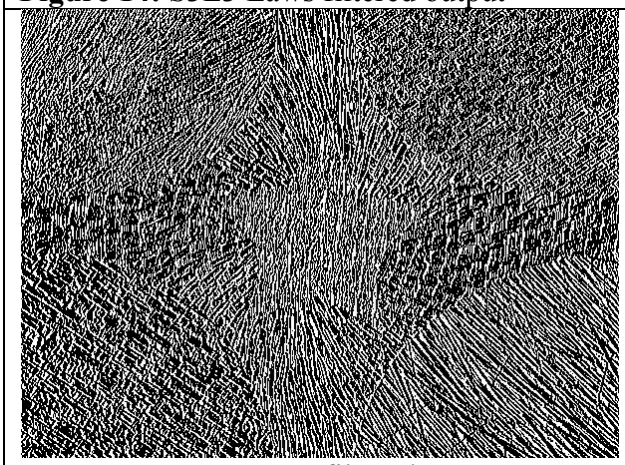


Figure 16: L3S3 Laws filtered output

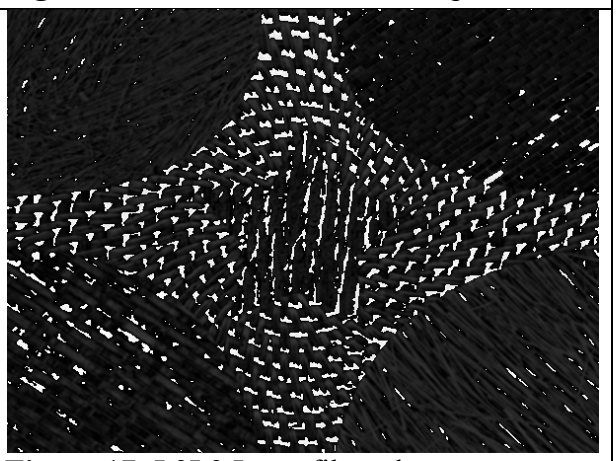
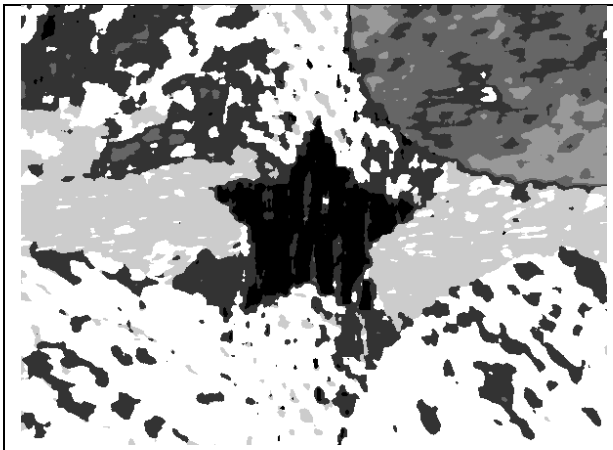
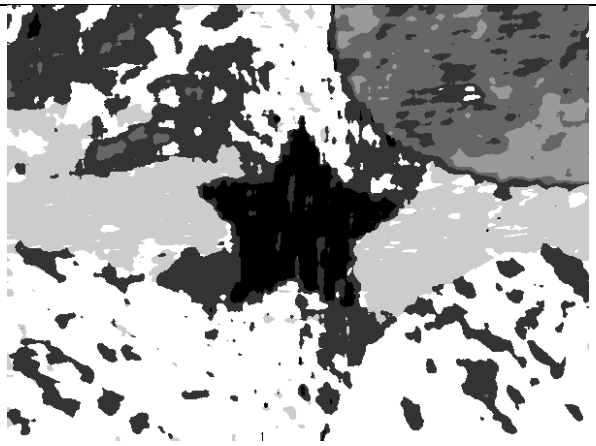


Figure 17: L3L3 Laws filtered output

Laws filter outputs



Window size=15



Window size=17

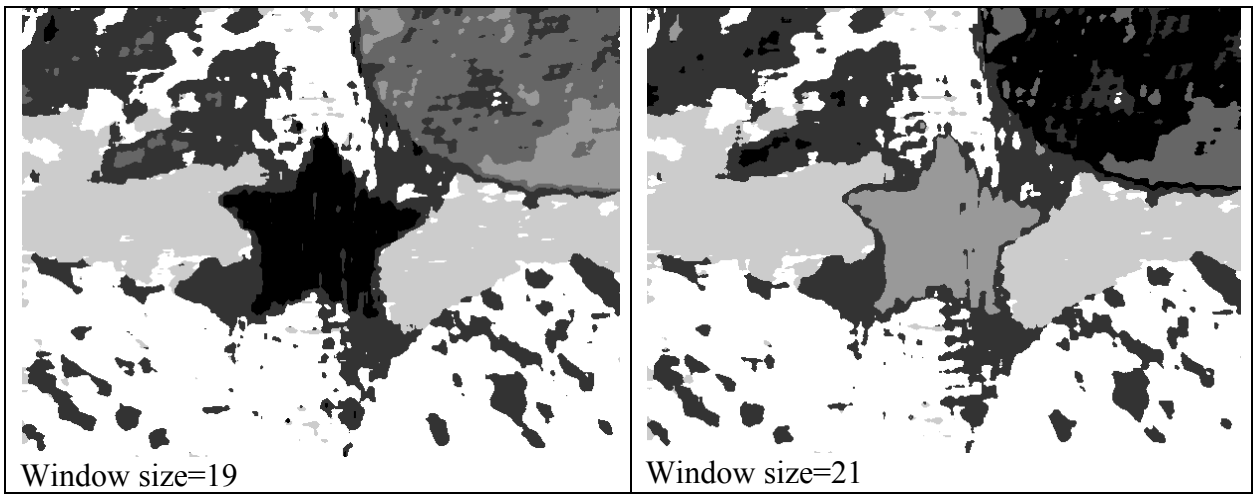


Figure 18: Normalization with L3L3 outputs for different window sizes

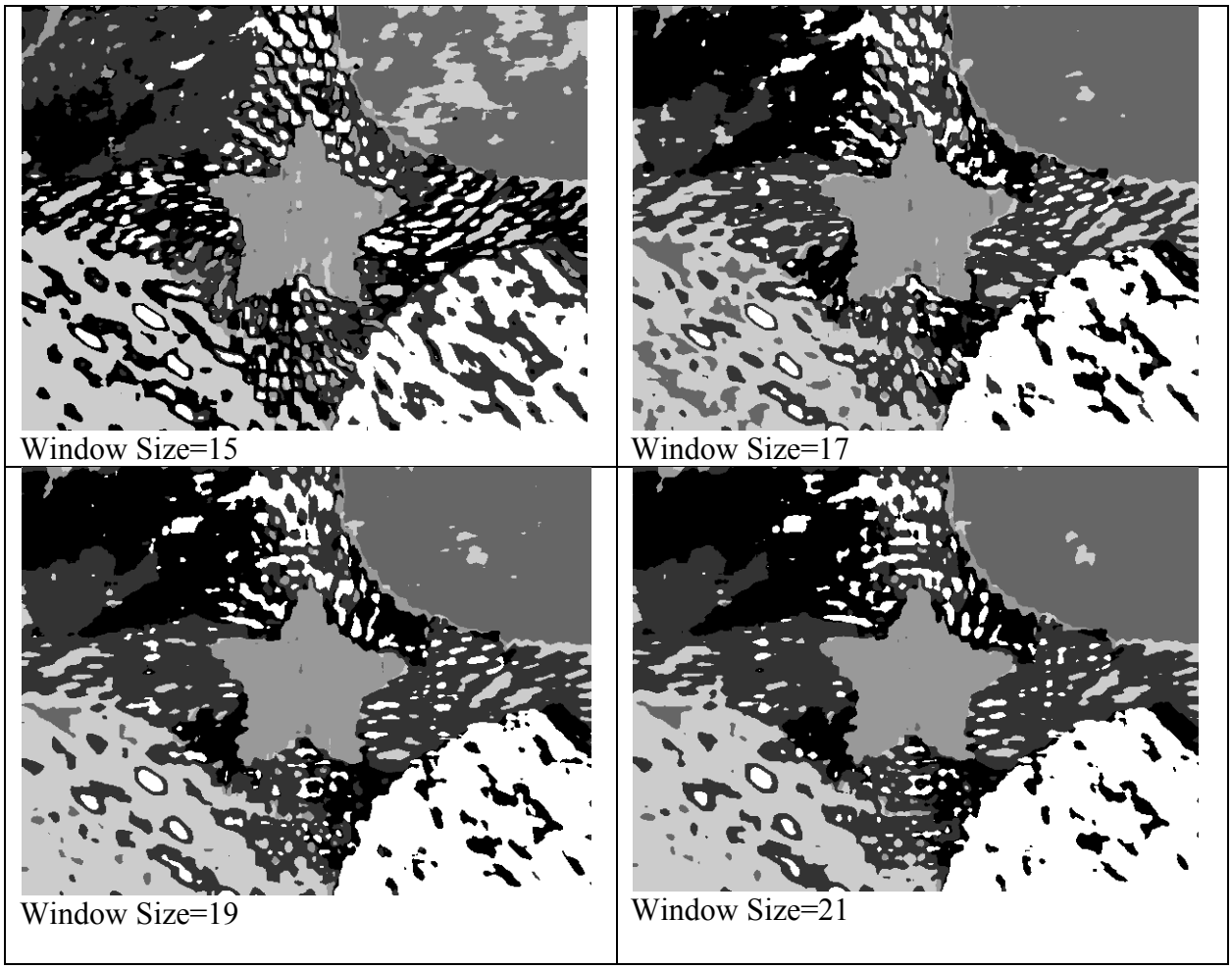


Figure 19: Un-normalized outputs for different window sizes

IV. Discussion

The segmentation of image was done by using the Laws filters for filtering. The normalization of all the features with respect to the L3L3 filter was done as the L3L3 filter has zero mean and is not a useful feature for classification.

The normalized and un-normalized outputs have been shown for different window sizes. The un-normalized output looks better than the normalized output as the segmentation is proper in that. 6 segmented regions can be observed in un-normalized outputs unlike normalized outputs which have 5 visually distinct regions.

By changing the window sizes, I deduced that as the size of the window increases, the segmentation quality increases. But once an appropriate window size is reached, the segmentation is not affected much. This can be observed in outputs of window sizes 19 and 21 as the segmentation result is almost the same in both of them.

According to me, the best window size is 21 as the segmented output for this window size is better in terms of its clarity.

c) Advanced- Dimension Reduction

I. Abstract and Motivation

Dimension Reduction plays an important role in Image Retrieval and Computer Vision Applications. In many image classification systems, the low-level features take a precedence whereas the high-level features which contain more information are considered at a later stage. That is why, high dimensional features are transformed to low dimensional features without losing important information. Principal Component Analysis is a method used for dimensionality reduction. In PCA, the higher dimensional data is reduced to lower dimensional data in such a way that the maximum variability in the data is reproduced.

The concept used in PCA is that the Eigen values and Eigen vectors for a high dimensional data are transformed to low dimensional data by finding the n largest Eigen values and finding the corresponding Eigen vectors for them. The value chosen by me is n=5. The 25D dataset is converted into a 5D dataset by using inbuilt MATLAB function for PCA.

II. Approach and Procedure

All the steps mentioned for Texture segmentation are used in PCA except that five 1D Laws filters are used instead of 3.
The Laws filters used are given below:

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]

These filters are divided by their absolute vales and the resultant filters are used to convolve the images.

$$L5 = \frac{1}{16} * [1\ 4\ 6\ 4\ 1] \qquad E5 = \frac{1}{6} * [-1\ -2\ 0\ 2\ 1] \qquad S5 = \frac{1}{4} * [-1\ 0\ 2\ 0\ -1]$$

$$W5 = \frac{1}{6} * [-1\ 2\ 0\ -2\ 1] \qquad R5 = \frac{1}{16} * [1\ -4\ 6\ -4\ 1]$$

This is how 25 5x5 Laws filters are generated and they are convolved with the DC removed input images.

Algorithm for Texture Segmentation using PCA:

Step 1: Read the contents of the input image "comb" into a 1D array whose dimensions are specified by imageHeight, imageWidth, bytesperpixel.
Step2: Convert 1D input image into a 1D image of double data format.
Step 3: Convert 1D double data format image into 2D arrays as the image is comprised of a single channel.
Step 4: Calculate the tensor product of the 5 Laws filters to get 25 5x5 Laws filters.
Step 5: Remove the DC component of each image by subtracting the average of all pixels in the window from each center pixel. The window size used here is 21x21. Refer to the equation for DC removal of image.
Step 6: Convolve the DC removed input images with 25 Laws filters to obtain 25 filtered outputs. Before convolving the images, pixel replication is performed to extend the boundaries.
Step 7: Calculate the localized energy for each pixel of the Laws filtered output images. The neighborhood taken for calculation of localized energy is changed every time. This step will generate the 25 energy arrays required for K-means clustering.
Step 8: Localize the energy of each pixel by dividing the value of each pixel in the 25 energy arrays by the L3L3 pixel energy value.
Step 9: Convert the 25 energy arrays into a format such that each pixel will have 25D feature values. Hence, the total number of feature vectors is (imageHeight*imageWidth) *25. These feature vectors are then used to implement the k-means clustering algorithm.
Step 10: Save the final Feature vector array into a text file and read it in Matlab.
Step 10a: Apply the inbuilt PCA function in Matlab to reduce the dimension size from 25 to 5. Finally, a 5D feature vector is created and exported from Matlab to C++.
Step 11: After reading the dimension reduced feature vector, K- means algorithm is applied
Step 11a: Initialize the 6 centroids arbitrarily.
Step 11b: Calculate the Euclidean distance of each pixel with the 6 centroids and assign the label for that pixel depending on the minimum Euclidean distance.

Step 11c: Update the centroids by finding the average of the feature Values in each of the cluster.

Step 11d: Repeat the steps iteratively until the centroids don't change for one iteration.

Step 12: Assign gray levels to each cluster after the k- means algorithm and display the output image using a 1D array.

III. Experimental Results

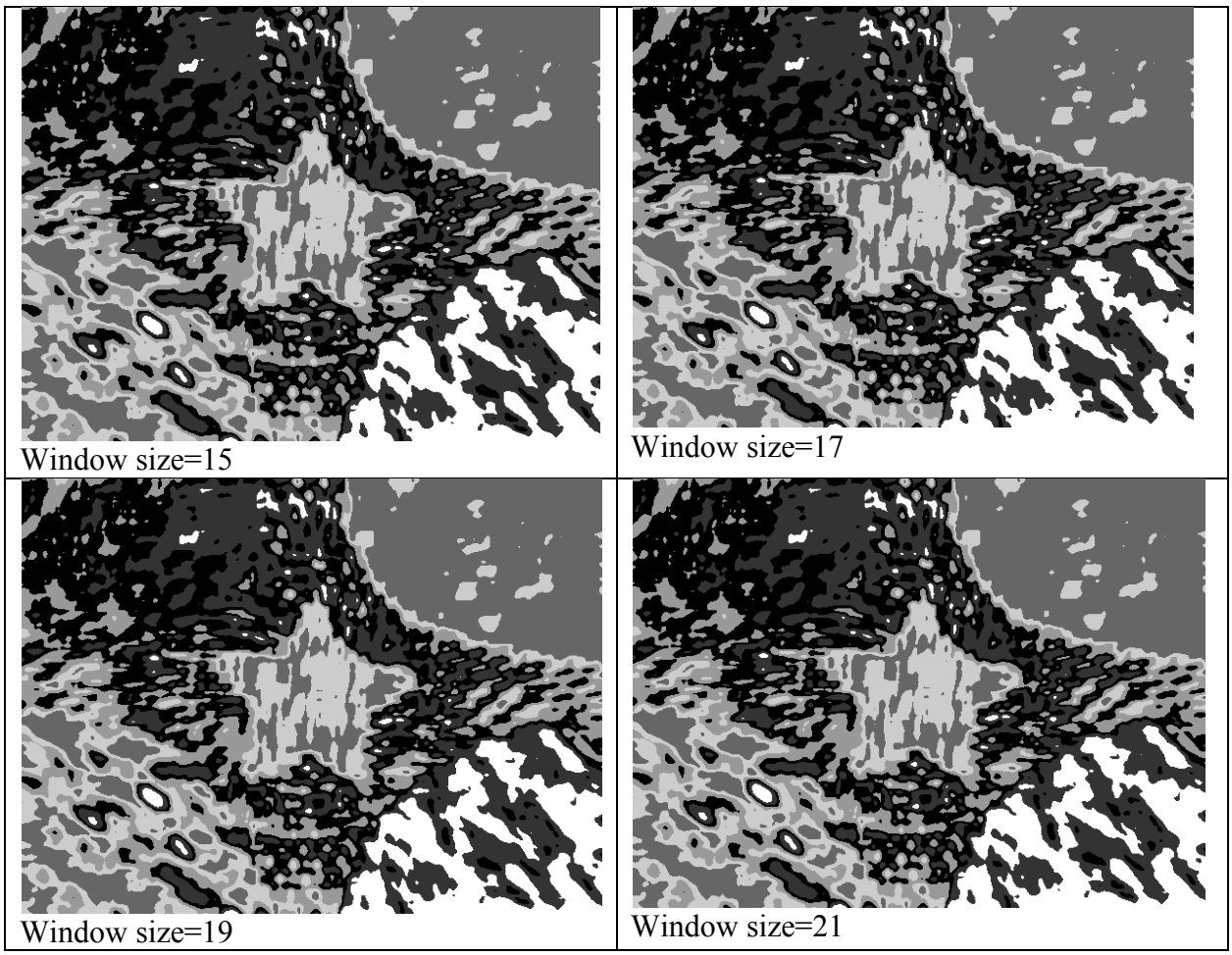


Figure 20: Un-normalized outputs for different window sizes using PCA

IV. Discussion

The PCA dimension reduction algorithm is applied on the texture segmentation algorithm and the following conclusions are made:

By changing the window sizes, I deduced that as the size of the window increases, the segmentation quality increases. But once an appropriate window size is reached, the segmentation is not affected much. According to me, the best window size is 21 as the segmented output for this window size is better in terms of its clarity.

I also deduced that dimension reduction does not prove to be extremely useful as the outputs obtained after dimensionality reduction through PCA are bad compared to the original outputs. May be some other form of pre-processing techniques like denoising or post-processing techniques like removal of small regions have to be performed to get a better output.
To get a better output compared to PCA I implemented the graph cut method, but it gave me memory problems.

Problem 2: Edge Detection

a) Basic Edge Detector

Edges are one of the most important attributes of an image. They describe the structure of an image. Edges are the points where there is a sudden change of intensity between pixels which is due to either a discontinuity in the intensity of pixels or due to change in first derivative of image intensity. However, due to smoothening of images while capturing them, sudden discontinuities have been replaced by change in intensities over finite distances.

As edge detection is used to detect local changes in intensity within an image, gradient is a good measure to detect these changes. There are various methods involved in edge detection:

i) Filtering: As changes in intensity of an image are susceptible to noise, filtering is used to enhance the performance of an edge detector in presence of noise.
ii) Enhancement: In order to detect edges effectively it is important to take into account the neighborhood of an image. This is done by considering the gradient magnitude of an image.
iii) Detection: Many points in an image can have non-zero values for the gradient irrespective of them being edges or not. Therefore, thresholding method is used to detect the strongest edges in the image.

Edge detection has a lot of applications in image processing, image analysis and computer vision. It is considered to be the first step in image analysis or understanding.

A) Sobel Detector

I. Abstract and Motivation

Sobel detector calculates the 2D spatial gradient changes in an image by emphasizing the high spatial frequency contents of the image corresponding to the edges. It is basically used to calculate the magnitude of absolute gradient of pixels in the image in horizontal and vertical direction.

II. Approach and Procedure

There are two filters used to calculate the gradient of the image in the horizontal and vertical direction respectively. These are 3x3 masks in which one mask is 90 degrees rotated version of the other. The two masks used to compute gradient in horizontal and vertical direction are given as follows:

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

Gx

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

Gy

These masks are applied separately to measure the gradients in horizontal and vertical direction and are later combined to produce magnitude gradient and orientation of the gradient for each pixel.
The gradient magnitude is given below:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

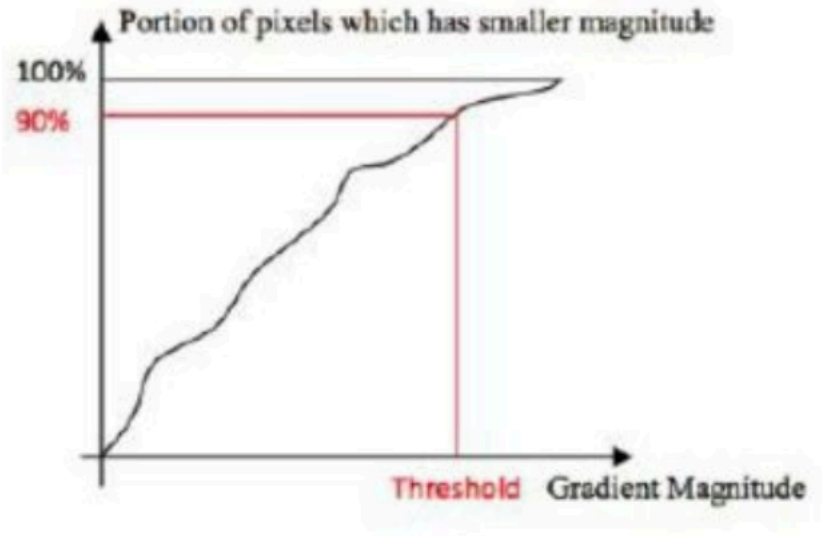
The orientation of edges relative to the pixels is given by:

$$\theta = \arctan(Gy/Gx)$$

As the image is a color image, it has to be converted to grayscale by using the luminosity method. The formula to convert a colored image to gray scale image is as given below:

Gray scale image= 0.21R component+0.72G component+0.07B component

After calculating the magnitude of gradient of the image an edge map is generated which is a binary image consisting of either 0(edge) or 255(background values). This edge map is prepared by observing the histogram of the intensity values and deciding the threshold. The threshold is decided in such a way that 90% pixels of the cumulative probability graph are considered as edges. The following plot gives a better understanding of how the threshold is set.



Task: To apply Sobel detector on two images and generating the best edge map for the image.

Algorithm for Sobel edge Detection:

Step 1: Read the input raw image 'Boat.raw" and "Noisy_Boat.raw" from a file into a 1D array whose dimensions are specified by image height, image width and Bytesperpixel for the image.
Step 2: Convert the 1D array into separate 1D arrays for R, G, B channels respectively.

Step 3: Convert the RGB image into a gray scale image by using the luminosity method formula mentioned above.

Step 4: Apply the Gx ,Gy kernels mentioned above for each pixel and find the magnitude of the image by using the magnitude formula mentioned above.

Step 5: As the values of the absolute magnitude as well as the Gx, Gy gradient magnitudes are quite high, they have to be normalized in the range of 0-255.To do this the max and minimum magnitude values for gradient have to be found out. The following formula is used to normalization:

$$Normalized\ image = \frac{input\ image - min}{max - min}$$

Step 6: The normalize array is sorted in increasing order of the intensities and the indexes of this sorted normalized array is stored. A cumulative distribution plot of this array is plotted in Matlab.

Step 7: As we have to include 10% of the pixels in the cdf plot to decide the threshold, the intensity value corresponding to the pixel threshold is found out.

Step 8: After deciding the threshold in Step 7, threshold the input image as follows:

Output image=0 if input image value> threshold

=255 if input image value< threshold

Step 9: Combine the thresholded output image into a 1D array used for displaying the edge map.

Step 10: Write the content of intensity values in a text file which can be read in Matlab to plot the histogram.

III. Experimental Results

Figure 21: Original Boat image

Figure 22: Original Noisy Boat image

Figure 23: Grayscale Boat image

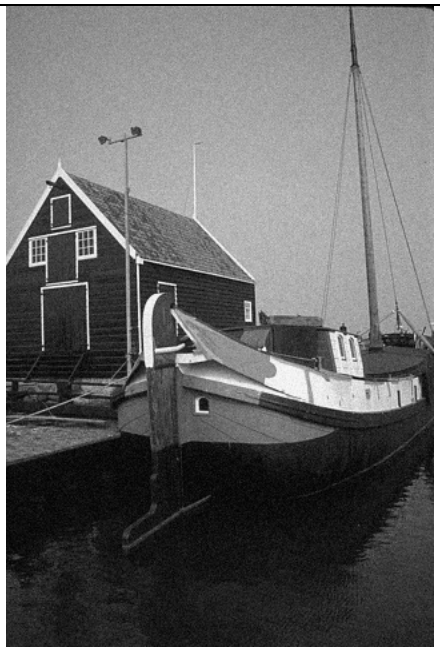


Figure 24: Grayscale Noisy Boat image

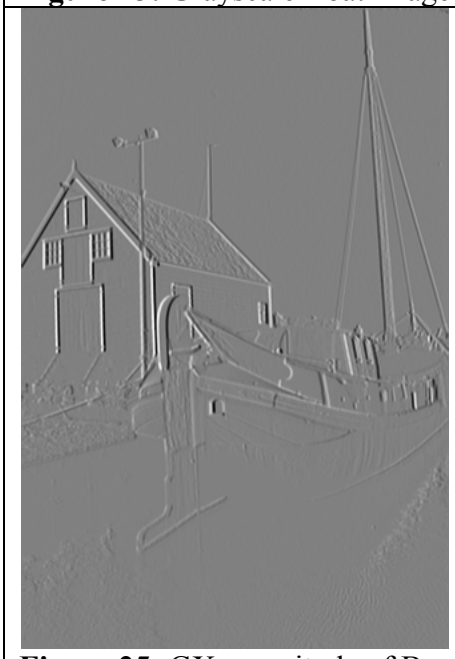


Figure 25: GX magnitude of Boat image

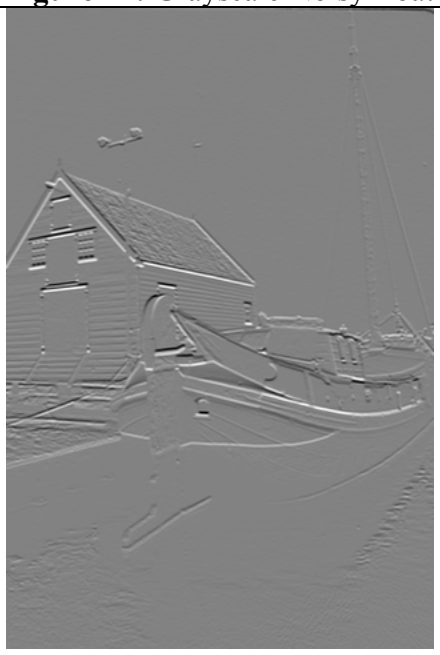


Figure 26: GY magnitude of Boat image

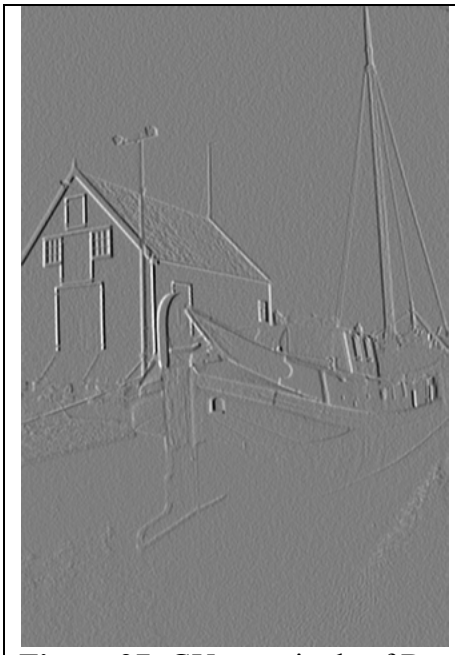


Figure 27: GY magnitude of Boat image

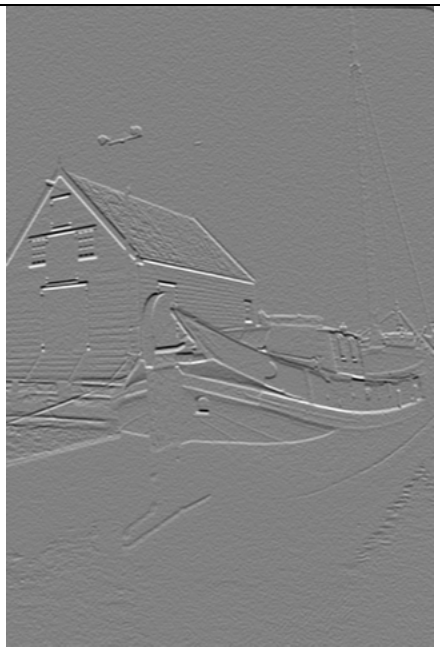


Figure 28: GY magnitude of Noisy Boat image

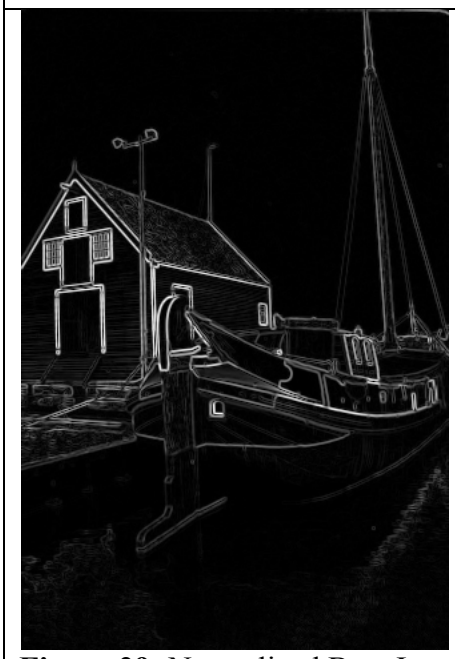


Figure 29: Normalized Boat Image

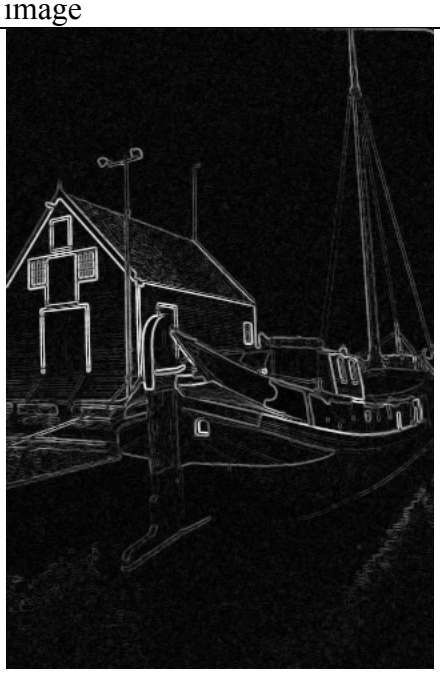


Figure 30: Normalized Noisy Boat Image

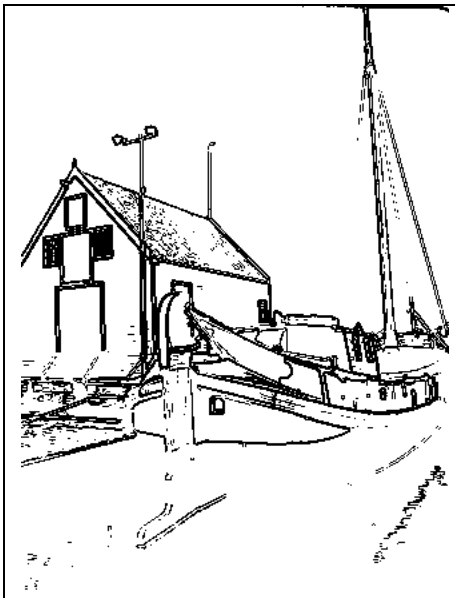


Figure 31: Final Edge map of Sobel detector for Boat image

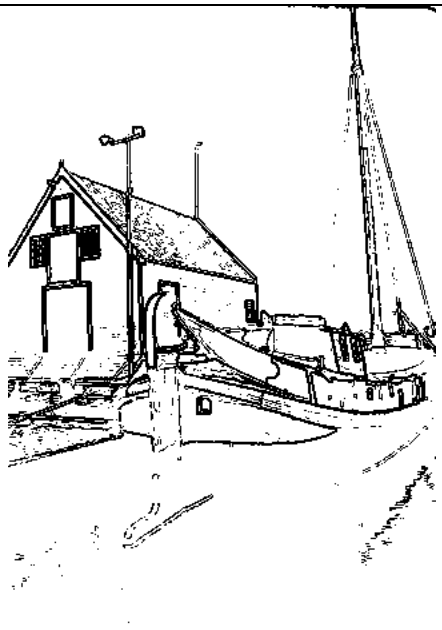


Figure 32: Final Edge map of Sobel detector for Noisy Boat image

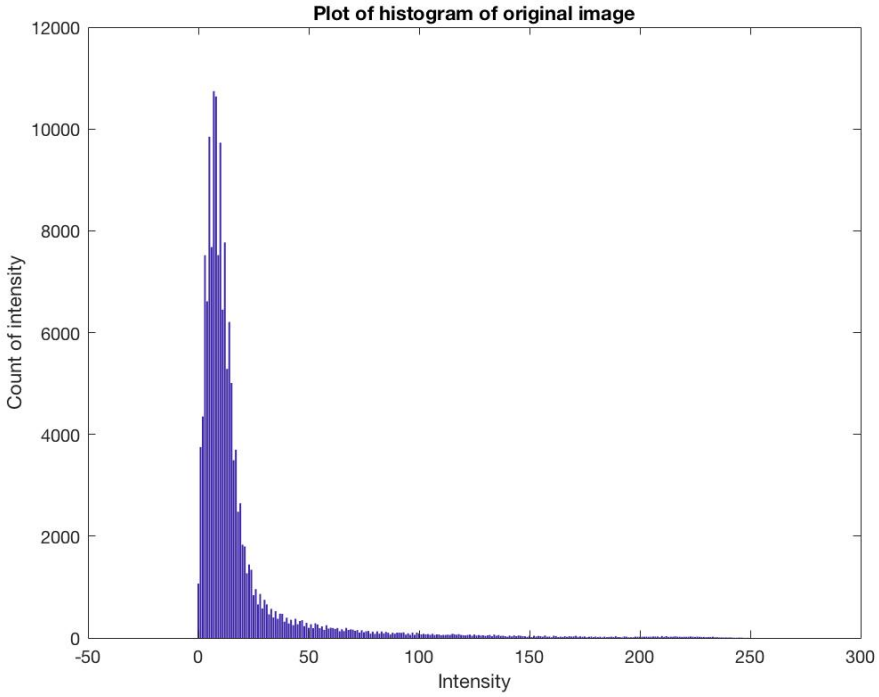


Figure 33: Histogram of intensity values for Sobel Detector

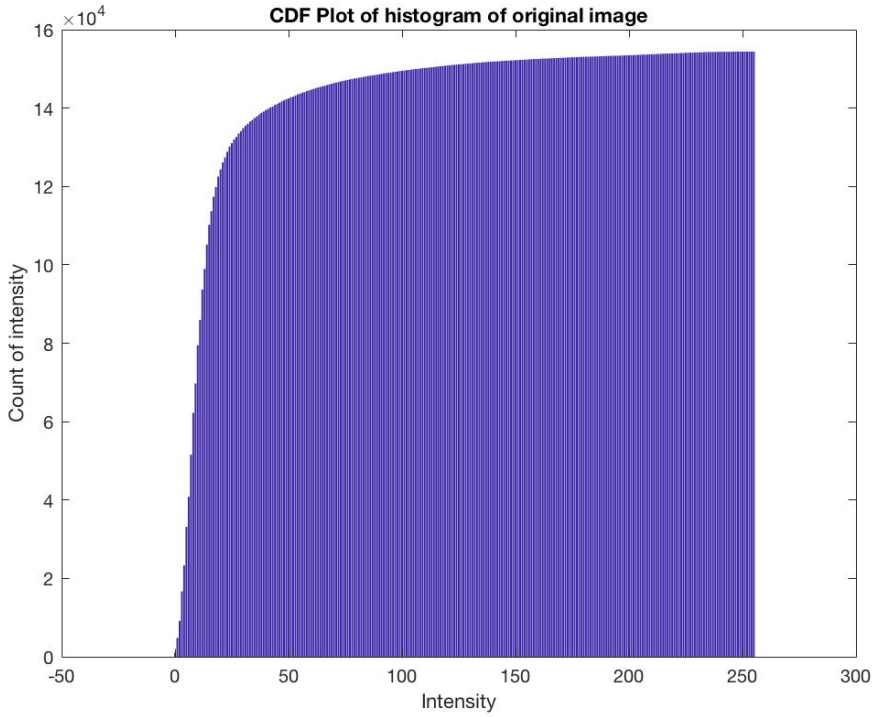


Figure 34: Cumulative plot of intensity values for Sobel Detector

IV. Discussion

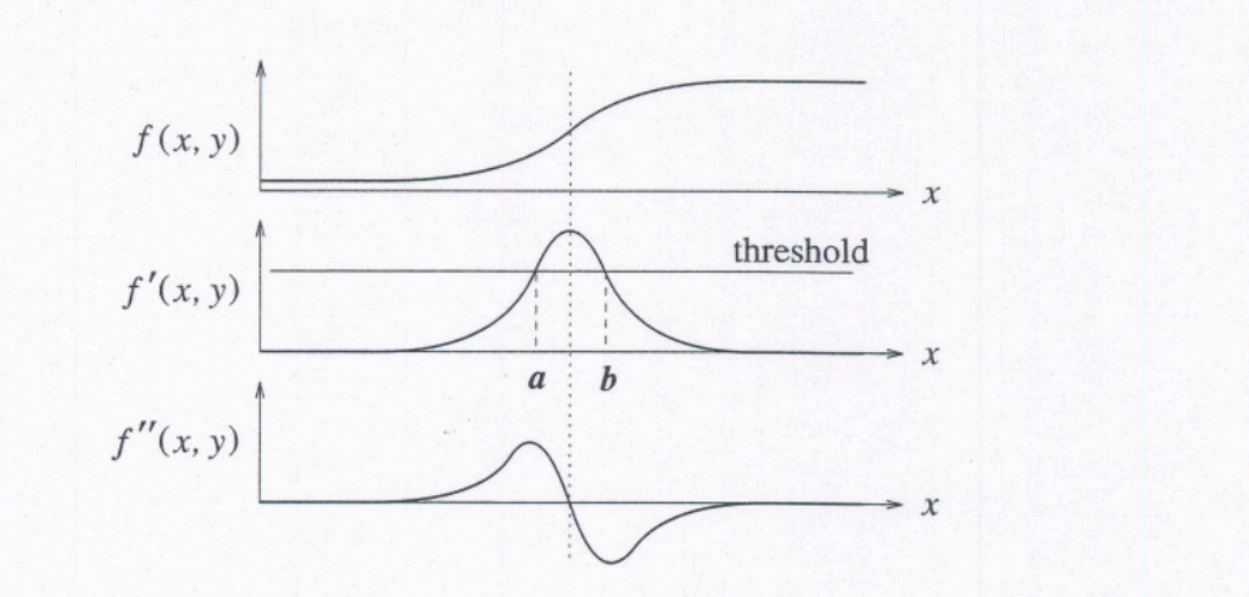
- The Sobel filter was used to calculate the change of gradient in x and y directions and this is how the absolute magnitude was obtained. The cumulative plot in figure 31 helped in deciding the threshold intensity value used to prepare the edge map.
- As seen from the histogram of intensities in figure 33, the Sobel detector has one knee point that is why one threshold is required to prepare the edge map.
- Sobel Detector is very susceptible to noise and hence noise filtering has to be done before applying the Sobel mask. Also, to detect proper location of edges Non-Maximum suppression is done after Sobel detection which can improve the result greatly.

B) Zero crossing Detector

I. Abstract and Motivation

In the Sobel detector, the first derivative of the image was computed. But this leads to a detection of a lot of edges which can be false edges as well. Hence, the Second derivative operator is used. The first derivative maximum corresponds to a zero crossing in the second derivative. Hence, zero crossings are monitored while using a second derivative operator.

As second derivative operators are susceptible to noise just like first derivative operators, noise removal operation has to be performed before taking the second derivative. Therefore, the second derivative operation i.e. finding the location of an edge is being done by the laplacian filter and the noise removal part is being done by the Gaussian filter. LOG is rotationally invariant as it just considers the magnitude, it does not take into account the direction of the edge.



The above figure gives a brief idea about how a ramp edge gets converted into a bell-shaped curve having edge at the maxima after taking the first derivative. This curve then gets transformed into a zero-crossing step function after taking the second derivative. As finding maxima is difficult, zero crossings are monitored for a second derivative filter.

II. Approach and Procedure

The Laplacian of an image is given as follows:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

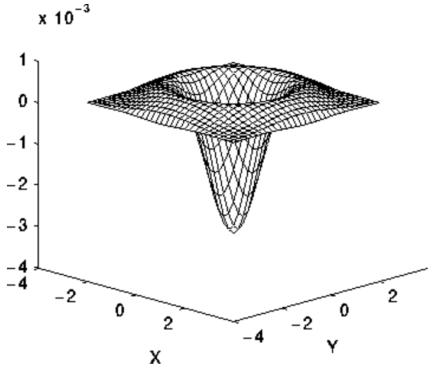
Where I denotes the intensity of the pixel of interest.

Instead of convolving the Gaussian filtered image with the laplacian operator, both the operations can be combined into one which will reduce the computation time. Hence, a hybrid filter of the Laplacian and Gaussian i.e. LOG is being used to convolve with the input images.

The 2D Gaussian having center zero and standard deviation σ is given as:

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

This is how the 2D LOG function looks like:



The 2D LOG filter used in this question is as follows:

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Figure 35: LOG filter with σ=1.4.

As the image is a color image, it has to be converted to grayscale by using the luminosity method. The formula to convert a colored image to gray scale image is as given below:

Gray scale image= 0.21R component+0.72G component+0.07B component

After calculating the LOG filtered image an edge map is generated which has intensity values between 0 to 255. As the histogram obtained from the LOG filtered image is a spike, two knee points have to be determined to convert the image into values -1, 0, 1. Then zero crossing detection is applied on this image to get the best edge map consisting of values 0(edge) and 255(background).

Task: To apply zero-crossing detector on two images and generating the best edge map for the image.

Algorithm for zero crossing edge Detection:

Step 1: Read the input raw image 'Boat.raw" and "Noisy_Boat.raw" from a file into a 1D array whose dimensions are specified by image height, image width and Bytesperpixel for the image.
Step 2: Convert the 1D array into separate 1D arrays for R, G, B channels respectively.

Step 3: Convert the RGB image into a gray scale image by using the luminosity method formula mentioned above.

Step 4: Apply the LOG kernel mentioned in figure 32 for each pixel and find the LOG filtered image.

Step 5: As the intensity values of the LOG filtered image are quite high, they have to be normalized in the range of 0-255.To do this the max and minimum magnitude values in the filtered image have to be found out. The following formula is used to normalization:

$$\text{Normalized image} = \frac{\text{input image} - \min}{\max - \min}$$

Step 6: The histogram of the normalized intensity values is plotted in Matlab to get the two knee points from this histogram.

Step 7: The normalized image is thresholded based on these two knee points.

If intensity values<knee point 1=output is -1

If intensity values>knee point 1 and intensity values<knee point 2=output is 0

If intensity values>knee point 1=output is 1

The output image is also thresholded to values 64,128,192 respectively.

Step 9: The thresholded image obtained in step 7 consisting of values between -1 and 1 is then applied to a zero-crossing algorithm wherein if the center pixel is 0 and the neighboring pixels in the 3x3 mask are -1 or 1, then it is detected as an edge and a value of 0 is assigned to it.

Else a value of 255 is assigned to the image. This is how binary thresholding is done on the image.

Step 10: Combine the binary thresholded output image into a 1D array sued for displaying the edge map.

III. Experimental Results

Figure 36: Original Boat Image

Figure 37: Original Noisy Boat Image

Figure 38: Gray Level Boat Image

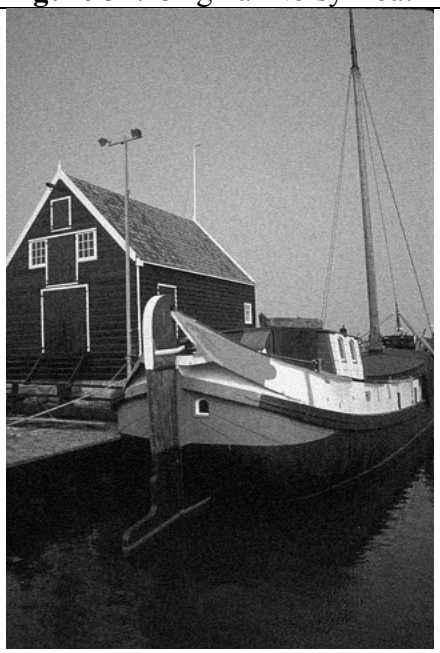


Figure 39: Gray level Noisy Boat Image

Figure 40: Normalized Boat image after LOG filtering

Figure 41: Normalized Noisy Boat image after LOG filtering

Figure 42: Ternary Thresholded Boat Image

Figure 43: Ternary Thresholded Noisy Boat Image

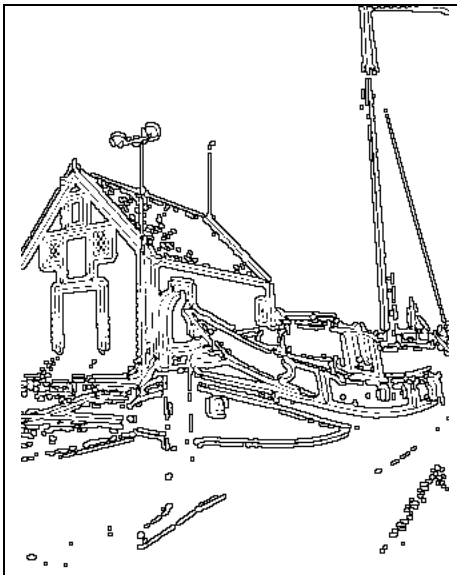


Figure 44: Final Edge map of zero crossing detector for Boat image

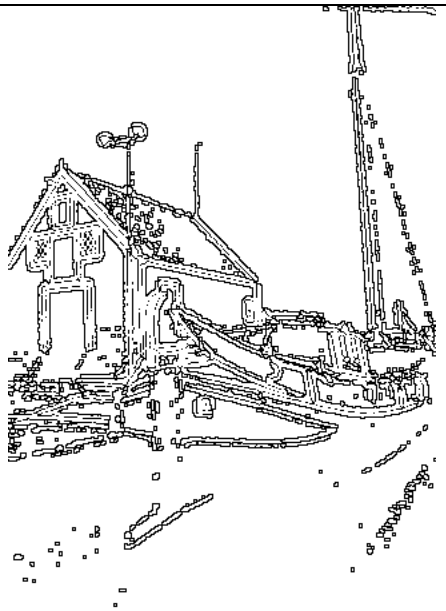


Figure 45: Final Edge map of zero crossing detector for Noisy Boat image

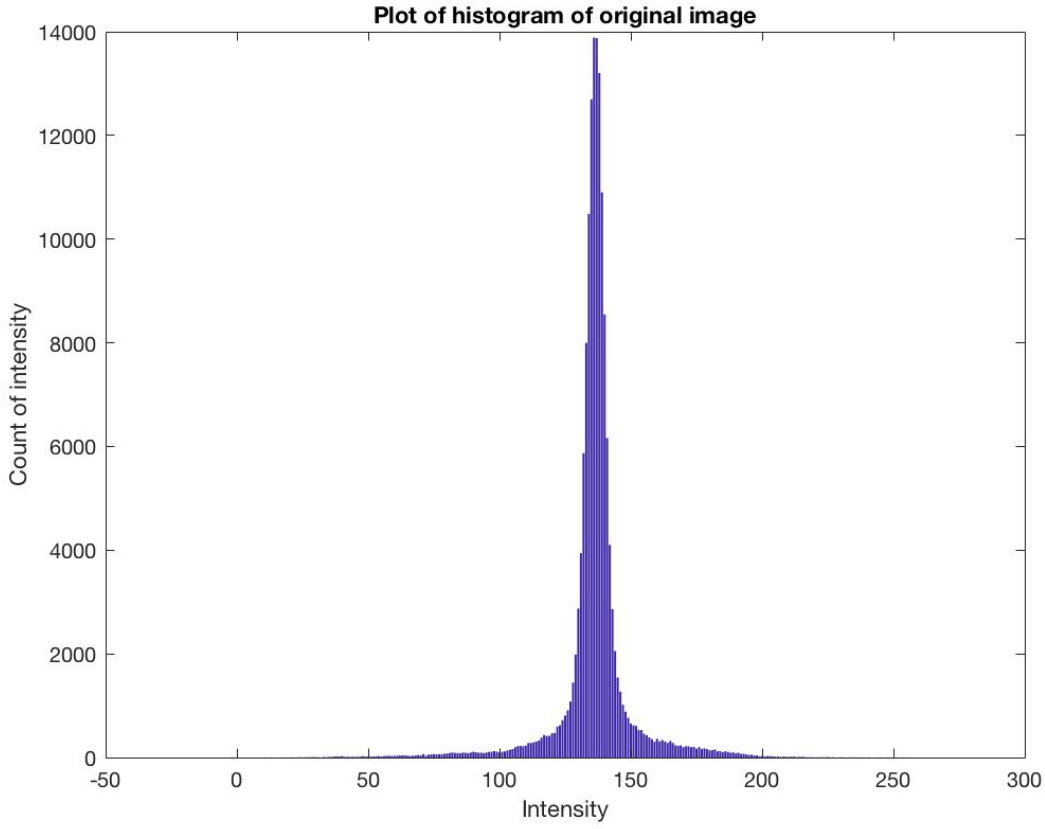


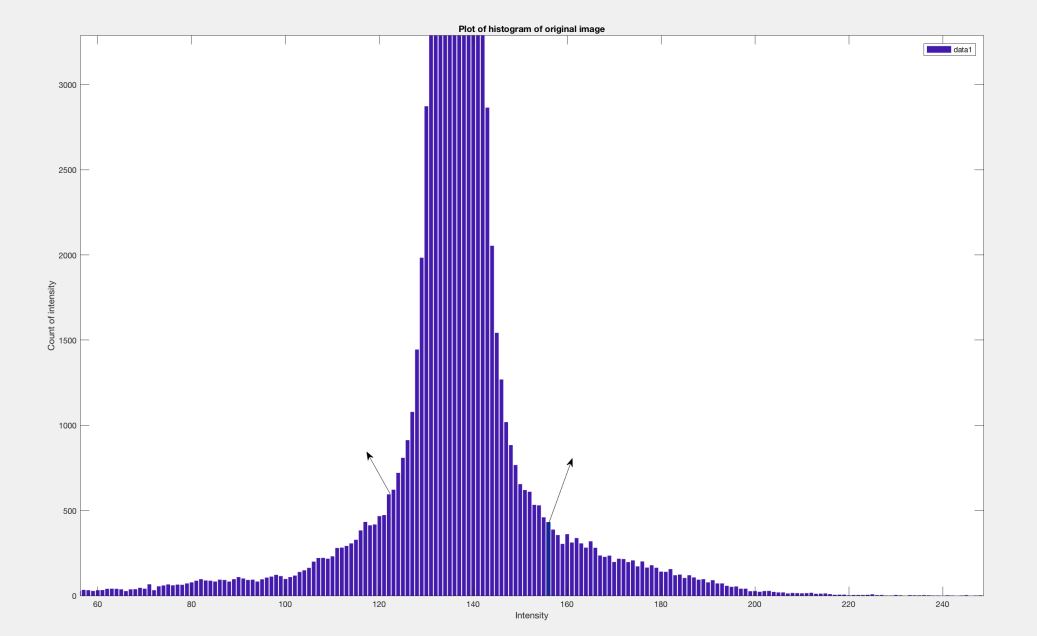
Figure 46: Histogram of intensity values of LOG filtered image

IV. Discussion

The zero-crossing detector is used to get the edges in the image. This filter has a better localization of the images, however, sometimes detects false images in the image. To threshold a given image, knee points have to be determined from the histogram of the filtered image.

Algorithm for Knee point selection:

As the obtained histogram is a symmetric histogram having 2 halves of the histogram from the center which are mirror images of each other, knee points can be determined by selecting intensity values at which the histogram intensity values start increases toward the spike. As the center of the histogram obtained in Figure 43 has center at 139, the knee point thresholds are chosen to be 122 and 156 respectively, which are equidistant from the center. As the histogram values start increasing at intensity value 122 and the sharp decrease ends at intensity value 156, those values are decided to be the thresholds.



If we observe closely at the outputs from Sobel and LOG, the outputs from LOG looks much better compared to Sobel outputs as more edges are detected in LOG especially near the roof and water. Also, if we closely observe the noisy image outputs and compare them with the Original image outputs it can be deduced that the LOG algorithm works better for noisy images as well. Some edges in the water parts of the original image were not detected in the non-noisy image but were detected in the noisy image. This proves that the LOG algorithm is more efficient in detecting the edges compared to the Sobel detection algorithm.

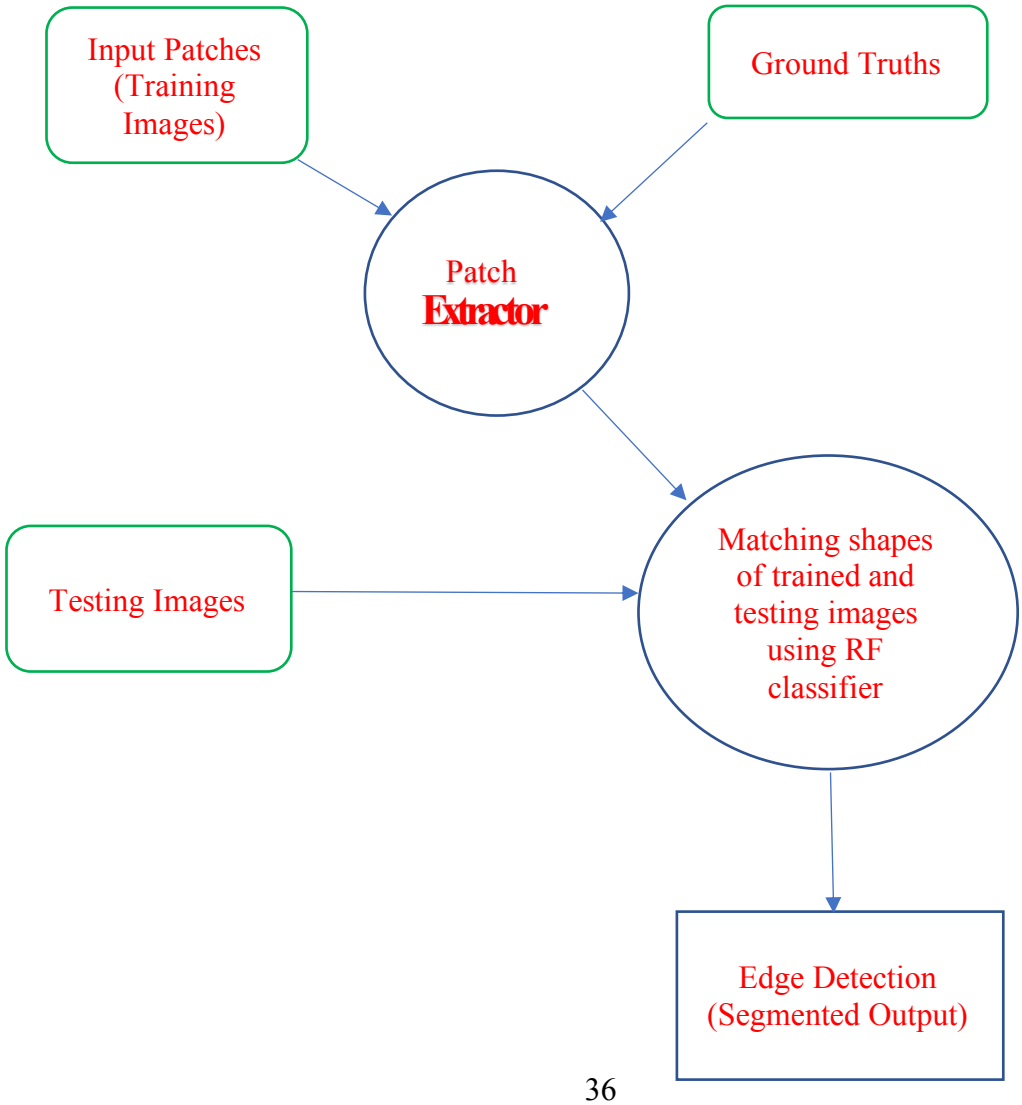
b) Structured Edge Detector [1]

I. Abstract and Motivation

As mentioned earlier, edge detection is a very important in field of computer vision, object recognition, image segmentation etc. Structured Edge Detection takes into account the structure of the edge rather than its location. Object detection with respect to background is done very effectively using a Structured edge. Therefore, a neighborhood of the pixel of interest is considered while labeling it as an edge. As neighboring pixels of an edge have similar characteristics, it's easier to detect edges if patches are considered. Due to similarity in patches, structured learning can be used as it works better for objects having similar characteristics. It is observed that used structured edge approach which generates edge map is computationally efficient.

II. Approach and Procedure

The flow chart for Structure Edge Detector algorithm is as given below:



Structured Forest algorithm is used to detect edges in an object using the Random Forest Classifier. The Random Forest Classifier takes input training images and ground truths to construct a forest having multiple decision trees. As the decision trees in the forest have high variability, they tend to overfit the data. Hence, an ensemble of the decision trees is used wherein the prediction of each decision tree is combined to generate the final output.

Structure edge detector considers the neighborhood of a pixel to detect the presence or absence of an edge. The edge patches formed from the patch extractor are classified into sketch tokens to take into consideration all the variability of different edge patterns of the input patches.

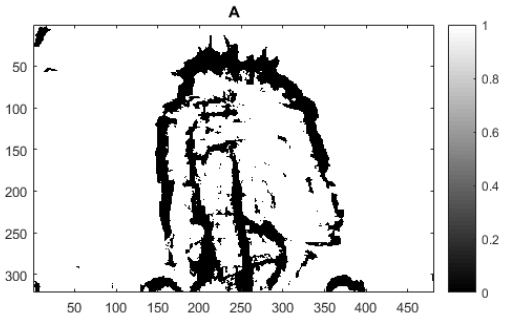
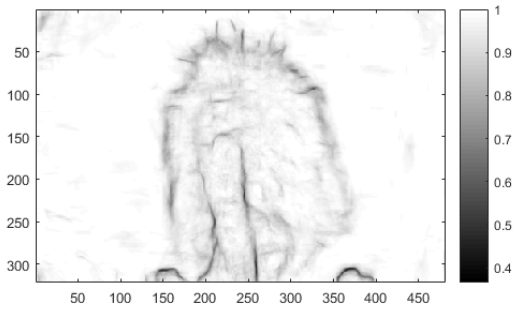
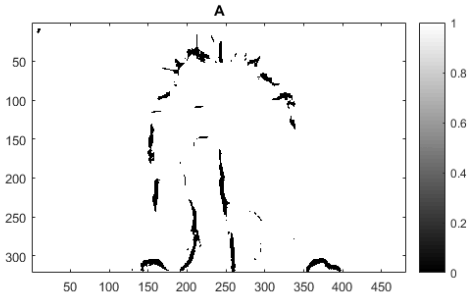
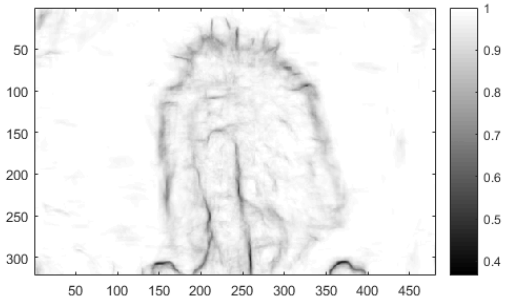
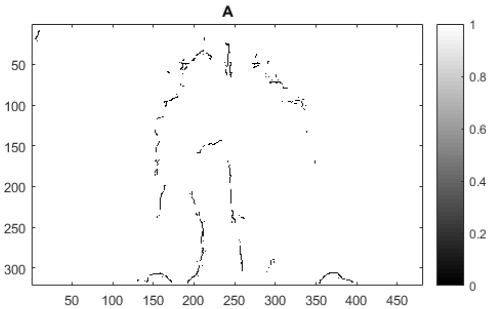
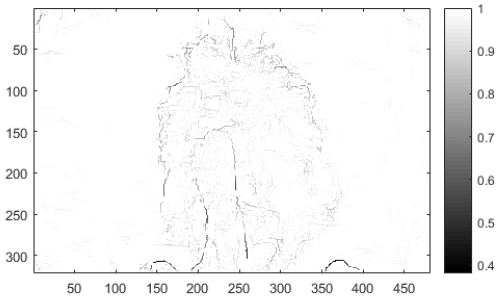
After a considerable amount of decision trees are added in the Random Forest Classifier, the testing images are given to the classifier. Random Forest uses the divide and conquer algorithm where it classifies the input by moving to the left or right of the current node and stops after reaching the leaf node. Each node in the tree has a split function $h(x, \theta_j) \in \{0, 1\}$ where if $h(x, \theta_j) = 0$ node j sends input to the left, else it sends it to the right.

A set of decision trees are trained in such a way that they give optimal split of data. The Random Forest Classifier matches the testing image patches to the trained classifier to make a decision about the presence of an edge. This algorithm works pretty well to give the correct edges describing an object.

III. Experimental Results

I varied the values of parameters and have displayed the changes in output.
The parameters changed in the testing part are Threshold value, Multiscale value, Non-Maximal Suppression value. The Number of threads, trees and Sharpen value are kept constant as they increase or decrease the speed. They don't affect the visual quality of the output.

Figure 47: Original Animal Image

Binary Edge Map	**Probability Edge Map**
	
Threshold=0.07, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0	Threshold=0.07, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0
	
Threshold=0.17, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0	Threshold=0.17, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0
	

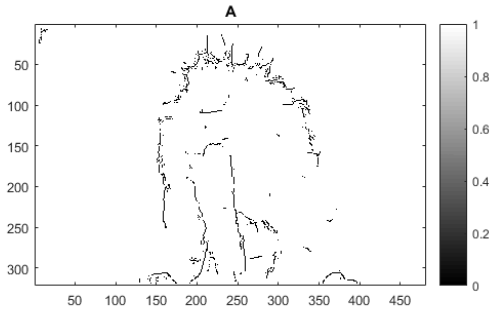
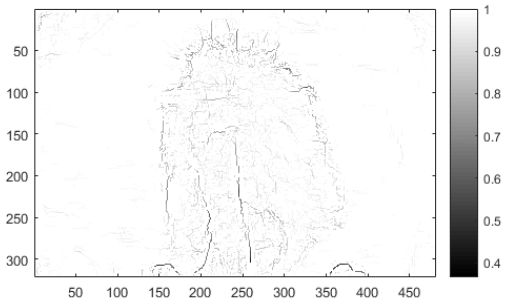
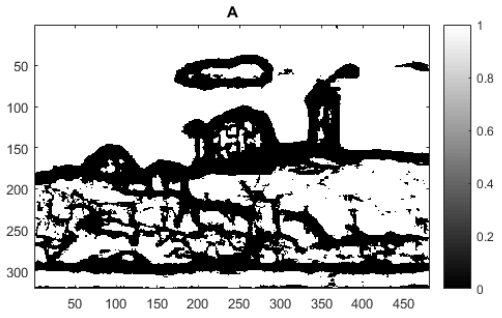
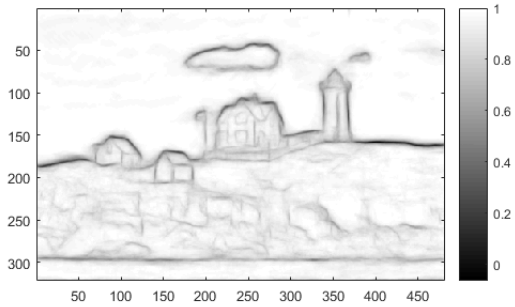
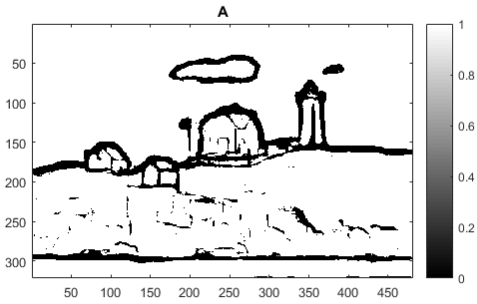
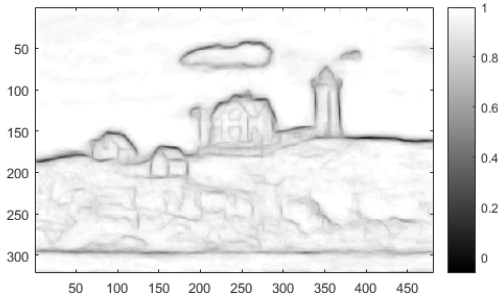
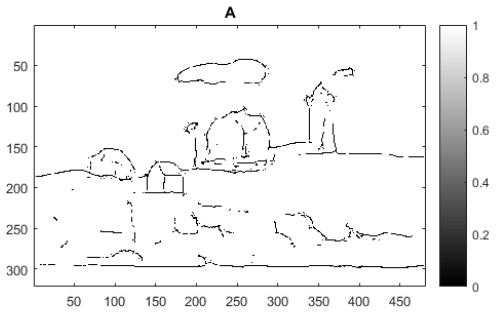
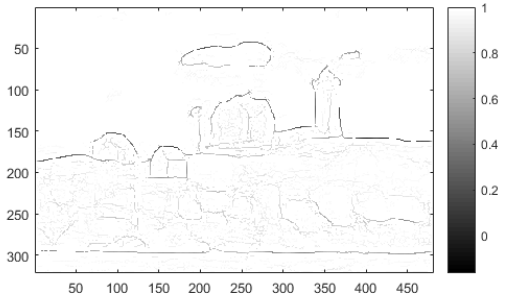
Threshold=0.07, Multiscale=0, Sharpen=0, number of trees=1, number of trees=4, NMS value=1	Threshold=0.07, Multiscale=0, Sharpen=0, number of trees=1, number of trees=4, NMS value=1
 **Figure 48: Threshold=0.12, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=1**	 **Threshold=0.12, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=1**

Figure 49: Original House Image

Binary Edge Map	**Probability Edge Map**
	
Threshold=0.07, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0	Threshold=0.07, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0
	
Threshold=0.17, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0	Threshold=0.17, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=0
	

Threshold=0.07, Multiscale=0, Sharpen=0, number of trees=1, number of trees=4, NMS value=1	Threshold=0.07, Multiscale=0, Sharpen=0, number of trees=1, number of trees=4, NMS value=1

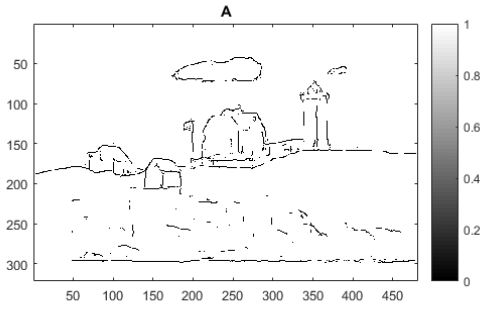
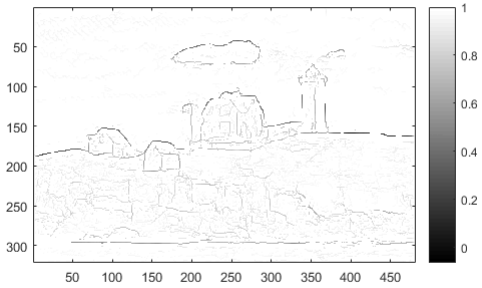

	

Figure 50: Threshold=0.2, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=1	**Threshold=0.2, Multiscale=1, Sharpen=0, number of trees=4, NMS value=1**

IV. Discussion

- The Algorithm and flow chart for structured edge is being explained above.
- Three parameters were changed in the MATLAB function and the corresponding outputs were obtained. A picture of the parameters changed is as given below:

```
%% 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
```

(Source: https://github.com/pdollar/edges)

- A binary edge map was prepared for the Animal and House images using the threshold values of 0.07 and 0.17. However, the best edge map according to me was obtained for threshold=0.2 for house image and threshold=0.12 for Animal image.
- After changing a lot of parameters, I concluded the following things:
  - Parameters such as sharpen, Number of trees and Number of threads are just used to increase the speed of computation, they don't affect the quality of edges generated.
  - NMS value is kept as 1 or 0. For NMS value 0 thicker edges are generated and for NMS value 1, thinner edges are generated. Hence, NMS value 1 is preferred more than 0.

- Multiscale value is used to obtain the accuracy with which edges are generated. For multiscale 0, less edges and noisy edges are detected, but for multiscale value 1, appropriate and more edges are detected.

- According to my observations, the best parameters are as follows:
  Threshold=0.2, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=1 for House
  Threshold=0.12, Multiscale=1, Sharpen=0, number of trees=1, number of trees=4, NMS value=1 for Animal.
  The corresponding Figure 48 gives the best visual edge detection for Animal and Figure 50 gives the best visual detection for House.
- Comparison between Sobel and Structured Edge Detector:
  The Sobel and LOG outputs are as displayed below:

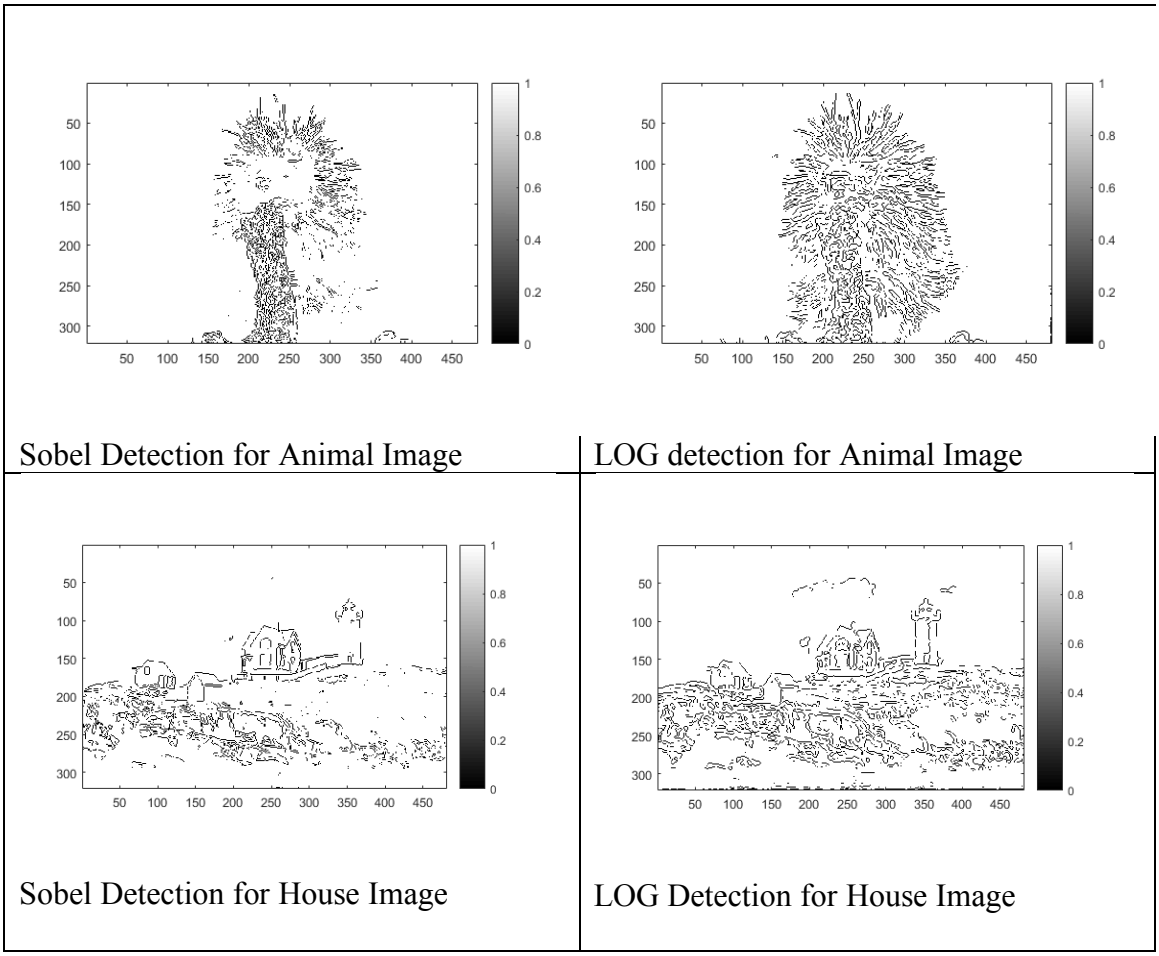


Figure 51: Outputs for Sobel and LOG edge Detector

From the outputs of Structure Edge (SE) and Sobel Detector it can be observed that the Sobel Edge detector has blocky patches whereas the SE output has distinct and clear edges. As Sobel detector is susceptible to noise, there are a lot of unwanted edges detected in the Sobel output. This is not the case of Structure Edge detector as the detector is trained first and hence is not affected by noise to a great extent. This can be

observed in the hairy parts of the animal as there are a lot of noisy edges in the hairy parts in Sobel and LOG compared to Structured Edge. Also, in the house image, the Sobel and LOG detectors detect too many edges in the lawn patch where there aren't so many edges in the actual image. So, I think Structured edge has better accuracy when it comes to detecting edges than LOG or Sobel.

c) Performance Evaluation [1]

I. Approach and Procedure

To evaluate the performance of an edge map, it is compared with ground truths. Ground Truths are how the humans perceive edges and because it is very subjective, 5 ground truths are prepared. Also, mean of the precision and recall is taken to take into account the subjective evaluation with respect to each ground truth.

$$\boldsymbol{Precision(P) = \frac{True\ Positive}{True\ Positive + False\ Positive}}$$

$$\boldsymbol{Recall(R) = \frac{True\ Positive}{True\ Positive + False\ Negative}}$$

Where True Positive is edge pixels in edge map coinciding with edge pixels in ground truth
False Positive is edge pixels in edge map coinciding with non-edge pixels in ground truth
False Negative is non-edge pixels in edge map coinciding with edge pixels in ground truth

As threshold is decreased, the value of precision increases, however the value of recall decreases. Hence, only precision and recall can't be used to judge the performance characteristics of an edge detector. Hence, F-measure is used.
The performance of the edge detection algorithm can be validated using F measure which is calculated using Precision and Recall values.

$$\boldsymbol{F\ measure = 2 * \frac{Presision * Recall}{Precision + Recall}}$$

A high value of F measure indicates that the edge detector detects best and appropriate edges to represent an object.

In this question, the probability edge maps are obtained for each image. As the values of the probability edge map are between 0 and 1, there are thresholded to get a binary edge map.

II. Experimental Results

The table to display the recall, precision and F-measure for each ground truth as well as the mean precision, mean recall and best F-score is given below:

Table for House Image using Structured Edge Detector

Ground Truth Number	Recall	Precision	F-measure	Threshold
Ground Truth 1	0.8757	0.7369	0.8003	0.26
Ground Truth 2	0.6758	0.7779	0.7233	0.23
Ground Truth 3	0.7101	0.8045	0.7544	0.25
Ground Truth 4	0.7454	0.6989	0.7214	0.22
Ground Truth 5	0.7534	0.5683	0.6479	0.19
Mean	**0.7878**	**0.8879**	**0.8349**	**0.2**

Table for Animal Image using Structured Edge Detector

Ground Truth Number	Recall	Precision	F-measure	Threshold
Ground Truth 1	0.6276	0.6421	0.6348	0.13
Ground Truth 2	0.6075	0.6337	0.6203	0.12
Ground Truth 3	0.3971	0.5023	0.4436	0.16
Ground Truth 4	0.6473	0.7395	0.6903	0.13
Ground Truth 5	0.5551	0.6099	0.5812	0.13
Mean	**0.6069**	**0.7656**	**0.6770**	**0.12**

Table for House Image using Sobel Edge Detector

Ground Truth Number	Recall	Precision	F-measure	Threshold
Ground Truth 1	0.6153	0.1626	0.2572	0.01
Ground Truth 2	0.5486	0.2356	0.3297	0.01
Ground Truth 3	0.5425	0.2037	0.2962	0.01
Ground Truth 4	0.6885	0.2528	0.3699	0.02
Ground Truth 5	0.8616	0.3030	0.4484	0.01
Mean	**0.6475**	**0.4021**	**0.4961**	**0.68**

Table for Animal Image using Sobel Edge Detector

Ground Truth Number	Recall	Precision	F-measure	Threshold
Ground Truth 1	0.5499	0.1467	0.2312	0.01
Ground Truth 2	0.5377	0.1690	0.2571	0.01
Ground Truth 3	0.4094	0.0900	0.1491	0.01
Ground Truth 4	0.6287	0.1890	0.2907	0.02
Ground Truth 5	0.5230	0.1512	0.2346	0.01
Mean	**0.5352**	**0.2428**	**0.3340**	**0.68**

Table for House Image using Laplacian of Gaussian Edge detection

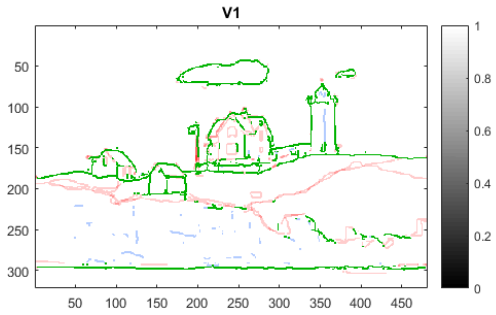
Ground Truth Number	Recall	Precision	F-measure	Threshold
Ground Truth 1	0.5892	0.1880	0.2851	0.02
Ground Truth 2	0.4996	0.2592	0.3414	0.01
Ground Truth 3	0.5372	0.2436	0.3352	0.01
Ground Truth 4	0.6350	0.2817	0.3902	0.01
Ground Truth 5	0.8045	0.3417	0.4797	0.01
Mean	**0.6023**	**0.4400**	**0.5107**	**0.05**

Table for Animal Image using Laplacian of Gaussian Edge detection

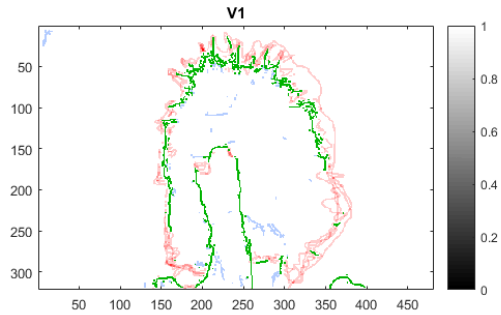
Ground Truth Number	Recall	Precision	F-measure	Threshold
Ground Truth 1	0.9007	0.1614	0.2738	0.01
Ground Truth 2	0.8532	0.1785	0.2952	0.01
Ground Truth 3	0.6209	0.0920	0.1603	0.01
Ground Truth 4	0.9542	0.1910	0.3182	0.01
Ground Truth 5	0.8283	0.1594	0.2673	0.01
Mean	**0.8418**	**0.2674**	**0.4059**	**0.54**

The Visualizations of matches of edges in an image with respect to the ground truths are given in the V image. In the V-image the true positives, false negatives and false positives are given using the following colors:
Green- True Positive
Blue- False Positive
Red- False Negative

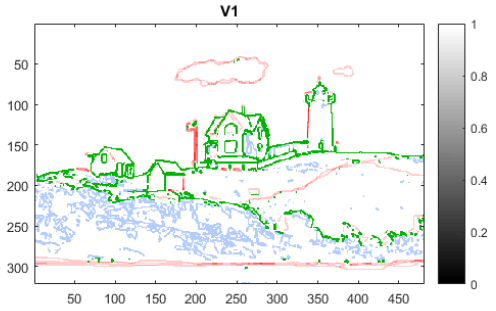
I have displayed the V images for all the three detectors here.



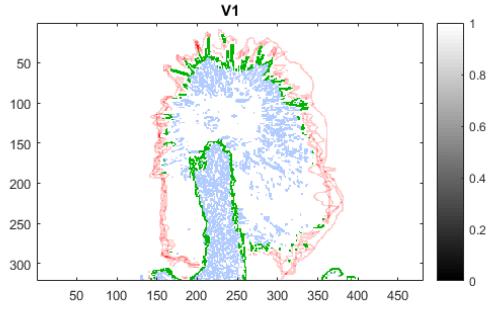
V image for House image using Structured Edge



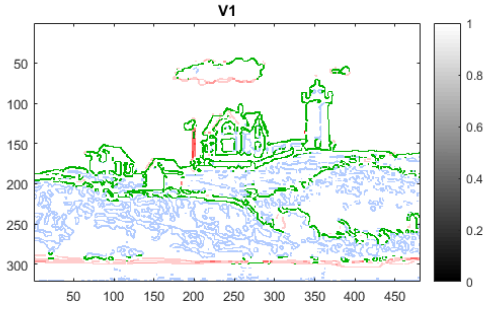
V image for Animal image using Structured Edge



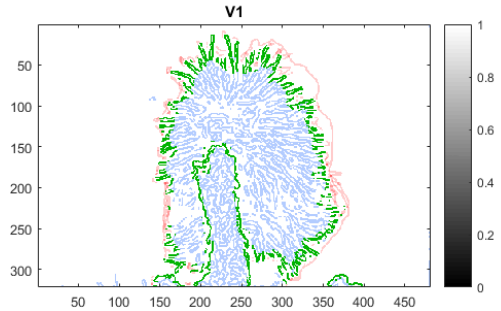
V image for House image using Sobel Edge



V image for Animal image using Sobel Edge



V image for House image using LOG Edge



V image for House image using LOG Edge

As it can be seen in the outputs of V edge, the Sobel and LOG outputs contain a lot of Blue portions which indicate false positives, as there are a lot of noisy edges detected in these two edge detectors compared to Structured Edge detector. Also, the green portions which indicate true positives are more in Structured Edge compared to Sobel and LOG edge detector.

III. Discussion

- F-score is used to evaluate the performance of an edge detector. A high value of F-score indicates that the edge detector is good. After observing the f-score values of Sobel, LOG and Structured edge detector it can be concluded that Structured edge detector is better than LOG and Sobel as it has a high value of score. The value of f-score for both house and animal images is in the range of 0.6-0.85 compared to that of Sobel and SE which is in the range of 0.3-0.5. Also, as given in the Visual Image evaluation process that Structured edge gives more true positives and less false positives.
- Sobel and LOG detect a lot of noisy edges which are not detected using Structured Edge as SE is first trained and then testing data is applied. Hence, it is more accurate in identifying false edges and detecting true edges.
- As F-measure is image independent, the F-measure for Animal and House images will vary. Intuitively I think segmenting the House image into edges is easier than segmenting Animal image as Animal image has a lot of occlusions and noisy points. Hence, there are high chances that the edges detected in Animal will be false edges. Out assumption is validated in our output as the F measure using Structured edge for House is 0.8349 whereas for Animal it is 0.6770.
- As f-measure is actually the harmonic mean of the precision and recall values, varying just one to a great extent won't change the F measure. There exists an inverse relationship between precision and recall. When Precision increases, recall decreases. Having equal values of precision and Recall will give a higher F measure. This can be proven as follows:
  If Precision and Recall add up to a constant,
  Precision (P) + Recall (R)=Constant (C)
  - R= C – P
  - Substituting in formula for F measure
  - F= 2 * (P * R)/(P+R)
  - Therefore, F= 2 * (P) (C – P)/ C.
  - To find the maximum, find the first derivative with respect to P and equate it to 0 if. F'=0
  - F = (2 * P * C- 2 * P * P) / C
  - F' is 2C-4P=0
  - Hence, P=C/2
  - Therefore, R=C/2 and R=P
  - Hence, we will get the maximum F-score when Recall = Precision

Problem 3: Salient Point Descriptors and Image Matching [2]

I. Abstract and Motivation

The features of an image are like the heart of an image. In applications such as object recognition, panorama, image stitching, gesture recognition etc. it is required to extract important features from the image. The extracted features have to be rotation invariant and scaling invariant. To obtain these features SIFT and SURF techniques are used. These techniques are robust as they extract features in the image that are scaling and rotation invariant.

In this question, the main aim is to identify key points of any image using SIFT and SURF techniques. These key points are called the salient features of the image. These salient points are then used for image matching using Brute Force and Flann based techniques. The performance and efficiency of the SIFT and SURF features is studied.

Sometimes while classifying an image into categories, different camera positions, variation in the object, illumination effects pose to be a problem. Hence, to reduce these undesirable effects, bag of algorithm is used which creates visual vocabularies from the training set and classifies unknown images from the testing set based on these vocabularies.

II. Approach and Procedure

a) Extraction and Description of salient points [3]

Feature extraction is a very important step in image processing. Scale Invariant Feature Transform(SIFT) and Speeded up Robust Features(SURF) are tools which help in identifying the key points of the image. SIFT was proposed by David G Lowe and is given in detail in [2]. To implement SIFT, OpenCV is used that contains SIFT detectors and extractors.
The following steps are used in SIFT feature extraction:

1) Scale Space Extrema detection

As we have to make the input image invariant to scaling, the scale space extremas are taken into consideration. The main goal of this step is to find the potential key points for the descriptors. The scale space extrema detection is done by applying Gaussian kernels in octaves to the image. The Gaussian Kernels are used as they blur the image protecting the boundaries and edges. Each octave is down sampled by a factor of 2 and each image in an octave is a smoothened by a factor of k which is chosen to be 1.414 in the paper. If the value of sigma is increased, a larger region is taken into consideration. Therefore, sigma acts as a scaling parameter. The optimum value of sigma is mentioned to be 1.6 in the paper.
The Laplacian of Gaussian basically acts as a blob detector and detects the blobs of various sizes in the image. A Gaussian Kernel with low sigma is useful for small corners but a kernel with high sigma is required for larger corners. This is how a local maximum can be found out in scale and space which indicates a key point in (x, y) at scale sigma. The application of Gaussian kernels in octaves can be shown in the following image:

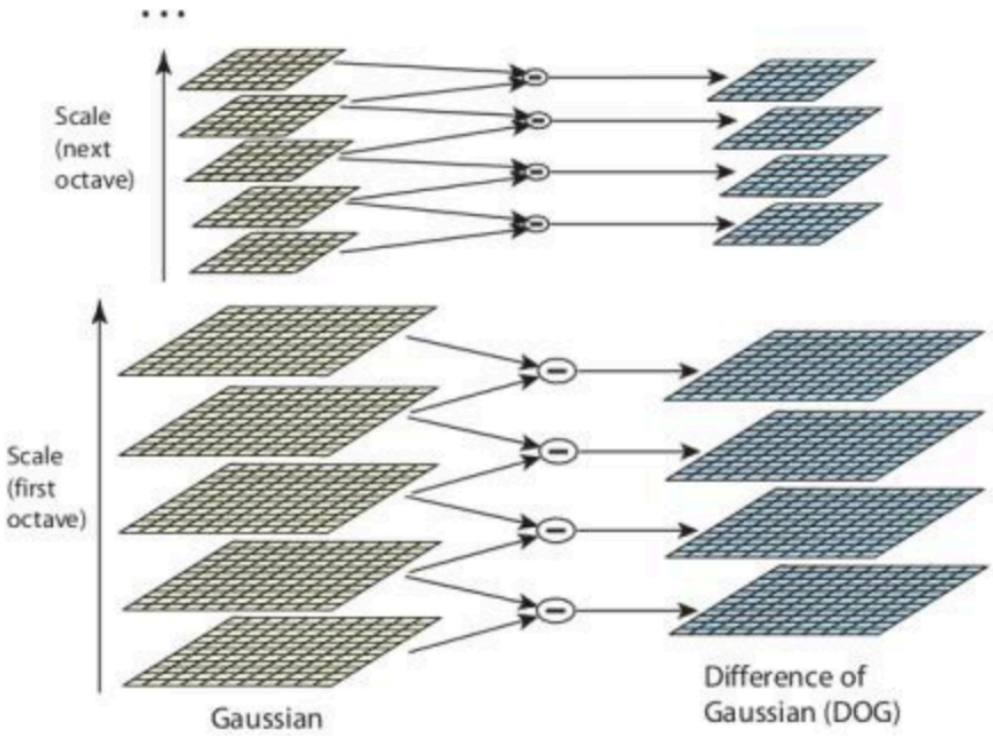


Figure 52: Scale Space Extrema Process (Source: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro)

After the application of Gaussian kernels, the Difference of Gaussian is taken into consideration. The Difference of Gaussian (DOG) is found by taking difference of 2 consecutive LOG's in the octave. DOG is used instead of LOG as it is easier to compute. After calculation of DOG, the local extremas of the image are calculated over scale and space. If a pixel is a local extremum it is a potential key point.

2) Key point Localization

Once the key points have been decided, they have to be refined by removing outliers from the image. The outliers can be in the form of low-contrast points or edges. As DOG is more susceptible to edges, some edges which have redundant information have to be reduced. To reduce the outliers, Taylor series expansion of the scale space extremas is done wherein the intensity points that lie below the threshold of 0.03 are removed. To remove the unwanted edges, Hessian matrix is used to detect the principal curvatures. Hessian matrix comprises of Eigen values and for an edge one Eigen value is larger than the other. To discard certain edge points the ratio of the largest Eigen value to the smallest is taken. If this ratio is greater than 10, the point of interest is considered as an edge and is discarded. This is how low contrast key points and edges are discarded and points are strong interest are retained.

3) Orientation Assignment

The invariance to rotation of an image can be achieved by taking orientation value at each key point. This is achieved by taking the neighborhood of the point of interest. The neighborhood of the key point is taken depending on the scale at which the point was detected. The gradient magnitude and direction is then calculated in the neighborhood. A histogram of 36 bins is calculated which takes into account the 360 rotation possible in an image. The orientation is then weighted by its gradient magnitude and a Gaussian weighted circular window with sigma that is 1.5 times the scaling factor. After this a histogram of the weighted orientation is plotted and the maximum value of the peak in the histogram is noted. Any peak higher then 80% of the maximum peak is taken into consideration for calculating the orientation. This process creates key points with similar scale but different orientations.

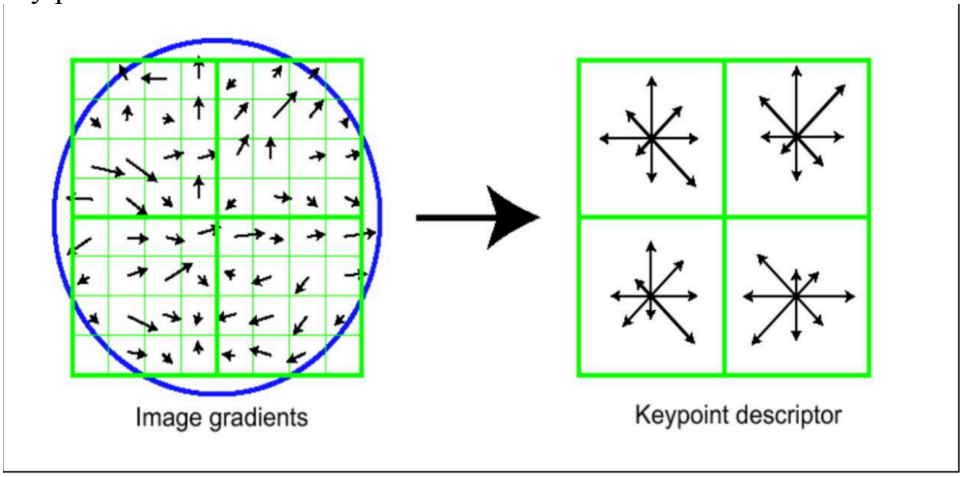


Figure 53: Orientation Assignment for each 4x4 block

4) *Key point descriptors*

The next step is to create key point descriptors. In the paper it is suggested that the histogram of the neighborhood orientations should to be considered as it reduces the effects to illumination, angle difference etc. Hence, a 16x16 neighborhood is considered from the pixel of interest. This neighborhood is then divided further into 4x4 blocks. There are 16 such blocks. The histogram of pixels in each of the 4x4 region is calculated and the data is divided into 6 bins. This is how 128 (16x8) bins histogram is created in every 16x16 neighborhood. Therefore, each key point has a 128 dimensional feature vector associated with it.

Speeded up Robust Features (SURF)

Speeded UP Robust Feature(SURF) was basically developed to speed up the process of SIFT as SIFT was slower. Hence, to achieve this, SURF uses the convolution with respect to box filters instead of DOG filters like in SIFT. The convolution of integral images with box filters is easy to calculate at every scale and this increases the speed to certain extent.

SURF can calculate key points at a high speed due to the fact that it finds integral of an image which works faster irrespective of the size of box filter. It also uses Non-Maximum Suppression to find the best key points for the image.

SURF also uses the sign of laplacian i.e. the trace of Hessian matrix for a particular interest point. Due to this the key points are detected in a better way without increasing the cost of computation as this calculation is done in detection step itself.

SURF is 3 times faster than SIFT while maintaining the performance rate constant. It works well in cases of blurred and rotated images but does not work well in cases of view point change and illumination change.

Task: To detect salient points in images using SIFT and SURF algorithms in OpenCV and studies their performance and efficiency.

Algorithm for Salient point detection in OpenCV:

Step 1: Read the Optimus Prime truck and Bumblebee images using imread() function.
Step 2: Create vectors for SIFT and SURF detectors.
Step 3: Detect the keypoints for SIFT detector using detector class of openCV.
Step 4: Use drawkeypoints() function in OpenCV to draw key points on the images and display them on the image using imwrite() function.
Step 5: Repeat Step 2 to 4 for SURF detector.

b) Image Matching [4]

By extracting the salient points using SIFT and SURF techniques, key point descriptors are prepared. The main reason to extract key points is to match them to another image to find the matching points. This concept is mainly required in object recognition and computer vision applications.

Key point matching: After extracting the features, image matching is performed by identifying the nearest neighbors. But sometimes, there can be two neighbors close to the test point. This mainly happened due to noisy patches in the image. In this case the ratio of the closest neighbor to the second closest neighbor is determined. If this ratio is extremely high, that match is discarded. The threshold used to discard such matching point is 0.8. Due to this thresholding, 95% of the false matches are discarded and 5% of good matches ae let go. This is how the algorithm effectively reduces unwanted matches.

After extracting the salient points using SIFT and SURF tools, Key point matching can be performed using Brute Force Matcher and Flann Matcher.

Brute Force Matcher: Brute Force Matcher uses a simple concept wherein it takes descriptors of one feature from one dataset and matches it with the descriptors of all other features in all other datasets using L2 norm. The feature having the closest distance is used for matching. The BF matching technique is exhaustive as it takes into account all the possible combinations.

Flann Matcher: FLANN stands for Fast Library for Approximate Nearest Neighbors. It is mainly used for larger datasets and higher dimensional data as it is faster compared to Brute Force matcher.

I have used Flann and Brute Force Matcher for matching the data points.

Task: To detect salient points in images using SIFT and SURF algorithms in OpenCV and perform image matching for different set of images.

Algorithm for Image Matching in OpenCV:

Step 1: Read the Optimus Prime truck,Ferrari 1,Ferrari 2 and Bumblebee images using imread() function.
Step 2: Create vectors for SIFT and SURF detectors.
Step 3: Detect and compute the keypoints for SIFT detector using detectandcompute class of openCV.
Step 4: Use match class in openCV to identify the matches between descriptors 1 and 2 of each image.
Step 5: Use drawmatches() function in OpenCV to match the key points in the images and display them on the image using imwrite() function.
Step 6: Repeat Step 2 to 5 for SURF detector.

c) Bag of Words

Bag of words algorithm was originally developed for documents wherein the frequency of occurrence of each word was noted as a vocabulary. This concept was then extended to images to describe an image with respect to it's best features by storing the histograms of these features. In a bag of words model, the frequency of occurrence of features for training images is stored in the form of vocabulary of visual words and the features of testing images are compared with this vocabulary to determine which classes they belong to.

The Bag of words model is implemented here using OpenCV using the Bag of words trainer. There are 3 testing images to generate the vocabulary and there is one test image to determine its closeness to one of the training images. To do feature extraction of an image, SIFT algorithm is used. The SIFT features extracted from each image are clustered into 8 groups by using the K-means algorithm. This is how a histogram of 8 bins is created for each image. Each feature in these 8 bins has a vector size of 128. Every bin in the histogram indicates the best possible features which are the centroids. To find the closest distance of the testing image to one of the training image, its Euclidean distance from each of the centroids is calculated, and based on the minimum distance it is assigned to the respective bin.

The **algorithm for Bag of words** can be summarized as follows:
Step 1: Load the training images and extract the features and descriptors from each image using SIFT feature extractor.
Step 2: Initialize the number of cluster required which are 8 in our case. Add the descriptors to the Bag of words trainer called the BOWKmeanstrainer in OpenCV. Cluster the Bag of Words trainer into 8 clusters to generate the vocabulary consisting of the best possible features in all the images as centroids.
Step 3: Calculate the Euclidean distance of each image with respect to the vocabularies by generating a histogram of frequency of occurrence of descriptors in the image.
Step 4: Calculate the absolute difference of the testing image with the training images with respect to all the 8 bins and decide which training image is the testing image closest to.

The outputs and histograms for all the images are displayed in the subsequent section.

III. Experimental Results

a) Extraction and description of salient points

Figure 54: SIFT Features Bumblebee car

Figure 55: SIFT features Optimus Prime car

Figure 56: SURF Features Bumblebee car

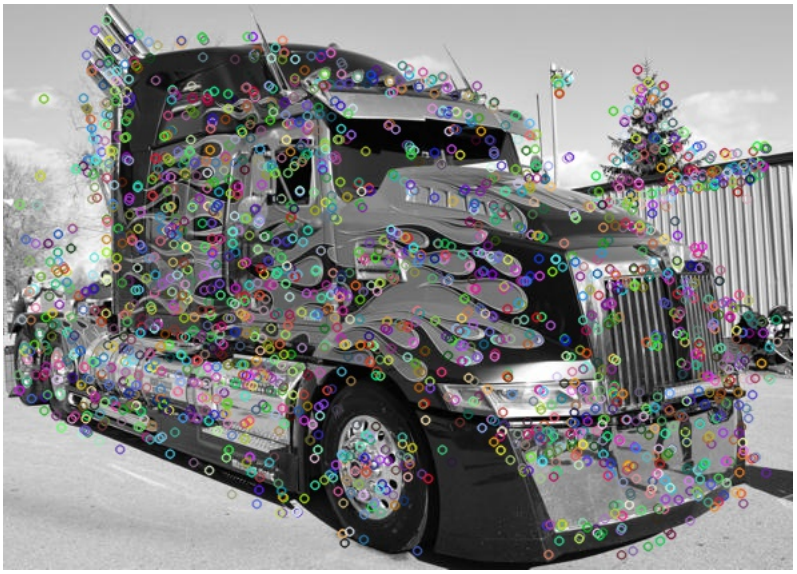


Figure 57: SURF features Optimus Prime car

b) Image Matching

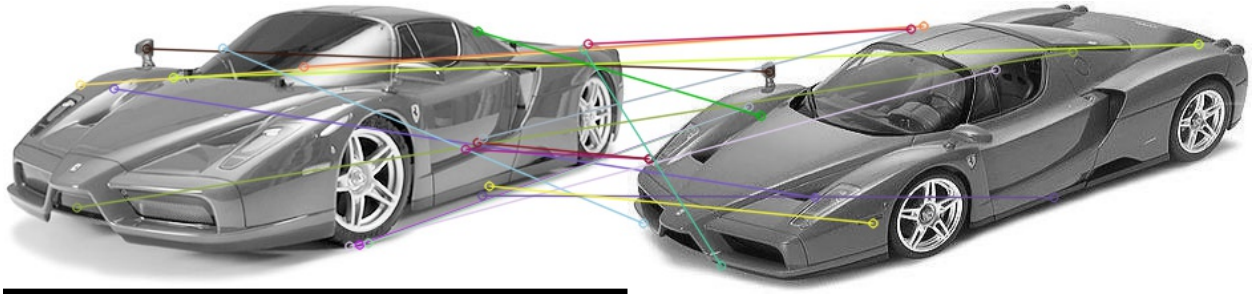


Figure 58: Matching points for SIFT features using Flann Matcher(minHessian=400)

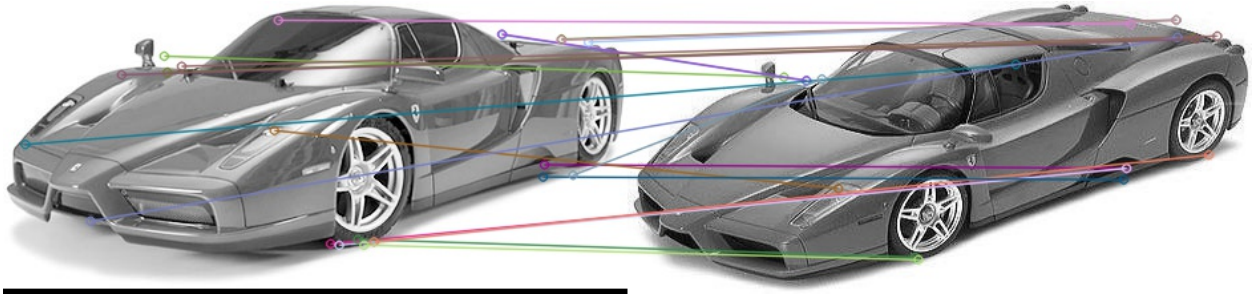


Figure 59: Matching points for SURF features using Flann Matcher(minHessian=400)

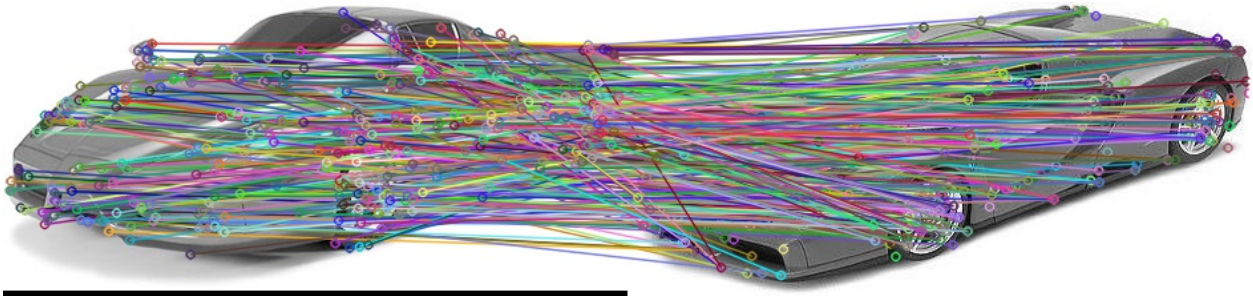


Figure 60: Matching points for SIFT features using Brute Force Matcher(minHessian=400)

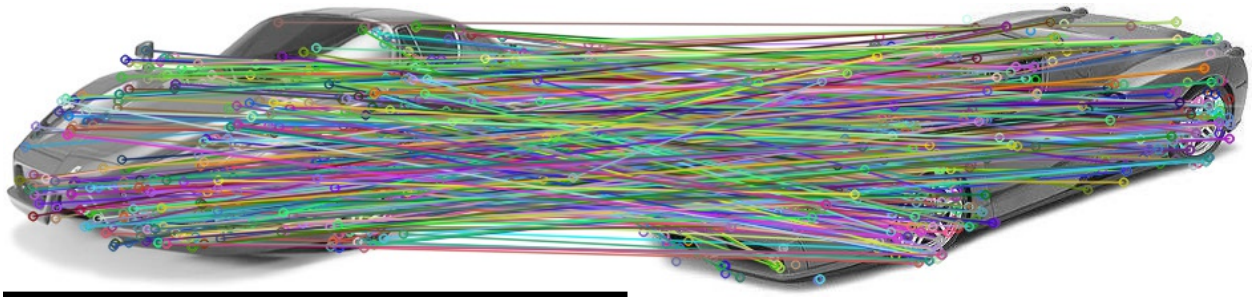


Figure 61: Matching points for SURF features using Brute Force Matcher(minHessian=400)

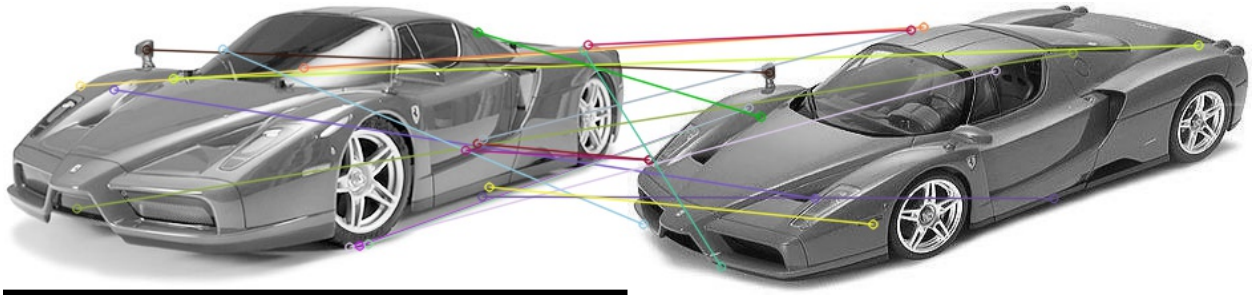


Figure 62: Matching points for SIFT features using Flann Matcher(minHessian=300)

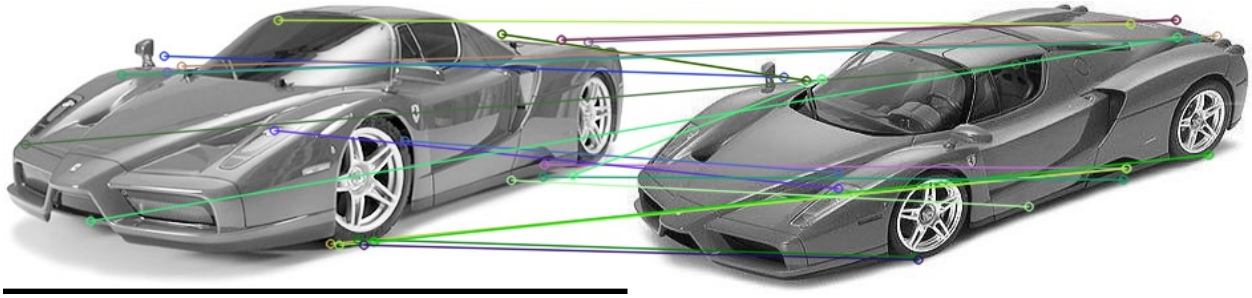


Figure 63: Matching points for SURF features using Flann Matcher(minHessian=300)

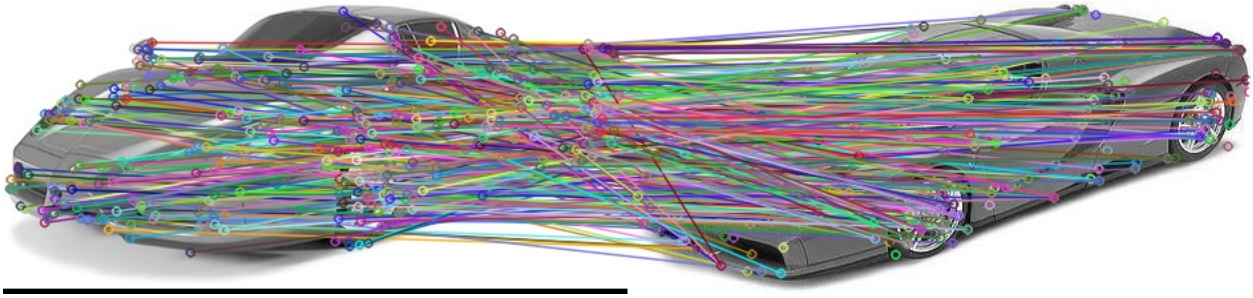


Figure 65: Matching points for SIFT features using Brute Force Matcher(minHessian=300)

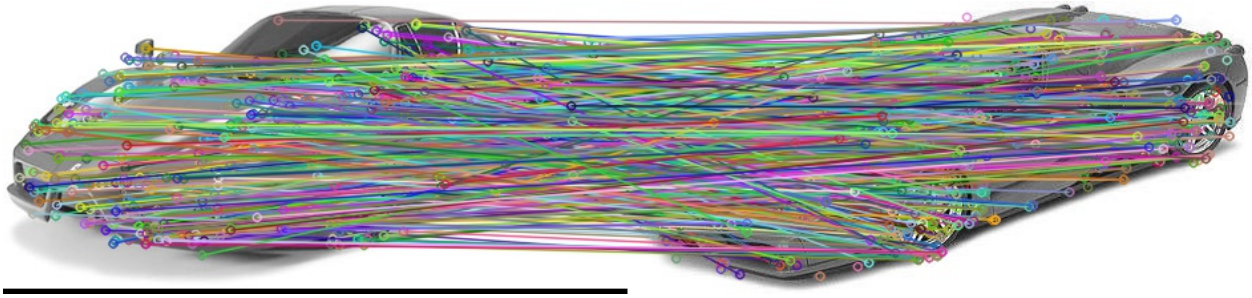


Figure 66: Matching points for SURF features using Brute Force Matcher(minHessian=300)

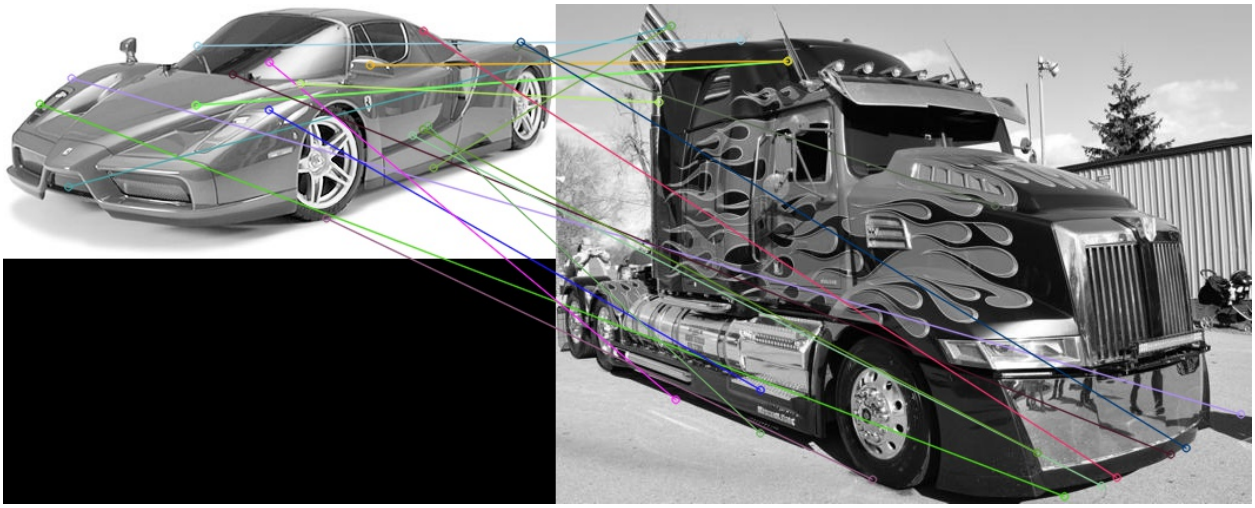


Figure 67: Matching points for SIFT features for Optimus prime and Ferrari 1 images using Flann Matcher(minHessian=400)

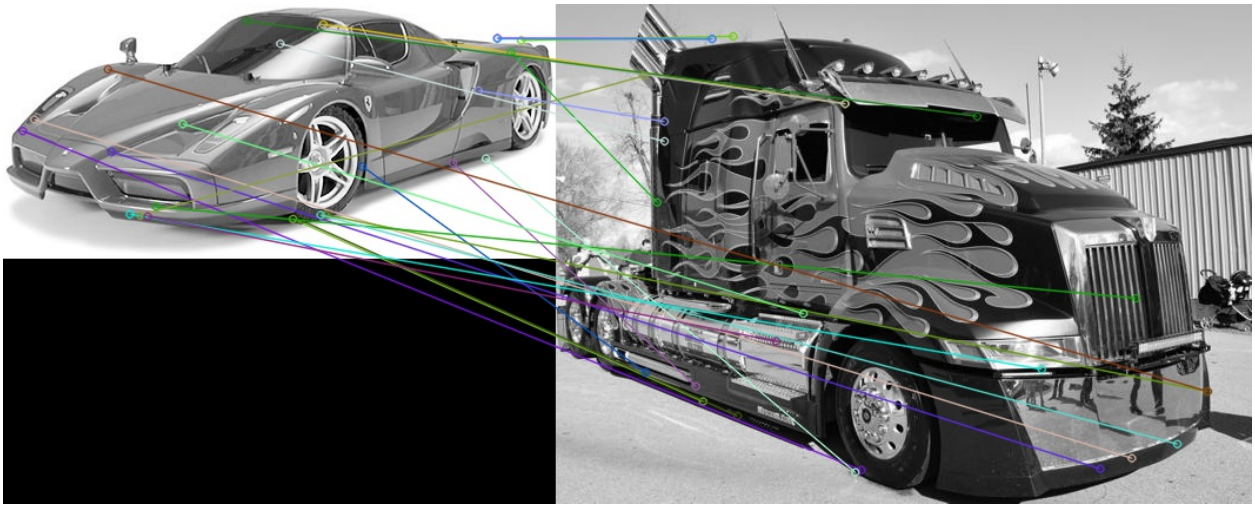


Figure 68: Matching points for SURF features for Optimus prime and Ferrari 1 images using Flann Matcher(minHessian=400)

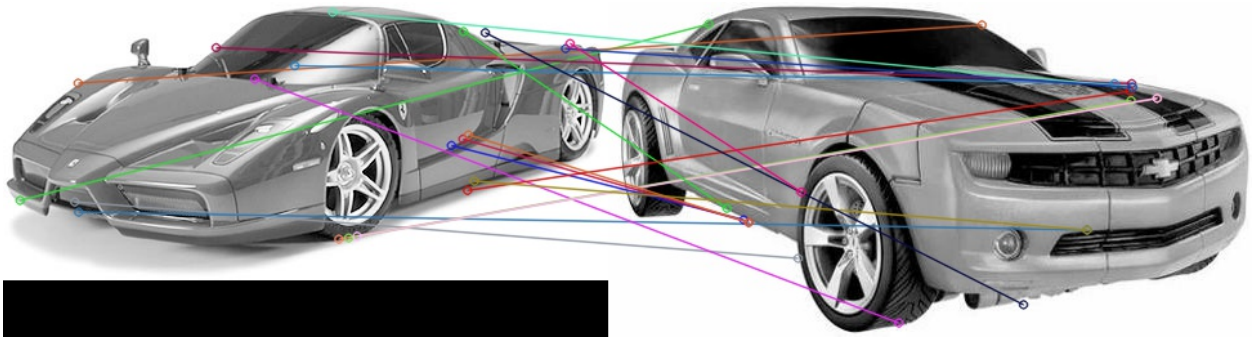


Figure 69: Matching points for SIFT features for bumblebee and Ferrari 1 images using Flann Matcher(minHessian=400)

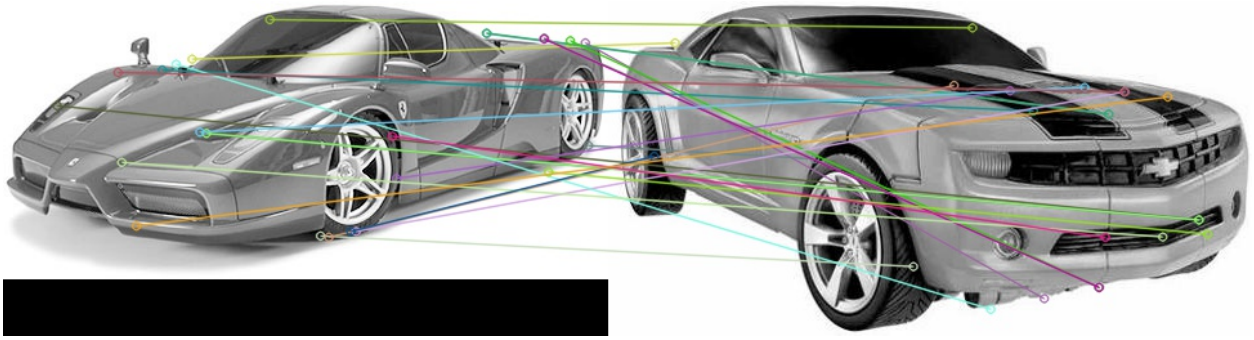


Figure 70: Matching points for SURF features for bumblebee and Ferrari 1 images using Flann Matcher(minHessian=400)

c) Bag of words

```
Error with respect to Ferrari 1 is: 10.125
Error with respect to Optimus Prime is: 30.5
Error with respect to Bumblebee is: 18

Final Conclusion is: Ferrari 2 is close to Ferrari 1
```

Figure 71: Final output to determine which image is Ferrari 1 closest to

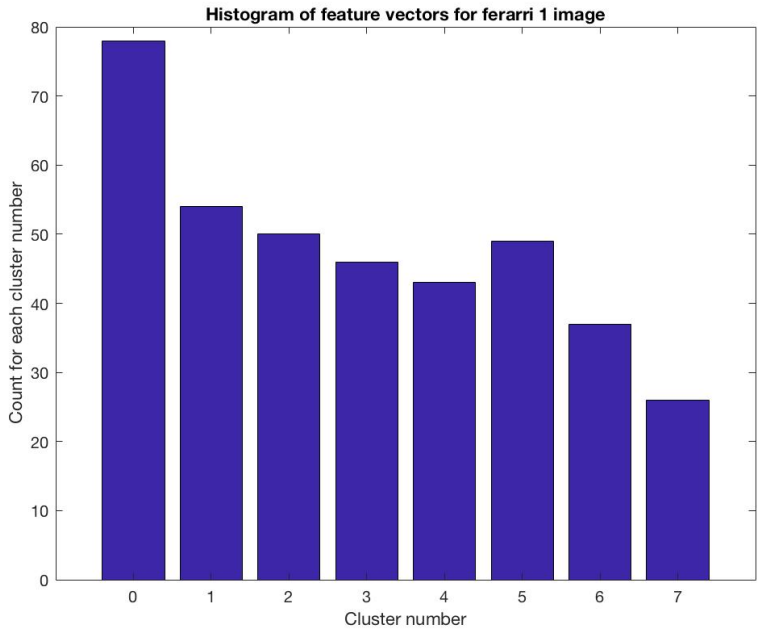


Figure 72: Histogram of Ferrari 1

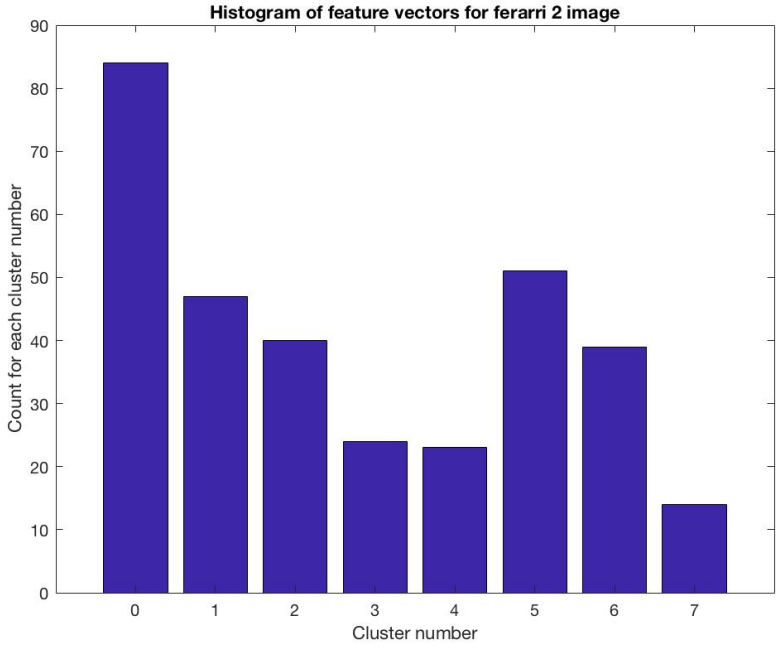


Figure 73: Histogram of Ferrari 2

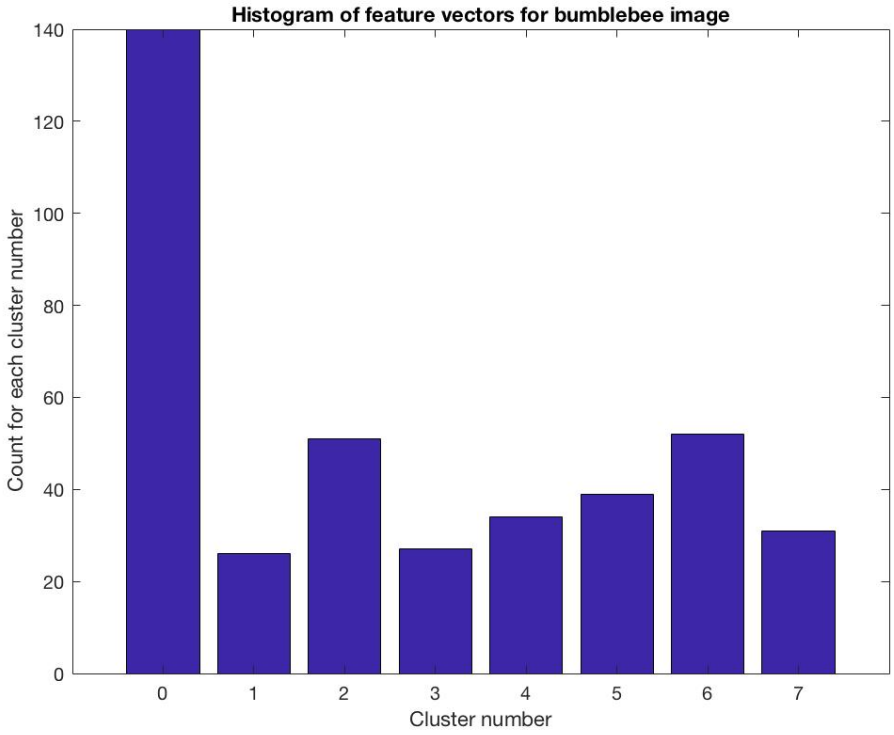


Figure 74: Histogram of Bumblebee

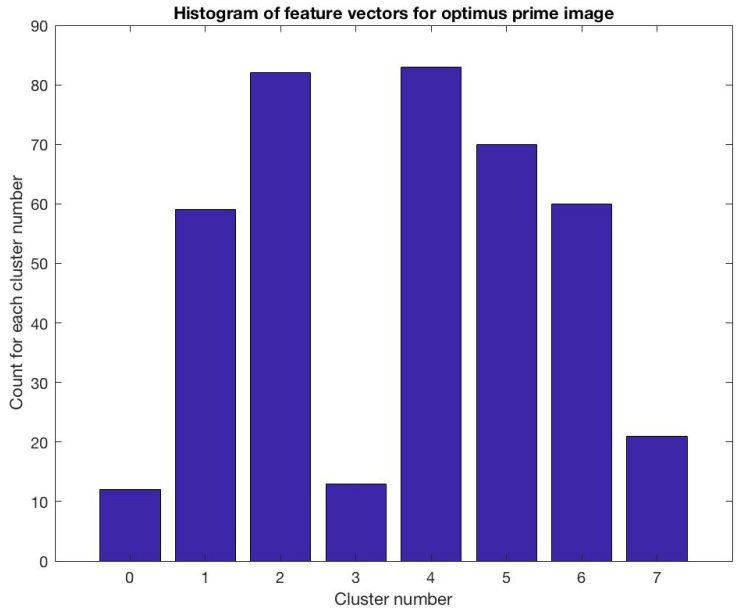


Figure 75: Histogram of Optimus Prime

IV. Discussion

a) Extraction and description of salient points

SIFT and SURF features were extracted in each image and it can be deduced that SURF feature extractor gives better and more points of interest than the SIFT detector. This happens due to the following reasons:

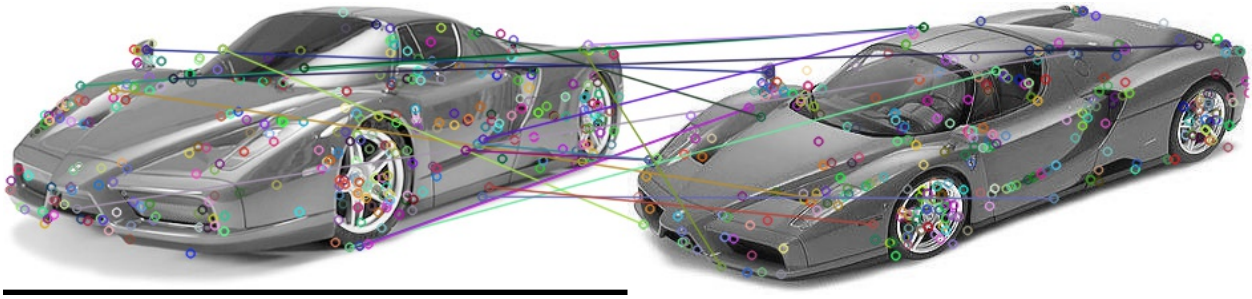
1) SURF uses integral images to generate scale space whereas SIFT uses convolution with DOG to generate the scale space. The integral images in SURF are then convolved with box filters of different sizes. This concept used in SURF decreases the computation time of feature vectors and hence SURF features are generated in 1/3rd time of what is required for SIFT features.
2) SURF uses quadratic grid of 4x4 neighborhood to generate the points of interest whereas SIFT uses 16x16 neighborhood to generate the points. Hence, the performance of SURF becomes better compared to SIFT.
3) SIFT works better for smaller datasets wherein the time of computation is not a constraint whereas SURF works better in larger dataset where the computation of features has to be done faster.
4) SIFT generates 128-Dimensional feature vectors whereas SURF uses Haar wavelet response to record 64-dimensional feature vectors.
5) SIFT works better for changes in illumination or changes in viewpoints of images as it uses integral images whereas SURF works better for blurred and rotated images as it used pyramidal structure of DOG.

After observing the outputs for SIFT and SURF feature vectors in Figures 54,55,56,57, it can be concluded that the SURF feature extractor algorithm generates more key points than SIFT algorithm especially while detecting edges in trees and wheels.

b) Image Matching

Image Matching requires extraction of salient features which is done using SIFT and SURF feature extractors. The matching algorithm works well as it matches almost all the points leaving out some points. I tried two Matching algorithms which are Flann Based matcher and Brute Force Matcher.
The following deductions are made according to the obtained outputs:

- Brute Force Matcher matches a lot of key points, but it also matches some noisy key points. For instance, while comparing Ferrari 1 and Ferrari 2 images, we can see that using the Brute Force matcher, the bottom of the car was getting mapped to the bonnet whereas some points on the wheels were getting matched to the upper part of the car etc.
- For different thresholds of Hessian matrix values, the number of matching points changes. As the value of threshold is increased, the number of matching points increased. This can be seen from Figure 66 and Figure 68 wherein the number of matching points using the Brute Force matcher has increased from Hessian threshold 300 to Hessian threshold 400.
- Also, according to me Flann Matcher works better than the Brute Force Matcher as appropriate points are matched using Flann Matcher rather than matching of noisy points using a Brute Force Matcher. To solve the problem of matching of noisy points using Brute Force matcher, I developed an algorithm wherein I calculated the minimum and maximum distance values of each match and kept only those values which had their distance less than 2 times the minimum distance. This decreased my matching points to a great extent and gave me only optimum points for matching.

- 

This is how the number of points of Brute Force matcher with Hessian value=400 have decreased compared to Figure 60 after using the above-mentioned algorithm.

- After observing a lot of threshold combinations, I concluded that the combination of Flann Matcher along with the Hessian Threshold of 400 gives better output as it uses

nearest neighbor approach. Also, SURF feature matching works better than SIFT feature matching as more salient points are matched in SURF.

- So, to compare two different sets of images, I used Flann Matching with threshold 400. As the viewing angle difference between Ferrari 1 and Bumblebee and Ferrari 1 and Optimus Prime is large, the matching is not very proper. Though some matching points are detected, a lot of noisy points are detected as well. In this case as well, SURF feature matching works better than SIFT feature matching as we can see that the glass of both the cars as well as the wheels are matched properly in SURF than SIFT.

c) Bag of words

The bag of words model for the images was prepared to test which image is Ferrari 2 closest to. From visual inspection it is obvious that Ferrari 2 is closest to Ferrari 1 as they are the same images with slight orientation difference. The output obtained from the algorithm coincides with our assumption as Ferrari 2 image has the minimum distance with respect to Ferrari 1 which can be seen in Figure 71.

The histograms for all the images have been plotted. After closely observing the histograms it can be observed that the histograms of Ferrari 1 and 2 are quite similar with some difference in the frequency of values in each bin. This difference is in regard to the orientation shift between both the images.

I tried changing the value of Hessian matrix parameter, but every time Ferrari 2 was getting classified as being close to Ferrari 1. This validates the code in terms of its efficiency.

Hence, the bag of words works well in determining the closeness of Ferrari 2 with the other three images.

REFERENCES

[1] https://github.com/pdollar/edges for edge detection question

[2] OpenCV documentation from www.opencv.org for OpenCV installation and reference source codes

[3] https://docs.opencv.org/3.0-beta/doc/tutorials/features2d/feature_detection/feature_detection.html

[4]https://docs.opencv.org/3.3.1/d5/d6f/tutorial_feature_flann_matcher.html

[4] www. wikipedia.com

[5] www.google.com