

EE 569

PROJECT #3

BY

PRANAV GUNDEWAR

USC ID: 4463612994

EMAIL: gundewar@usc.edu

Table of Contents

Problem 1: TEXTURE ANALYSIS AND SEGMENTATION

- a) Texture Classification
- b) Texture Segmentation
- c) PCA and Post Processing

Problem 2: EDGE DETECTION

- a) Basic Edge Detector
- b) Structured Edge
- c) Performance Evaluation

Problem 3: SALIENT POINT DESCRIPTORS AND IMAGE MATCHING

- a) Extraction and Description of Salient Points
- b) Image Matching
- c) Bag of Words

PROBLEM 1 – TEXTURE ANALYSIS AND SEGMENTATION

TEXTURE CLASSIFICATION

Task: Generate nine 5*5 filters and classify the input texture images into clusters

ABSTRACT AND MOTIVATION

Texture of an image gives an overview of how an image might look like. It can be considered as pattern that repeats periodically and that can be segmented and classified further. This texture information is very useful in image segmentation and classification problems in industry. Second part of the task is to classify 12 texture images into 4 clusters using K means clustering (unsupervised) and visual inspection which is supervised.

APPROACH AND PROCEDURE

We have to construct 9 Laws filter using kernels for edge spot and waves.

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

The general overview of approach is stated below

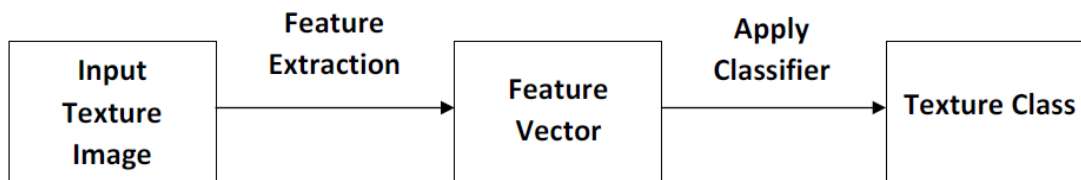


Figure 1 – Overview of Texture Classification

For every input image, we will 9 features of every image using devised Laws filters.

Laws filter can be calculated using tensor product of 2 kernels.

So, for 3 5*5 kernels, we will get 9 filters namely-

E5E5, E5S5, E5W5, S5E5, S5S5, S5W5, W5E5, W5S5, W5W5.

Tensor product is shown below

$$L5^T L5 = \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} [1 \quad 4 \quad 6 \quad 4 \quad 1]$$

$$L5^T L5 = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

In texture classification, we apply each filter from filter bank and calculate average energy of an entire image which is particular feature required. This process is repeated for all filters and all images.

The obtained feature vector is of dimensions M*N where M is number of images and N is number of Laws filter used.

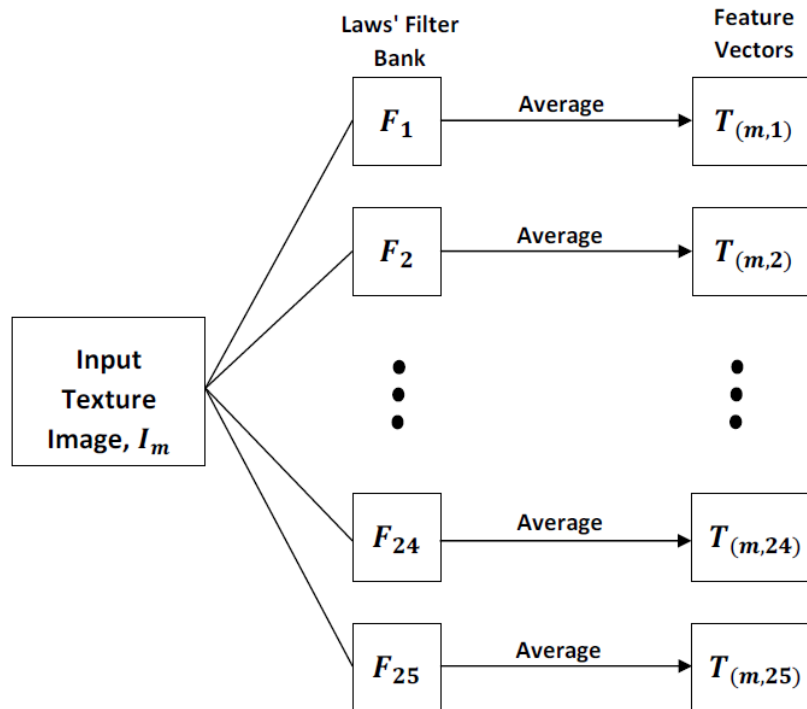


Figure 2 – Laws filter bank

Where,

I_m is input image, F_1 to F_{25} are Laws filters and T is respective feature calculated by

$$T_{(m,n)} = \frac{1}{KL} \sum_{i=0}^K \sum_{j=0}^L F_n(i,j)$$

After feature vector has been calculated, we are required to cluster the features in 4 clusters using K means clustering.

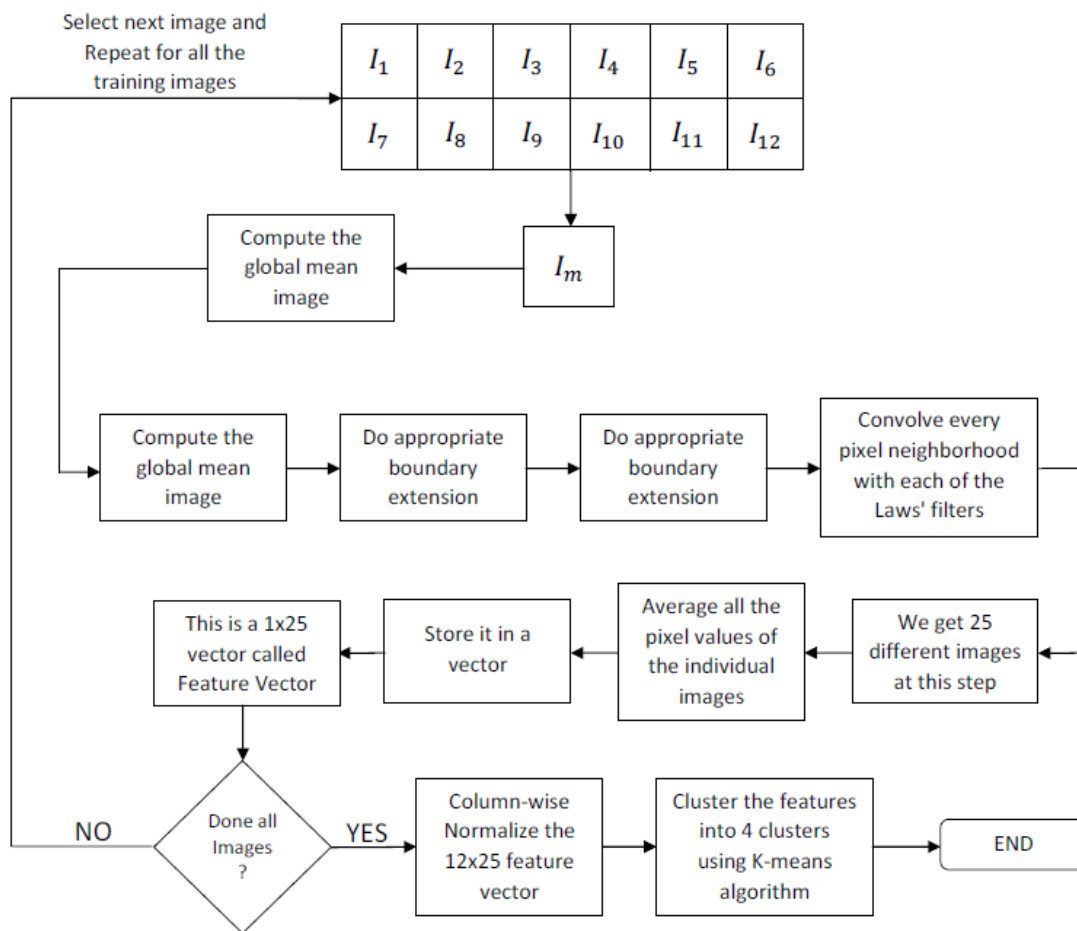


Figure 3 – Flow chart for Feature Extraction

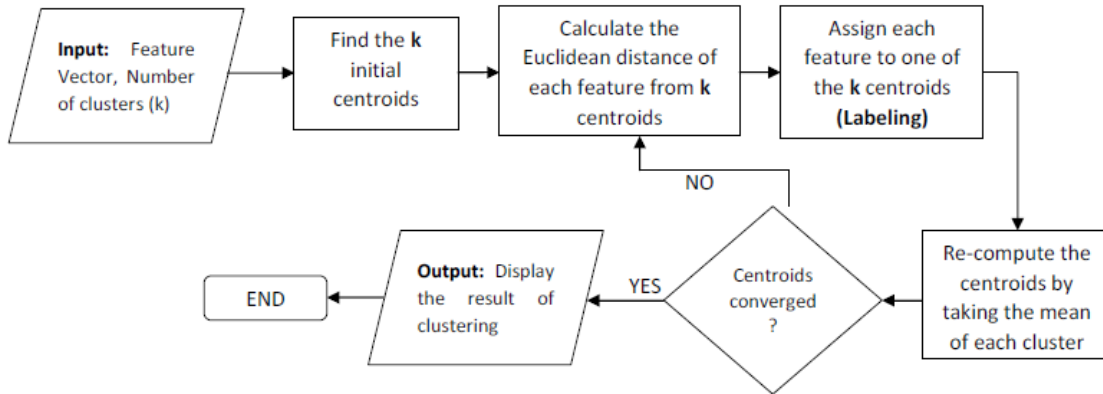


Figure 4 – Flow chart for K Means Clustering

STEP 1 – Choose random initial K centroids

STEP 2 – Assign each feature to one of the determined centroid clusters. It is found out by Euclidean distance between feature point and centroid

STEP 3 – Once all the features are assigned to any particular labels, we update the centroid using

$$C_K = \frac{1}{\sum_i I(\text{Label}(i, j) = K)} \sum_i I(\text{Label}(i, j) = K) T(i, j)$$

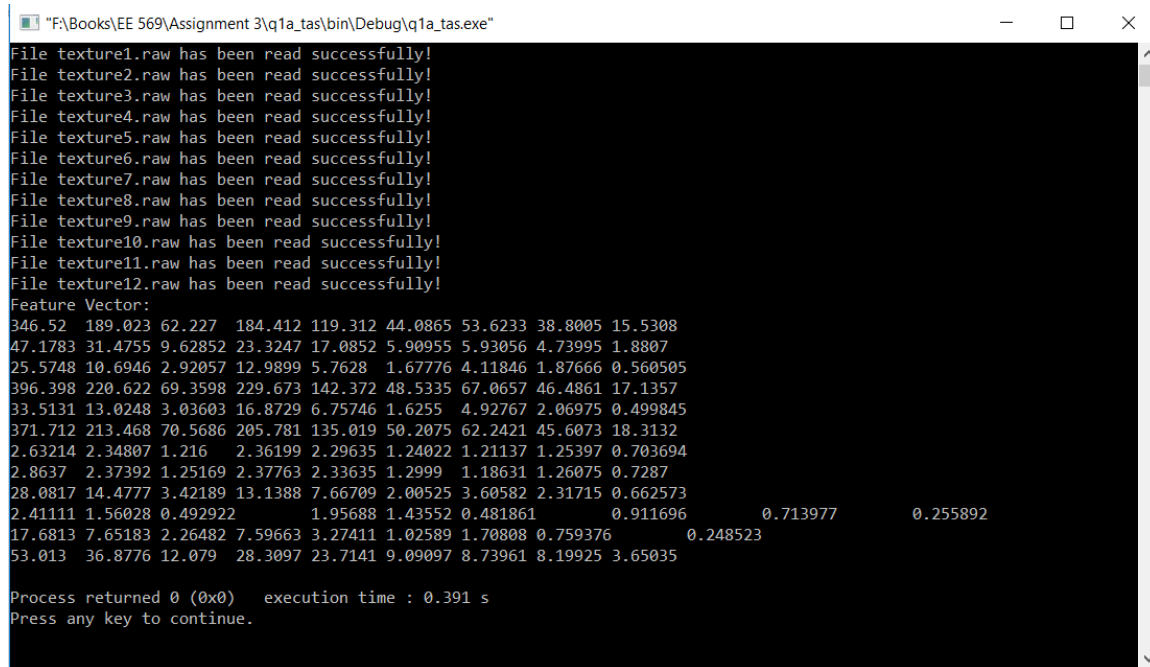
Where,

Label is indicator function, K is number of Clusters

STEP 4 – Repeat until process iteratively converges that there is no significant change in centroid mean.

STEP 5 – Speed up the process by choosing intuitively random features as centroids.

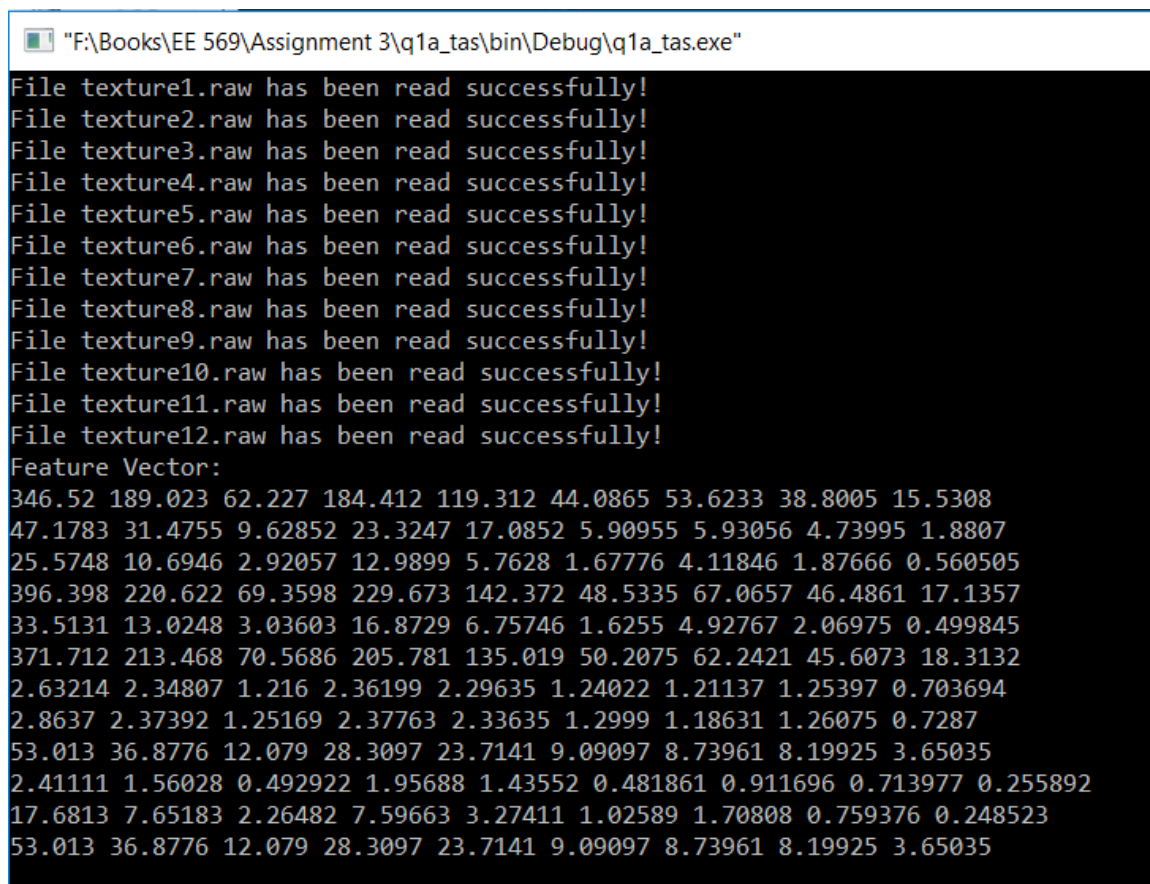
EXPERIMENTAL RESULTS



```
"F:\Books\EE 569\Assignment 3\q1a_tas\bin\Debug\q1a_tas.exe"
File texture1.raw has been read successfully!
File texture2.raw has been read successfully!
File texture3.raw has been read successfully!
File texture4.raw has been read successfully!
File texture5.raw has been read successfully!
File texture6.raw has been read successfully!
File texture7.raw has been read successfully!
File texture8.raw has been read successfully!
File texture9.raw has been read successfully!
File texture10.raw has been read successfully!
File texture11.raw has been read successfully!
File texture12.raw has been read successfully!
Feature Vector:
346.52 189.023 62.227 184.412 119.312 44.0865 53.6233 38.8005 15.5308
47.1783 31.4755 9.62852 23.3247 17.0852 5.90955 5.93056 4.73995 1.8807
25.5748 10.6946 2.92057 12.9899 5.7628 1.67776 4.11846 1.87666 0.560505
396.398 220.622 69.3598 229.673 142.372 48.5335 67.0657 46.4861 17.1357
33.5131 13.0248 3.03603 16.8729 6.75746 1.6255 4.92767 2.06975 0.499845
371.712 213.468 70.5686 205.781 135.019 50.2075 62.2421 45.6073 18.3132
2.63214 2.34807 1.216 2.36199 2.29635 1.24022 1.21137 1.25397 0.703694
2.8637 2.37392 1.25169 2.37763 2.33635 1.2999 1.18631 1.26075 0.7287
28.0817 14.4777 3.42189 13.1388 7.66709 2.00525 3.60582 2.31715 0.662573
2.41111 1.56028 0.492922 1.95688 1.43552 0.481861 0.911696 0.713977 0.255892
17.6813 7.65183 2.26482 7.59663 3.27411 1.02589 1.70808 0.759376 0.248523
53.013 36.8776 12.079 28.3097 23.7141 9.09097 8.73961 8.19925 3.65035

Process returned 0 (0x0)   execution time : 0.391 s
Press any key to continue.
```

Figure 5 – Feature vector



```
"F:\Books\EE 569\Assignment 3\q1a_tas\bin\Debug\q1a_tas.exe"
File texture1.raw has been read successfully!
File texture2.raw has been read successfully!
File texture3.raw has been read successfully!
File texture4.raw has been read successfully!
File texture5.raw has been read successfully!
File texture6.raw has been read successfully!
File texture7.raw has been read successfully!
File texture8.raw has been read successfully!
File texture9.raw has been read successfully!
File texture10.raw has been read successfully!
File texture11.raw has been read successfully!
File texture12.raw has been read successfully!
Feature Vector:
346.52 189.023 62.227 184.412 119.312 44.0865 53.6233 38.8005 15.5308
47.1783 31.4755 9.62852 23.3247 17.0852 5.90955 5.93056 4.73995 1.8807
25.5748 10.6946 2.92057 12.9899 5.7628 1.67776 4.11846 1.87666 0.560505
396.398 220.622 69.3598 229.673 142.372 48.5335 67.0657 46.4861 17.1357
33.5131 13.0248 3.03603 16.8729 6.75746 1.6255 4.92767 2.06975 0.499845
371.712 213.468 70.5686 205.781 135.019 50.2075 62.2421 45.6073 18.3132
2.63214 2.34807 1.216 2.36199 2.29635 1.24022 1.21137 1.25397 0.703694
2.8637 2.37392 1.25169 2.37763 2.33635 1.2999 1.18631 1.26075 0.7287
53.013 36.8776 12.079 28.3097 23.7141 9.09097 8.73961 8.19925 3.65035
2.41111 1.56028 0.492922 1.95688 1.43552 0.481861 0.911696 0.713977 0.255892
17.6813 7.65183 2.26482 7.59663 3.27411 1.02589 1.70808 0.759376 0.248523
53.013 36.8776 12.079 28.3097 23.7141 9.09097 8.73961 8.19925 3.65035
```

Figure 6 – Feature vector when texture 14 is used for texture 9

```
Final Class Labels:  
Images belong to Class Rock: 1 4 6  
Images belong to Class Grass: 2 12  
Images belong to Class Weave: 3 5 9 11  
Images belong to Class Sand: 7 8 10  
  
Process returned 0 (0x0)   execution time : 0.461 s  
Press any key to continue.
```

Figure 7 – Final Class labels

```
Final Class Labels:  
Images belong to Class Rock: 1 4 6  
Images belong to Class Grass: 2 9 12  
Images belong to Class Weave: 3 5 11  
Images belong to Class Sand: 7 8 10  
  
Process returned 0 (0x0)   execution time : 0.458 s  
Press any key to continue.
```

Figure 8 – Final Class labels when Texture 14 is used

DISCUSSION

The K means clustering has been done on obtained feature vector. The boundary extension that was applied is based on pixel replication.

The discriminant power of features can be found out using overlapping data points after filter has been applied. Highly overlapped data denoted weak discriminant power. The minimal overlapping of data points denoted very strong discriminant. Also, another point to consider is variance. Higher the variance, better is discriminant power and vice-versa.

Hence feature (E5*E5) has strongest discriminant power whereas feature (W5*W5) has lowest discriminant power.

For unsupervised clustering, texture 9 was wrongly classified into class weave instead of Grass. Accuracy was 100% of the clustering when texture 14 image was used instead of texture 9.

We can reduce computations and improve accuracy by feature reduction using PCA.

TEXTURE SEGMENTATION

Task: Apply 9 laws filters to extract 9 grey images and use k means clustering to segment composite image into 6 cluster. for visual representation, divide the image into 6 grey levels.

ABSTRACT AND MOTIVATION

Texture segmentation is used widely in machine learning problem that needs K-means clustering to cluster the feature points by generating a codebook and extract neat features out of it and finally segment the image with color codes to represent a segmented image [6]. K means clustering is unsupervised where you do not know data assignment.

APPROACH AND PROCEDURES

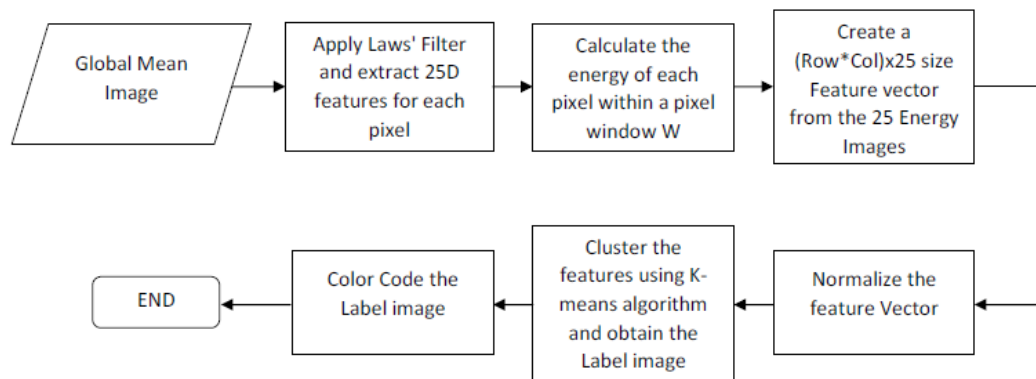


Figure 9 – Flow Chart for Image Segmentation

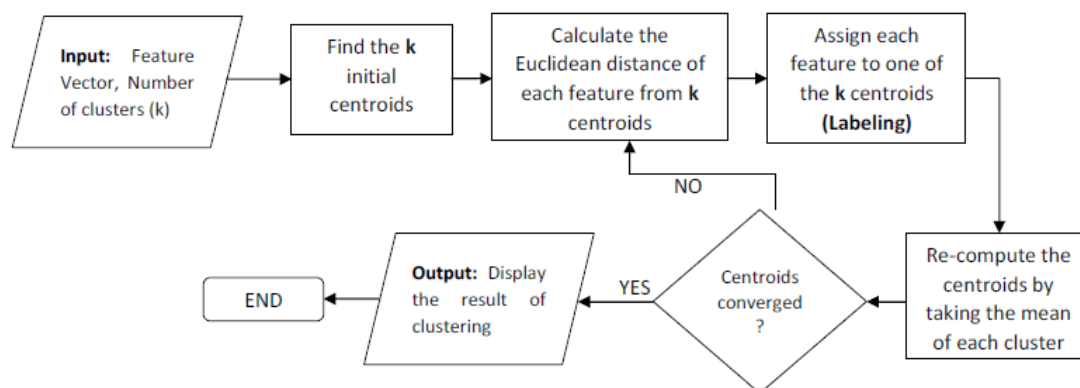


Figure 10 – K means clustering

Theoretical Approach:

1. Energy feature computation- We are supposed to use windowed approach to calculate energy for each pixel. I have tried various windows 13*13, 51*15, 25*25 etc. Typically window size of 13*13 or 15*15 is used.

$$Energy = \frac{1}{(WindowSize * WindowSize)} \times \sum_{i=0, j=0}^{i=Height, j=Width} (Img(i, j) * Img(i, j))$$

2. Energy feature normalization- Window size is 13 or 15. The feature extracted by L5L5 is not useful for texture classification and segmentation. Hence use this feature to normalize the data.
3. Segmentation- Using K means clustering, segment the composite mixture image into 6 grey levels.

EXPERIMENTAL RESULTS

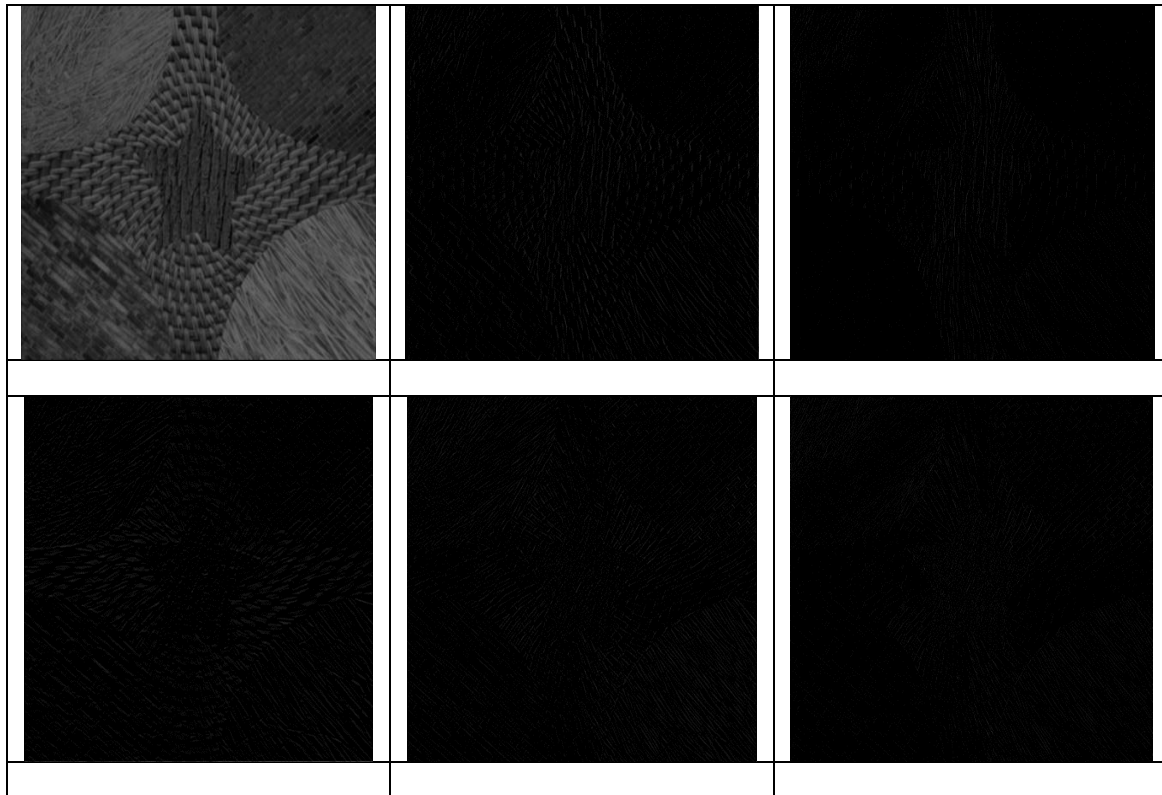




Figure 11 – 9 Gray scale images using Laws filters

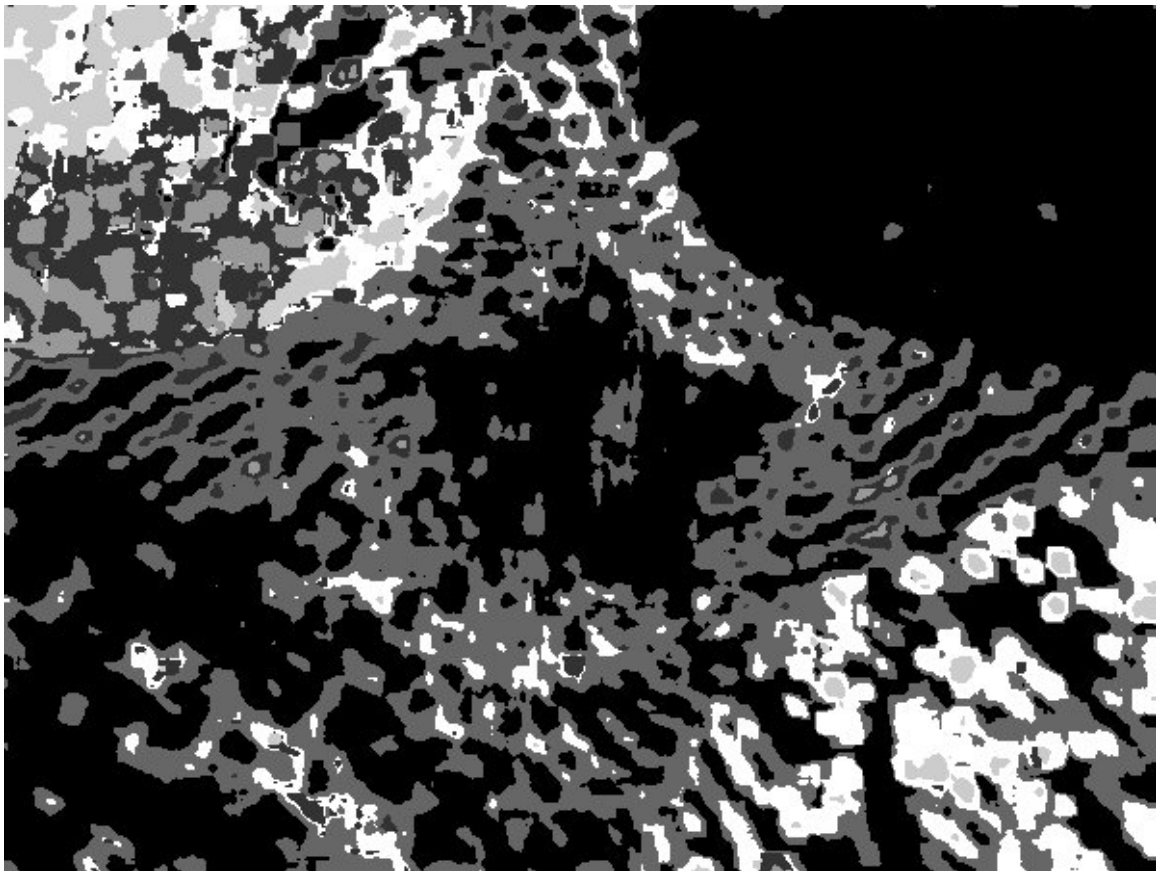


Figure 11 – Normalized Segmentation Output

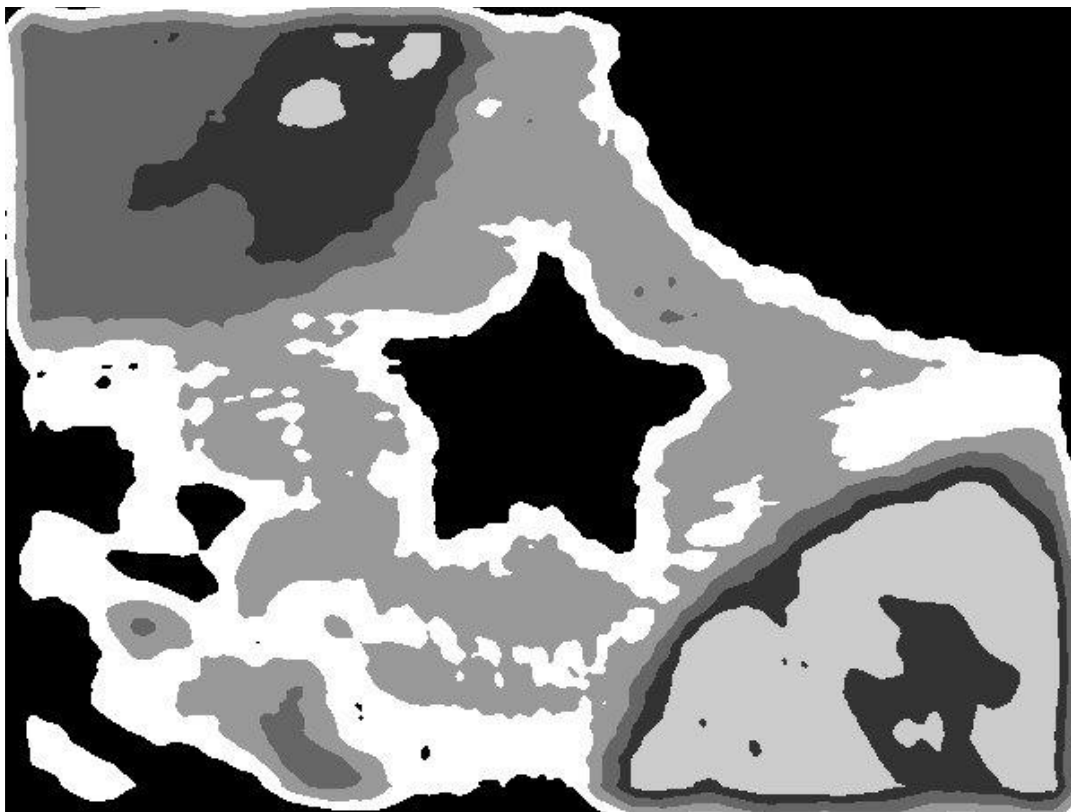


Figure 11 – Segmentation Output

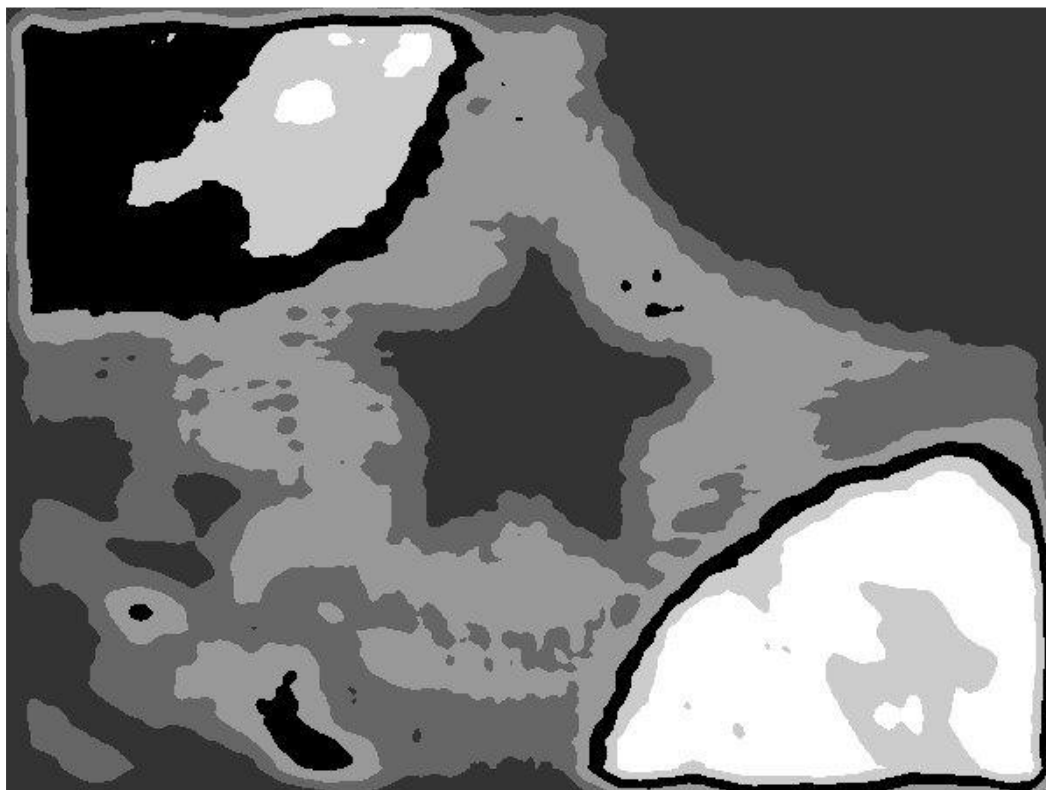


Figure 12 – Segmentation Output $W = 31$

DISCUSSION

The final segmented output is shown in figure 11. I have tried many windows and for window size 15*15, I got this better output. Segmented image still lacks the clarity. As window size increases, segmentation result gets better. But after certain point, it start connecting different regions resulting into bulgy and blocky output.

I have normalized using L3L3. The segmented output using normalization isn't great. It is shown. Hence, I decided not to go forward with normalization. So, I implemented K means for these 9 features instead of 8.

There is tradeoff between smoothness and accuracy of the segmentation. I observed this as I increase window size, accuracy reduces but smoothness increases i.e. segmentation loses its details.

ADVANCED – PCA & POSTPROCESSING

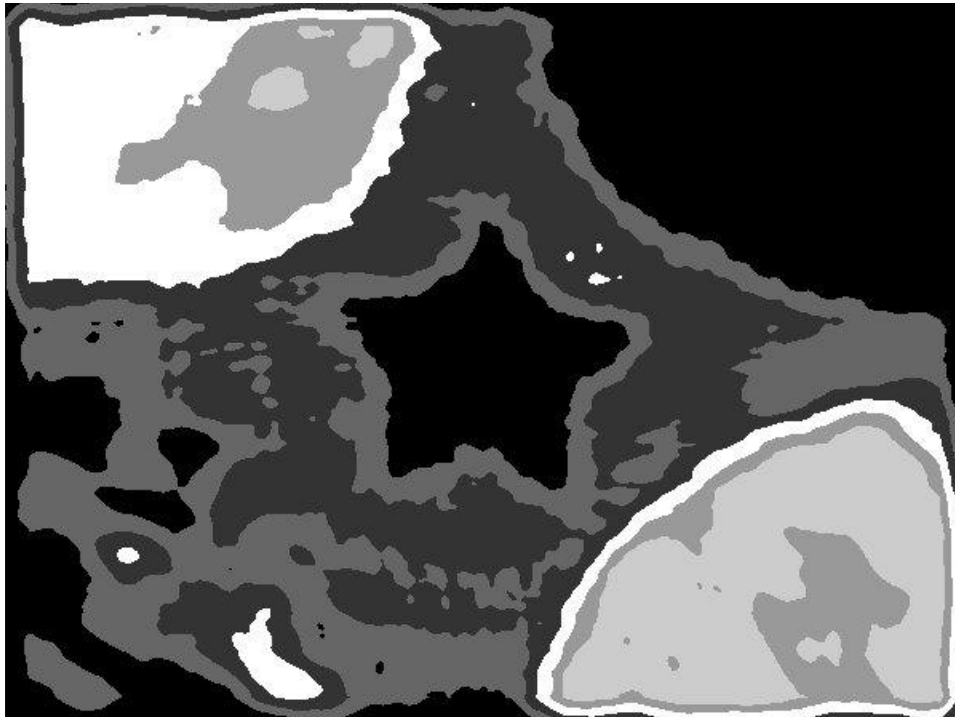


Figure 13– PCA output for 16 features

DISCUSSION

I used 25 5x5 Laws Filters and use the Principal Component Analysis (PCA) method to reduce the dimensions of the feature vectors for Window Size=15.

For PCA, I choose number of features to be $K=5$ & 16 for window size 15 for which I got the best output. Output with 16 features has been shown in figure.

To improve further results, I would prefer advanced texture segmentation. It includes two spirals for coarse and fine tuning of parameters for segmentation.

PROBLEM 2 – EDGE DETECTION

BASIC EDGE DETECTOR

Task: Implement the sobel and LoG edge detectors to clean and noisy boat images

ABSTRACT AND MOTIVATION

Edge and contour detection are the fundamental operations tackled in Image processing and Computer Vision algorithms. These techniques are used to extract structural information from the image to reduce data to be processed. Edges are the location where there is sudden change of intensity and hence frequency domain, it is considered as high frequency. Many filtering techniques are used to reduce high filtering noise in variety of applications like object detection, object tracking and image segmentation.

For this part, we will implement sobel edge detection and Laplacian of Gaussian edge detection and carry out performance analysis.

APPROACH AND MOTIVATION

SOBEL EDGE DETECTION: The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically, it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image [3].

Sobel operator consists of pair 3*3 pair convolutional kernels. One is rotation of another by 90°.

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

Gx

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

Gy

Figure 14 – Sobel Kernels

Gx and Gy are tensor products of 2 row matrices. X coordinate defined here is increase in right direction and y as down direction. At every point, gradient values in each direction has been calculated. These values are then combined to find absolute gradient at each point.

Gradient magnitude is given by

$$|G| = \sqrt{G_x^2 + G_y^2}$$

Angel of orientation is given by

$$\theta = \arctan(G_y/G_x)$$

DESIGNED ALGORITHM FOR SOBEL:

STEP 1 – Read the input image and convert it to gray scale.

STEP 2 – Construct the two sobel kernels Gx and Gy

STEP 3 – Multiply gray scale image by 2 kernels to find orientation is x and y direction and find gradient magnitude by the formula stated.

STEP 4 - Find max and min value obtained for magnitude of gradient. Using these values, normalize GX and Gy between 0-255.

STEP 5 – Find CDF of magnitude array to find threshold with desired probability and using this threshold, binarize the output edge image.

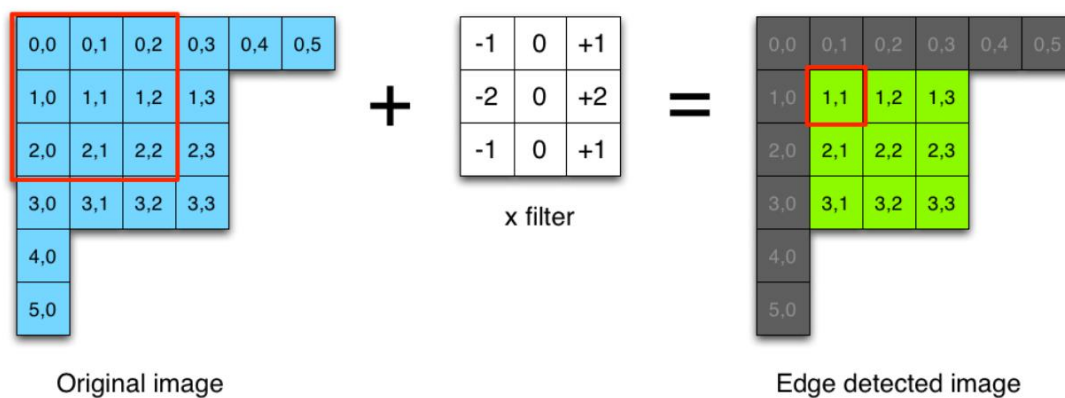


Figure 15 –Overview of Sobel edge detection

LAPLACIAN OF GAUSSIAN (LOG) EDGE DETECTION: The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single grey level image as input and produces another grey level image as output [4].

Laplacian $L(x,y)$ of an image with pixel intensity $f(x,y)$ is given by

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

To include smoothing Gaussian filter, combining Laplacian and Gaussian functions-

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

Since input image is represented in set of discrete pixels, approximated discrete convolutional kernel of second derivatives of Laplacian has been calculated. Two commonly used kernels are

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 16 – Laplacian Kernels

DESIGNED ALGORITHM FOR LOG:

STEP 1 – Read the input image and convert it to gray scale.

STEP 2 – Construct the LoG kernel shown in figure

STEP 3 – Multiply gray scale image by kernel to find gradient value. Now find histogram of this array.

STEP 4 - From histogram, find 2 knee points by calculating slope at every point and if slope is greater than a fixed value, knee point is selected.

STEP 5 – LoG operator takes second derivatives and it will be zeros when image is uniform and for change, it will give positive response for darker side and negative for lighter.

STEP 6 – If pixel intensity falls below negative knee point, assign it gray value as 64 for ternary edge map. If it is greater than positive knee point, assign it 192 else 128.

STEP 7 – From this edge map and ternary map, use pattern to detect any edges and assign it as 255 else 0.

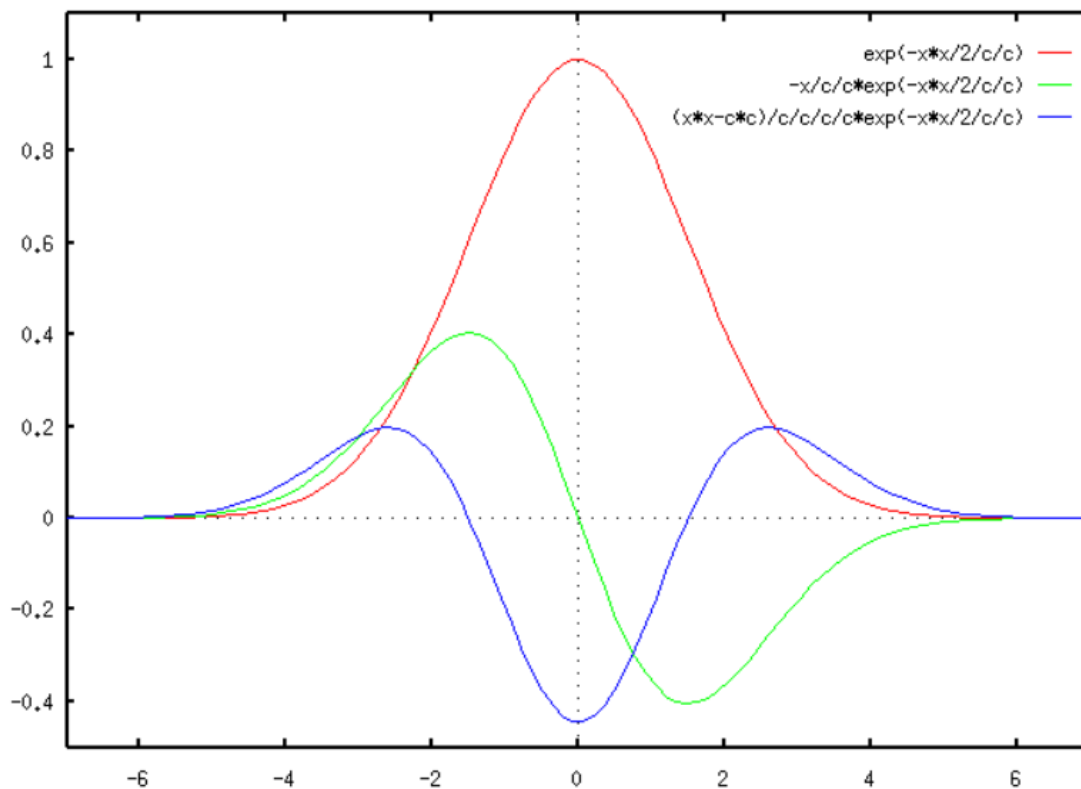


Figure 17 – Gaussian and its first and second derivatives

The edges in the image can be obtained by these steps:

Applying LoG to the image, detection of zero-crossings in the image and threshold the zero-crossings to keep only those strong ones.

The last step is important as it is needed to suppress the weak zero crossings mainly noises.

EXPERIMENTAL RESULTS

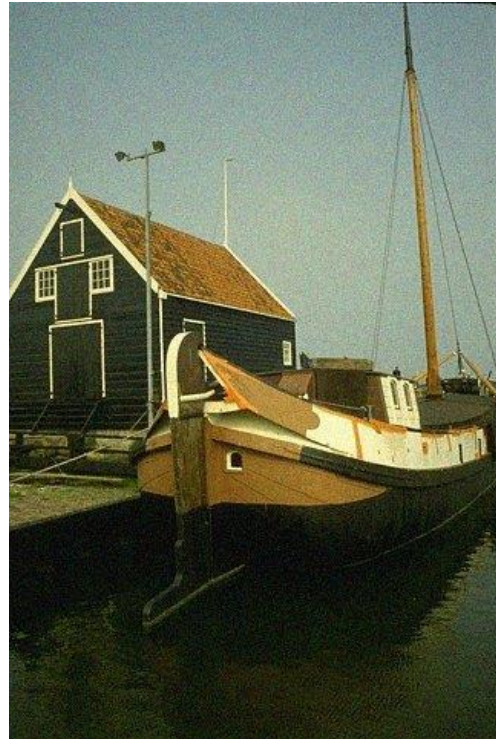


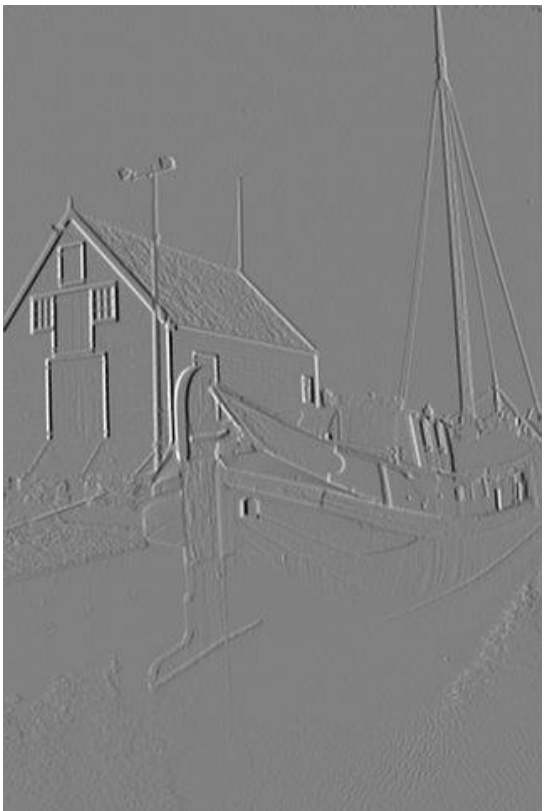
Figure 18 – Input Boat and Noisy Boat Image and their gray scale images



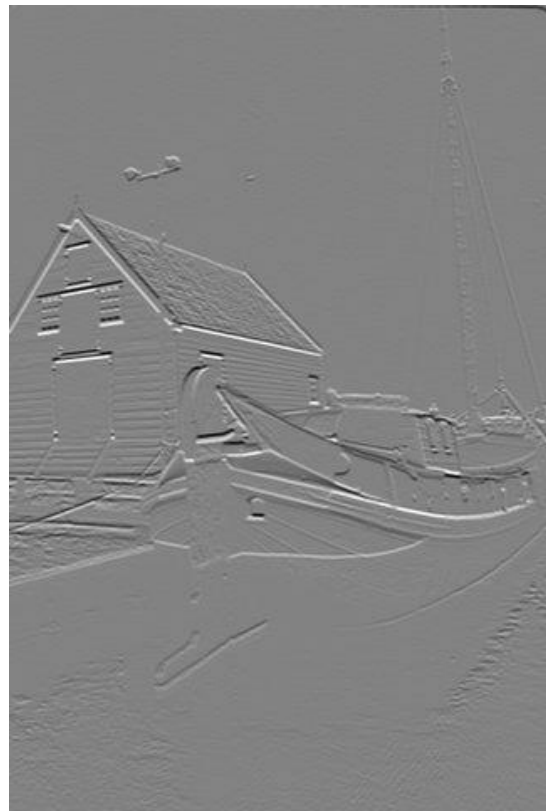
Boat Edge Output



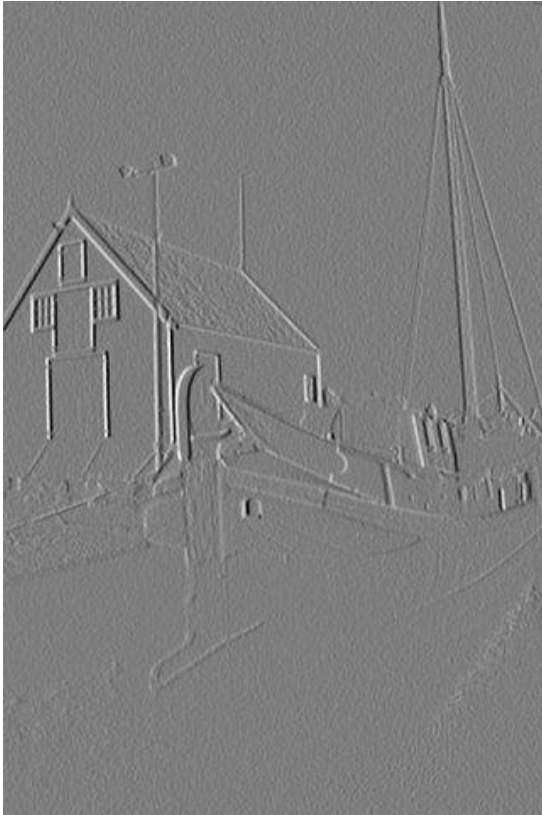
Noisy Boat Output



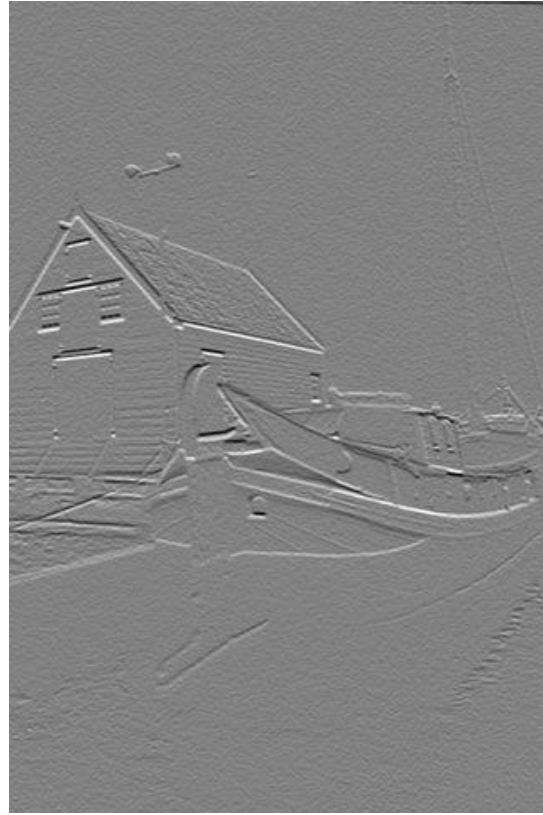
Gradient X of Boat



Gradient Y of Boat



Gradient X of Boat Noisy



Gradient Y of Boat Noisy

Figure 19 – Sobel Output for Boat and Noisy Boat



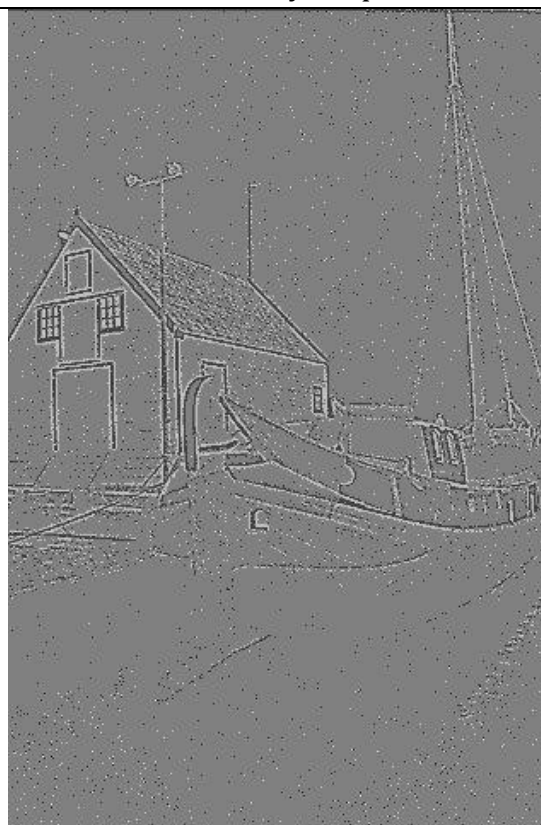
Boat Output



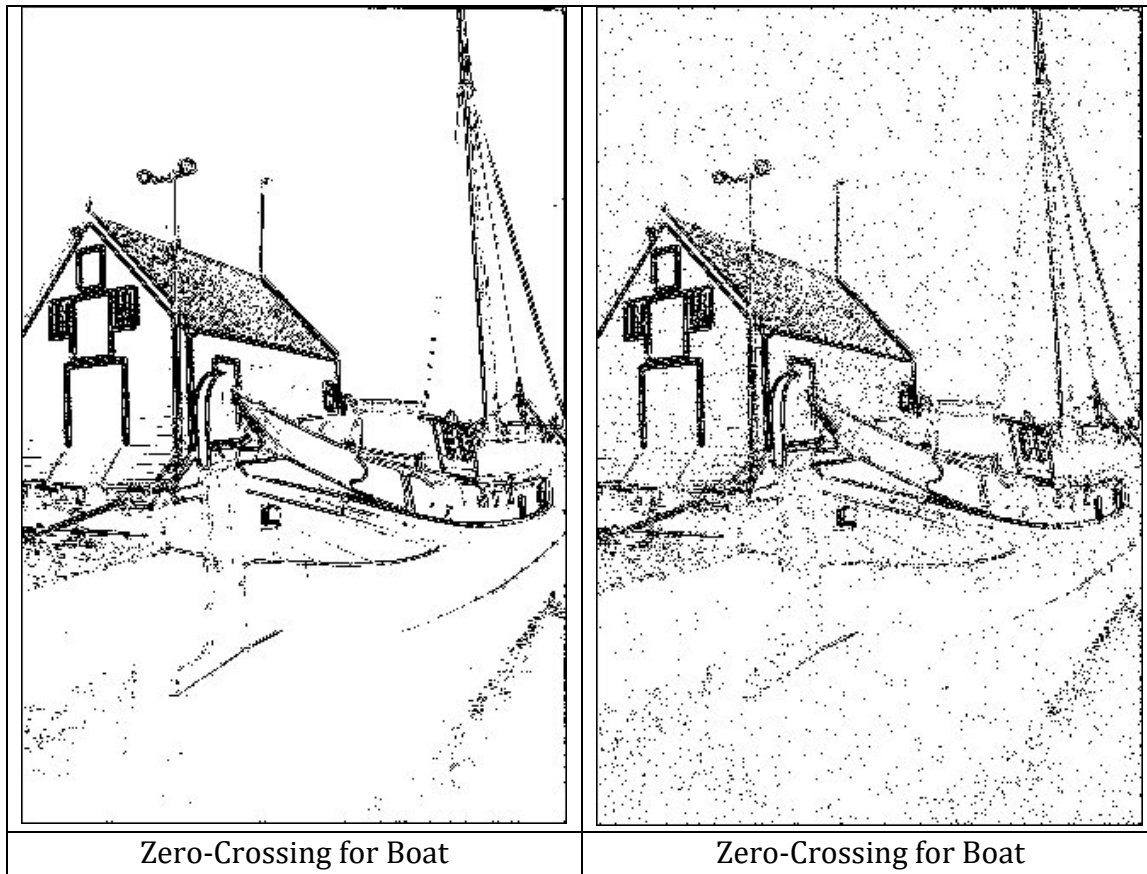
Boat Noisy Output



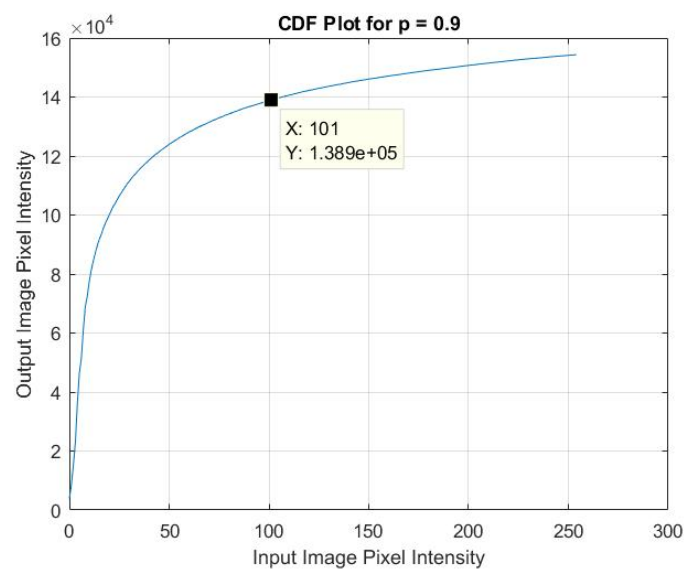
Ternary Output for Boat



Ternary Output for Boat Noisy



The table shows various output stages for noisy and original image. Magnitude of image has been normalized using CDF method.



DISCUSSION

The figure above shows output for all the stages for LoG and sobel edge detector.

Edge detection shows low and high threshold combinations. Because of very bright texture of water in front boat, we see loss of edges for boat for that part.

For sobel detector, original boat image has better accuracy but most of the noise from image has been successfully removed from the image and observing from edge and final images, Boat and Boat_noisy has almost similar edge detection response. We can tune gradient assumption so get better output. For Noisy boat, sobel has far better output than LoG.

We can definitely tune the parameters to like sigma or threshold to increase or decrease accuracy of edge detection. The smaller the value of sigma, better are the intricate detailed edges observed. I have used by default value of sigma which is square root of 2. The LoG filter has better edge detection accuracy because it smoothens the image to remove noise hence widely used in real time applications. Since LoG is derivative filter, it is used to find areas of rapid changes in an image. Laplacians are computationally faster to process and compute the result. Ternary image is found out by comparing patterns as stated in algorithm above. The knee location is found out using threshold calculated for 10% value around zero crossing. From ternary image, edges are found out by traversing entire image and comparing with predefined pattern.

Edge detection can be improved by using canny edge detection. It models edges as sharp changes in brightness between pixels. This method is based on point pixel property and expansion of sobel detector with an additional layer of intelligence. This algorithm finds all the edges at minimal error rate.

STRUCTURED EDGE

Task: Apply structured edge detector to extract edge segments

ABSTRACT AND MOTIVATION

Contour detection and structured edge are widely used in computer vision and image segmentation algorithm. Traditional algorithms like LoG, Canny uses gradient, orientation combined with non-minimal suppression has several disadvantages. Structured forest is modern and state of art algorithm used to find edges. This algorithm is based on image patch property. For every image patch, it takes structural advantages to learn about image behavior and efficiently detect edge. It takes likelihood pixels present in an image patch whether they are edge or point.

These patches are classified into tokens using random forest classifier. Random forests are collection of T decision trees. For every input sample, different trees and ensembled into one model producing one output.

Structured edge detection algorithm is given below-

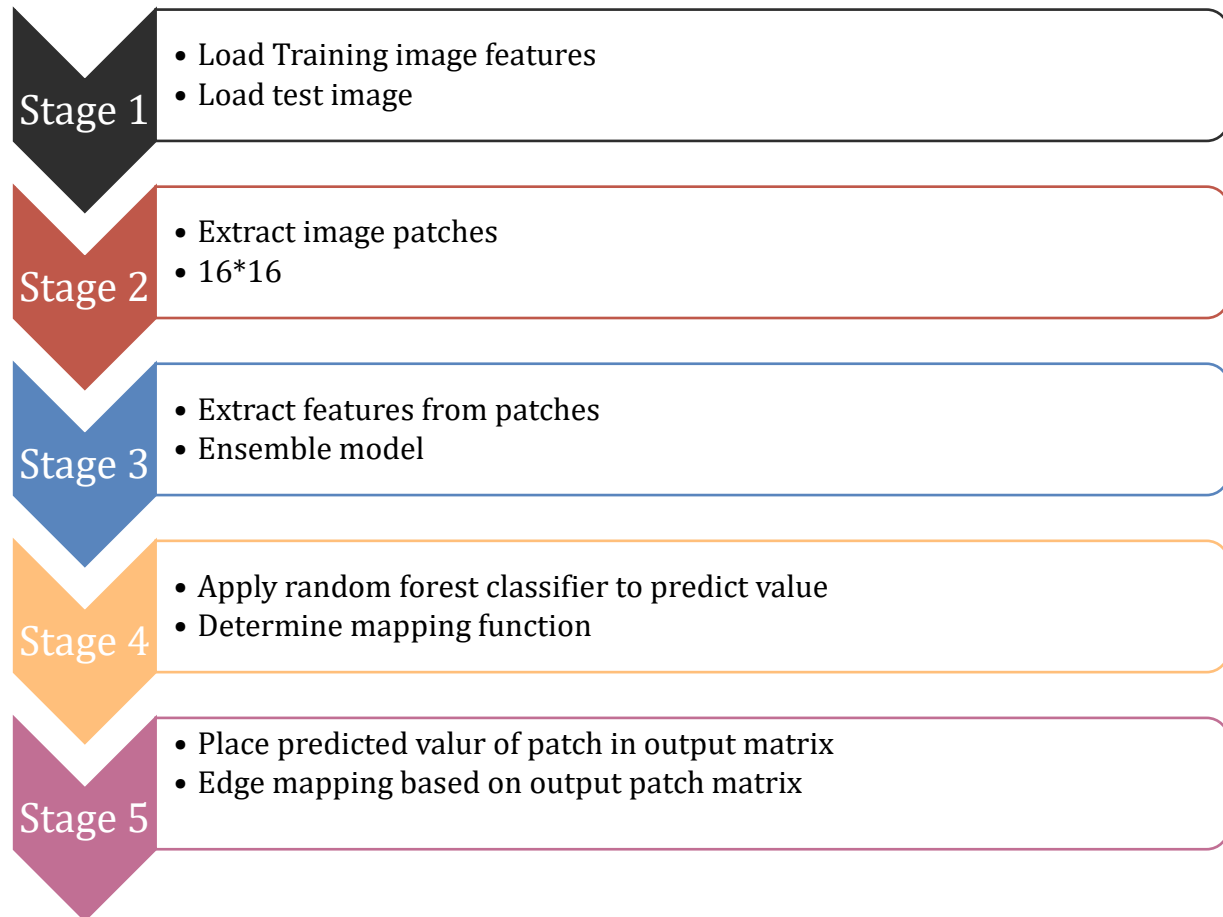


Figure 18 – Structured edge algorithm

APPROACH AND PROCEDURE

The random forest is based on divide and conquer approach. The basic idea is to put set of weak rule base combined to form strong learner i.e. rule base. The basic element is decision trees which is weak learner and they are ensembled together.

The random forest is trained by local image patches and ground truth images and forming a structured output having condition whether certain combination would classify as edge or not. After the model has been trained by number of images, it is then tested using obtained 16 segments and test image and segmented output is obtained

from classifier. In decision tree, an input that belongs to space A as an output that belongs to a space B. The input or output space can be complex in nature. For example, we can have a histogram as the output. In a decision tree, an input is entered at the root node. The input then traverses down the tree in a recursive manner till it reaches the leaf node [4].

A set of such trees are trained independently to find the parameters in the split function that result in a good split of data. Splitting parameters are chosen to maximize the information gain [4].

EXPERIMENTAL RESULTS:

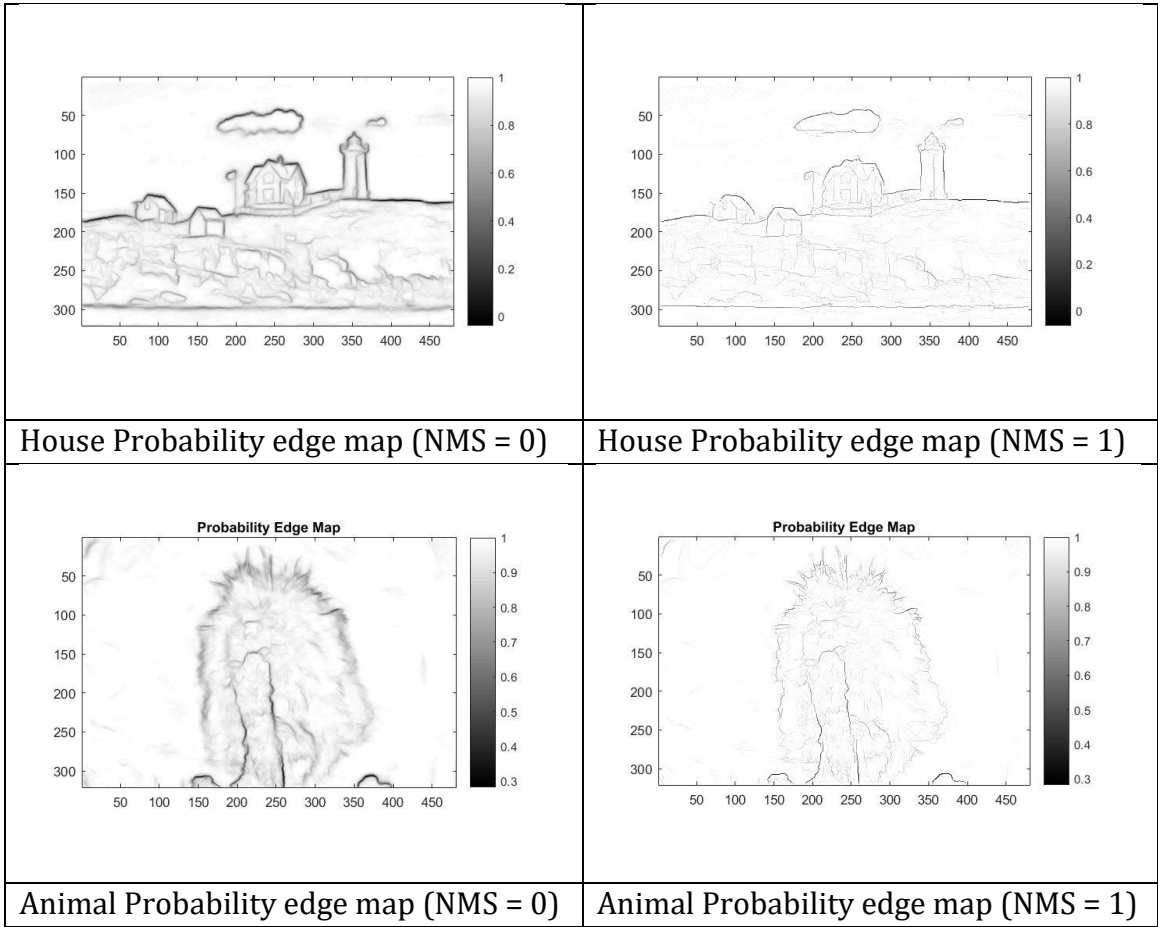
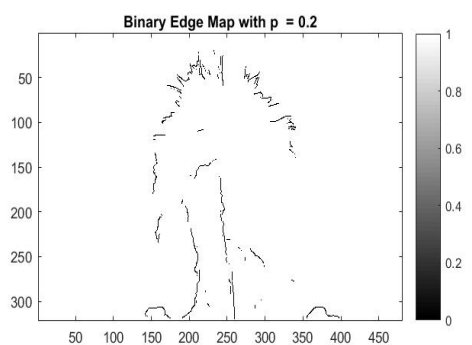
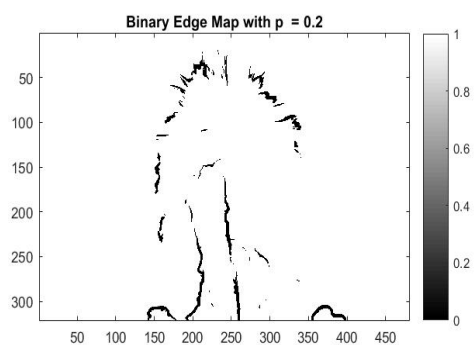
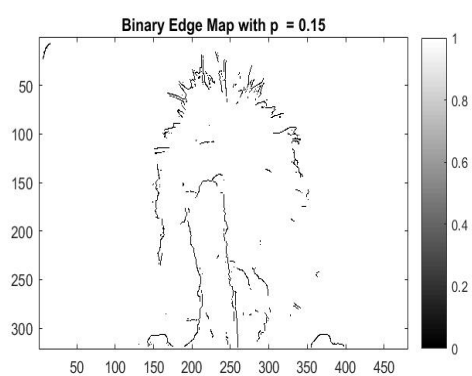
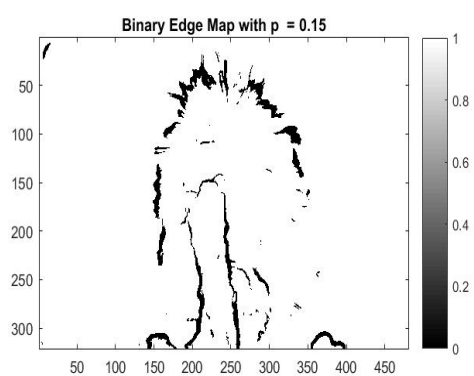
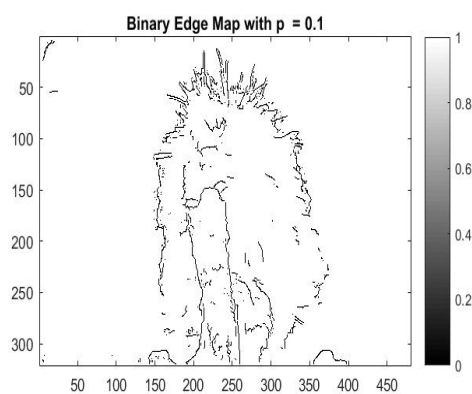
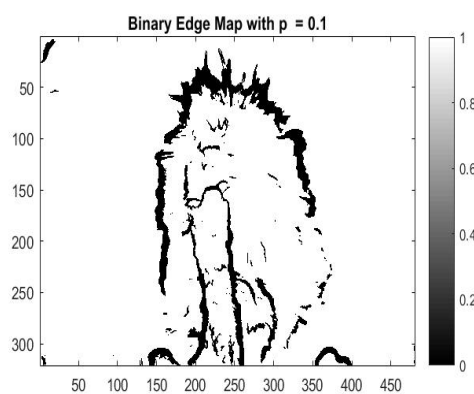
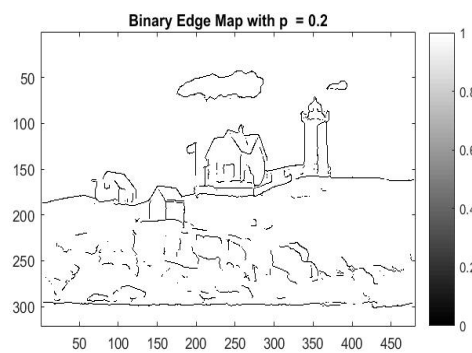
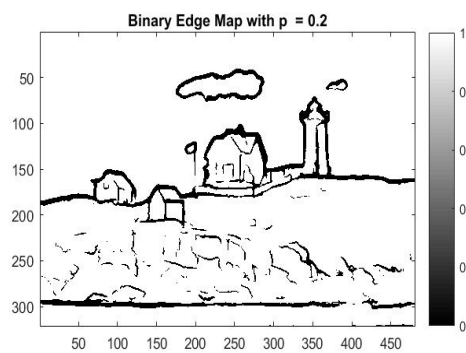
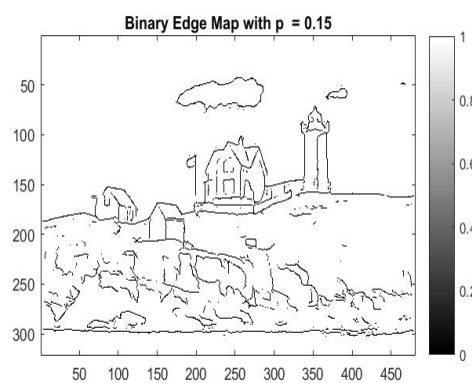
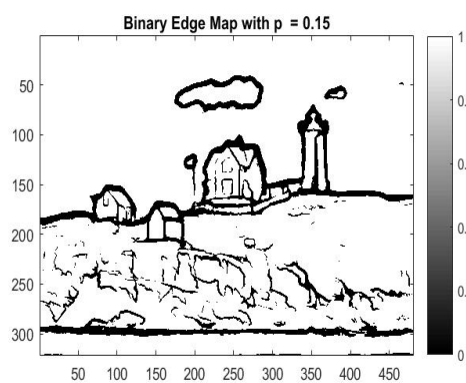
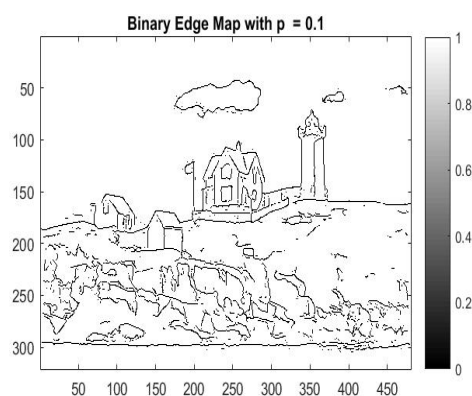
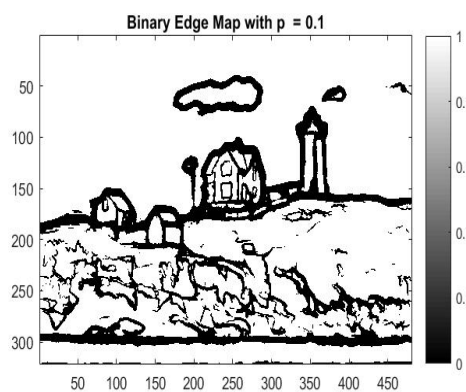


Figure 20 – Probability Edge Map



Animal Binary Edge Map (NMS=0) Animal Binary Edge Map (NMS=1)



House Binary Edge Map (NMS=0)

House Binary Edge Map (NMS=1)

DISCUSSION

Figure shows output for House and Animal of probability edge map as well as binary edge map for various thresholds.

The parameters used to set probability images are

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

Figure 21– Parameters for edgesDemo.m

Structured edge performs very well for edge detection even on the visual front. Owing to its extensive algorithm, it has intelligent layer that allows it to display only important edges thereby reducing noise and unwanted edges. It has better performance than canny, sobel and Log edge detection method. This algorithm is a fast-paced algorithm. This is because other algorithms are not intelligent and depend upon gradient, orientation and many other parameters.

For less value of sharpen parameters, edge map has diffused edges. To tackle this problem, I have kept sharpen = 2 to get sharp edges. Multiscale parameter is set to 1 to make algorithm run edge detector on original size I, half resolution of I and double resolution I. Tree evaluation parameter is set to 4 which enables algorithm to select set of 4 alternating trees from available 8. Edges can be thinned using NMS parameter hence output for both NMS = 0 and 1 is shown.

Structured edge mimics human handwriting and hence widely used in industry.

PERFORMANCE EVALUATION

Task: Perform quantitative comparison between edge maps obtained from different edge detectors.

ABSTRACT AND MOTIVATION

We can find accuracy and performance of different edge detectors using mean recall, precision and F measure concept. These terms are used to quantify the performance of different machine learning algorithms.

Edge detection algorithms can never come close to edges detected by humans. Ground truths are the edges detected by humans. To cover perceptions of all humans, we are comparing edge map with various ground truths to get a better idea about performance of algorithm. So we take mean performance of ground truth.

From precision and recall, F can be calculated as

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

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

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}.$$

Where,

True Positive-Edge pixels in the edge map coincide with the ground truth.

False Positive- Pixels in the edge map coincide with the non-edge pixels ground truth.

True Negative- Non- edge pixels in the edge map coincide with the non-edge pixels ground truth.

False Negative- Non- edge pixels in the edge map coincide with the edge pixels ground truth [5].

APPROACH AND PROCEDURES

I have used pDollars toolbox to implement this code.

In MATLAB, performance evaluation is done by:

[thrs, cntR, sumR, cntP, sumP, V] = edgesEvalImg(BinaryMap, ground truths, varagin);

where,

thrs - the threshold values

P = cntP./sumP

R = cntR./sumR

F = From formulae.

EXPERIMENTAL RESULTS

Structured Edge Detection F measure table for stated parameters:

Animal Image	Precision (P)	Recall (R)	F Measure (F)
GT1	0.6954	0.6151	0.6528
GT2	0.6324	0.6560	0.6440
GT3	0.4947	0.4036	0.4445
GT4	0.6786	0.7357	0.7060
GT5	0.5457	0.6886	0.6089
Mean	0.7379	0.6917	0.7141

House Image	Precision (P)	Recall (R)	F Measure (F)
GT1	0.6477	0.8576	0.7380
GT2	0.7704	0.6594	0.7106
GT3	0.7123	0.6975	0.7048
GT4	0.6075	0.7149	0.6568
GT5	0.5413	0.7499	0.6287
Mean	0.8223	0.7784	0.7998

House Image	Precision (P)	Recall (R)	F Measure (F)
Structured Edge	0.8223	0.7784	0.7998
Sobel	0.2306	0.8868	0.3660
LoG	0.1873	0.9519	0.3130
Canny	0.2109	0.9115	0.3425
Prewitts	0.2331	0.8769	0.3683

Animal Image	Precision (P)	Recall (R)	F Measure (F)
Structured Edge	0.7379	0.6917	0.7141
Sobel	0.1877	0.8594	0.3081
LoG	0.1778	0.9803	0.3010
Canny	0.1817	0.9811	0.3066
Prewitts	0.1878	0.8504	0.3076

DISCUSSION

Tabulated results of every value F, P and R has been described for every case for House and Animal image.

F measure gives us evaluation of performance of edge detection. Based on F values, we can say that structured edge has highest value and hence best performance for edge detection.

F measure is image dependent. By intuition, we can say that it is easier to get higher F score. House has well defined edges than Animal. House image than Animal image because it is simpler to segment the house image as it has quite a big distance from source as well as small dimensions. But on the other hand, animal as object has higher dimension than house as well as it is very close to source hence a bit difficult to segment. Another reason is house less number of objects in background compared to animal image.

F measure is measurement of test accuracy in statistics. It is dependent upon both P and R value. If any of the value is low or has huge difference in between those, then that results into bas F score. Hence there always in trade-off between P and R value.

Assume $P + R = C$.

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

$$\text{Hence } F = 2 * (R - C) * R / C$$

Taking derivative wrt R , we get $4 * R = 2 * C$

Solving equations, we get

$$P = R.$$

F measure reaches maximum value when precision equals recall.

PROBLEM 3 – SALIENT POINT DESCRIPTORS AND IMAGE MATCHING

EXTRACTION AND DESCRIPTION OF SALIENT POINTS

Task: Extract and show SIFT and SURF features of input vehicle images.

ABSTRACT AND MOTIVATION

Salient point extraction and description is widely used for tasks specifically such as object recognition, video tracking, navigation, gesture recognition and image stitching. Feature detection is the process of computing the abstraction of the image information and making a local decision at every image point to see if there is an image feature of the given type existing in that point [1]. The features are usually invariant to image scaling and rotation, change in illumination and 3D camera viewpoint to some extent. In this section, we have used SURF and SIFT algorithms that extract local features stated above.

Scale Invariant Feature Transform (SIFT) is a feature detector developed by Lowe in 2004. Although SIFT has proven to be very efficient in object recognition applications, it requires a large computational complexity which is a major drawback especially for real-time applications. Speed up Robust Feature (SURF) technique, which is an approximation of SIFT, performs faster than SIFT without reducing the quality of the detected points [1]. SIFT and SURF both uses descriptor and detector. The performance and efficiency on both algorithms has been studied depending on the features.

APPROACH AND PROCEDURE

The overview for extraction of features using SIFT algorithm is given below.

SCALE SPACE EXTREMA DETECTION: This is initial step. To ensure key points scale invariant, we consider images represented on different scale. The scale space consists of n octaves where each octave is down sampled by 2 every step. This is done by applying Gaussian filter with different sigma. After smoothing the images, differences of Gaussian (DoG) is taken. DoG is an approximation of LoG as calculating LoG is computationally very expensive and takes a lot of time.

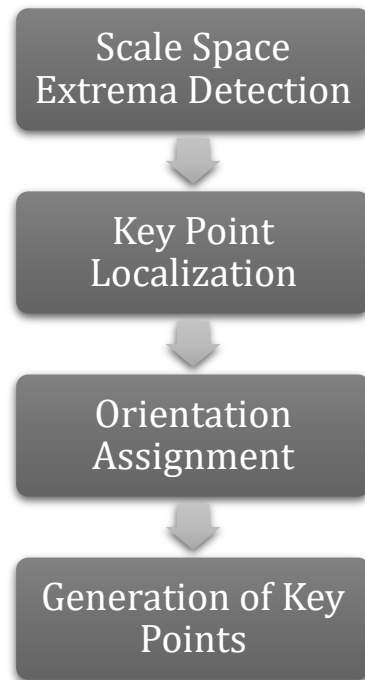


Figure 22 – The overview of SIFT algorithm

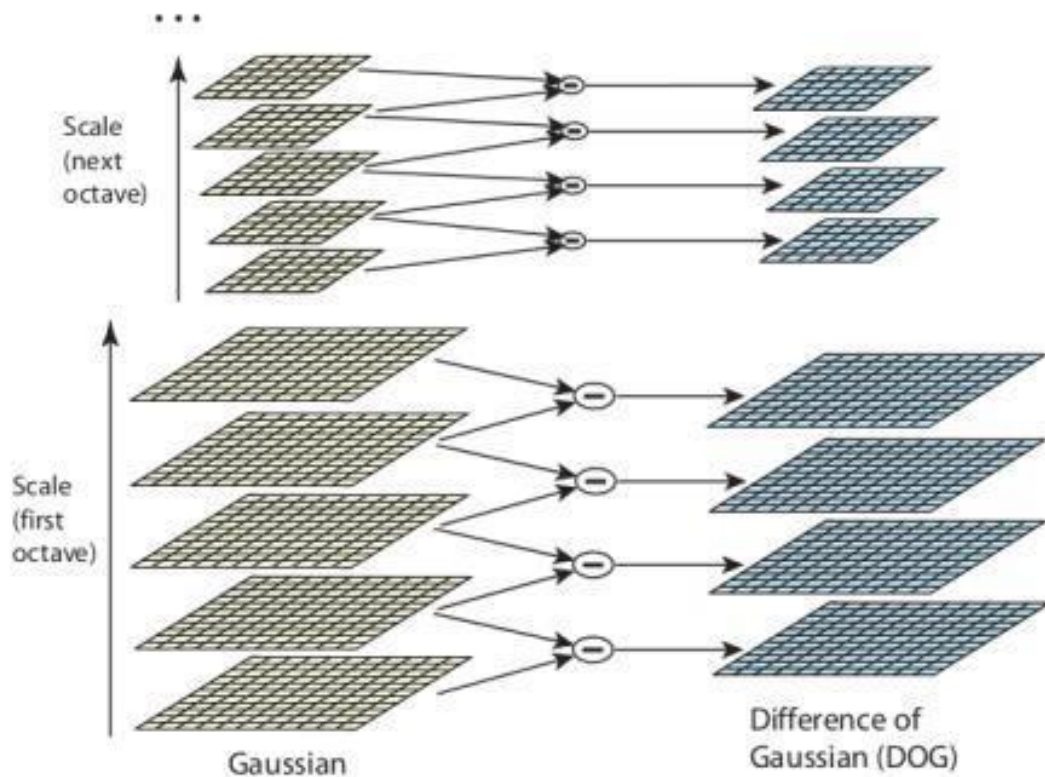


Figure 23 – Different scales of the same image and the Laplacian of Gaussian

For the Gaussian blurred image,

$$\frac{\partial G}{\partial \sigma} = \sigma \Delta^2 G \quad \text{Heat Equation}$$

$$\sigma \Delta^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \Delta^2 G$$

Typical values : $\sigma = 1.6$; $k = \sqrt{2}$

Figure 24 – Difference of Gaussian

KEY POINT LOCALIZATION: Using very fast approximations, we will find key points. The points of interest may be weak feature or part of an edge. To avoid such errors and outliers, we use Taylor series expansion to localize sub points needed. We locate extrema of DoG which can be done by scanning through each image and then identify minima and maxima.

We also computer Hessian matrix and compare its eigen value. Filter out low contrast points using scale space values at previous locations and filter edge responses using Hessian Matrix. Depending upon ratio of eigen vectors, we either discard or keep. It takes ratio of trace of determinant and according to heuristic values, it finds the maximum value.

ORIENTATION ASSIGNMENT: To obtain rotational invariance, we assign particular value to every key point found above. This is done by taking histogram of neighbourhood pixels. The neighbourhood size depends upon the scale of the point. Orientation of each pixels in window is weighted by its magnitude and gaussian weighted circular window with 1.5 times scaling factor. Max orientation value is found out using histogram. To improve the robustness, 80% of the maximum value points are considered.

KEY POINT DESCRIPTORS: We need to find blurred image of the nearest scale and then sample the points around key point found out in previous descriptor. Using the orientation computed previously, now rotate the gradients and coordinates with orientation. Now divide each region into several sub-regions with several bins. Once we select the key point orientation, the feature descriptor is then calculated as a set of orientation histograms on 4×4 -pixel neighbourhoods. We obtain orientation data from the gaussian image closest to the key point scale. The resolution of histogram has been set as 8. This forms 128 histogram bins in the 16×16 neighborhood. The 128 bins are

taken as a feature vector for a key point. The vector is normalized to tackle illumination effect.

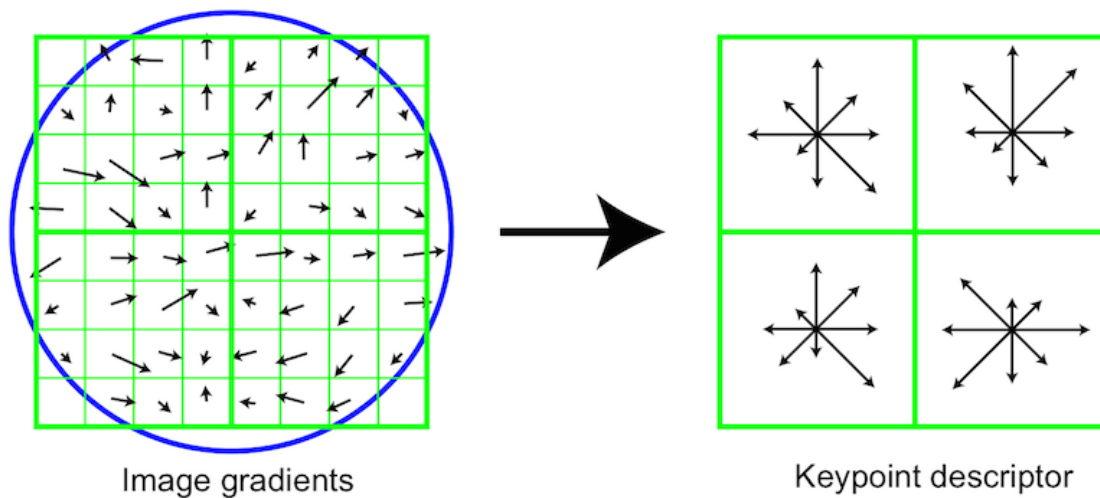


Figure 24 – The gradient orientation and Normalization (SIFT)

The overview for extraction of features using SURF algorithm is given below.

SURF stands for Speeded Up Robust Features which can be used for tasks such as object recognition, image registration, classification or 3D reconstruction. It is partly inspired by the (SIFT) descriptor. The standard version of SURF is several times faster than SIFT and is more robust against different image transformations than SIFT [1].

To detect key points, SURF uses integer approximation of Hessian blob detector. This is used to speed up the filtering process to obtain key points using Hessian matrix similar to SIFT. We create integral images in SURF because mask convolution is independent of filter size. Integral image is of the same size of the image to be analyzed and the value of integral image at any location can be determined as the sum of the intensity values lesser than or equal to the location where we evaluate the integral image.

SURF algorithm:

1. Finding interest Points- The determinant of Hessian matrix is found out using product of eigen values. We use second order LoG derivative for filter for Hessian matrix. We find sum of intensities for squares present in matrix by calculating sum of intensities multiplied by weight vectors of net sum. Evaluation of filter is done using threshold.
2. Major Interest Points- Use 3*3 NMS below and above scale space for every octave. Interpolation of points is done to obtain correct scale.

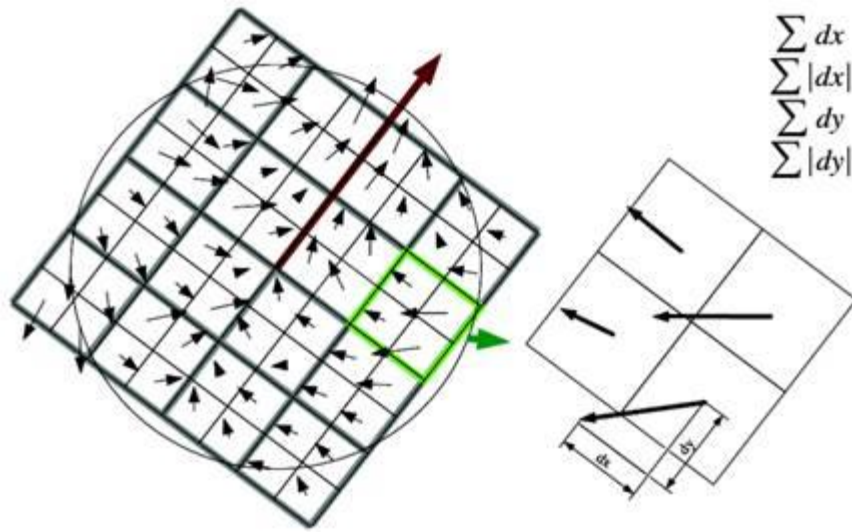


Figure 25 – SURF Descriptor

3. Feature Direction- Using Harr transform, we get pixels available in circular radius of 6 times scale space. From Harr transform, get direction of maximum weight to get maximum orientation.
4. Generating feature vectors- Get vectors for 16 sub regions generated from square description window. These are computed in direction of rotation.

EXPERIMENTAL RESULTS



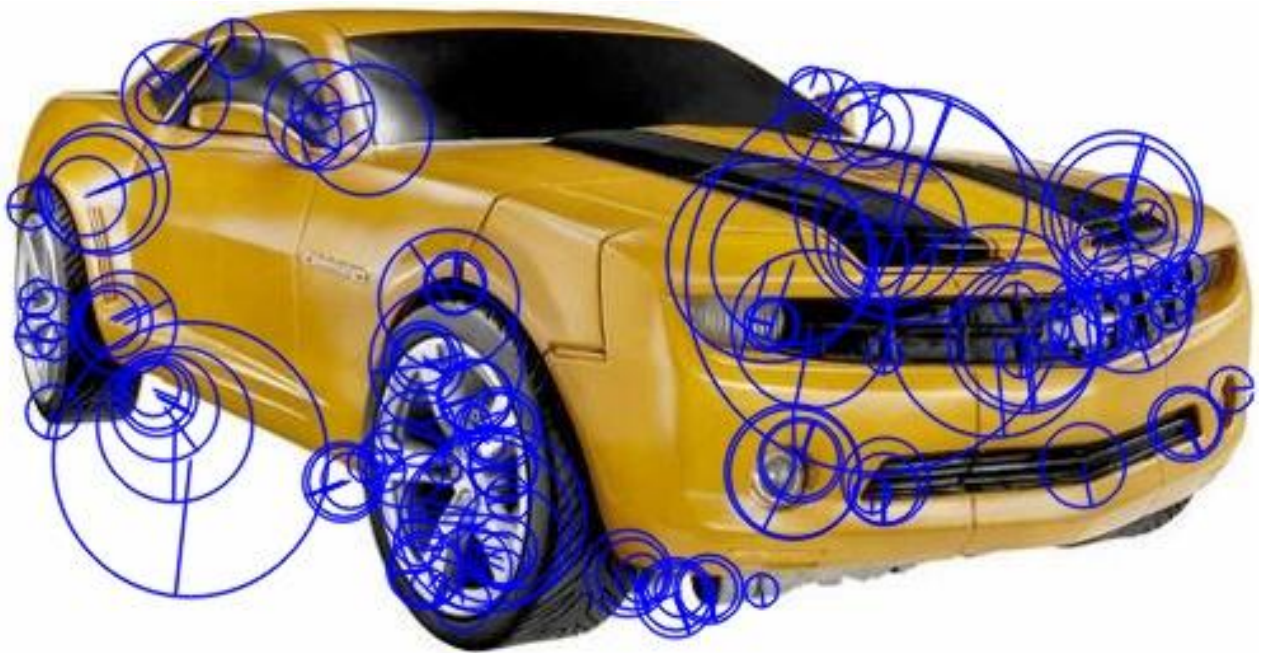




Figure 26 – SURF and SIFT features of input

DISCUSSION

The outputs for SIFT and SURF algorithms for image key point generation has been shown in image. The obtained key points are of same size irrespective of their importance or weight. We can check properties of last parameter using drawKeyPoints() function. It also shows the orientation of the key points. The size of circle denoted importance of features and its prominence. The number of key points can be varied by changing Hessian value for SURF and SIFT.

Figure shows features different Hessian matrix and key points. SURF is faster in computation than SIFT. SIFT achieves strong features (number of features) compared to SURF. SIFT is widely used even though little slow because of its ability to show more number of features. For smaller datasets, difference for execution is not that much.

SIFT	SURF
Uses DoG as scale space convolved with Gaussian	Uses integral image as space scale
Computationally expensive and slower than SURF	Computationally faster than SIFT with better key points

NMS is used to remove outliers from Hessian matrix	Hessian matrix is used for key points detection
Descriptors are calculated for 128D histogram	Descriptors are calculated with 64D using Harr transform
To find key points, 16*16 window is used	4*4 grid for sub-regions along with Harr response is used for
Pyramidal structures of DoG are used which may fail in case of similar backgrounds	Integral image is used which works better for illuminance change and blurring

IMAGE MATCHING

Task: Extract and apply SIFT and SURF features to object matching.

ABSTRACT AND MOTIVATION

To find relevant matches between 2 images, we generally extract features from images. Flann based matcher can be used to match key point from features calculated. Brute force is simplest and basic matcher. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

FLANN stands for Fast Library for Approximate Nearest Neighbors. It contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features. It works faster than BFMatcher for large datasets. For FLANN based matcher, we need to pass two dictionaries which specifies the algorithm to be used, its related parameters [2].

APPROACH AND PROCEDURES

After feature extraction using OpenCV, I have used Flann based matcher.

DEVELOPED ALGORITHM for Image matching using SIFT:

STEP 1 – Read the input image.

STEP 2 – Detect and compute key points and descriptors in both images using SIFT detector.

STEP 3 – Use Flann based matcher to match key points and draw the correspondence using drawMatches()

STEP 4 – Select best edges based on ratio test mentioned in paper (ratio = 0.75)

STEP 5 – Display the image with key point and matching lines.

DEVELOPED ALGORITHM for Image matching using SURF:

STEP 1 – Read the input image.

STEP 2 – Detect and compute key points and descriptors in both images using SURF detector with Hessian parameter set as 400.

STEP 3 –Use Flann based matcher to match key points and draw the correspondence using drawMatches()

STEP 4 – Select best edges based on ratio test mentioned in paper (ratio = 0.75).

STEP 5 – Display the image with key point and matching lines.

EXPERIMENTAL RESULTS

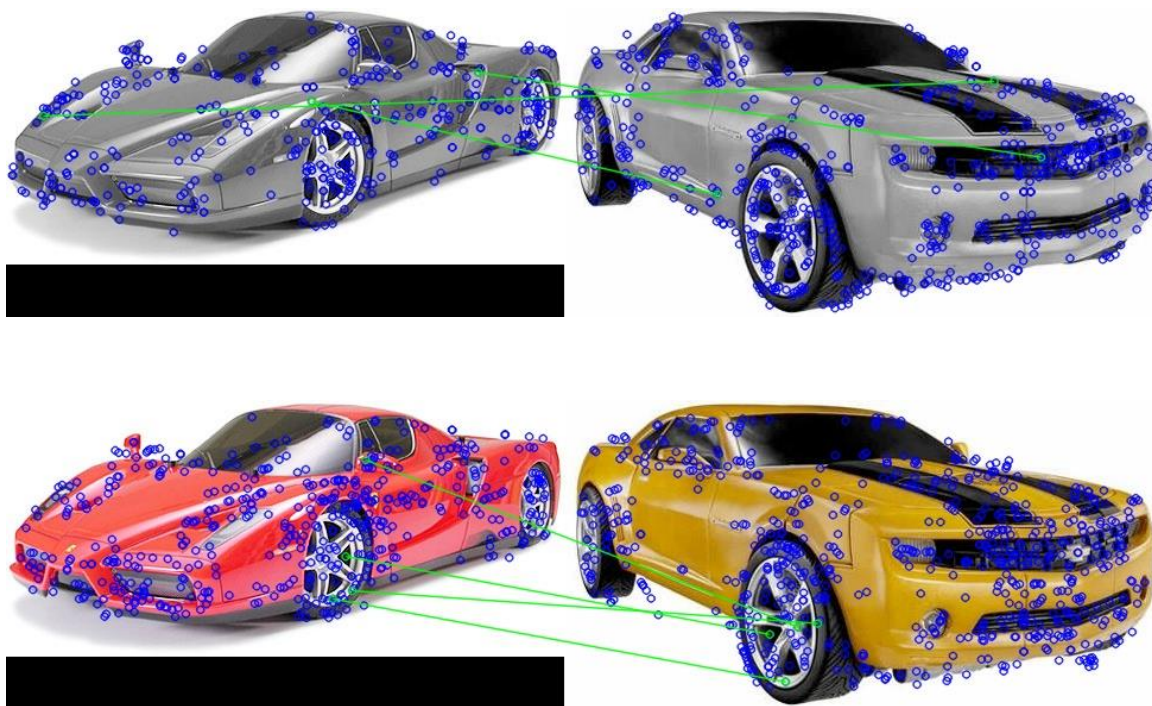


Figure 27 – SIFT and SURF matching between Ferrari 1 and Bumblebee using Flann Based Matcher

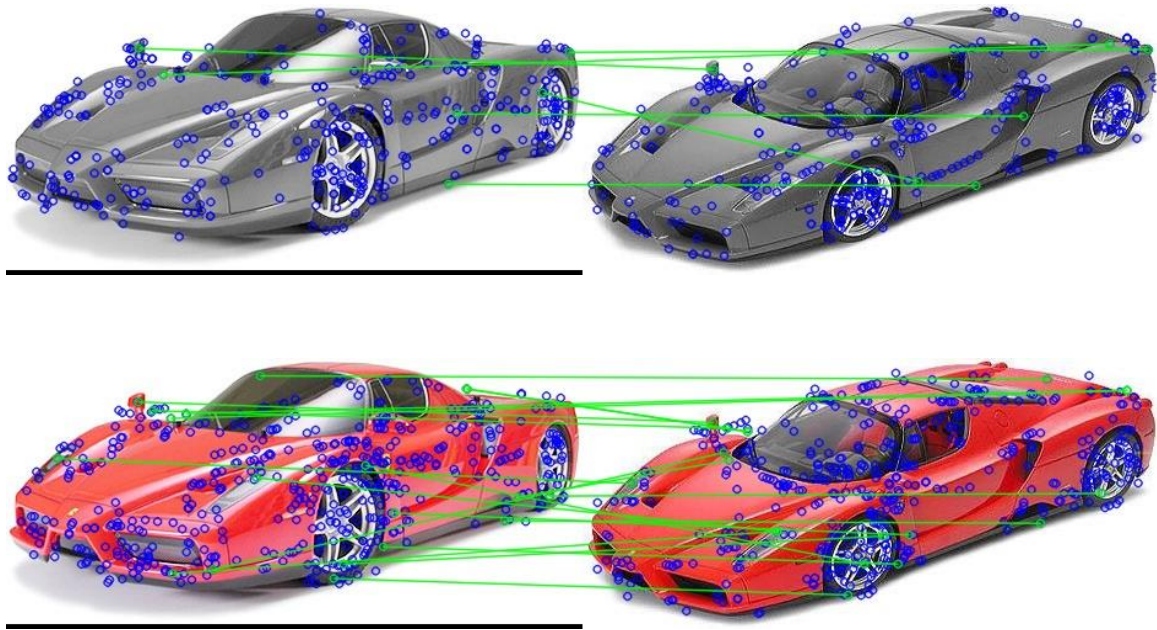
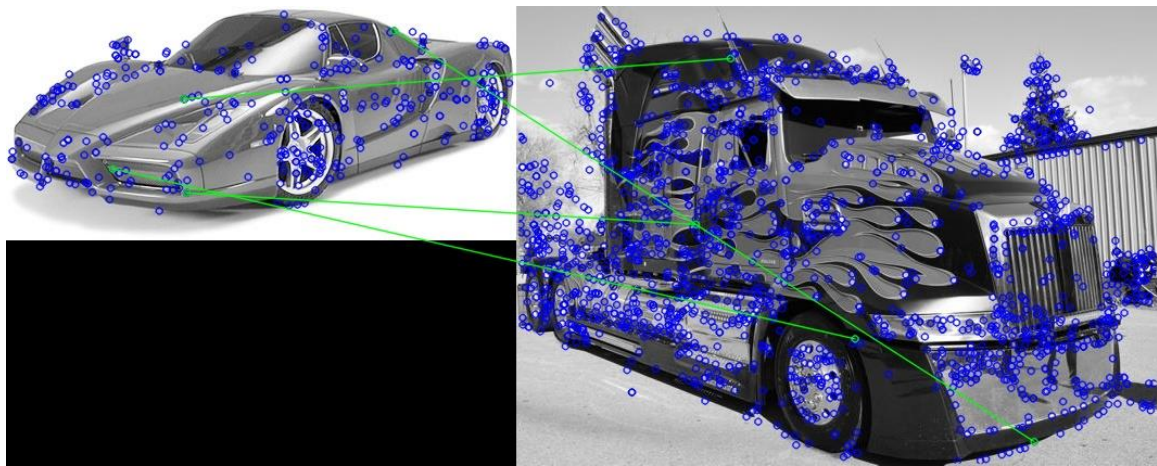


Figure 28 – SIFT and SURF matching between Ferrari1 and Ferrari 2 using Flann Based Matcher



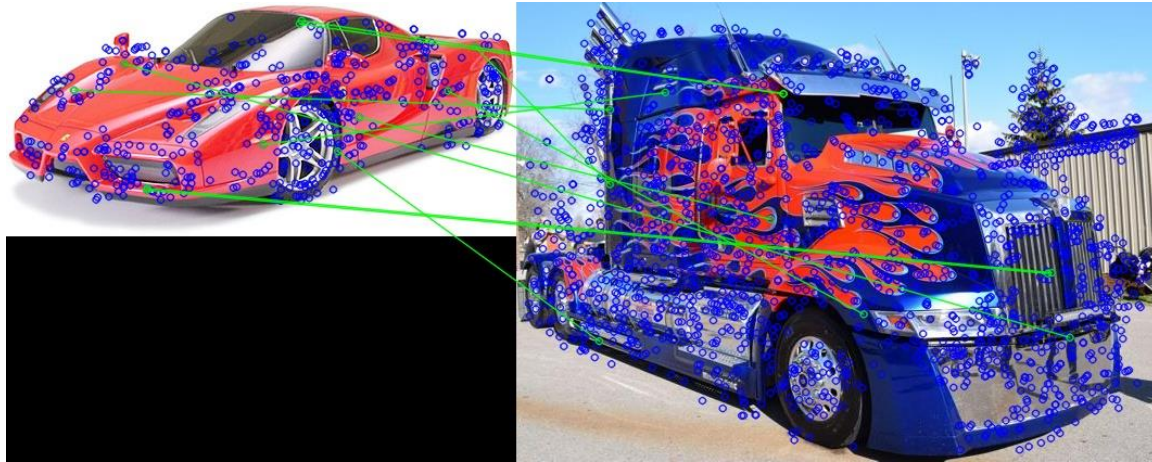


Figure 29 – SIFT and SURF matching between Ferrari1 and Optimus Prime using Flann Based Matcher

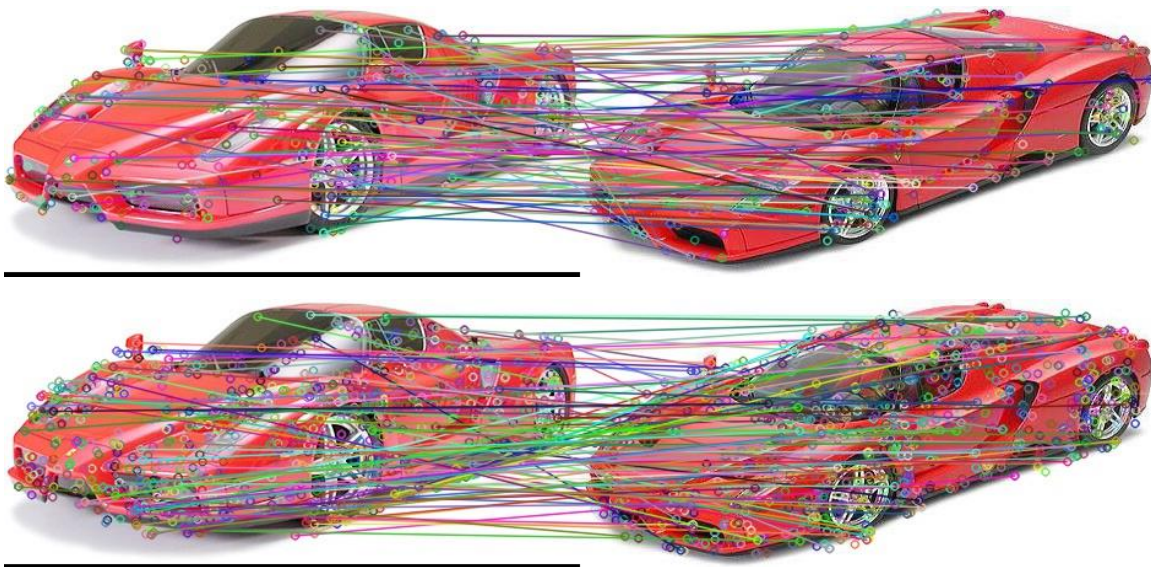


Figure 30 – SIFT and SURF matching between Ferrari1 and Ferrari 2 using Brute force matcher

DISCUSSION

All the images displayed above has only correspondence of few top features having ratio greater than 0.75 using ratio test. It means if number of matched features crosses a certain threshold, then the two images match else do not. For input image, matching of all the features is done and threshold on certain matching is set. For feature matching, multiple algorithms are there like brute force. Flann based matcher

From figures above, we can conclude that SURF produces better and more matching lines than SIFT. Ferrari 1 and 2 has most point matches than rest two images. Due to different orientation, size and height objects there were quite a few errors for matching for all three images. Bumblebee and Optimus Prime has different orientation which resulted into less key point matching.

There are certain parameters which define pass or failure for image matching.

1. Key points might be far apart due to dimension of object being far apart which may lead to incorrect matching. We can implement bilinear interpolation to resize the image as feature extraction is invariant to scaling.
2. Orientation and viewing angle can be different which generally results into wrong key points being matched not getting proper match.
3. Brute force matches find match for every key point which is time consuming and leads to noisy key points. Flann based matcher can be used to tackle this problem.

BAG OF WORDS

Task: Apply K means clustering to extracted SIFT features from three images to form a codebook for 8 clusters. Create codewords of all images and match Ferrari's codewords with rest images.

ABSTRACT AND MOTIVATION

The motive behind bag of words is to create and learn a visual vocabulary from the training set and classify unknown images based on the vocabulary. It is based on documentation classification and data management. It is occurrence of count of each words and plotting histogram of each key word. The dictionary is then clustered into several clusters

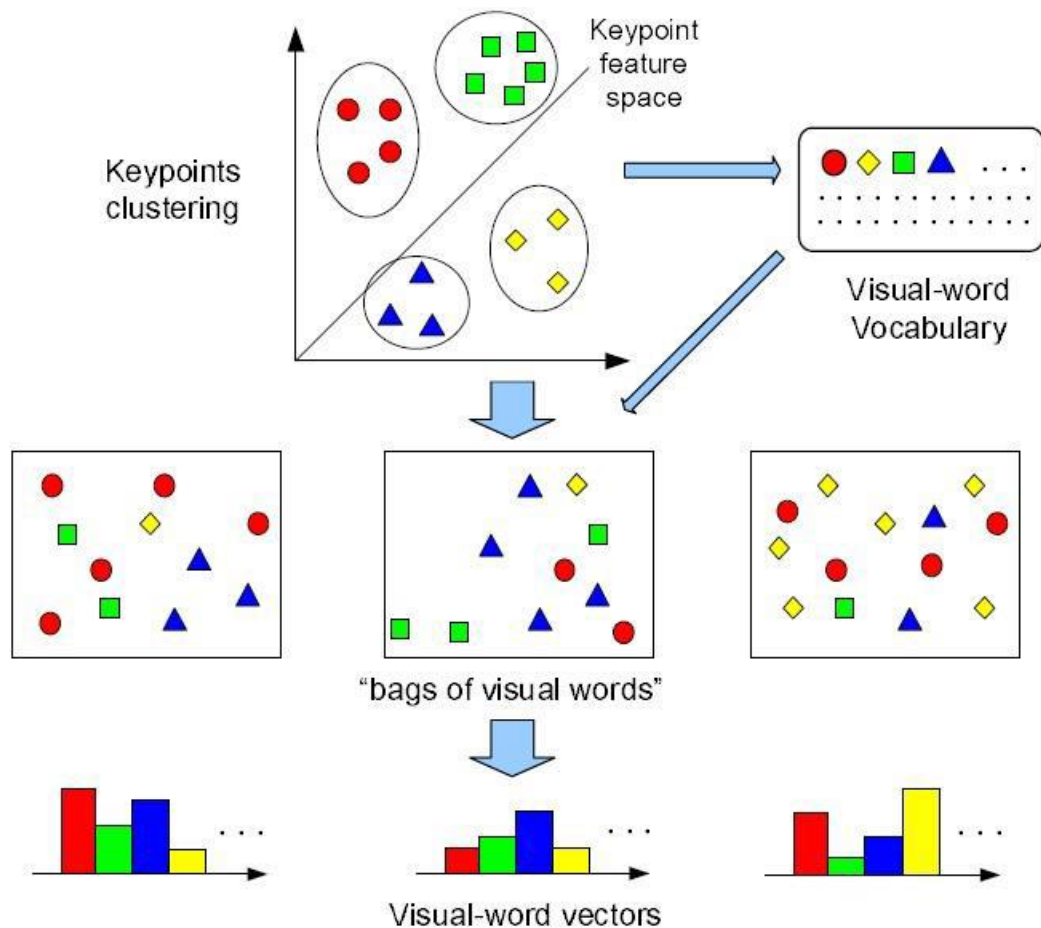


Figure 31 – Bag of words algorithm

APPROACH AND MOTIVATION

For bag of words training and testing, following algorithms has been developed.

DEVELOPED ALGORITHM:

STEP 1 – Extract the local features of the image using SIFT.

STEP 2 – Apply the K means clustering for K clusters and vocabulary of length K

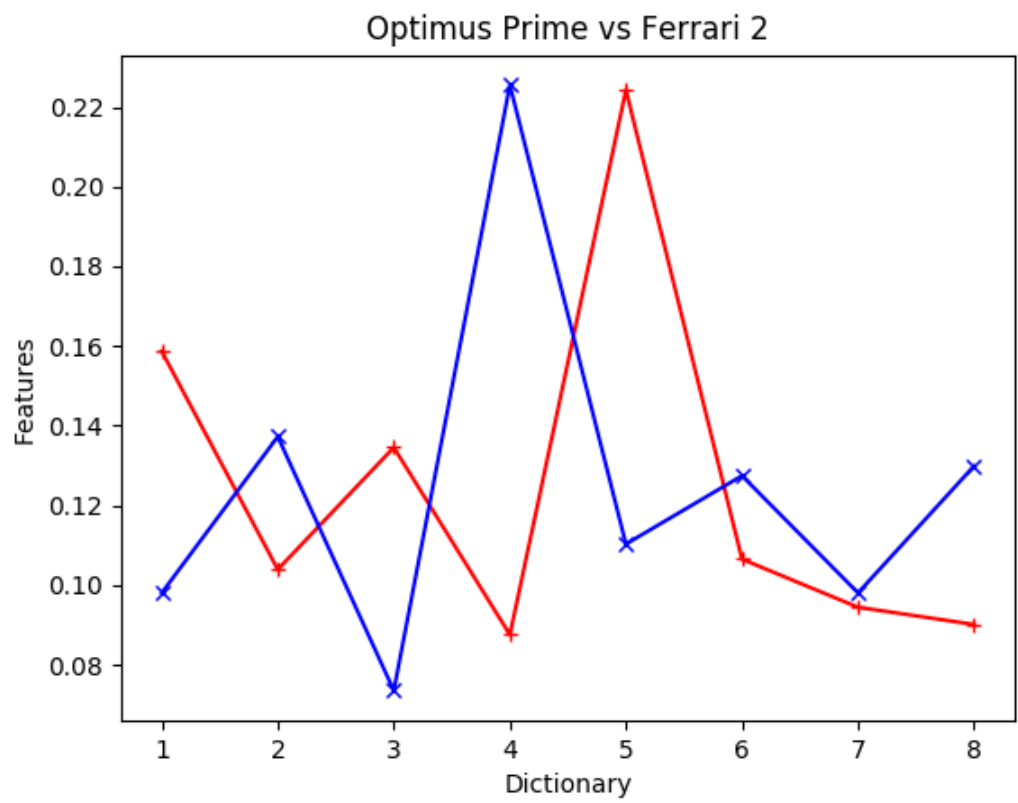
STEP 3 – Save centroids of the clustering is .yaml file.

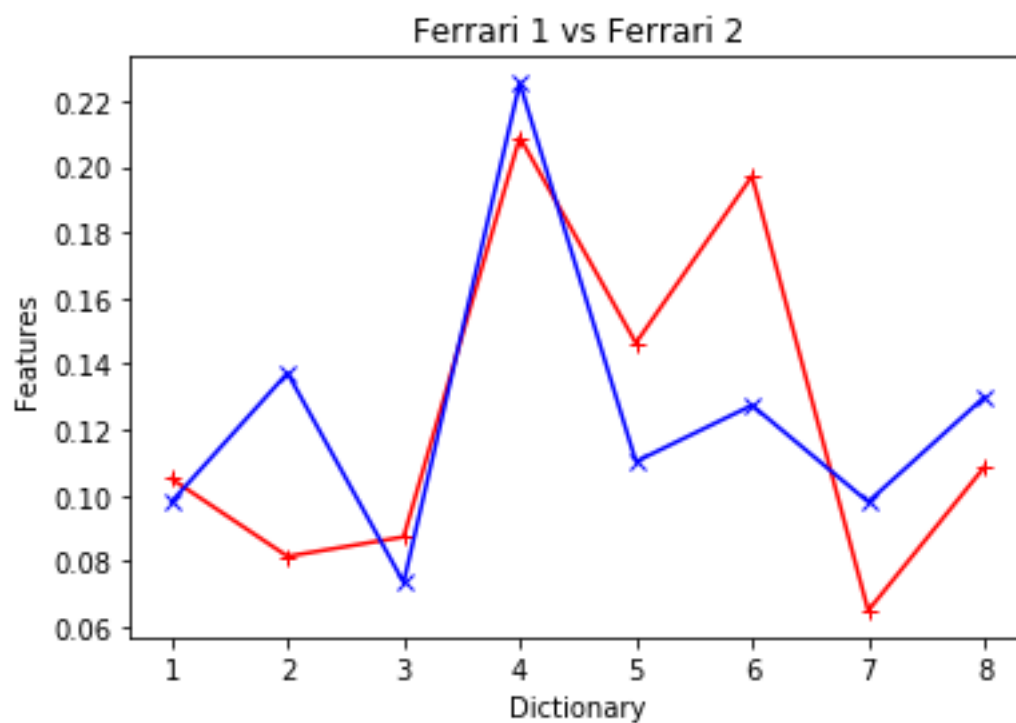
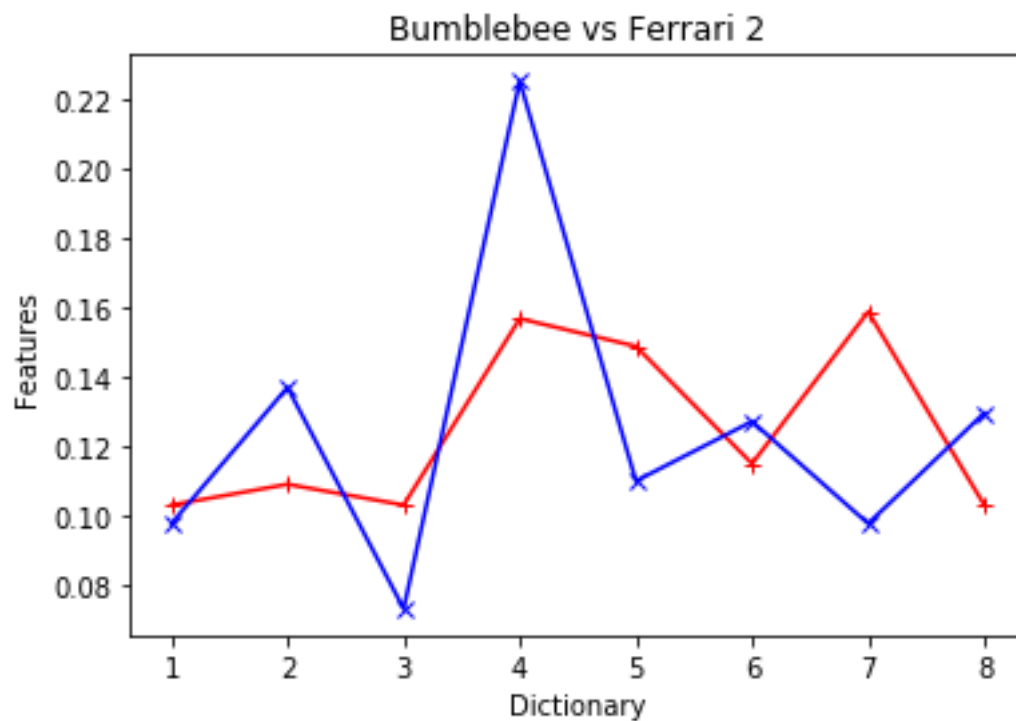
STEP 4 – Create a Flann based matcher to test the image with trained vocabulary.

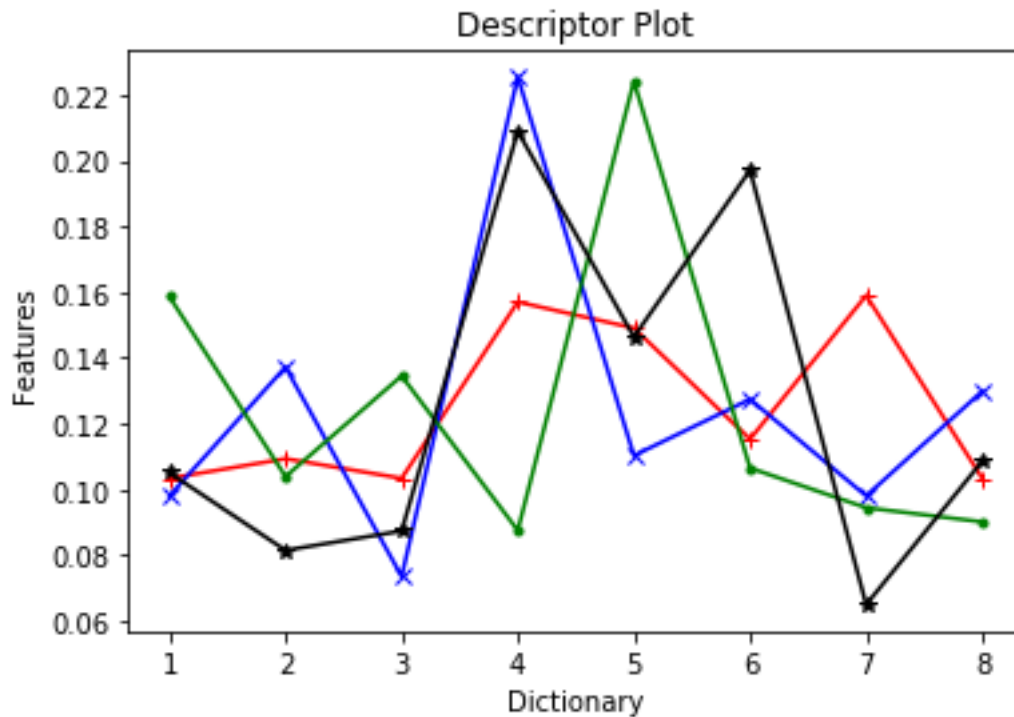
STEP 5 – Detector will detect and computer the key points of test image and and trained dictionary and will return image tag of labels.

STEP 6 – Using Euclidean distance, calculate closeness of test image with training images.

EXPERIMENTAL RESULTS







Ferrari 1 is closest to the Ferrari 2!

Figure 33 – Bag of Words output

DISCUSSION

For Hessian values and threshold, we have trained the dictionary and tested it using Ferrari 2 image. Comparison for every train image with test image has been done and descriptor plot has been shown in above figures.

We can observe degrees of similarity or dissimilarity between two different images through feature plot. Dimension, orientation and quality of image plays an important role while calculating the parameters.

Bag of words dictionary works very well for variety of descriptors given they have highest match for every Hessian value. This algorithm uses K means clustering. Hence initial value of K, random initialization of centroid also plays important part for clustering and accuracy. For Hessian value of 400, Ferrari 2 has strongest resemblance with Ferrari 1. This resemblance can be virtually observed thereby proving the degree of similarity.

REFERENCES

- [1] David Lowe's paper
- [2] https://docs.opencv.org/3.3.0/dc/dc3/tutorial_py_matcher.html
- [3] <http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>
- [4] <https://pdollar.github.io/files/papers/DollarPAMI15edges.pdf>
- [5] Class notes and assignment material
- [6] https://en.wikipedia.org/wiki/Image_segmentation