EE-569 INTRODUCTION TO DIGITAL IMAGE PROCESSING

HomeWork # 3

SUBMITTED BY,

HARIKRISHNA PRABHU
3333077042

PROBLEM 1: TEXTURE ANALYSIS AND SEGMENTATION

AIM:

To implement texture analysis and segmentation algorithm based om the 3x3 laws filter discussed in class or 5x5 filters constructed using the tensor product of the 5 1-D kernels shown in the table.

Table 1: 1D Kernel for 5x5 Laws Filters

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

(a) Texture Classification:

To develop a Texture classification algorithm and implement that on the given 12 texture images. Classify those 12 images into the following 4 categories of images as shown below.

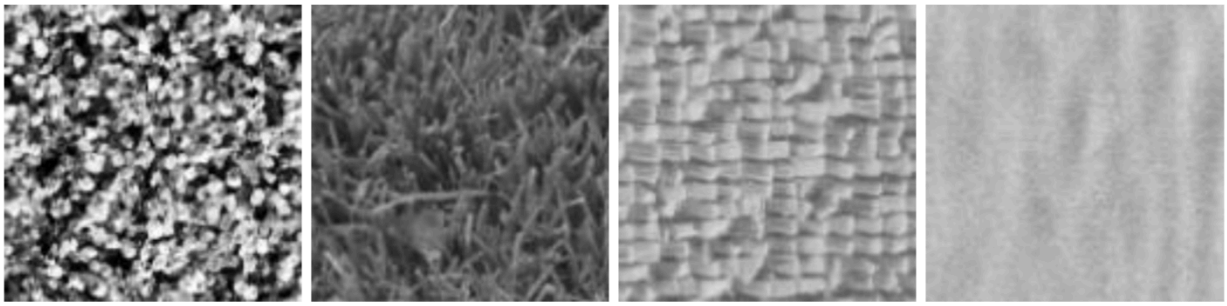


Figure 1: Rock, Grass, Weave and Sand.

Cluster your images into the 4 types, report and compare them by checking the texture images by eyes.

(b) Texture Segmentation:

To develop and implement Texture segmentation algorithm on the given image (Composite texture image) and assign ‘K’ different grey levels to ‘K’ different clusters with each representing one type of texture.

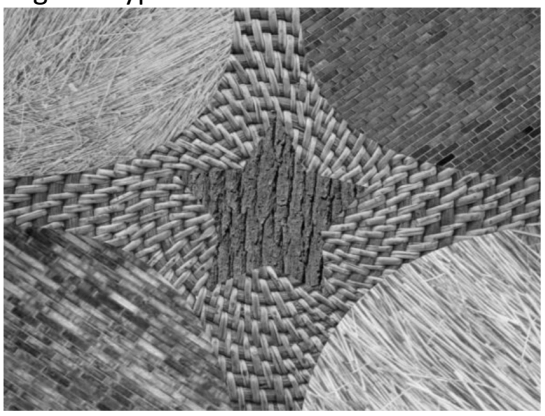


Figure 2: Composite Texture image.

(c) Advanced Texture segmentation:

To develop some techniques to improve the segmentation result.

(i) May use all 25 filters from the 1-D kernels.

(ii) Adopt PCA to perform dimension reduction and then apply your text segmentation algorithm.

ABSTRACT AND MOTIVATION:

Texture: The texture of an image gives is a piece of information that tell us about the spatial arrangement of color and its intensity in a selected part of the image or the entire image. It can be considered as an entity of mutually related pixels. In image processing, Image Textures can be artificially created or be found in nature. Image textures play a very important role in segmenting the image.

Texture classification and segmentation plays a very important role in the field of Image processing and especially in computer vision. It is very intuitive that, textures are basic patterns/building blocks for a particular part of the image or the entire image itself. So, we see that texture is a statistical information for an image and there can be any number of different textures for the given image. Here our motive is to observe and understand different textures and its patterns, and to get into an understanding on how to classify them. In Texture classification, the goal is to assign an unknown sample image to one of the sets of the known texture classes. This process has 2 phases, (I) The learning phase, where we build the model for texture content and (II) The testing phase where the features of the unknown image is extracted, compared with the developed model and is classified into one of its pre-set classes. In our question, we apply k-means algorithm to cluster the images based on the features extracted from the images. Once we become aware of how it is classified, we can extend our scope of analysis to texture segmentation.

A major problem with textures is that in the real world they are often not uniform, due to changes in orientation, scale or other visual appearance. Sometimes the degree of computational complexity becomes very high.

NOTE: When we say features of an image, it can be the image's spatial structure, contrast, roughness, orientation, etc.

APPROACH AND PROCEDURE:

(a) Texture Classification:

A general overview of texture classification can be shown as,

INPUT IMAGE → (Applying filters & Feature extraction) → FEATURE VECTORS → (Applying classifier) → CLASSIFIED TEXTURE IMAGE

The main aim is to implement texture classification by extracting good features. By good features we mean the features that aren't scale, rotation or transform variant.

In this part, we'll be using the following 3 (1-D) kernels for texture classification.

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

- Using this 3 1-D kernels, we construct 9 5x5 law's texture filter using tensor product.
- Before applying the filters to the images, it is always better to normalize the image using the global mean. So that the unnecessary high frequency components are taken care of.
- We apply each of the 9 filters constructed above to the given image. For each filter applied image we calculate the energy measure for the filter's response using the formula

$$\textbf{Energy} = \frac{1}{N*M} * \sum_{i=0}^{N} \sum_{j=0}^{M} (I(i,j))^2$$

- This way for each image we'll have computed 9 features, that is, 9 energy values for a given image using each of the 9 filters. This will give us a feature vector matrix that is 12x9 in dimension. 12= # of images, 9= # of filters.

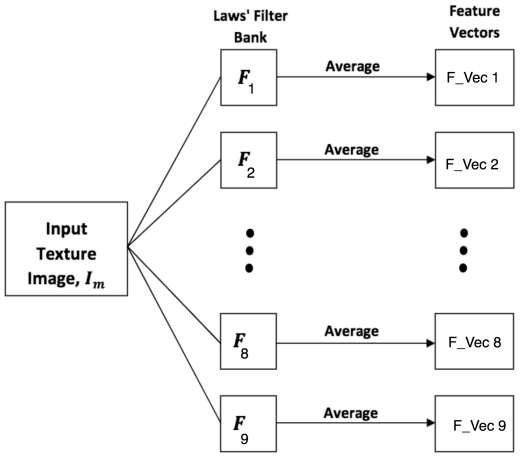


Figure 3: Feature vector created using 9 filters.

- From visual inspection we can say that, among the 12-input images for classification we can see that 3 belongs to each class, thereby we have 4 texture classes. For computationally classifying the input images, we use k-Means clustering.
- K-Means clustering:

The basic working principle of

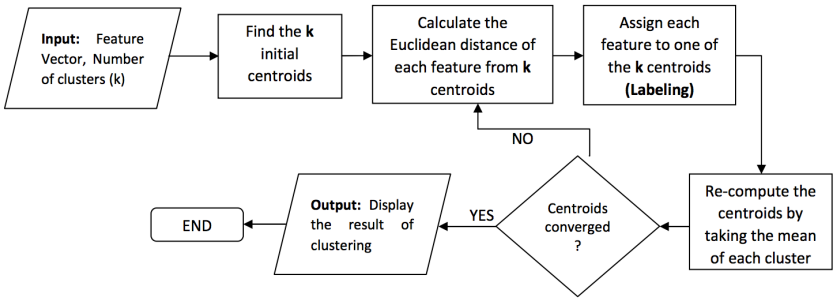


Figure 4: k-Means clustering algorithm flow chart.

(1) Here the feature vectors are obtained from the above explained steps.

- (2) Since we know that we have to classify the given 12 images into the known 4 types, we set 4 initial centroids while performing the k-means.
- (3) Then the Euclidean distance for each of the data point in the feature vector is calculated with each of the newly assigned centroids. The distance formula is shown below.

$$\mathrm{dist}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Distance formula (Here i=1,2,…,9)

- (4) Now, we'd have calculated 4 distance values for each data point from the 4 cluster centroids. Depending upon which distance is the least, that image is assigned to that particular cluster type.
- (5) Once all the images have been assigned to each cluster, we re-compute the 4 centroid values with the data points belonging to its cluster.
- (6) If the new centroid is same as the previous one, proceed with the next step, else go to step (3).
- (7) Once the centroids have converged, we can display the classified images.

After Textured images are classified using k-means, we visually inspect them and check if the images classified to a particular cluster are of same texture type.

(b) Texture Segmentation:

(NOTE: Here we use combination of 3 1-D kernels discussed in class to construct the 3x3 filters.)

E 3 = [-1 0 1], S 3 = [-1 2 1] and L 3 = [1 2 1]

The procedure for texture segmentation is similar to that of classification in a general view.

- In texture segmentation, we compute the feature extraction for each pixel in the image. Whereas in texture classification we computed the features for the image as whole. Our image is 600x450 in size and we use 9 3x3 filters here. So, we'll have a feature vector whose data dimension is 600x450x9. While calculating the energy for determining the feature, we used different window sizes. Say, 5x5,11x11,13x13...so on.
- All kernels mentioned and used above have a zero-mean except for $L3^T L3$. $L3^T L3$ is typically not a useful feature, and it is used for the normalization of all other features.

- By visually observing the input Composite texture image, we can say that there are 6 different textures and hence we initialize 6 initial cluster centroids in k-Means for classifying the textures.
- The same procedure for k-means is followed in this part of the question also.
- Once each pixel in the image is clustered/classified to a particular type, we assign grey levels to pixels belonging to each cluster type. We use the following grey levels in segmenting the image based on texture.
- Gray levels (0, 51, 102, 153, 204, 255) to denote six segmented regions for the output image.

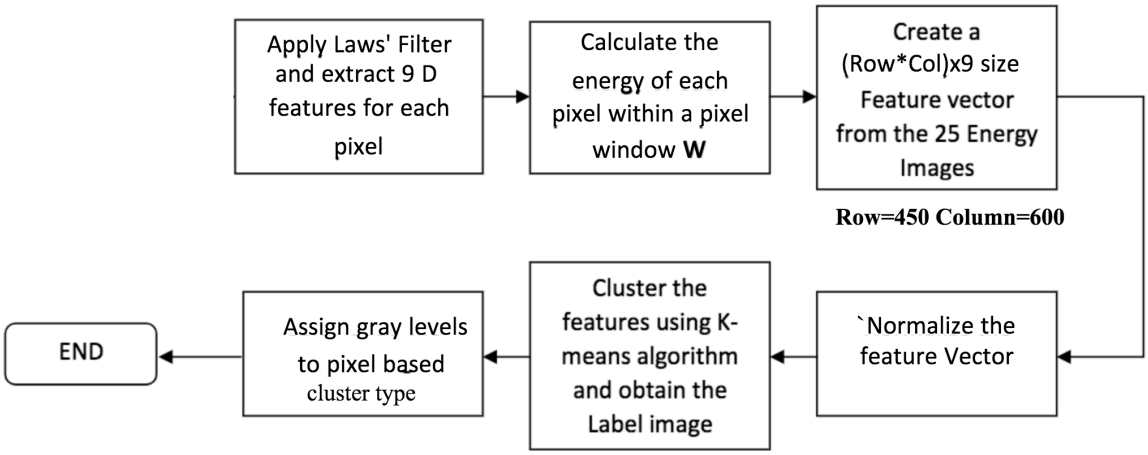


Figure 5: Image Segmentation flow chart.

(c) Advanced Image Segmentation:

Here we just make two small changes,

- We include all the 5 1-D kernels, thereby producing 25 different filters through tensor product.

L5 L5, L5 E5, L5 S5, L5 W5, L5 R5,
E5 L5, E5 E5, E5 S5, E5 W5, E5 R5,
S5 L5, S5 E5, S5 S5, S5 W5, S5 R5,
W5 L5, W5 E5, W5 S5, W5 W5, W5 R5,
R5 L5, R5 E5, R5 S5, R5 W5, R5 R5.

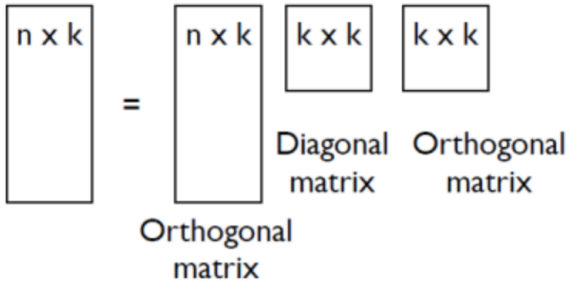
By applying all these filters, we will get a 600x450x25 features feature vector.

- Then we write this 600x450 x 25 features feature vector into a .csv file and we use 'Python' programming to implement PCA.

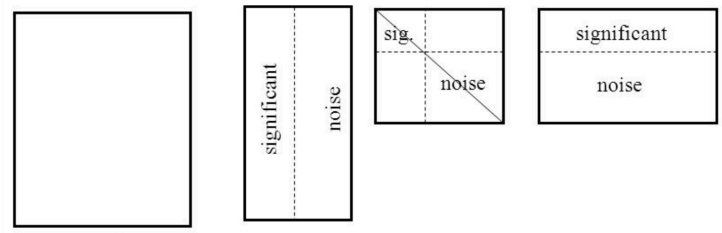
  - ❖ There, we read the input .csv file into a 2-D array, then fit the data for applying Principle component analysis and then transform the input full dimensional data into reduced dimensional data set.
  - ❖ Then this reduced dimensional data set is read back into our original program and k-means clustering is applied to this reduced dimensional feature vector.

Mathematically, PCA can be represented as

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$$



where X= original data set or actual, U and V are the Eigen vector matrix (U and V are the same), and D is the Eigen value matrix that is diagonal in nature. This is how we split our data set.



Once we have diagonalized our data, we take only the sig part of the matrix omitting the rest as noise. This signal part of our actual data is the reduced dimensional new feature vector. This diagonal matrix arranges itself in the order of decreasing importance. That is, the one on the top has highest importance. Likewise, we chose top 5 features or top 10 feature or any other constrained based features through this PCA technique.

- After PCA, this reduced dimensional feature vector is used inside our k-means algorithm and the texture segmentation is complete like in part (b).

EXPERIMENTAL RESULTS:

(a) Texture Classification:

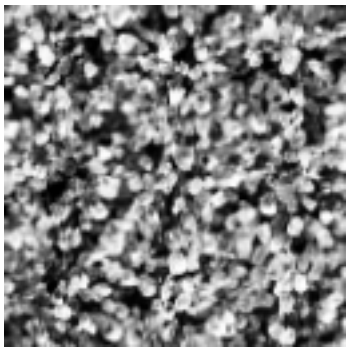


Figure 6: Texture 1

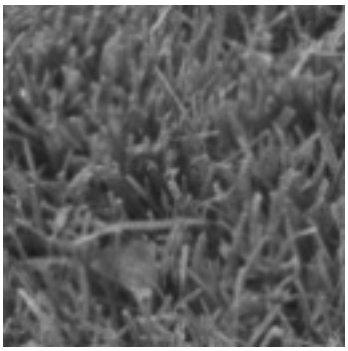


Figure 7: Texture 2

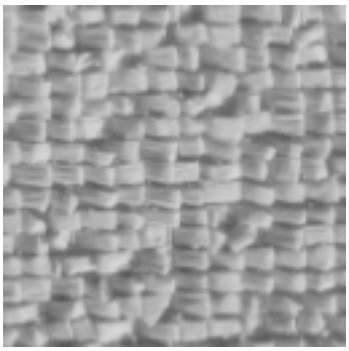


Figure 8: Texture 3

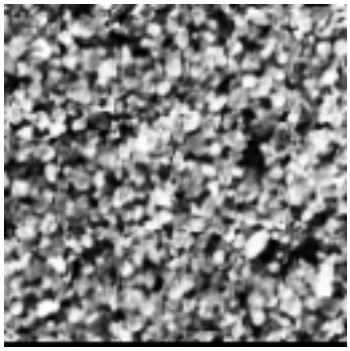


Figure 9: Texture 4

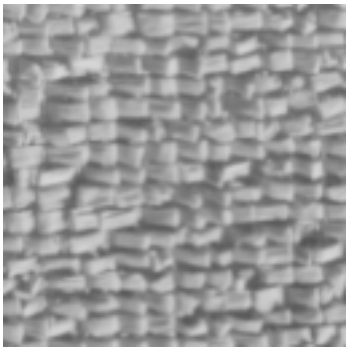


Figure 10: Texture 5

Figure 11: Texture 6

Figure 12: Texture 7

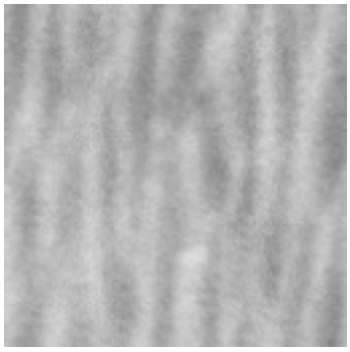


Figure 13: Texture 8

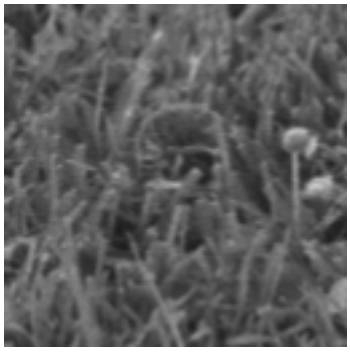


Figure 14: Texture 9

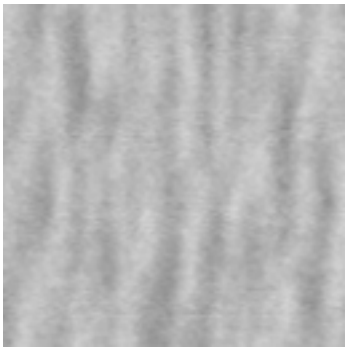


Figure 15: Texture 10

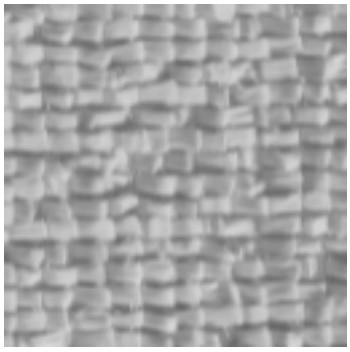


Figure 16: Texture 11

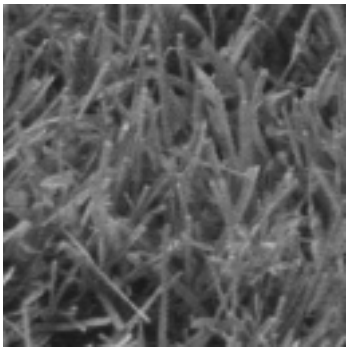


Figure 17: Texture 12

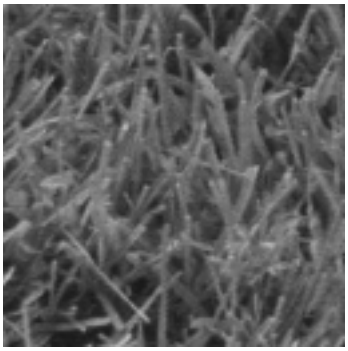


Figure 18: Texture 14 (Non-noisy figure 9)

```
Solve for
Q1.Texture Classification
Q2.Texture Segmentation
Q3.PCA
1
okay..
Enter the following: #columns, #rows, BytesPerPixel
128
128
1
Okay...

#img:1  The feature vectors are:
344.018 186.061 53.7423 188.6   120.666 39.0028 61.2936 44.24   15.4946

#img:2  The feature vectors are:
46.3627 23.5475 5.9298  31.2466 17.4793 4.7905  9.4237  5.95967 1.86328

#img:3  The feature vectors are:
25.2418 13.25   4.1517  10.6901 5.88723 1.89792 2.88702 1.69371 0.561593

#img:4  The feature vectors are:
391.84  234.002 66.6463 218.695 145.28  46.687  67.3022 48.348  16.8481

#img:5  The feature vectors are:
32.8793 17.1203 4.95169 12.9497 6.98586 2.11117 2.94539 1.65613 0.502228

#img:6  The feature vectors are:
365.772 207.763 61.9568 210.696 137.178 45.9181 68.5212 50.1652 18.1288

#img:7  The feature vectors are:
2.59011 2.37605 1.21018 2.32695 2.33501 1.26194 1.19087 1.25474 0.703943

#img:8  The feature vectors are:
2.80714 2.38672 1.1836  2.36005 2.37355 1.26954 1.23078 1.31358 0.728702

#img:9  The feature vectors are:
27.7474 13.3577 3.61613 14.4099 7.85595 2.35592 3.30248 1.98583 0.654637

#img:10 The feature vectors are:
2.36254 1.96011 0.90753 1.56228 1.46236 0.723782    0.486474    0.48598 0.25616

#img:11 The feature vectors are:
17.5163 7.67694 1.72073 7.64914 3.34064 0.772847    2.242   1.03679 0.250214

#img:12 The feature vectors are:
51.7588 28.281  8.68804 36.7679 24.2198 8.3276  11.9055 9.19143 3.66408


Iteration:1
    Img:1=(k)1  Img:2=(k)3  Img:3=(k)2  Img:4=(k)1  Img:5=(k)3  Img:6=(k)1  Img:7=(k)4  Img:8=(k)4  Img:9=(k)2  Img:10=(k)4 Img:11=(k)2 Img:12=(k)3
Iteration:2
    Img:1=(k)1  Img:2=(k)3  Img:3=(k)2  Img:4=(k)1  Img:5=(k)2  Img:6=(k)1  Img:7=(k)4  Img:8=(k)4  Img:9=(k)2  Img:10=(k)4 Img:11=(k)2 Img:12=(k)3
Iteration:3
    Img:1=(k)1  Img:2=(k)3  Img:3=(k)2  Img:4=(k)1  Img:5=(k)2  Img:6=(k)1  Img:7=(k)4  Img:8=(k)4  Img:9=(k)2  Img:10=(k)4 Img:11=(k)2 Img:12=(k)3

Program Exit. Thank You
```

Figure 19: Classified with Texture 9

```
Solve for
Q1.Texture Classification
Q2.Texture Segmentation
Q3.PCA
1
okay..
Enter the following: #columns, #rows, BytesPerPixel
128
128
1
Okay...

#img:1  The feature vectors are:
344.018 186.061 53.7423 188.6   120.666 39.0028 61.2936 44.24   15.4946

#img:2  The feature vectors are:
46.3627 23.5475 5.9298  31.2466 17.4793 4.7905  9.4237  5.95967 1.86328

#img:3  The feature vectors are:
25.2418 13.25   4.1517  10.6901 5.88723 1.89792 2.88702 1.69371 0.561593

#img:4  The feature vectors are:
391.84  234.002 66.6463 218.695 145.28  46.687  67.3022 48.348  16.8481

#img:5  The feature vectors are:
32.8793 17.1203 4.95169 12.9497 6.98586 2.11117 2.94539 1.65613 0.502228

#img:6  The feature vectors are:
365.772 207.763 61.9568 210.696 137.178 45.9181 68.5212 50.1652 18.1288

#img:7  The feature vectors are:
2.59011 2.37605 1.21018 2.32695 2.33501 1.26194 1.19087 1.25474 0.703943

#img:8  The feature vectors are:
2.80714 2.38672 1.1836  2.36005 2.37355 1.26954 1.23078 1.31358 0.728702

#img:9  The feature vectors are:
51.7588 28.281  8.68804 36.7679 24.2198 8.3276  11.9055 9.19143 3.66408

#img:10 The feature vectors are:
2.36254 1.96011 0.90753 1.56228 1.46236 0.723782    0.486474    0.48598 0.25616

#img:11 The feature vectors are:
17.5163 7.67694 1.72073 7.64914 3.34064 0.772847    2.242   1.03679 0.250214

#img:12 The feature vectors are:
51.7588 28.281  8.68804 36.7679 24.2198 8.3276  11.9055 9.19143 3.66408


Iteration:1
    Img:1=(k)1  Img:2=(k)3  Img:3=(k)2  Img:4=(k)1  Img:5=(k)3  Img:6=(k)1  Img:7=(k)4  Img:8=(k)4  Img:9=(k)3  Img:10=(k)4 Img:11=(k)2 Img:12=(k)3
Iteration:2
    Img:1=(k)1  Img:2=(k)3  Img:3=(k)2  Img:4=(k)1  Img:5=(k)2  Img:6=(k)1  Img:7=(k)4  Img:8=(k)4  Img:9=(k)3  Img:10=(k)4 Img:11=(k)2 Img:12=(k)3
Iteration:3
    Img:1=(k)1  Img:2=(k)3  Img:3=(k)2  Img:4=(k)1  Img:5=(k)2  Img:6=(k)1  Img:7=(k)4  Img:8=(k)4  Img:9=(k)3  Img:10=(k)4 Img:11=(k)2 Img:12=(k)3

Program Exit. Thank You
```

Figure 20: Classified with Texture 14

(NOTE: Features are computed by sum of squares div by sq of the dimension)

Textures: 1, 4 and 6 are same.

Textures: 3, 5 and 11 are same.

Textures: 2, 9 and 12 are same.

Textures: 7, 8 and 10 are same.

And is inspected visually also.

(b & c) Texture Segmentation and Advanced

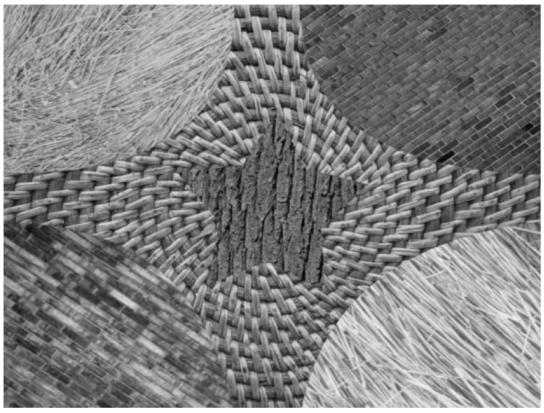


Figure 21: Input image

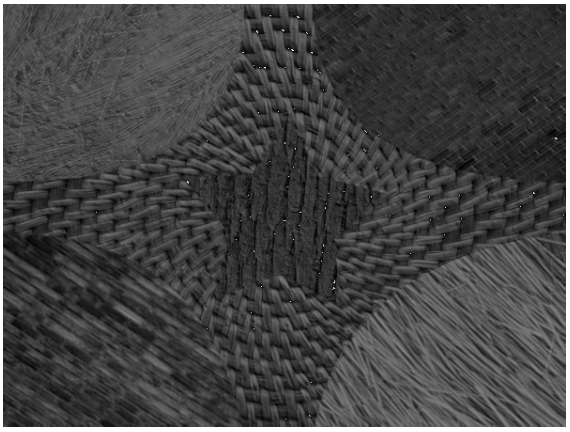


Figure 22: Filter 1 E3E3

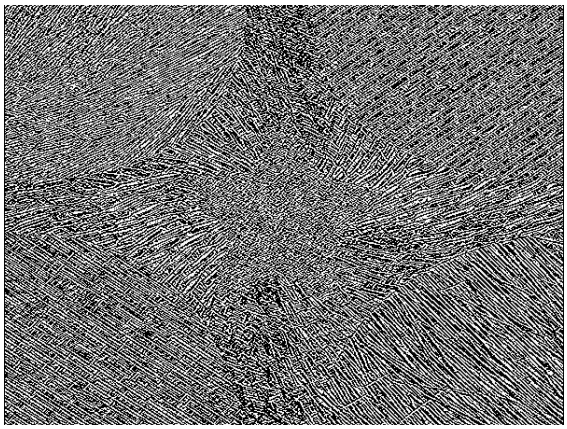


Figure 23: Filter 2 E3S3

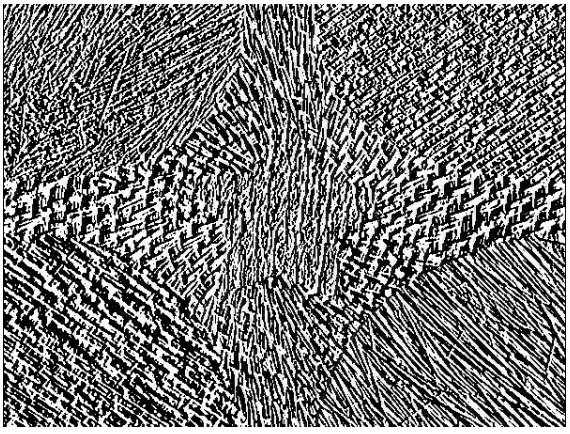


Figure 24: Filter 3 E3L3

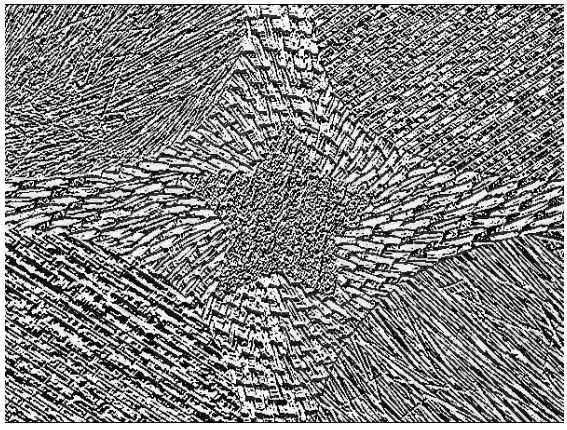


Figure 25: Filter 4 S3E3

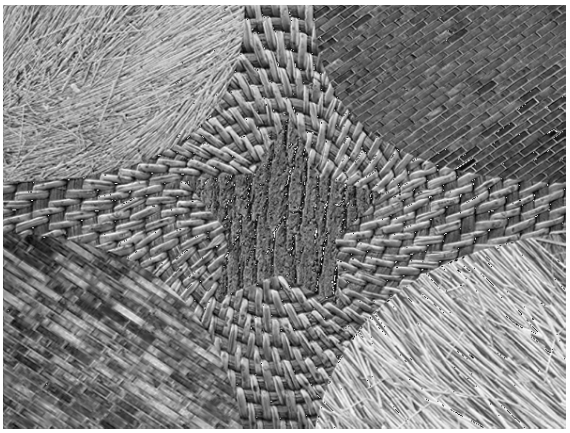


Figure 26: Filter 5 S3S3

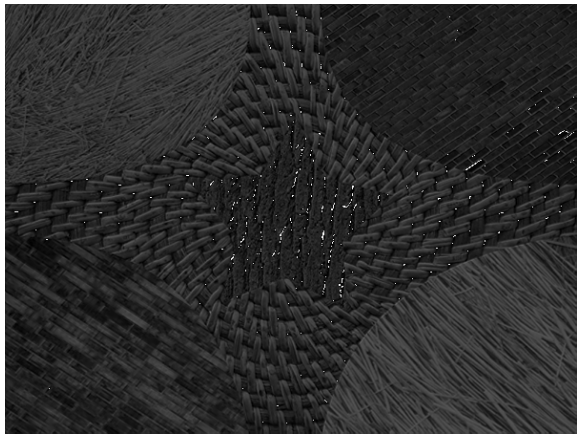


Figure 27: Filter 6 S3L3

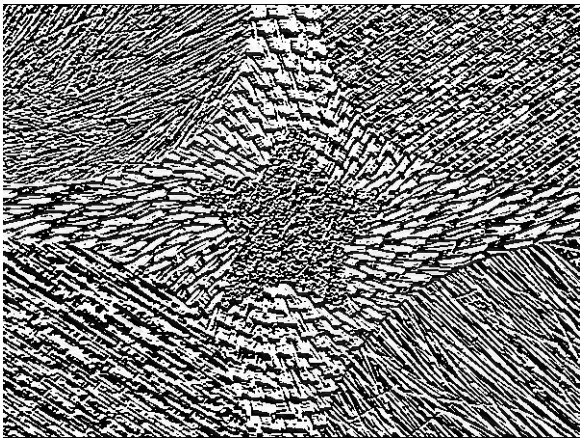


Figure 28: Filter 7 L3E3

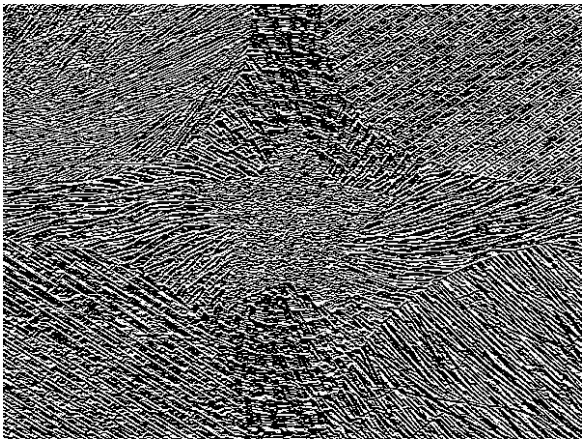


Figure 29: Filter 8 L3S3

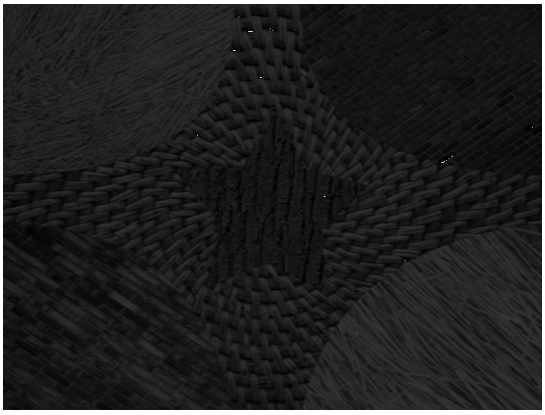


Figure 30: Filter 9 L3L3

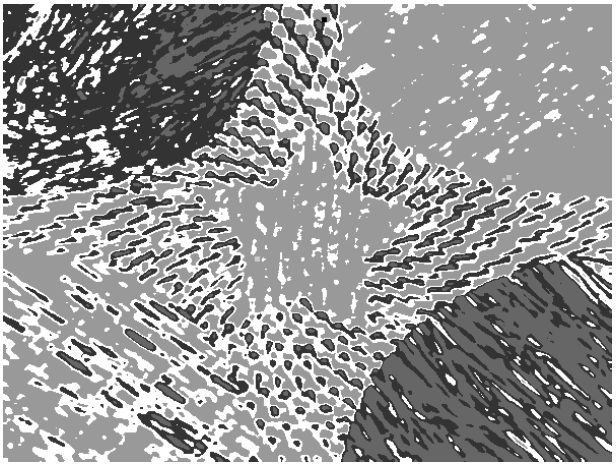


Figure 31: Filter 5x5

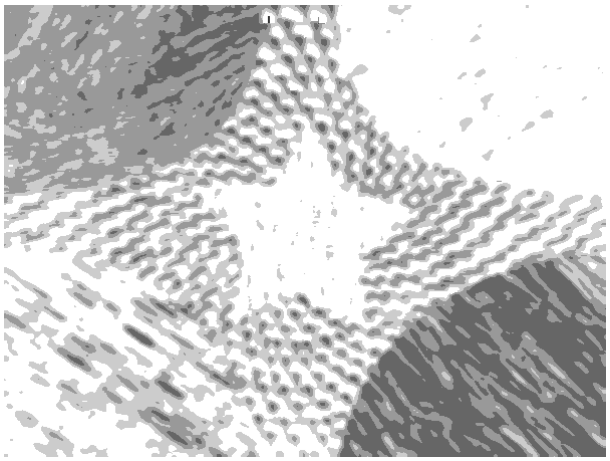


Figure 32: Filter 7x7

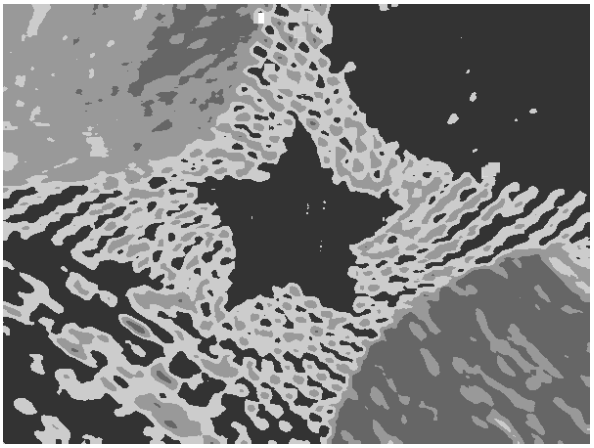


Figure 33: Filter 11x11

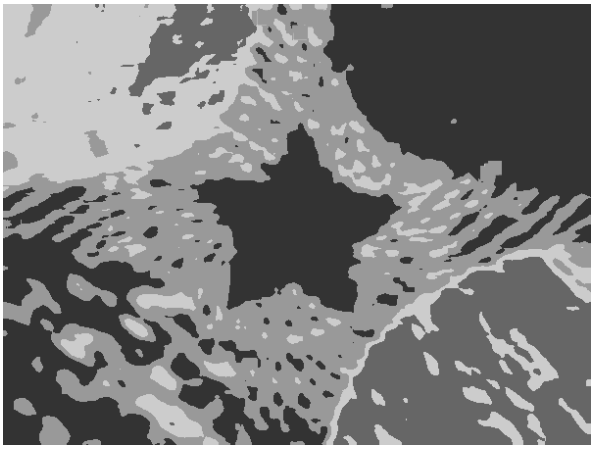


Figure 34: Filter 15x15

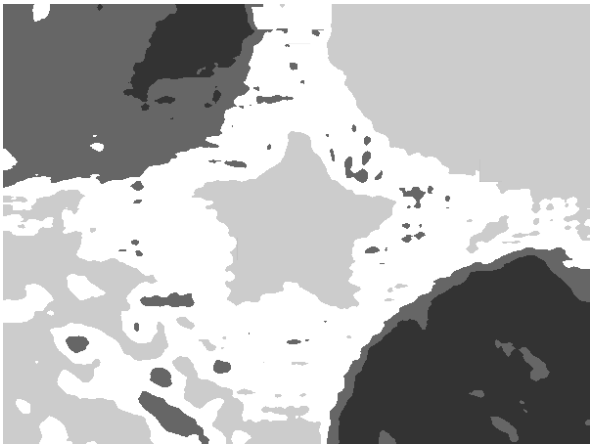


Figure 35: Filter 23x23

PCA:

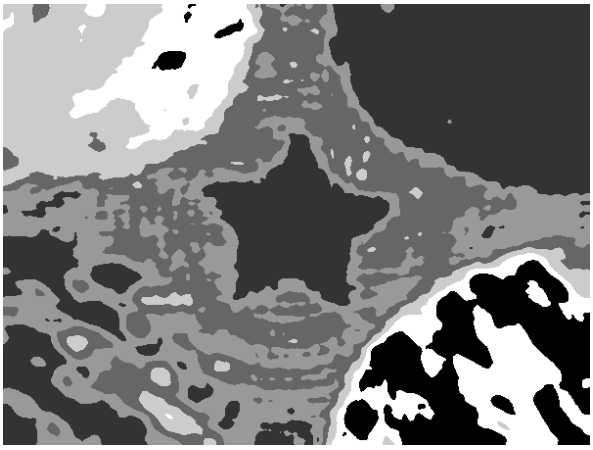


Figure 36: 23x23 PCA for 10

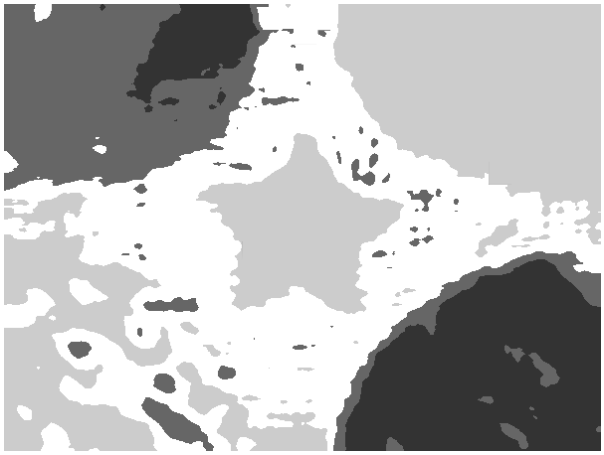


Figure 37: 23x23 PCA for 5

DISCUSSION:

(a) Texture Classification:

Finally, we have designed and implemented an algorithm for texture classification. This includes extraction of features and applying k-means clustering. We can see from the output that,

(1) When texture 14 is used in place of texture 9.

* Images 1, 4 and 6 are clustered together and represents the 'Rock' texture.
*Images 3, 5 and 11 are clustered together and represents the 'Weave' texture.
* Images 2, 14 (9 in program) and 12 are clustered together and represents the 'Grass' texture.
*Images 7, 8 and 10 are clustered together and is represented as 'Sand' texture.

(2) When texture 9 is actually used in place of texture 9.

* Images 1, 4 and 6 are clustered together and represents the 'Rock' texture.
*Images 3, 5, 9 and 11 are clustered together and represents the 'Weave' texture.
* Images 2 and 12 are clustered together and represents the 'Grass' texture. *Images 7, 8 and 10 are clustered together and is represented as 'Sand' texture.

Though we can visually comprehend that the textures 9 and 14 belong to the same texture pattern, texture 9 image seems to be over smoothened and smudged a bit. A comparison of texture 14(Grass) , texture 9 and texture 5(Weave) is shown below.

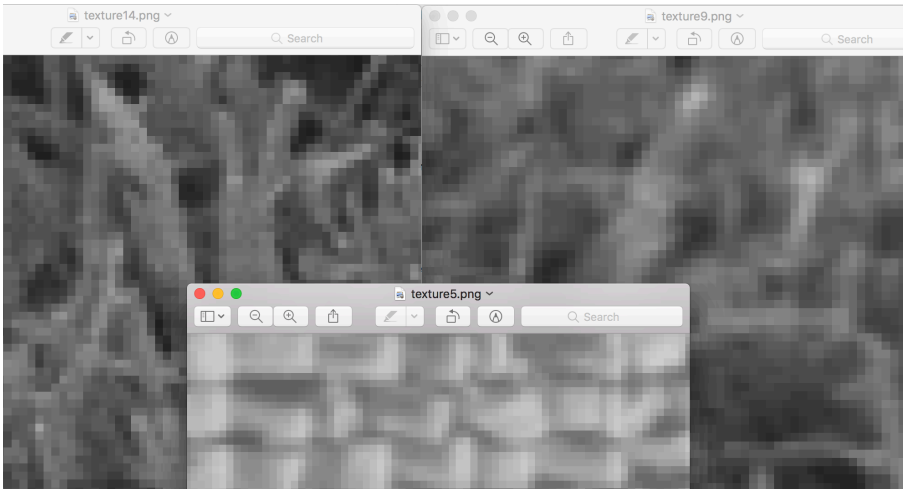


Figure 38: Texture 14(left), Texture 5 (bottom) and Texture 9 (right).

The above given image is a zoomed image of all the three textures. Here we can understand the difference between texture 14 and 9. Texture 14 has sharp pixel edges and the tonal changes are crisp. Whereas in texture 9, the changes are smoothened and looks a little bit of weave-ish (Perhaps a weak weave pattern). We can observe this from the feature vectors also. As shown in Figures 19 and 20, The feature vector for img 9 (actual texture image 14) is similar to feature vectors of textures 2 and 12 (Grass type). For img 9 (actual texture image 9), the feature vector values fall in the vicinity of feature values of textures 3, 5 and 11 (Weave type). This may be the reason as on why texture 9 is misclassified and why texture 14 is correctly classified.

(b & c) Texture Segmentation:

In this part, we have implemented Texture segmentation and further applied Principle Component Analysis for dimension reduction for the features extracted from the image with each filter applied.

Here, I have shown the output image for the image after application of each of the 9 filters. They are shown in the Figures 22 through 30. These images give a vivid representation of the textures in the image.

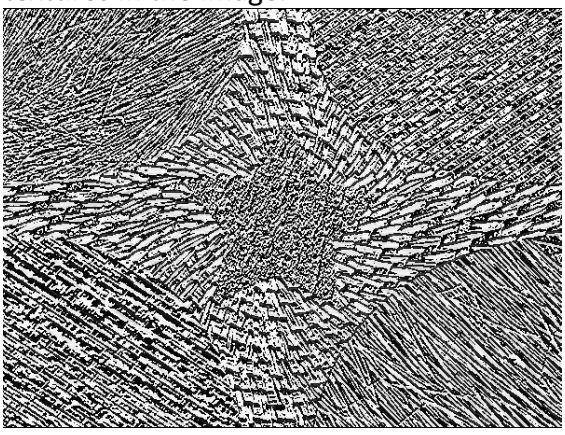


Figure 25: Filter 4 S3E3.

We can observe that the texture patterns are distinctively different for each of the texture or from image, we can distinctively figure out different textures.
As explained in the instructions, L3T L3 is of no use and it is clearly seen in the figure 30. The output produced has no meaningful information.
Figures 31 through 35 shows the Texture segmentation output for various window sizes. From these outputs, we can understand the fact that,
as the window size for calculating the energy measure increases, we get a better segmented image. Example. Texture segmented image with window size 23x23 (figure 35) has better output than the texture segmented image of window size 7x7 or 11x11 (figure 32 or 33).
So, as window size increases, better is the segmentation result.

NOTE *k-Means*: One has to keep in mind the fact that, the accuracy of the k-Means clustering algorithm largely depends on the initial values of the centroids and also the cluster length. For us, cluster length won't play a major part as our cluster length is a constant for each part of the question. The only thing that changes is the initial centroid values. As we have only lesser number of data points, randomly assigning the centroid won't help the system to converge as desired. Hence what we do is that, we randomly assign one of the data points as the initial centroid for each of the centroids. By this way we can have a surety that the centroids will converge much better than the previous case.
Warning: In k-Means as mentioned, the starting point plays a very important role in the convergence of the data, however big the data set is. So, one must be experienced enough to select a good starting point or the general rule of thumb is to use any of the data points as the centroid.

We've tried applying PCA for the extracted feature vector whose dimension is 600x450x25. We see that the outputs differ slightly based on the number of reduced dimension (n) [feature vector dimension is 600x450xn] but for 9 features, both with (c) and without (b) PCA produce the same image. This is shown in figures 36 and 37.
From the experimental results, it is quite evident that visually the output segmented images have no much difference, but the computational time efficiency increases as the feature dimension reduces.

PROBLEM 2: EDGE DETECTION

AIM:

To implement various edge detection techniques.

(a) Basic Edge Detector:

To implement the following edge detection techniques to the given images Boat and Boat_Noisy.

(i) Sobel Edge Detector.

(ii) Zero Crossing Detector.

Observe and compare the results of (i) and (ii).

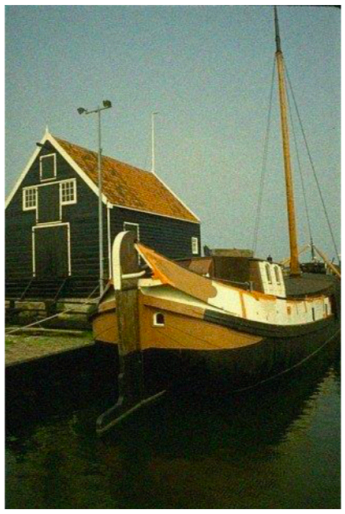


Figure 39: Boat (left) and Boat_Noisy (right).

(b) Structured Edge:

(i) To give a brief explanation about the working algorithm of Structured Edge Detector along with a flow chart.

(ii) To explain the construction of decision tree and the principle of the Random Forest (RF) classifier.

(iii) To apply SE detector on the following images,

Figure: Animal

Figure: House

And to state the parameters chosen and justify it.

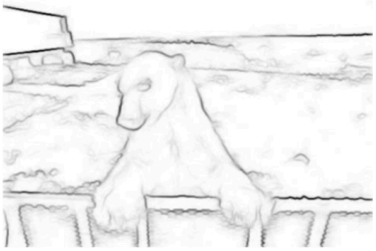
Compare and comment the visual results of Sobel and Laplacian of the Gaussian along with SE detector.

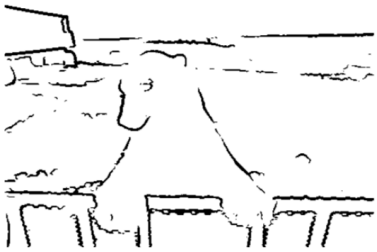
(c) Performance Evaluation:

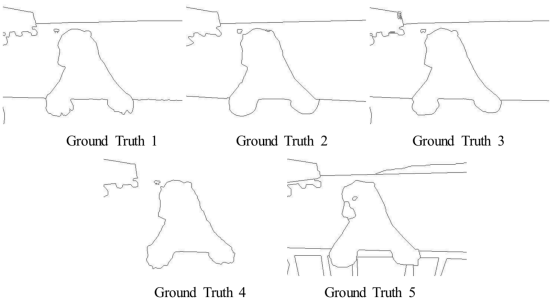
To perform a quantitative comparison between different edge maps obtained by different edge detectors.

(i) To calculate the precision and recall for each ground truth and the corresponding F- measure for the generated edge map. Then find mean precision and mean recall and calculate F-measure for the generated edge map using this mean precision and recall. Comment on the performance of different edge detectors.
(ii) Is F-measure Image Dependent? For which of the given images is F-measure easier to get high? Provide an intuitive answer.
(iii)Discuss the rationale behind F – measure? Is it possible to have high F-measure if the precision is significantly higher than recall or vice-versa? If sum of precision and recall is constant, show that F-measure reached maximum when precision is equal to recall.
An example for image and its ground truth is shown below.


Bear


Probability edge map


Binary edge map (with p>0.2)



ABSTRACT AND MOTIVATION:

An edge can be defined as a set of contiguous pixel positions where an abrupt change in the intensity (either gray or RGB) value occurs. And Edge detection is an image processing technique for finding the boundaries of objects within images which is computed using this discontinuity.
Most of the shape information of an image is enclosed in edges. That is why we detect these edges in an image. Edge detection is a useful technique that plays an important role in image segmentation. These edges detected along with application of filters helps us in enhancing the sharpness of the image. Mostly they are used in Image Pre-processing. In other words, edge detection is used in low level image processing to produce good images. And for high level image processing techniques, good images are very much required and hence edge detection is used as an image preprocessor.
Basic edge detectors are, (i) Sobel, (ii) Zero-crossing (iii) Prewitt, (iv) Robinson compass, (v) Structured edge detector and many more.
Any of the edge detection problem can been implemented in two ways, they are
(i) Point Pixel Property and

(ii) Image patch property.
Sobel edge detector is an example for implementing Point pixel property and Structured edge detector is an example for Image patch property. We can see that the structured edge detector has a machine learning characteristic in it with all those probability edge maps and other related attributes.
Random forests play a significant role in implementation of complex and efficient edge detection techniques. It will be briefly explained in our discussion part.

APPROACH AND PROCEDURE:
(a)(i) Sobel edge detector:

The Sobel–Feldman edge detector or popularly called as Sobel edge detector is an operator that performs first order 2-D spatial gradient measurement for an image. That is, it computes the edge detection in both horizontal and vertical axis.

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

Gx

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

Gy

Figure 40: X-Gradient and Y-Gradient.

These are the 3x3 convolution kernels to compute edge detection in horizontal (Gx) and vertical (Gy) directions. This detector operates as a first order derivative and simply computes the difference in pixel intensities in the edge region. In Gx, that is, the horizontal convolutional kernel, computes the difference between the *right and left* neighboring pixels and likewise the vertical kernel computes the *top-bottom* difference.
Procedure:
Note: Sobel edge detector works well with grey scale images only.

- We check whether the input image is color or grey, and if it is a color image, it is converted into grey.
- Once we have a grey image at our disposal, we implement the computation of the X-gradient (Gx) and the Y-gradient (Gy) using the 3x3 convolutional kernels applied to the input grey image separately. (kernels are shown in the above figure).
- Gx is the Right-Left pixel difference and Gy is the Top-Bottom pixel difference.
- The values obtained after applying the horizontal and the vertical filters are normalized to 0-255 range. This normalization is done to bring the computed values into the scope of range of colors.
- Then we compute the 'G' or the gradient magnitude using the formula

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

.

- The edge orientation can be computed using,

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

, where 'theta' is the angle value.

- The gradient magnitude 'G' is also normalized to 0-255 range. This image pertaining to the G value is the edge detection map.

Note: Here throughout our problem, we have implemented white as the background color and black for the detected edge.

- The map produced is not binary in nature, but rather a like a probability edge map. We need to threshold it and produce a binary image that then can be implemented in higher levels of image processing.
- Here, we compute the threshold from the cumulative distribution function (CDF) for the given image. From the obtained CDF, we fix a % value for the pixel and by mapping that pixel % to the CDF's index, we'll obtain the threshold value.
- Using this threshold value, we binarize the given image and generate a new image with white color as background and black color indicating detected edges.

(a)(ii) Zero crossing detector: As the name suggests, zero crossing edge detector's function is to detect the zero-crossing point.

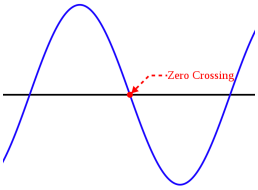
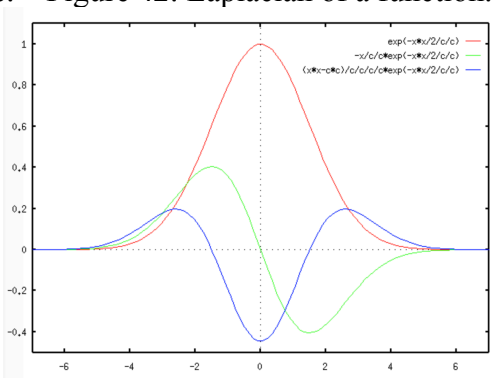


Figure 41: Zero crossing.

The core of this zero-crossing detector is the *Laplacian* filter. While all the above-mentioned edge detectors in the abstract section are first order derivative based edge detectors, zero-crossing edge detector is a second order derivative based edge detector. The Laplacian in it fuels the second order derivative.

Figure 42: Laplacian of a function.



In this figure, the RED line denotes the function f(x). The GREEN line indicates the first order derivative of that function and the BLUE line indicates the second order derivative of the function respectively. Once the Laplacian filter provides the second order derivative, the zero-crossing detector detects the two zero crossing points. These zero crossing points essentially indicates the presence of an edge in that part of the image.

Laplacian of the Gaussian: It is called Laplacian of the Gaussian because, the Laplacian filter detects not only the edges but the noise in the given image. And in order to overcome/remove these noises, we apply a Gaussian filter over the Laplacian filter. Hence this filtering operation together is called as the Laplacian of the Gaussian. This filter generally brings about the change

in sign of the second-order derivative every time it detects an edge. And this sign change is made use by the zero-crossing detector and proceeded further.

Procedure:

- Like the Sobel edge detector, Zero-crossing edge detector also works for grey images only. So, we convert any given input color image into its grey scale image.
- Once we obtain the grey scale image, we apply the LoG filter to it. When we plot the histogram of the pixel values obtained after application of the LoG filter. We will have a plot that has a steep symmetric spike. From this plot, we need to find. Out the knee points.

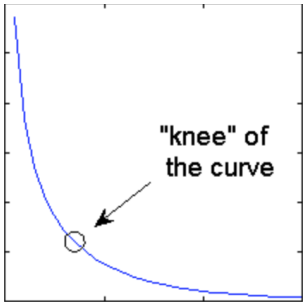


Figure 43: What is a knee point.

  - We'll be finding 2 knee points from our obtained plot.
    - ❖ Algorithm to find the knee point:
    - ❖ Obtain the Probability Density Function (PDF) of the image. From the PDF,
      - Traverse through the PDF in forward direction, compute the difference in the PDF value of the present index and the previous index. Once the difference between the PDF values keeps increasing (indicating the steep rise in the PDF of the plot) [a numerical threshold value is set as a threshold after manually observing the plot], we set the first instance of the continuous increase in the differences as the first knee point.
      - The same is performed in reverse direction of the plot and the second knee point is found.

- Once we obtain the two-knee points of the generated curve, we partition the LoG histogram into three values. Namely, -1 for all those points lying to the left of the left most knee point. 0 for all those points lying in between the two knee points and +1 to all of those points lying to the right of the right most key point.
- This -1,0,1 level is mapped to the grey levels 64,128 and 192 respectively. This image thus obtained is called the Ternary map of the given image.
- Same -1,0,1 image is used for zero-detection, that is, edge detection. We know that in Gaussian, change in sign (essentially crossing the zero point) indicates the presence of an edge. So, it is sufficient for us to check for this change.
- For every pixel, we check if there is a change in the pixel value of its neighbors. That is, If there is a 1 or -1 surrounding a 0 pixel and if there is a 0 surrounding the -1 or 1 pixel. If we observe any such change, then we write the new image with '0' for that particular pixel denoting the edge and if not '255' denoting background or non-edge portions of the image.

(b) Structured Edge Detector:

Traditional Edge detection techniques makes use of gradient magnitude and gradient orientation to figure out the edges in an image. But structured edge is a highly complex method used for detection of edges. It implements random forests and includes decision trees as well. As mentioned earlier, Structure edge is a patch based image processing technique. Here a patch surrounding the current pixel is taken and the likelihood of an edge being present is computed.

(i) Decision Tree: Decision tree is a tree like graph model. It is a map of the possible outcomes of a series of related choices. It allows us to weigh possible actions against one another. It consists of 3 types of nodes, namely, (i)Decision node- Square in shape, (ii) Chance node- circular in shape and (iii) End nodes – triangular in shape as shown in the figure below

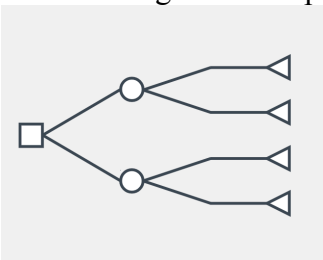


Figure 44: Decision Tree

The problem with decision tree is that they tend to over fit the input data. Hence, we bring the concept of Random trees.

(ii) Random Forests:

Random forest is a collection of 'N' independent decision trees and its output is the combination of outputs from each of the individual tree added using ensemble model.

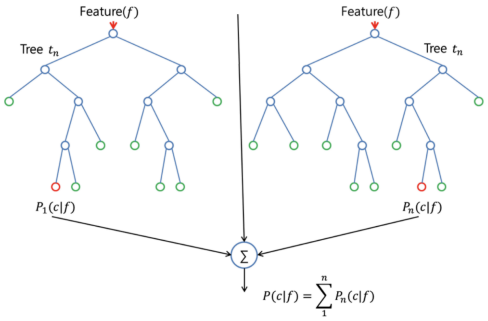


Figure 45.1: Random forest model.

The decision tree consists of a number of rhetoric questions as shown in the figure. Likewise, output is obtained from each of the decision trees and then combined using ensemble learning. The function of random forest depends on the type of problem to be solved. For a classification problem, it functions based on the "Majority Voting" and for regression type, it uses "Mean Averaging".

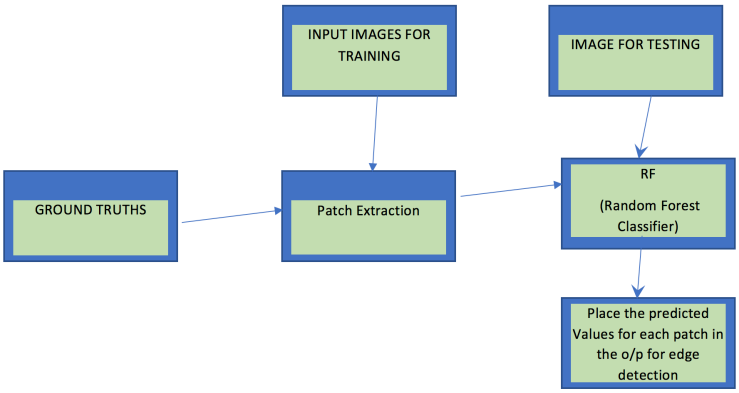


Figure45.2: Structured Edge Detection algorithm

Procedure:

- Image patches and taken and are classified into sketch tokens (a group of tokens represents the local edge features). This classification is done with the help of voting measure provided by the random forest. (Here the function of the random forest is of "Majority Voting" type).
- This classifier is trained by comparing the image patch from our input image and the corresponding ground truth image and generating an output of whether the combination is an edge or not.

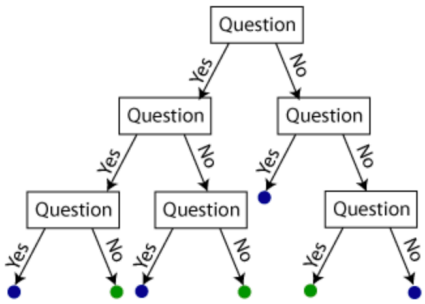


Figure 46: Working of decision tree.

- ❖ In the decision tree, the input traverses from the top to the bottom and reaches the leaf node. Each node is a decision check node where the decision parameter is checked and if true, it traverses down through the right-hand side branch of the tree and if false, in the left. This is done iteratively and is stops when it reaches the leaf node.
- ❖ So, basically each node is of a rhetoric question or a split function.

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

- ❖ Likewise, the same operation is performed for all the 'N' decision trees and is added in an ensemble learning method.

Ensemble Model is used for combining multiple predictions obtained from multitude of decision trees. Each leaf node also stores a edge map (Multiple overlapping edge maps) along with the learned mask. Thus, predictions can be combined quickly and simply by averaging.

After performing the above steps, we get a ***probability edge map*** on which we can apply thresholding to get the binary edge map.

(c)Performance Evaluation:
The goal is to quantitatively compare the outputs of edge maps produced using different edge detectors. For this we need an edge map produced by humans. The performance is evaluated by comparing the machine generated edge map and human designed edge map. This performance evaluation is denoted by a parameter called the "F- measure or 'F' ".
It is given by the formula,

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

Where P = Precision and R = Recall.

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

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

True Positive = Edge pixels in the image coincide with the edge pixels in the ground truth.
True Negative = Non-edge pixel in the image coincides with the non-edge pixel of the ground truth.
False Positive = Edge pixels in the image correspond to non-edge pixels in the ground truth.
False Negative = Non-edge pixels in the image correspond to the edge pixel in the ground truth.

But opinions of people are different and to handle such situation, we consider a number of ground truths in place and take the mean value for all the parameter we consider. Example, Mean Precision, Mean Recall, Mean F- measure.

EXPERIMENTAL RESULT:

(a) (1). Sobel Edge Detector:

Figure 47: Boat RGB

Figure 48: Boat_Noisy RGB

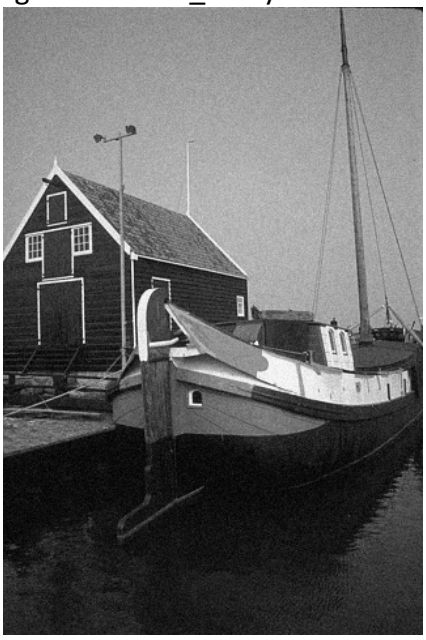


Figure 49: Boat Grey

Figure 50: Boat_Noisy Grey

Figure 51: Boat_GX

Figure 52: Boat_Noisy_GX

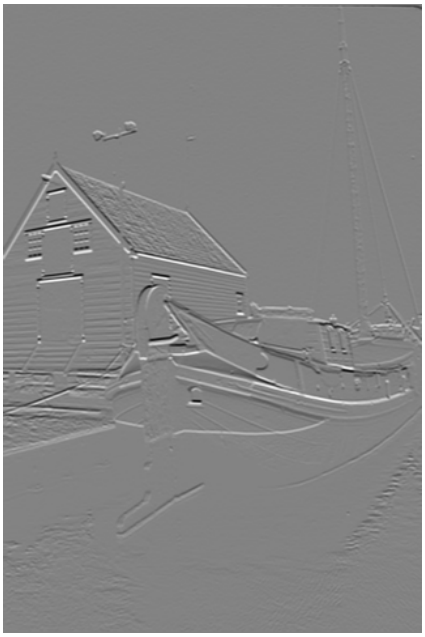


Figure 53: Boat_GY

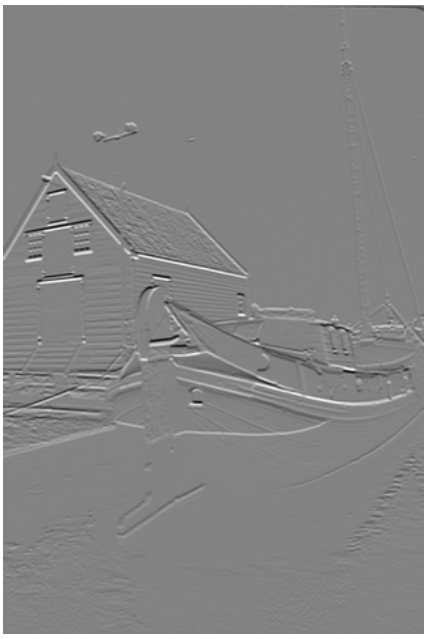


Figure 54: Boat_Noisy_GY

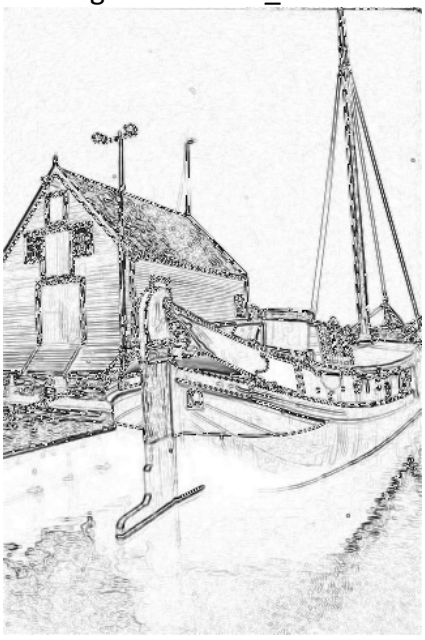


Figure 55: Boat without thresholding

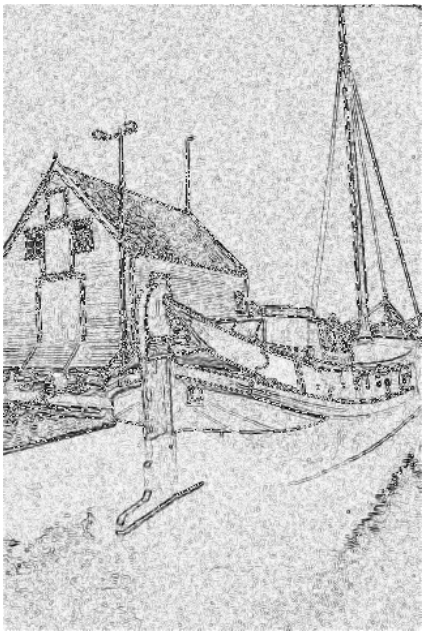


Figure 56: Boat_Noisy without Thresholding

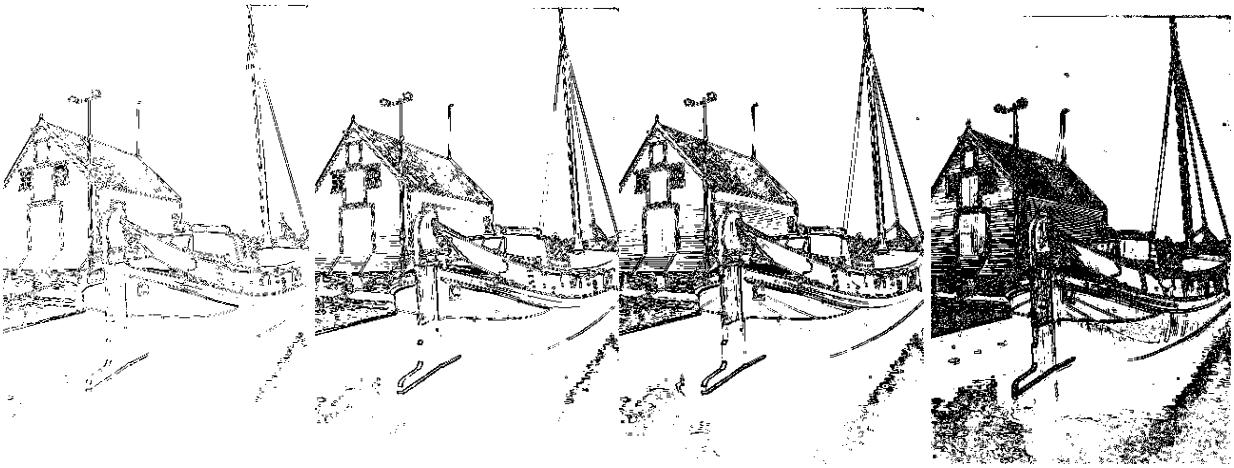


Figure 57: Boat Threshold = 95, 90, Threshold = 85, Threshold = 70.
T_value=162

```
For:95%
Threshold=162
For:95%
Threshold=165
```

```
For:90%
Threshold=104
For:90%
Threshold=110
```

```
For:85%
Threshold=71
For:85%
Threshold=83
```

```
For:70%
Threshold=28
For:70%
Threshold=55
```

T_value =165

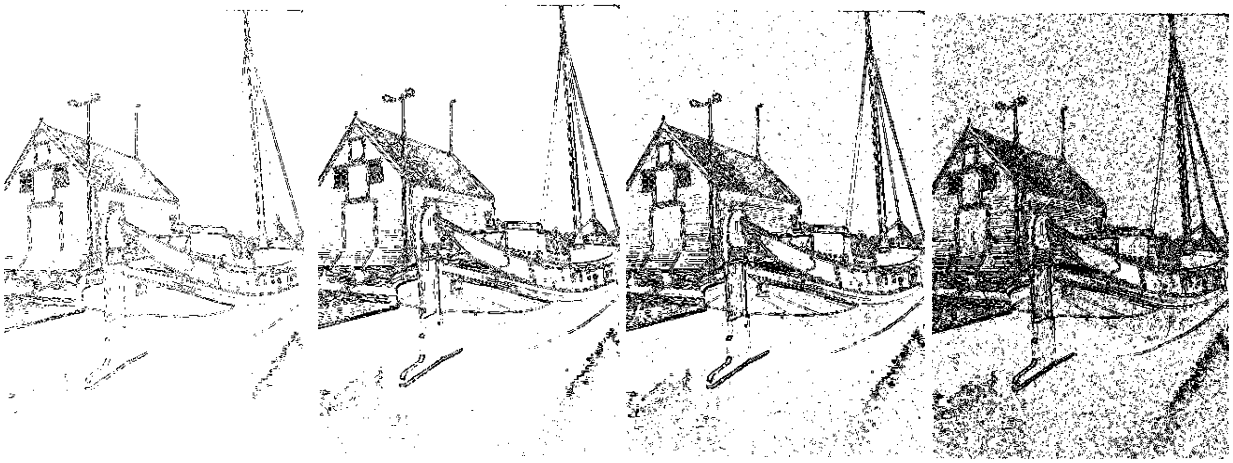


Figure 58: Boat_Noisy Threshold = 95, = 90, Threshold = 85, Threshold = 70.

(b) Zero Crossing Detector:

Figure 59: Boat RGB

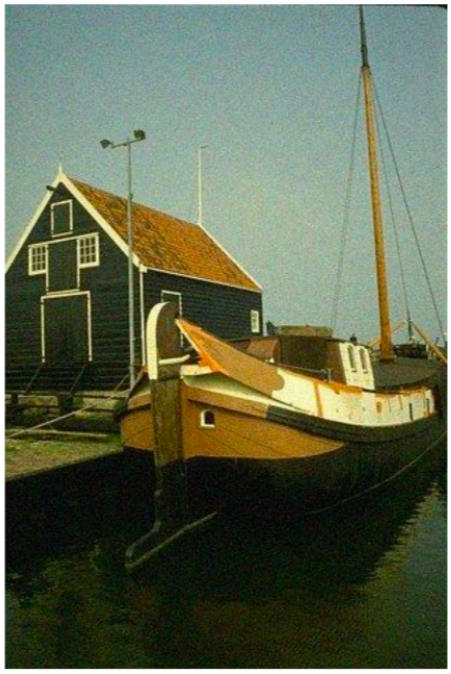


Figure 60: Boat_Noisy RGB

Figure 61: Boat Grey

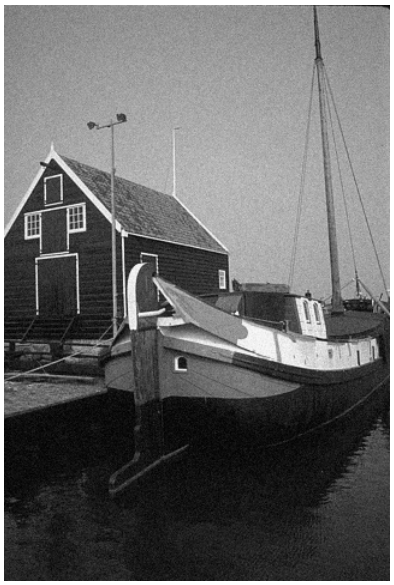
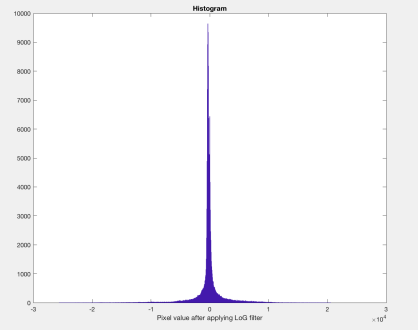


Figure 62: Boat_Noisy Grey



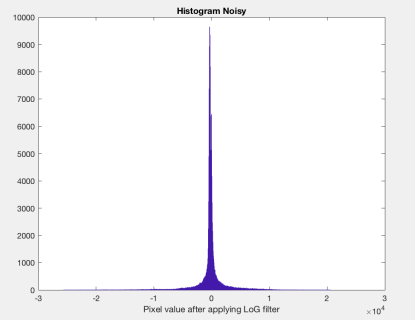
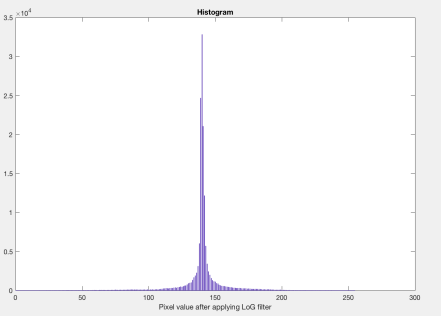


Figure 62.2: Histogram before Normalization Boat(left) and Boat Noisy (right)



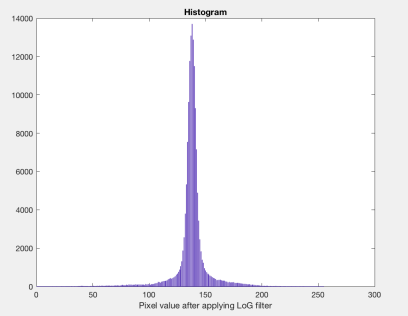


Figure 62.3: Histogram after Normalization Boat(left) and Boat Noisy (right)

Figure 63: Ternary Map (TM) 115,165
Knee point 1 = 115, Knee point 2 =165.

Figure 64: TM 110,165

Figure 65: LoG 115,165

Figure 66: LoG 110,165

Figure 67: TM 125,155

Figure 68: TM 120,155

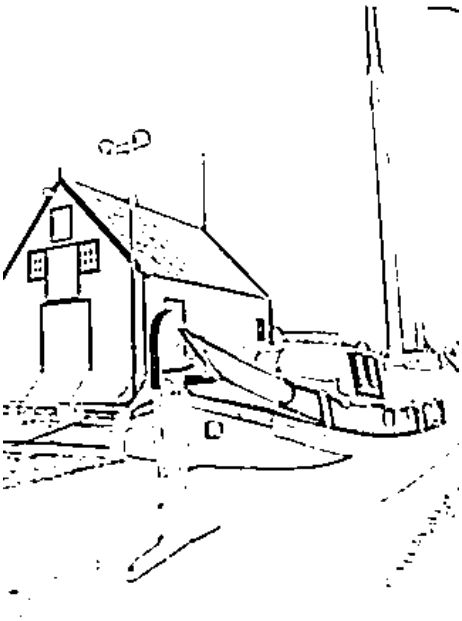


Figure 69: Log 125,155

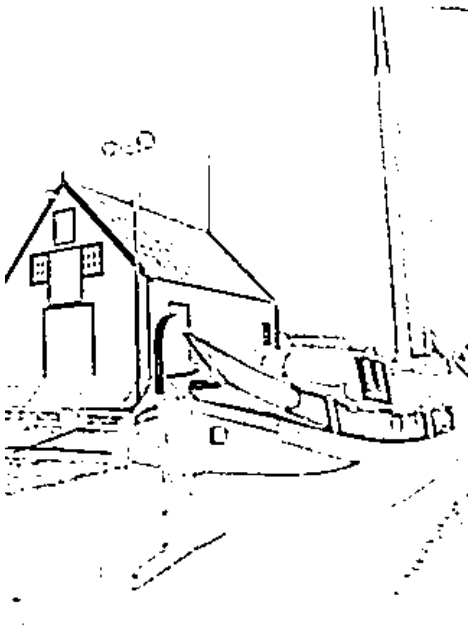


Figure 70: Log 120,155

Figure 71: TM 130,150

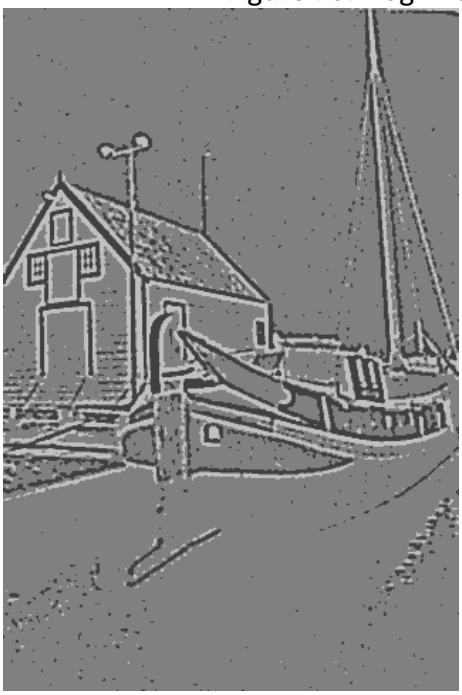


Figure 72: TM 130,150

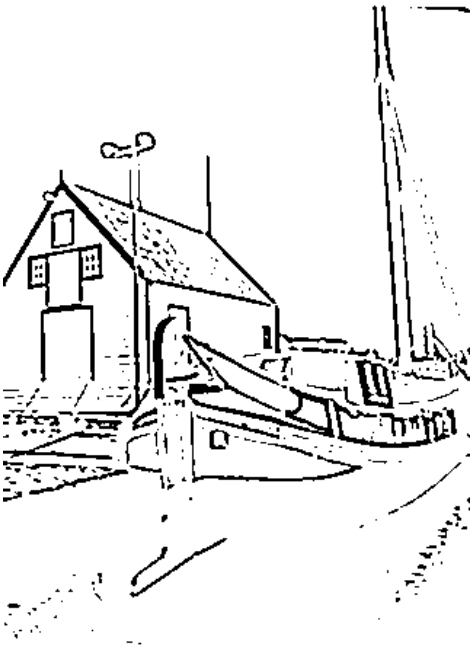


Figure 73: Log 130,150

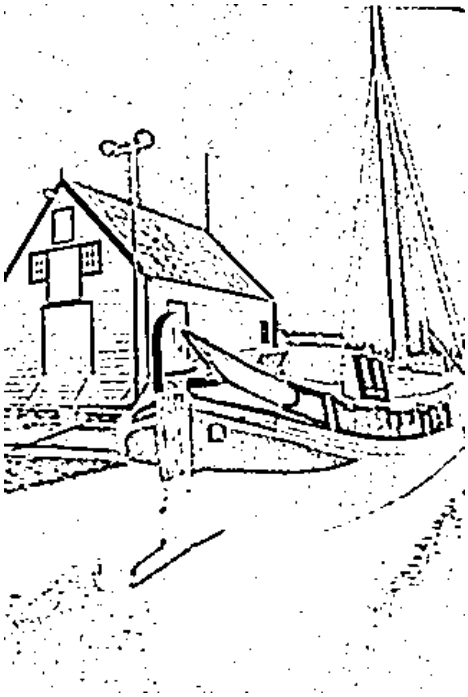


Figure 74: Log 130,150

Figure 75: TM 130,150

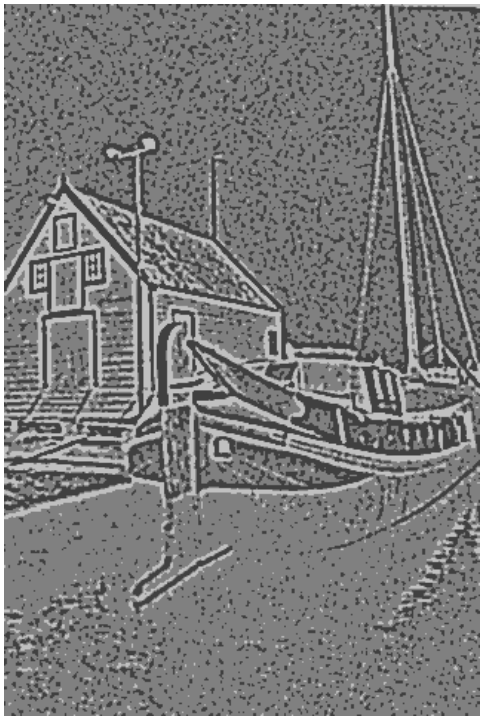


Figure 76: TM 130,150 +5

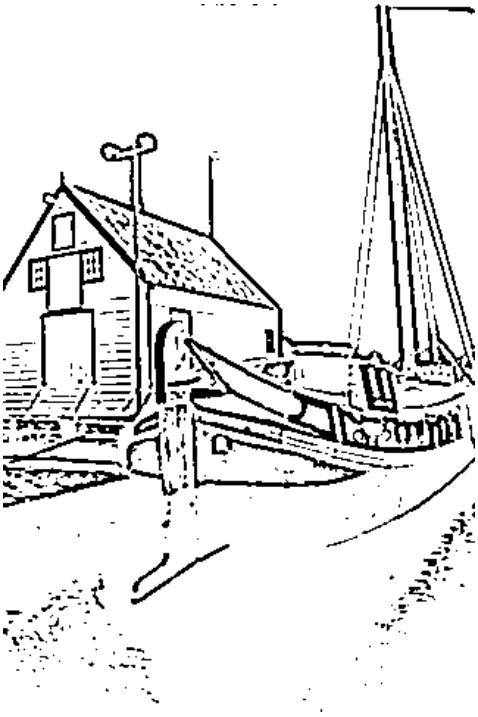


Figure 77: Log 135,155

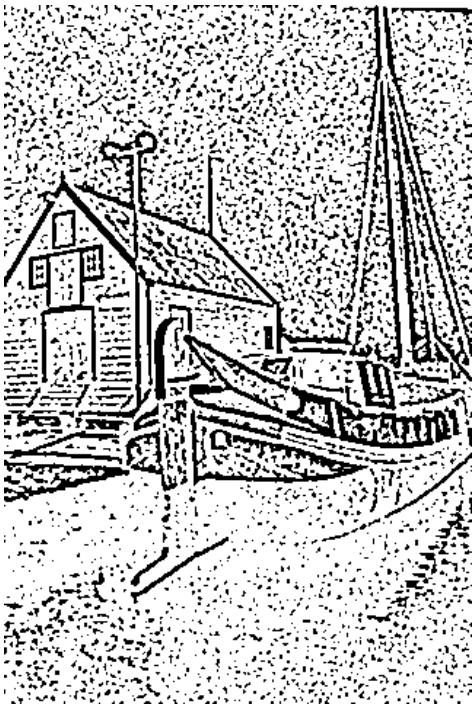


Figure 78: Log 135,155

(b)Structured Edge:

Figure 79: Animal RGB

Animal probability edge map

Figure 80: Animal t=70%

Figure 81: Animal t=80%

Figure 82: Animal t=85%

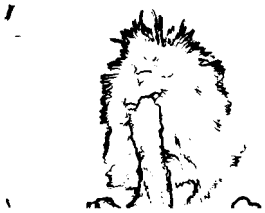


Figure 83: Animal t=90%

Figure 84: Animal t=95%

Given House RGB image.

Figure 85: House RGB

Probability Edge Map

Figure 86: House t=70%

Figure 87: House t=80%

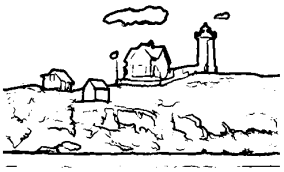


Figure 88: House t=85%

Figure 89: House t=90%

(c) Performance Evaluation:

ANIMAL: Table 2

Ground Truth	Mean Precision	Mean Recall	F- measure
1	0.747734	0.588935	0.658902
2	0.772659	0.521407	0.622642
3	0.447885	0.426619	0.436993
4	0.776738	0.619073	0.689001
5	0.692897	0.551247	0.614008

Animal:	all ground truths together.	LOG	SOBEL
F-measure =	0.683946	0.308491	0.300991
Threshold [](in Black BG) =	0.88 (0.12)	0.94 (0.06)	0.959 (0.041)

HOUSE: Table 3

Ground Truth	Mean Precision	Mean Recall	F- measure
1	0.696624	0.815224	0.751272
2	0.795425	0.646176	0.713074
3	0.768863	0.658217	0.709251
4	0.557339	0.711120	0.624908
5	0.525885	0.700567	0.600786

House:	all ground truths together.	LOG	SOBEL
f-measure =	0.782383	0.366043	0.312934
Threshold [] (in black BG) =	0.7700 (0.2300)	0.4400 (0.56)	0.5100 (0.4100)

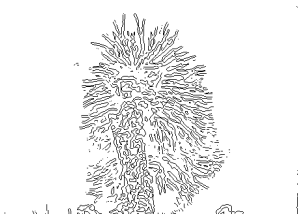


Figure 90: Animal LOG

Figure 91: Animal SOBEL

Figure 92: House LOG

Figure 93: House SOBEL

DISCUSSION:

(a) Basic Edge Detection:

(i) SOBEL: The X- gradient (GX) and Y-gradient (GY) of the given Boat and Boat_Noisy images are shown in the figures 51,53 and 52,54 respectively. From the obtained images, we can vividly see that the GX contains all the horizontal edges and the GY contains all the vertical edges. This clearly shows the horizontal and vertical edge detection.

For various threshold values [95%,90%,85%,70%], we've provided its equivalent binary image (With white as the background and black as the detected edge.). Images are shown for both Boat (clean) and Boat_Noisy. Refer Figures 55 through 58.

In the Boat_Noisy image, we see that the noise pixels are falsely detected as edges. Even though humans can classify noise from edge in the edge map produced, for machine level, they are still detected edges according to the algorithm.

The advantage of using Sobel edge detector is that we can modify the weight values in the kernels. Sobel detectors are slow in computational speed, but because of their larger kernel size, they smoothen the input images making the operator less sensitive to noise.

(*ii) Zero-Crossing detector:* The Laplacian of the Gaussian filter is applied to the input grey image. This Grey image is normalized to the range 0-255. The histogram for the pixel values after application of LoG filter is shown in the figures 62.2 and 62.3. for both normalized and un-normalized value.

From the histogram, respective knee points are retrieved and these knee points are used as threshold values to partition the image into -1, 0 and 1.

- This -1, 0, and 1 image is used to produce the ternary map by mapping the -1 to 64, 0 to 128 and 1 to 192 or in the reverse order respectively.

- ❖ And the same -1, 0, and 1 map is used to detect the zero crossing and create an edge map. Edge exists (i) If a zero is surrounded by -1 or 1 and (i) -1 or 1 is surrounded by 0. Thus we create a binary edge map for the given image.

For different knee points, the respective ternary map and the edge map is shown in the figures 63 through 78.

The simplest way to threshold is to threshold at 0, which will produce an image containing foreground and background pixels. Then a boundary detector function (marks if a background pixel is surrounded by a foreground pixel) can be applied to mark the edges. The advantage of using Zero-Crossing and LoG is that it is easy to detect edges and its orientation but it is highly sensitive to noise.

Comparing SOBEL with the ZERO-CROSSING Detector under visual basis, we can say that zero crossing detector (LoG…) has produced better edge maps than the Sobel. They are finer, confidently appearing solid edges on the edge map. Whereas in Sobel, the edges are slightly grainy in appearance.

(b) Structured Edge:

(I & ii) A detailed explanation for the functioning of Structured edge (SE) detector along with Random Forest (RF) classifier, that is implemented within the SE, is given in the Approach section of this problem. Also refer to figures 44 and 45. Functioning of Structured edge detector, Random forest and Decision tree will be vividly explained.

Parameters: [6]
Image Patch size: 32x32
Segmentation Mask: 16x16
Number of trees in the Random Forest to train: 8
Tree stope at a tree depth: 64
Number of samples after PCA (Principle Component Analysis): 256
Number of classes for Splitting: 2
Sharpness: 2
NMS: False

The probability edge map and its respective binary edge maps for different threshold values are shown in the figures 79 through 84 for the image ‘ANIMAL’, [Threshold = 70%, 80%, 85%, 90% and 95%]and 85 through 89 for the image ‘HOUSE’ [Threshold = 70%, 80%, 85% and 90%].
As the name suggests, the “Probability Density Function” is created using the probability of a pixel to be an edge. Darker the pixel is, greater is its chance of being an edge in the given image and vice versa. This probability edge map is used and on it, threshold is applied (p>0.1 [black-BG, white edge] or 0.9 [White BG, Black edge]) and respective binary edge map is obtained.
Comparing the edge maps produced by the three edge detectors we’ve discussed so far, we can clearly say that the edge map produced by the SE detector is much better than the edge maps produced by Sobel or Zero-crossing detectors. See

	Animal	House
Structured Edge Detector	Figure 83 [90% T]	Figure 89
Sobel Edge Detector	Figure 90	Figure 92
Zero-Crossing Detector	Figure 91	Figure 93

And also Figures 57 and 58.

These figures sufficiently give us an understanding in the differences between the edge maps generated by the Sobel, Zero-crossing and the Structured edge detectors.

(c) Performance Evaluation:

(i) The values for precision, recall and F-measure for each individual ground truths and the mean precision, mean recall and F-measure computed using mean precision and recall for all the ground truths combined is shown in the Tables 1 and 2 for the images Animal and House respectively.

The ***program parameters*** that were set while computing these parameters are,

Multiscale: 0
Sharpness: 2
Number of decision trees: 8
Number of threads: 4
NMS (Non-Maximal Supression): 0
thrs: 99 [that is, it'll give an array of size 99]
Tolerance: 0.0075
Thinning Factor: 1

By looking at the tables 2 and 3, we see that the F-measure for SE generated edge map is greater than any of the F- measures computed, that is, Sobel and LOG. In the order of decreasing F- measure (Highest to lowest) we can state that the edge map produced by

SE edge detection > LOG edge detection > Sobel Edge detection.

And also, we that the F- measure for house is greater than the F- measure for animal.

(ii) Yes, F- measure is image dependent. This could be easily seen in the observation of F- measure values of House and Animal. F-measure (House) > F-measure (Animal). This is because, from the given images it can be seen that the House image has better separation of gradient magnitude between edges than the Animal image. Also since F- measure is dependent on the ground truths and in turn these ground truths are dependent on humans, drawing ground truths for House image is more precise and finer than drawing ground truths for Animal. The Animal has so many edges concentrated in a particular area and for humans, it is difficult to make a keen note on all of its edges properly. This is the reason for the F- measure value to be higher for House than for Animal image.

(iii) F- measure can be given by,

$$F = [2 \times (P \times R)] / [P + R]$$

It is intuitive that P and R values go hand in hand. So, when P increases, R decreases and Vice versa. So, the term [P + R] is always a constant and the only term that'll differ is [P x R].
So, when One is large, automatically the other one becomes smaller and the numerator value decreases. Hence for F- measure to be maximum, P must be equal to R.
Say P+R is a constant C,

➔ P = C- R
➔ F = [2 x ((C- R) x R)]/[C-R+R]
➔ F = [2 x (C - R) x R]/[C] ➔ F = [2 x (CR - RR)]/[C]

taking the first derivative and substituting F'=0,

➔ 2C – 4R = 0
➔ R = C/2

so therefore P = C – R ➔ P = C/2.
Therefore, P and R are equal in order to obtain Maximum F – Measure.

PROBLEM 3: SALIENT POINT DESCRIPTORS AND IMAGE MATCHING
AIM:
(a) Extraction and Description of Salient Features:
To extract and show both Scale Invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF) features for the following images,

Figure: Optimus Prime Truck

Figure: Bumble Bee Car

Compare these results in terms of efficience and performance.
(b) Image matching:
(i)To Extract and show the SIFT features for the given race cars,

Figure: Ferrari 1

Figure: Ferrari 2

And show the corresponding SIFT pairs between the 2 images.
(ii) Repeat for SURF features.
(iii) Perform the same operation with (I) Ferrari 1 and Optimus Prime Truck and (II) Ferrari 2 and BumbleBee car.
Comment on the matching results and explain why it works or fails.
(c) Bag of Words:
(i) To extract the SIFT features from the following three images, Optimus Prime truck, BumbleBee car and Ferrari 1, and create a codebook by applying k-Means to the SIFT features. To create a codebook for all four images and match ferrari 2's code words with other images. Show the result and discuss your observation.

ABSTRACT AND MOTIVATION:
The main aim of this problem on how we compare two images. For human eyes it is very intuitive to describe the features of an object in the image and compare them with others. But when it comes to machine level, it is quite tricky than the former. In general the features of an image include edges, corners, blobs , ridges etc. And detecting features and extracting it is a

very important task in image processing, computer vision and other such related fields. In applications, this is used is object detection, video tracking, panaroma etc. We have to extract features that are meaningful in computational level. We must make sure that the features that are chosen as the image's attributes are invariant to scaling, rotation and partially invariant to illumination (to some extent, invarient in 3-D view). Such algorithm for feature extraction are Scale Invariant Feature Transform and Speeded Up Robust Features.
Once meaningful features are extracted, we can compare and match them based on their similarity. Such operations play important role in the field of Computer Vision.

APPROACH AND PROCEDURE:
For an Image, the features can be extracted and described using the the following algorithms.
(i) SIFT – Scale Invariant Feature Transform and
(ii) SURF – Speeded Up Robust Features.
The extractio of features using SIFT is described below,
SIFT Algorithm:

- Scale Space Extrema Detection: This is done to achieve the scale invarianceness of the key points in the image. Here we consider an input image on different scales. As we increase the scale for an image, the variance of the points in the image decreases. And these points that have passed through this scaling/gaussian blurring process are said to be storng key points. Given below is an example where an image is scaled up for different sigma values.

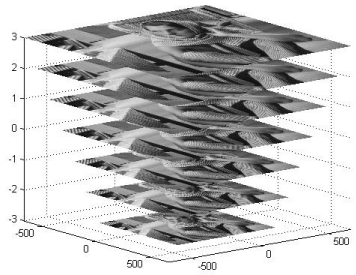


Figure: Scaling

The scale space consists of 'n' octaves and each octave is downsampled by a factor of k= 2 (author: k= sqrt(2)). After this step Difference of the Gaussians (DoG) is computed which helps in approximation of the LoG. The DoG gives us an edge map to detect points of interest and
these points are selected such that they are retained as original even after scaling processes.

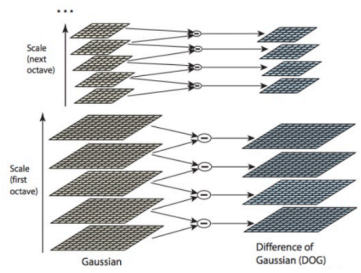


Figure: Scale: LoG and DoG.

- Key Point Localization: The aim of this step is to get rid of outliers and other bad key points. Bad key points include Edges and low contrast regions. Eliminating such keypoints makes the algorithm perform faster and in a more robust manner. This can be done by performing Taylor series expansion and remove all those points that lie below the set threshold value (say 0.3). We also compute the hessian matrix and compute it's Eigen values. If the ratio of the first eigen value and next is greater than 10, then we have encountered an edge and so we discard it. Corners do not face such problems.
- Key Point Orientation: In this step, we figure out the orientation of each of the key points. For this we consider a 16x16 patch around the key point. That neighbourhood information acts as the SIFT descriptor that aids in computing our SIFT feature vector. The 16x16 patch is broken into 16 4x4 patches and for each sub block 8 bin orientation histogram is created. Thus we will have 16x8 = 128 bin features (i.e. 128 D feature vector.)

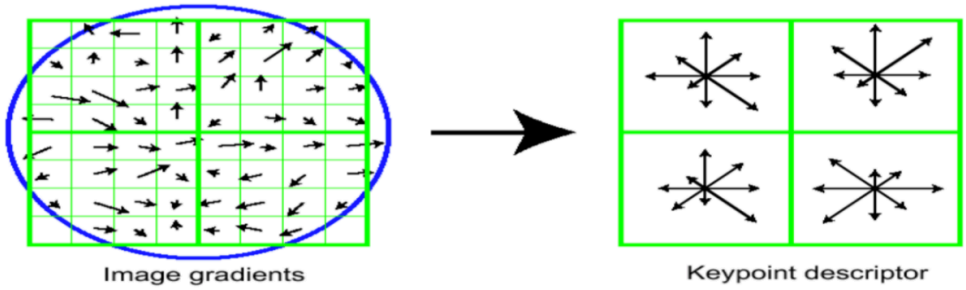


Figure:Histogram for Orientation.

Thus a the key points become rotation invariant also.

Thus we have extracted keypoints the keypoints that are invariant to scaling and rotation. And these keypoints are marked on the input image and can be shown as the output for obeservational purposes.

SURF Algorithm:

SURF is an economic and computational friendly algorithm, that is, It produces more features even for smaller objects in lesser computational time.

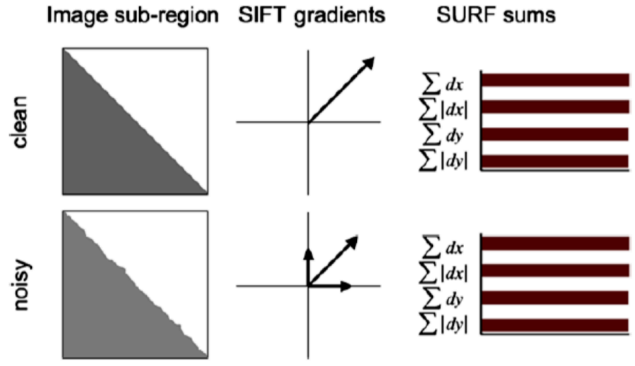
- SURF makes use of the Intergral filter, which enhances the speed of operation. [It operates faster irrespective of the filter size.]
- Once the Intergral image is created, we compute the Hessian matrix and use this combination to find the sum of intensities or squares of intensities in the hessian box filters and to it is added a weight vector. This step gives us a weak confidence on key points.

  Note: LoG filter is used in the Hessian matrix.
- To the filter manipulated image above, we perform a 3x3 Non-Maximal Supression below and above the scale space of each octave. This step find all the major key points or interest points.
- To extract the feature direction/orientation detection, we apply Haar transform. Then a 20s X 20s size patch is taken (where s denotes the size) around the neighbourhood of

the keypoint and is divided into 4x4 sub regions. For each sub region, Horizontal and Vertical wavelet responses are taken and a vector is formed like shown below.

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|).$$

This represented in mathematical form gives us a 64 D feature vector.



- The descriptors/feature vectors hold the actual features at each keypoint in the respective images.

(b) Image Matching:

- Two images that are to be compared is read as the input images of the system. These images are converted into greyscale images.
- Corresponding features of the converted grey scale images are extracted either using SIFT or SURF. (SIFT gives Nx128 feature matrix and SURF gives Nx64 feature matrix). The descriptors hold the actual features at each key point in the respective images
- Now we have to compare the descriptors of the two input images. This step can be done by using many different algorithms. A few of them are, knn matcher, BruteForce Matcher and Flann Matcher etc. (For this problem I have used Brute Force Matcher BFM).
- Now we sort the matched descriptor in an order of their distances and we apply a threshold to display only the top 10 or 30 or N features of the images. So far we have just marked the matching features, now we join them using lines and display the image showcasing the object and its common key features.

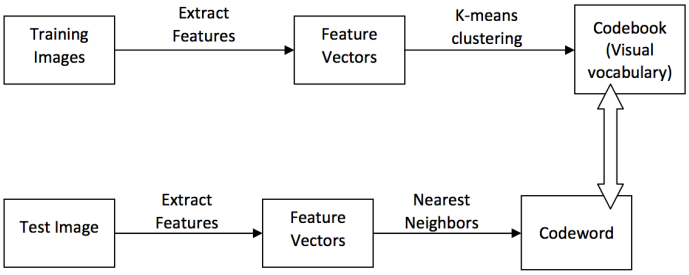
Brute Force Matcher:

It is the most simplest form of matching algorithm. After we compute the features from both the images. It compares the distance between one key point from the image 1 with all the other keypoints of image 2. The key point from image 2 that produced the least distance is assigned as the match for the key point from image 1. This is repeated for all the keypoins of both the images.

(c) Bag of words:

Bag of Word or Bag of visual words is one of the most popular means used in the field of Image classification. These words are basically the features of the image.

- We extract the features of the image using SIFT technique. SIFT features stand a good chance to be the optimum feature extraction technique in BoW concept as the order of the feature vector is not important.
- We apply k-Means cluster to this extracted feature vector to build our own code book. In k-Means, we take k = 8 bins and the code book is basically a dictionary that consists of code words made of descriptors.



EXPERIMENTAL RESULTS:

(a) Extraction of Key points.

Figure : OptimusPrime Truck

Figure: BumbleBee Car

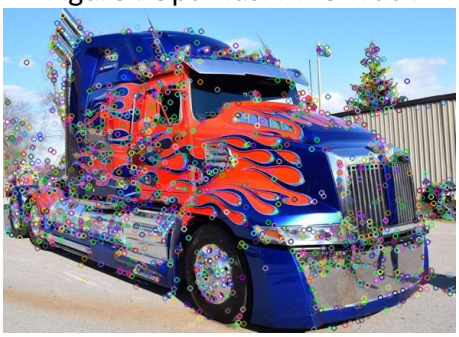


Figure : OptimusPrime Truck SIFT features

Figure: BumbleBee Car SIFT features

Figure : OptimusPrime Truck SURF features

Figure: BumbleBee Car SURF features

```
In [14]: runfile('/Users/rickerish_nah/Documents/t
rickerish_nah/Documents/test')
Time take for SIFT to run: 0.26518702507019043

In [15]: runfile('/Users/rickerish_nah/Documents/t
rickerish_nah/Documents/test')
Time take for SURF to run: 0.20658087730407715
```

Figure: Showing SIFT is better than SURF

(b) Image Matching:

Fig: Ferrari 1

Fig: Ferrari 2

Fig: Ferrari 1 SIFT features

Fig: Ferrari 2 SIFT features

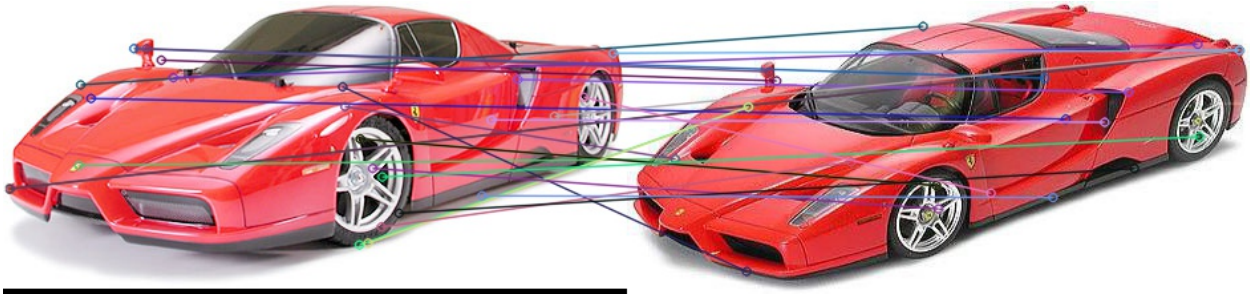


Fig: Ferrari SIFT matching threshold=75 %

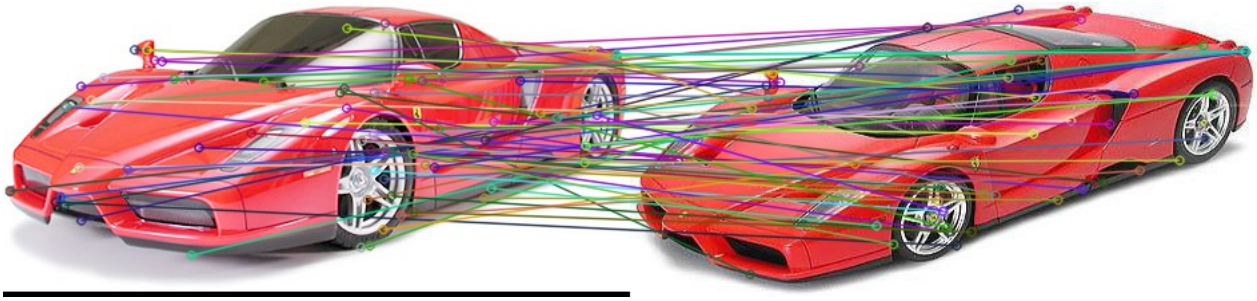


Fig: Ferrari SIFT matching threshold=85%

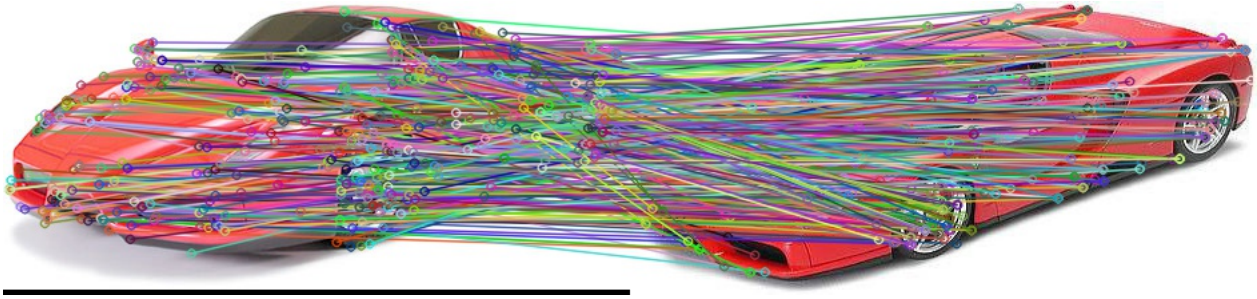


Fig: Ferrari SIFT matching threshold=100%

Fig: Ferrari 1 SURF features

Fig: Ferrari 2 SURF features

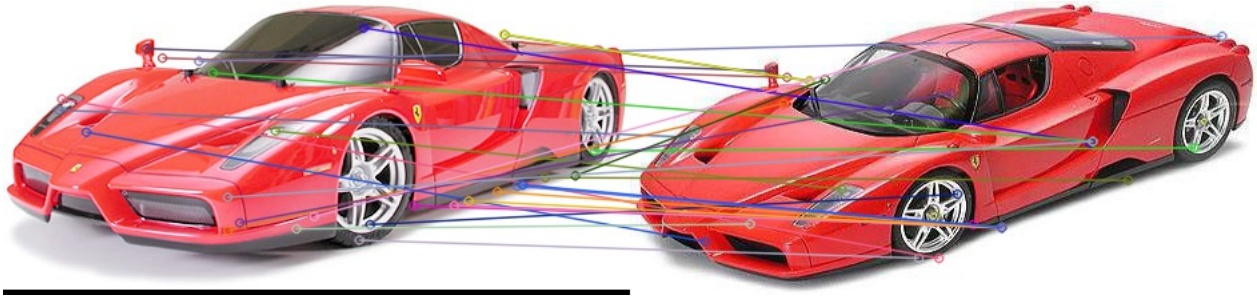


Fig: Ferrari SURF matching threshold=75%

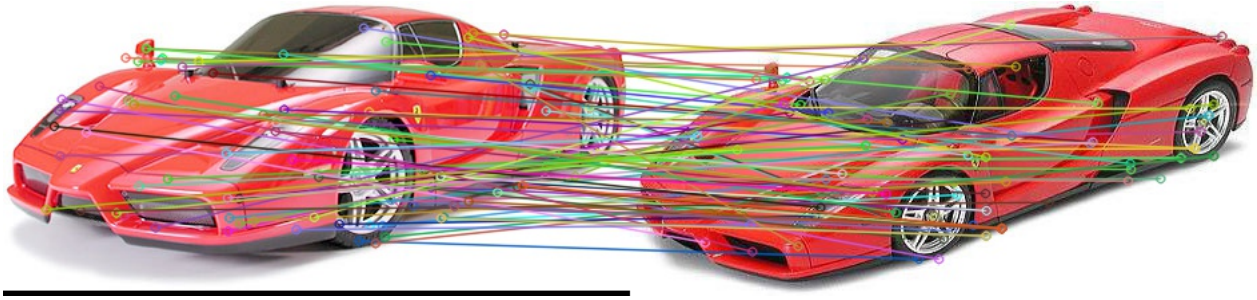


Fig: Ferrari SURF matching threshold=85%

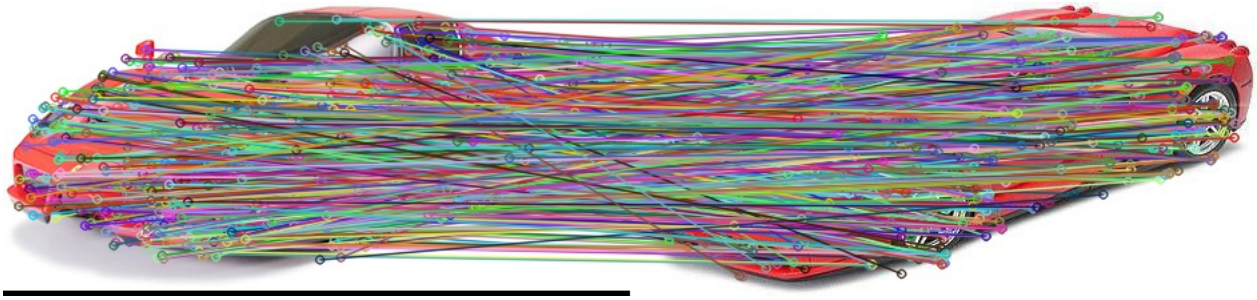
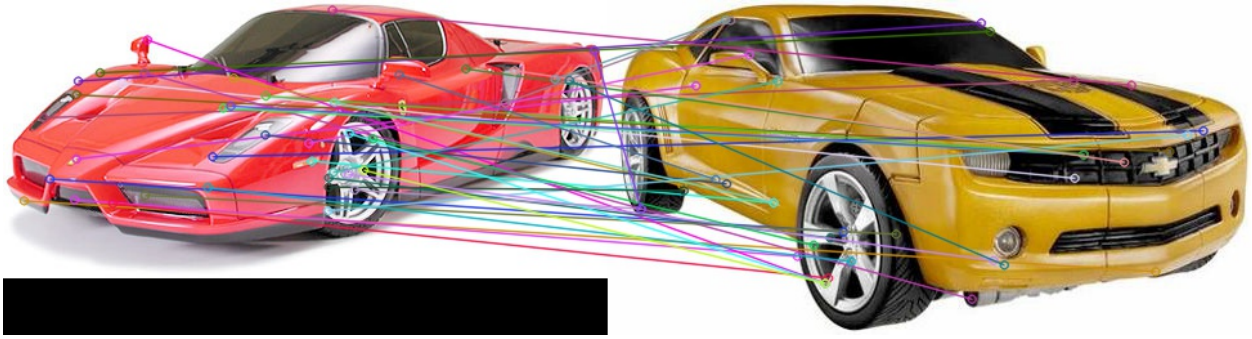
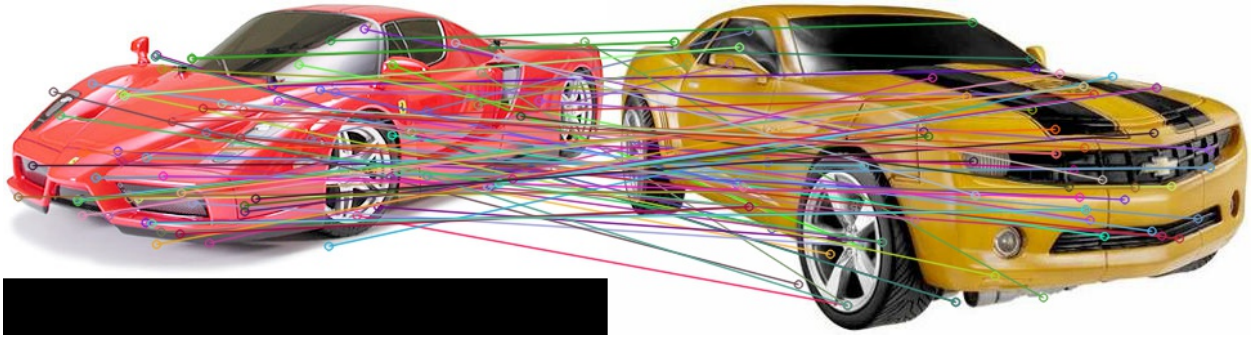


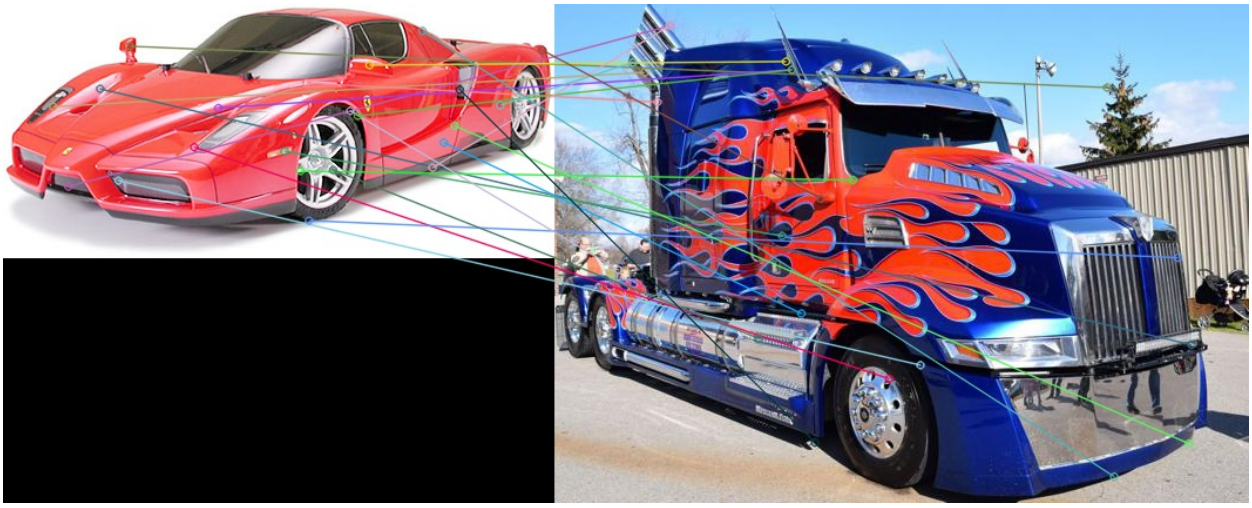
Fig: Ferrari SURF matching threshold=100%



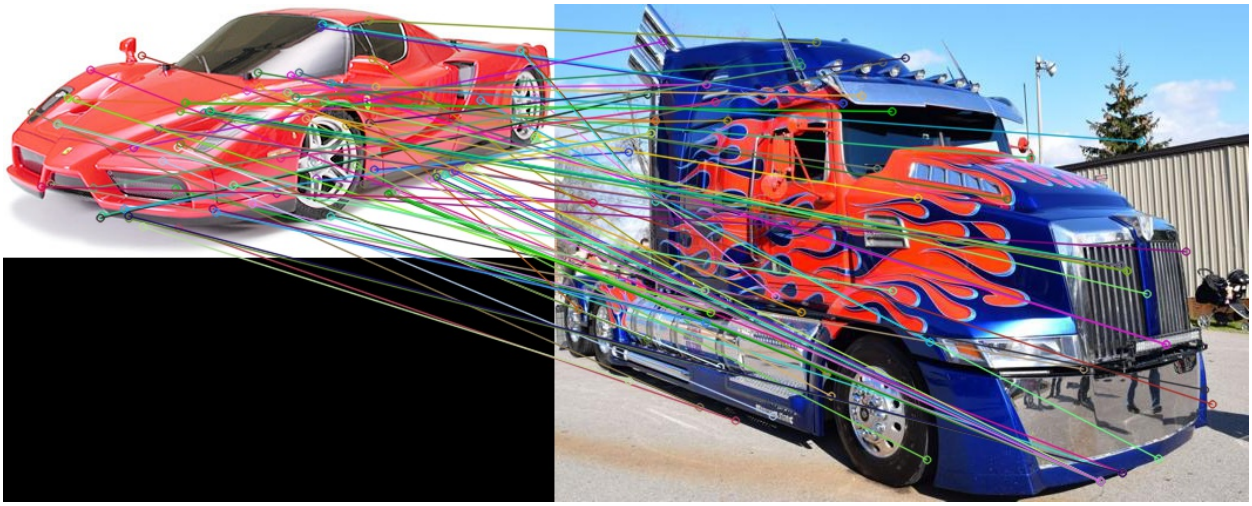
SIFT



SURF



SIFT



SURF

(c) Bag of Words:

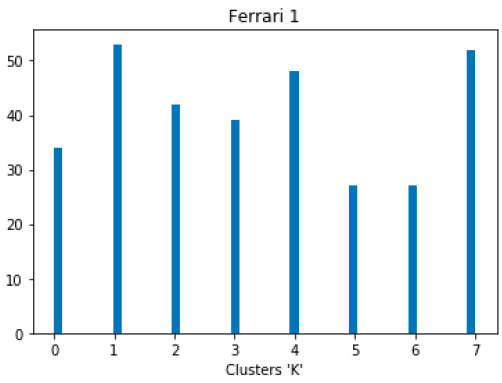


Figure: Ferrari 1 histogram

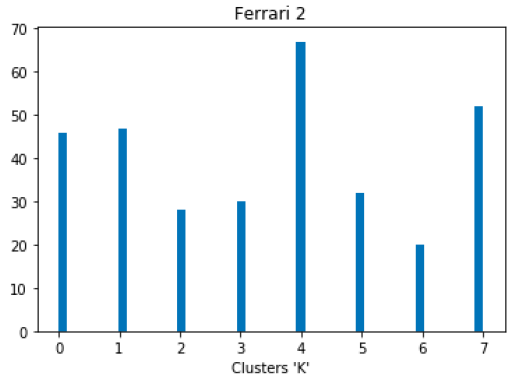


Figure: Ferrari 2 histogram

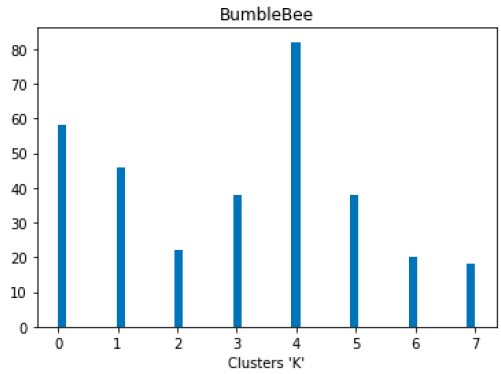


Figure: Bumble Bee car histogram

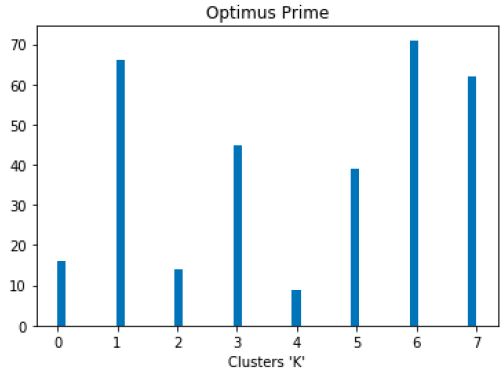


Figure: Optimus Prime histogram

DISCUSSION:

(a) The key points were successfully extracted using the SIFT and SURF techniques.
It can be seen from the program run that SIFT takes 0.26 and SURF takes 0.20 seconds to compute. Not only SURF is faster but also it extracts more key points than SIFT.
(b) Image Matching is performed here. Refer to the figures in the experimental result section. For Ferrari 1 and 2 we see that there more key points that are matched and so is for Ferrari 1 and Bumble Bee.
But when we see Ferrari 1 and Optimus prime, we can say that they have only few features in common and hence the matching of key points is less dense.
(c) In this part we have plotted the historgram for each of the Image fed into the trained Classifier. (A;lso called as codebook comparison). From the plotted histograms it can be clearly seen that the historgrams of Ferrari 1 amnd Ferrari 2 are nearly the same.
When we compare Bumble Bee with the Ferrari, it still produces reasonable matches and has a similar histogram.
But when we compare the Optimus Prime truck with the ferrari, we have onlt few features matching and as a result we can see that their histograms are distinctive or different in form.

REFERENCES:

[1]http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OJALA1/texclas.htm
[2]Wikipedia
[3]http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm
[4]https://www.tutorialspoint.com/dip/sobel_operator.htm
[5] http://homepages.inf.ed.ac.uk/rbf/HIPR2/LaplacianoftheGaussian.html
[6] "Fast Edge Detection using Structured Forests", Piotr Dollar and C. Lawrence Zitnick, arXiv Journal, 2015.