

EE 569 Homework #3

Maroof Mohammed Farooq

maroofmf@usc.edu

7126869945

Table of Contents

| | |
|--|----|
| 1. Problem 1 | 3 |
| 1.1 Abstract and Motivation | 3 |
| 1.2 Approach and Procedures | 3 |
| 1.2.1 Texture Classification..... | 3 |
| 1.2.2 Texture Segmentation..... | 4 |
| 1.3 Experiment Results..... | 5 |
| 1.4 Discussion..... | 7 |
| 2. Problem 2 | 9 |
| 2.1 Abstract and Motivation | 9 |
| 2.2 Approach and Procedures | 9 |
| 2.2.1 Extraction and Description of Salient points | 9 |
| 2.2.2 Image Matching..... | 10 |
| 2.2.3 Bag of Words..... | 10 |
| 2.3 Experiment Results..... | 11 |
| 2.4 Discussion..... | 14 |
| 3. Problem3 | 15 |
| 3.1Abstract and Motivation | 15 |
| 3.2 Approach and Procedures | 15 |
| 3.2.1 Canny Edge Detection | 15 |
| 3.2.2 Structured Edge..... | 15 |
| 3.2.2 Performance Evaluation..... | 16 |
| 3.3 Experiment Results..... | 17 |
| 3.4 Discussion..... | 20 |
| 4. References | 21 |

1. Problem 1

1.1 Abstract and Motivation:

Texture of an image is like a fundamental structure that obeys statistical properties. Image textures have similar structures that repeat periodically. These textures have a certain degree of randomness to it. It is important to study different textures present in an image as they provide important details required for image segmentation. In essence, texture provides important information about spatial arrangement of intensity values.

For segmentation purposes, it is important to differentiate textures based on their properties. In general, we attempt to differentiate textures using K means clustering. This would require us to extract features from the image and represent them on the feature space. In this problem we apply 5x5 laws filter that was proposed by Laws in the 1980s. These filters have certain properties that are discussed in detail. We obtain the energy values from each of the 25 filter responses for a texture type and represent them as a point in 25D feature space. Then clustering algorithms are applied and its performance are evaluated. We also attempt to study feature importance and feature reduction using Principal Component Analysis.

The main motivating factor behind studying texture classification lies behind improving image segmentation results. In this problem, we attempt to segment two texture mosaic based on the features mentioned above. We use a window based method for computing energy. In this problem, we have discussed the effects of different window sizes on segmentation quality.

1.2 Approach and Procedures:

1.2.1 Texture Classification

This is a typical machine learning problem. We need to extract good features and deploy K means to cluster the feature points. We are give twelve images of four different cluster types which are used for training. We test the accuracy of clustering on the so formed cluster centroids.

To implement this, we first generate a 5x5 laws filter from the five 1D filters provided to us. Each of the 2d laws filter has a specific frequency response that filters out certain frequencies. The frequency response of these filters is discussed in section 1.4. Each 1D filter is combined with another filter by taking a tensor product to produce a 5x5 filter. These 2d filters are stored in a hash map. As any typical machine learning problem, we can describe this implementation using 4 steps:

- **Pre-Processing:** After the generation of the 2d filters, we read each of the input images and subtract the DC component from it. This ensures that we don't have unnecessary high feature values as DC component has high energy but less information. This is one of the preprocessing steps we do before feature extraction.
- **Feature Extraction:** We then convolve the preprocessed image with the 25 laws filter stored in the filter bank. Each training image would give us 25 output images that represent the filter response. We then calculate the energy of each filter response by apply the following formula:

$$\text{Energy} = \frac{1}{N*M} * \sum_{i=0}^N \sum_{j=0}^M (I(i,j))^2 \quad \text{-----(Equation 1.1)}$$

This is applied to each training image's filter response. Finally, we will have a 25D feature vector for each training image. Each value of the feature vector represents the energy of the filter response for a given image. We then use this information to perform clustering/classification.

- **Dimensionality Reduction:** We have 25 features which may not be equally important. The feature importance is discussed in section 1.4. It then becomes necessary for us to reduce the dimensions and hence reduce the time taken for clustering/classification. Dimensionality reduction can be seen as a process of deriving a set of features that reproduce most of the variability in a given dataset.^[1] We have implemented Principal Component Analysis that reduces the feature space from 25 dimensions to 3 dimensions. Basically PCA calculates the Eigen values and Eigen vectors of higher dimensional data space and reduces it to n dimensions whose basis are the Eigen vectors corresponding to n largest Eigen values. Here, n is equal to 3. To implement this, we utilized OpenCV's PCA function to calculate the principal components and project the high dimensional data onto a 3D plane.
- **Clustering:** In order for us to classify an unknown texture type based on a set of known texture types {Rock, Grass, Weave, Sand}, we first cluster the features of these known textures into k points. So, we know that our feature space should consist of 4 clusters. We pass this to a K means clustering algorithm and determine the clustering centroid. K means is an iterative algorithm. It first assumes the cluster centroids at random and then computes the sample means spanned by the cluster centroid based on Euclidean distance. This step terminates when we have reached our maximum iteration limit or when the changes in cluster centroids are less than a threshold. We have taken different values of these terminating conditions and validated the result.

1.2.2 Texture Segmentation

To segment different textures from an image, we utilize the laws filter feature extraction and k means to form a segmented result. Firstly, we remove the DC component from the image. We do this to neglect high energy values that give very less information. We follow the steps given in section 1.2.1 and apply all the 25 laws filters on the input image. To perform energy value extraction, we consider a window of size n and compute the energy for that window. We then replace the center pixel with its calculated energy value.

Normalization was performed to remove the effect of high feature values. As we can see that filter L5-L5 multiplies the image and increases its intensities, we prefer to reduce its effects by dividing all feature values by L5-L5 value. Later k means was performed and the output labels were obtained from it. We then proceed to color label the segmented output and display it.

1.3 Experiment Results:

| Image Name | Actual Label | Output 1 | Output 2 | Output 3 | Output 4 |
|------------|--------------|----------|----------|----------|----------|
| Texture 1 | Rock | 1 | 1 | 1 | 1 |
| Texture 2 | Grass | 0 | 0 | 0 | 0 |
| Texture 3 | Weave | 3 | 3 | 3 | 2 |
| Texture 4 | Rock | 1 | 1 | 1 | 1 |
| Texture 5 | Weave | 3 | 3 | 3 | 2 |
| Texture 6 | Rock | 1 | 1 | 1 | 1 |
| Texture 7 | Sand | 2 | 2 | 2 | 3 |
| Texture 8 | Sand | 2 | 2 | 0 | 3 |
| Texture 9 | Grass | 0 | 0 | 0 | 0 |
| Texture 10 | Sand | 2 | 2 | 2 | 3 |
| Texture 11 | Weave | 0 | 0 | 0 | 0 |
| Texture 12 | Grass | 3 | 3 | 3 | 2 |

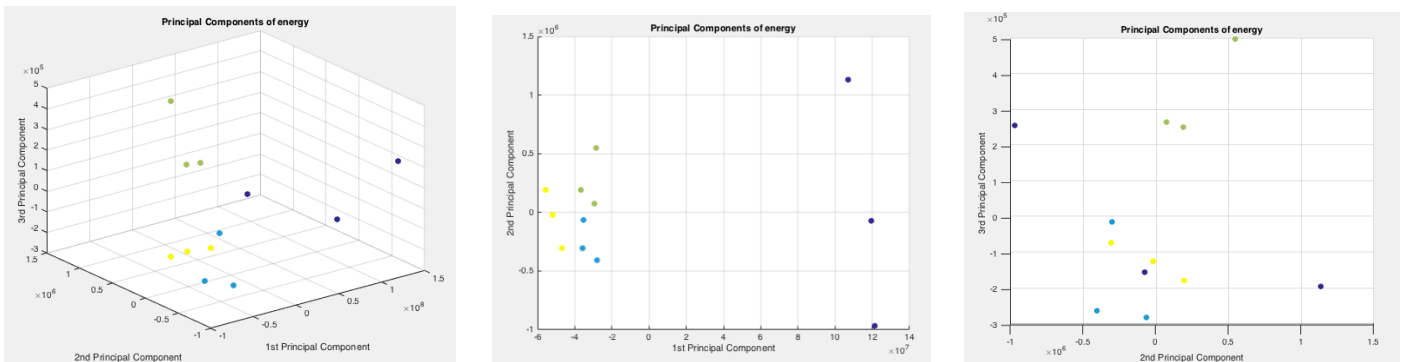
Table 1.1: K means output labels

| Output Number | Feature Size | Total attempts | Epsilon Value | Max Iterations | Initial points |
|---------------|--------------|----------------|---------------|----------------|----------------|
| Output 1 | 25 | 1 | 0.1 | 40 | RANDOM |
| Output 2 | 3 | 1 | 0.1 | 40 | RANDOM |
| Output 3 | 25 | 1 | 0.001 | 3 | RANDOM |
| Output 4 | 3 | 1 | 0.001 | 3 | RANDOM |

Table 1.2: K means output parameters selected

| Output Number | Rock | Grass | Weave | Sand | Average |
|---------------|------|--------|--------|--------|---------|
| Output 1 | 100% | 66.67% | 66.67% | 100% | 83.34% |
| Output 2 | 100% | 66.67% | 66.67% | 100% | 83.34% |
| Output 3 | 100% | 66.67% | 66.67% | 66.67% | 74.5% |
| Output 4 | 100% | 66.67% | 66.67% | 100% | 83.34% |

Table 1.3: K means output performance



(a) 3D View of feature points

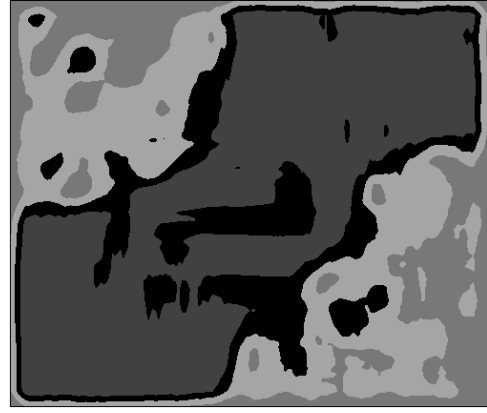
(b) 1st vs 2nd Principal Components

(c) 2nd vs 3rd Principal Components

Figure 1.1: PCA projected data points where each color represents each class

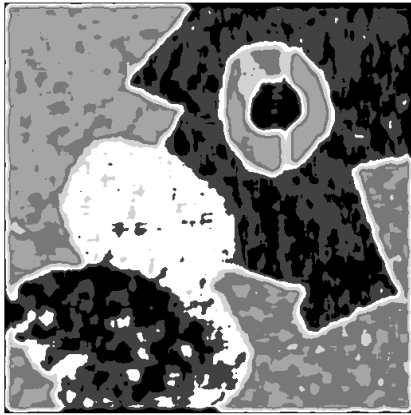


(a) Window Size = 15

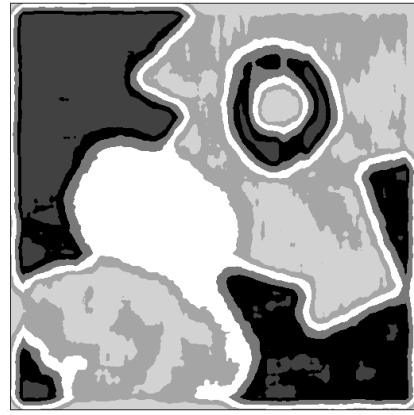


(b) Window Size = 31

Figure 1.2: Texture Segmentation of comb_1



(a) Window Size = 15



(b) Window Size = 31

Figure 1.3: Texture Segmentation of comb_2

1.4 Discussion:

It is important to study the frequency response of laws filter. We will learn how to appreciate the contribution of each energy value (feature) if we look at the frequency response. In the figures below, we can see the filter response for different filter combinations. From the filter responses, we can conclude the following:

- L5- Low pass filter with cut off frequency at 0.5
- E5- Band pass filter with lower cut off at 0.1 and upper cut off at 0.7
- S5- Band pass filter with lower cut off at 0.2 and upper cut off at 0.8
- W5- Band pass filter with lower cut off at 0.3 and upper cut off at 0.9
- R5- High pass filter with cut off frequency at 0.5.

When we convolve these filters with the input image, we capture different frequency contents from different axes. Some images like rock will have high output values when a combination of R5-R5 is passed. This shows how we can obtain features that are directly dependent on the filter response. The features are obtained by calculating the energy of the entire filter response. The energy value will be high if a particular filter response is suited with the image.

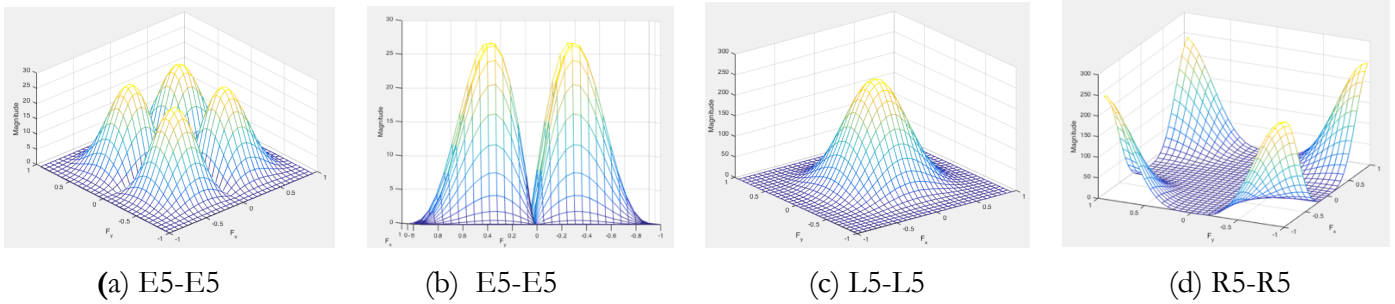


Figure 1.4: Laws filter response

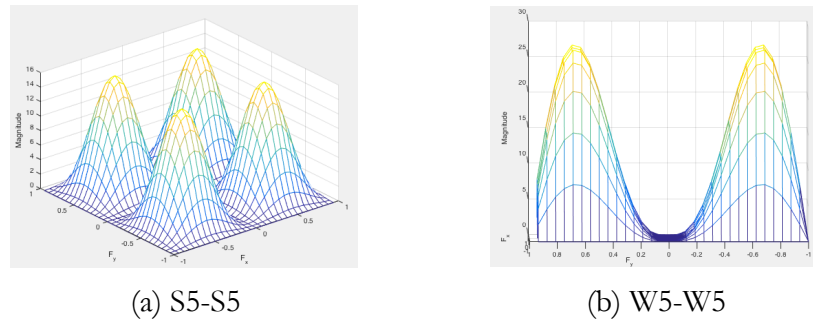
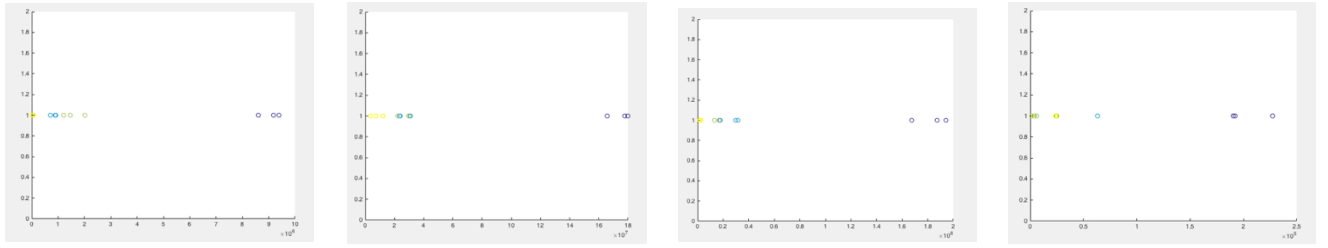


Figure 1.5: Laws filter response

To compare the discriminant power of features, we plot each feature on a 1d plane with different class labels and looked if the feature values overlapped with other classes. We can see the images in figure 1.6 to get a flavor of what was done. Here we can see how the data points overlap in the case of figure 1.6 (b), (c) and (d). This shows that the features which have many overlapping points will not classify the data with high accuracy. On the other hand, if we look at figure 1.6 (a) we can find that it has a really strong discriminant power which can classify all the texture types perfectly.



(a) E5-E5

(b) L5-L5

(c) L5-S5

(d) R5-R5

Figure 1.6: Feature selection; Here different colors belong to different classes.

Analyzing the features in this fashion, we can conclude that the strongest feature is E5-E5 and the weakest feature is R5-R5. It can clearly be seen in figure 1.6 (a) and (d).

Feature reduction doesn't seem to have a significant change in the performance of the clustering algorithms. In table 1.3 we can see that the performance of K means with 25D and 3D features are same. Upon further investigation, we found out that K means with 3D features converges faster than when 25D features are used. This can again be seen in table 1.3 where the iteration limit was set to 3. These experiments were tried multiple number of times and the output accuracy was found to be consistent.

From Figure 1.2 and 1.3 we can see the quality of segmentation based on different window sizes. We don't get a good result in the case of Comb_1 because the texture that is at the center can not be capture properly when we consider a small window size. If we want to capture the texture that is at the center, we need a large window size as the edges in that texture is not getting noticed in its neighboring pixels. To improve this segmentation, we can train our system with the nearest neighbor classifier for each of the texture and take windows of varying sizes to accommodate proper capture of textures. Additionally, L5-L5 was removed because it created many outliers and didn't contribute well to the clustering.

2. Problem 2

2.1 Abstract and Motivation

In applications such as object recognition, video tracking, navigation, gesture recognition and image stitching, we need a means to extract meaningful features that can describe the image. These extracted features are required to be scale invariant and rotation invariant to provide robustness. SIFT and SURF are such algorithms that extract important local features in an image that are scale and rotation invariant.

In this problem, we implement SIFT and SURF algorithms to compare their performance and efficiency. We also utilize these features to perform object matching using Brute Force matching technique. We will make an attempt to study its deficiencies and how we can overcome it in detail.

Sometimes we need to perform object categorization and localization to enhance the understand of an image. The biggest challenges in object classification are introduced due to problems such as different camera positions, illumination differences, internal parameters and variation within the object. These problem are effectively solved by the bag of words model. The underlying idea of this model is to create and learn a visual vocabulary from the training set and classify unknown images based on the vocabulary. This model coupled with robust features such as SIFT or SURF can provide surprisingly good results. We will explore this problem in detail in the following sections.

2.2 Approach and Procedures

2.2.1 Extraction and Description of Salient Points:

The general rationale behind extracting as good features as possible is that it contributes heavily towards the classification accuracy. “Garbage in, Garbage out” is rightly quoted to show the importance of feature engineering in any classification or clustering problems. So, this gives us an opportunity to explore better features for the purpose of image classification.

SIFT was proposed by David G Lowe and is given in detail by the author in [2]. SIFT stands for Scale Invariant Feature Transform. To implement this, we have used OpenCV that contains SIFT detectors and extractors. Following are the steps that are taken to extract SIFT features:

- **Scale Space Extrema Detection:** To make the key points scale invariant, we consider the input image on different scales. This is done by applying Gaussian smoothening with different values of sigma. The scale space consists of n octaves with 3 images in each octave. Each octave is down sampled by a factor of 2 following the first octave. Additionally, each images in a octave is smoothened by a sigma value that is a multiple of k . The author heuristically found the value of k to be $\sqrt{2}$. Similarly, the value of sigma was found out to be 1.6. After applying the Gaussian smoothening, the difference of each Gaussian is taken. This is to approximate the LoG function as the latter takes a lot of time to compute. The DoG gives us an edge map. To detect the point of interest from this edge maps, we iterate through all the possible points and select the point of interest which lies in the local maxima and minima at a given scale. This is done by comparing a given point at different scales.
- **Key point Localization:** In the previous step, we found out the interest points. These points of interest may contain weak/noisy points due to low contrast or they may be part of an edge. To avoid such outliers, we compute the Taylor series expansion of DoG and remove the points that lie below a threshold of 0.3. We also compute the hessian matrix and compare its Eigen values. If the ratio of first and second Eigen values are greater than 10 that means that the point of interest is an edge point. So we discard it.

- **Orientation Assignment:** To achieve rotational invariance, we assign an orientation value to each key point. This is done by first taking a neighborhood around the point of interest. The neighborhood size depends on the scale at which the point was detected. Then a histogram of 36 bins are created that represents 360 degrees. The orientation of each pixel in the window is weighted by its gradient magnitude and a Gaussian weighted circular window with sigma that is 1.5 times the scaling factor. The weighted orientation is plotted on the histogram and the maximum histogram value is selected. All the orientations that lie within 80 percent of the maximum are considered to increase the robustness.
- **Key point Descriptors:** Now it is important for us to create feature values for a point of interest. The author suggested to take the histogram of the neighborhood orientation as features as they provide variations against illumination effects, viewing angle difference etc. To do this, we first consider a 16x16 neighborhood from the interest point. This neighborhood is divided into 4x4 units from which histogram of the pixels in each of the unit is calculated. The resolution of histogram was taken to be 8. This in turn forms 128 histogram bins in the 16x16 neighborhood. The 128 bins are taken as a feature vector for a key point. This vector is normalized to a unit vector to cancel the illumination effects.

In the experimental results, we can see all the key points for different images. Similar to SIFT method, we have implemented SURF method on openCV. SURF stands for speeded up robust Features. This provides a faster way to obtain the feature points. A general comparison of SIFT vs SURF is done in the discussion below.

2.2.2 Image Matching

In the above problem we were able to appreciate the importance of good features and extraction of SIFT and SURF features. The underlying reason why we extract features from an image is to find relevant matches among images. In this problem we attempt to perform image matching. In addition to the steps mentioned above, we perform the following:

- **Keypoint matching:** After extracting SIFT features from two images, perform nearest neighbor search to locate matching keypoints. It may so happen that there would be two neighbors that are close to the test point. In this case we take the ratio of the first best and second best distance and threshold it. If the ration is above 0.8, we reject the matching. This way we reduce a lot of irrelevant matches.

After SIFT or SURF feature extraction using OpenCV, we have used Brute Force matcher and FLANN based matchers. The result is displayed in section 2.3.

2.2.3 Bag of Words

In this problem, we attempt to perform classification on an unknown test image based on the previously trained image. The implementation was done using OpenCV library. The following steps were taken to execute this:

- **Step 1:** The training and testing images were read and a bag of words trainer was initialized with 8 clusters. From each training image we have extracted SIFT key points and descriptors and added it to the bag of words trainer.
- **Step 2:** They features were then clustered with the help of k means. This step builds the vocabulary of visual words that are used to describe the image.
- **Step 3:** After getting the vocabulary, we draw out the histogram of each input image. Since we have 8 clusters, we can now represent the histogram as eight bins. Each train image has its own histogram vector which is of length 8.

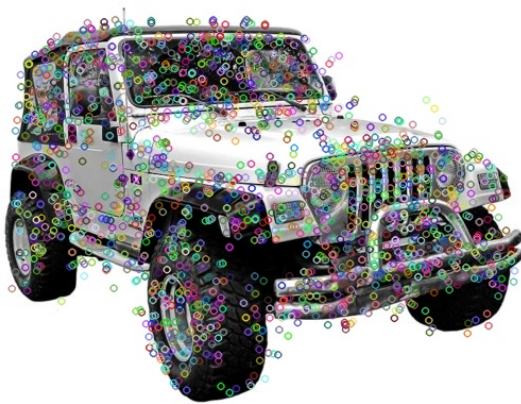
- **Step 4:** Now we take the test image and follow step 1 to 3.
- **Step 5:** We now compare the histogram of the test image with the training images. The histogram with the closest values will be the one that the test image belongs to. To compare histograms, we have computed chi square distance which is give by:

$$distance = \frac{1}{2} * \sum_{i=0}^{n-1} \frac{(x_i - y_i)^2}{x_i + y_i} \text{ -----(Equation 2.1)}$$

where x_i and y_i are histogram values.

Based on the above steps, we have obtained pretty accurate results discussed in the next sections.

2.3 Experiment Result



(a) SURF Features Points



(a) SIFT Features Points

Figure 2.1 : Feature points for jeep image



(a) SURF Features Points



(a) SIFT Features Points

Figure 2.2: Feature points for bus image



(a) SURF Features Points



(a) SIFT Features Points

Figure 2.3: Feature points for rav4_1 image



(a) SURF Features Points



(a) SIFT Features Points

Figure 2.4: Feature points for rav4_2 image

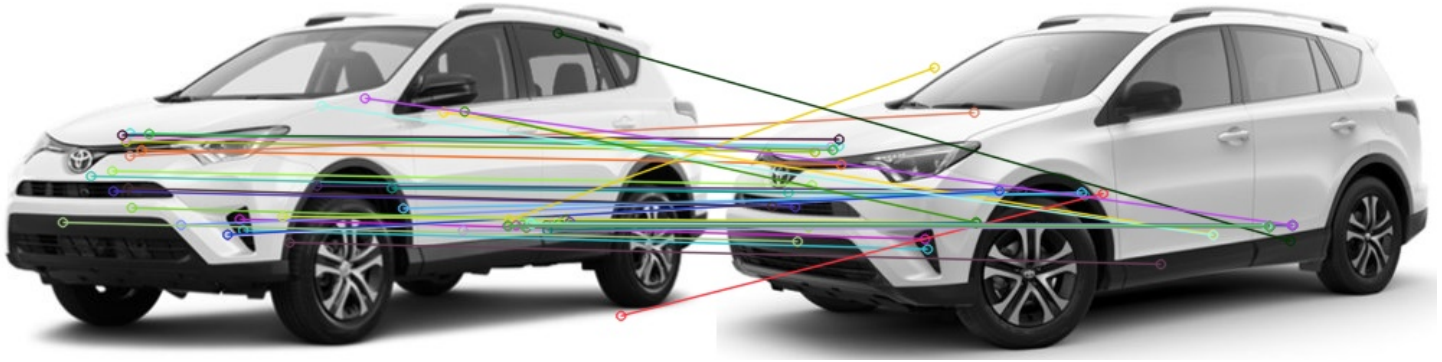


Figure 2.5: Feature matching using SIFT

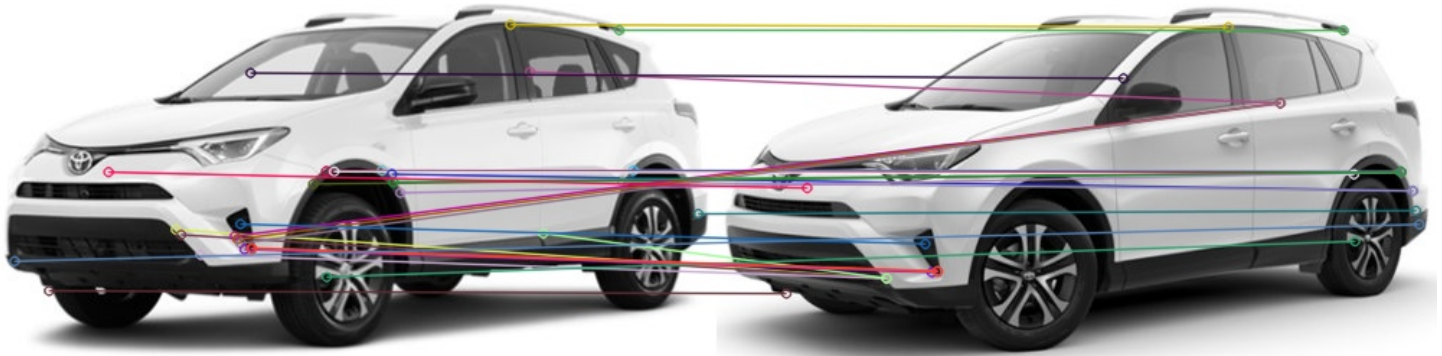


Figure 2.6: Feature matching using SURF



Figure 2.7: Feature matching using SIFT

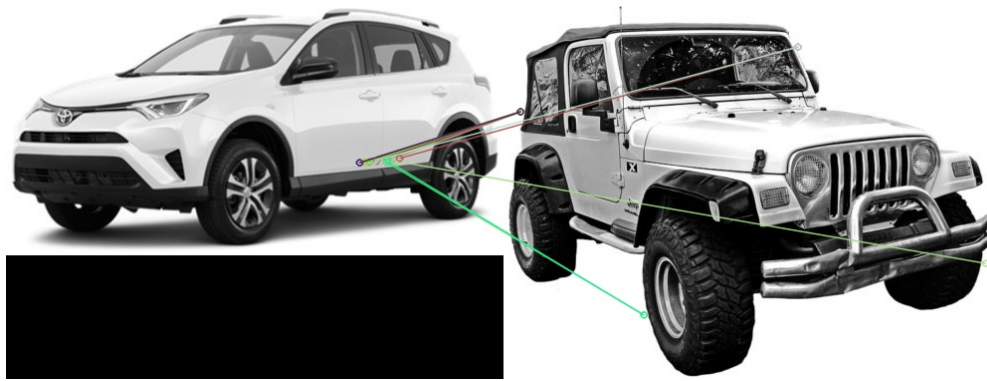


Figure 2.8: Feature matching using SURF

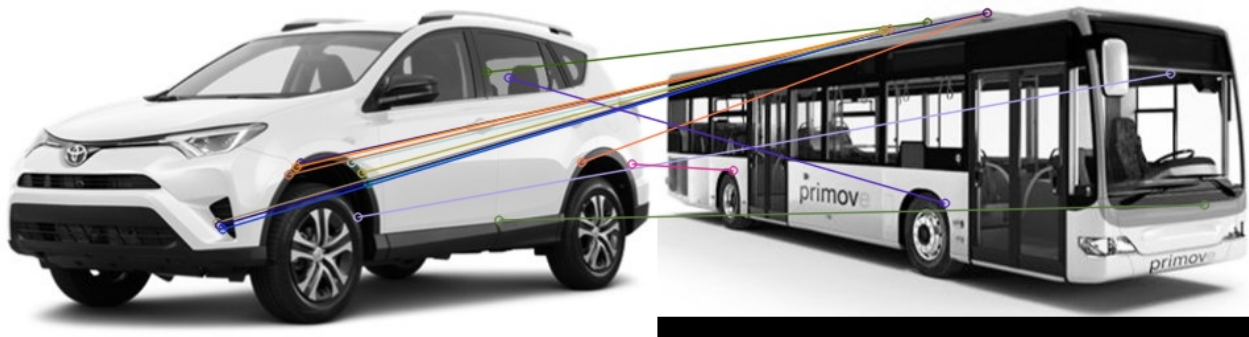


Figure 2.9: Feature matching using SIFT

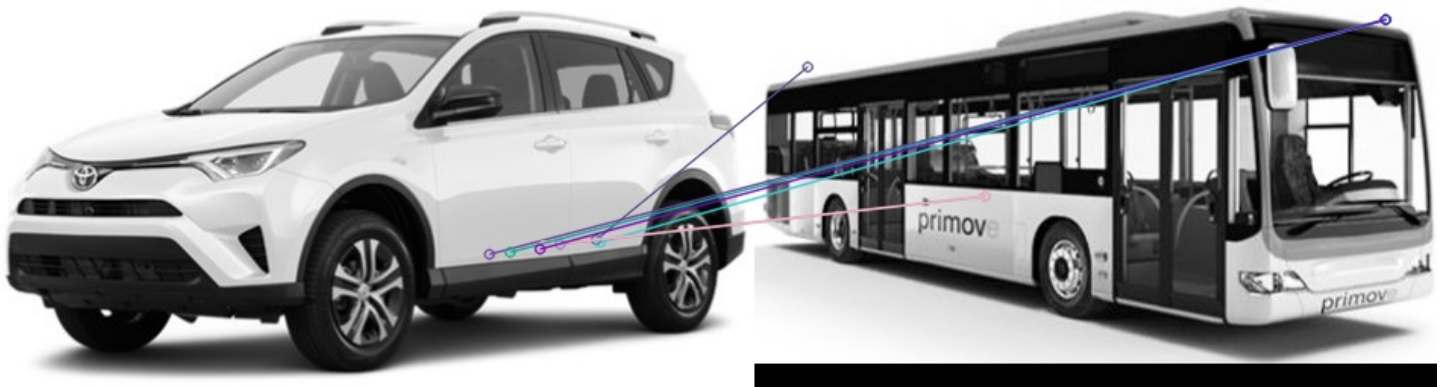


Figure 2.10: Feature matching using SURF

| Test Image | Rav4_1 (Chi Square Distance) | Jeep(Chi Square Distance) | Bus(Chi Square Distance) |
|---------------|------------------------------|---------------------------|--------------------------|
| Rav4_2 | 0.0072 | 0.0141 | 0.04808 |

Table 2.1: Chi square value for the test image compared with the training images

2.4 Discussion

SIFT and SURF features were extracted from each of the input image and are displayed in figures 2.1 to 2.4. We can clearly observe that SURF has more interest points than SIFT. To understand this disparity, we need to study the differences between them:

- The scale space for SIFT is obtained by DoG convolved with different Gaussian filter parameters where as in SURF, different sized box filters are convolved with integral image to form the scale space.
- In the Keypoint detection stage, SIFT utilizes non maximum suppression and eliminates outliers with the use of Hessian matrix. On the other hand, SURF utilizes Hessian matrix to determine the points of interest.
- In SIFT the orientation is calculated by taking the weighted orientations of the surrounding pixels. On the other hand, SURF uses a sliding orientation window of size $\pi/3$ to detect the dominant orientation of the Gaussian weighted Haar wavelet responses at each point of interest.
- The keypoint descriptors for SIFT is done by taking the histogram of orientation in a 16×16 neighborhood whereas in SURF a quadratic grid with 4×4 square sub regions are considered and the wavelet responses are computed relative to the orientation of grid.

In paper [3], we can see how the SIFT method compares with the SURF feature extraction. SURF computes meaningful features with as much as a third of the time taken by SIFT feature extractor. It also provides as good results as SIFT would. This time difference comes due to the optimization used in SURF. Surf builds its scale space and interest point detector using integral images which is much faster than SIFT way of generating its scale space.

In figures 2.5 to 2.10 we can see the matching performance among different input images. A few modifications were made to SIFT before matching. Since the Brute Force matcher computes a match for every point, it was necessary to filter out noisy matches. To do this, we have calculated the maximum and minimum values of distance between each match. Later we only considered the matches that were less than twice the minimum value. Hence this way we can filter out matches that were not relevant. Again, this type of match filtering may give errors when an object is being compared with different objects. This is because the Brute force method tries to match irrelevant points and in doing so it sets up the minimum distance value. This can be solved by using a FLANN based matcher. It uses the nearest neighbor's technique.

From table 2.1 we can observe that the rav4_2 image have been correctly classified to rav4_1. The chi square statistic for each pair of test and train images was performed and the lowest value of chi square was selected. The lower the value of chi square distance the better it matches with the training image.

Problem 3

3.1 Abstract and Motivation

Image edge can be defined as the points where there is a sudden change in grayscale values. Edge detection is a critical problem in most of the computer vision applications. It is used in image processing pipelines to achieve tasks such as object detection, scene detection, object tracking and so on. Edge detection is a very challenging problem because there is a whole new level of interaction between pixels in an image like brightness variations, colors, texture, continuity, symmetry etc. that needs to be considered before calling a particular pixel as an edge point. In this problem we will apply Canny edge detector and Structure Forests for fast edge detections with a view point of segmenting a particular object from the image.

Canny edge detection is basically an extension to Sobel edge detection. This algorithm adds a layer of intelligence over the Sobel edge detector as it performs double hysteresis thresholding to remove weak unconnected edges from the image. Additionally, Canny edge detection doesn't require any training samples and it is really fast. This method can be used in real time applications and it can be parallelizable.

Structure Forests method for edge detection takes the advantage of the structure present in the local image patch to train a random forest classifier. This work was done with the motivation to apply effective edge detection in real time. In this problem, we will explore the working and quality of result of this method.

3.2 Approach and Procedures

3.2.1 Canny Edge Detector

Canny edge detection was performed using Opencv's Canny operator. First the image was smoothened by a Gaussian filter of size 3x3 to remove unwanted noise. Then a threshold was calculated using the following formulas:

$$\begin{aligned} \text{lowerThreshold} &= \max(0, (1 - \sigma) * \text{median}) \text{ -----(Equation 3.1)} \\ \text{upperThreshold} &= \min(255, (1 + \sigma) * \text{median}) \text{ -----(Equation 3.2)} \end{aligned}$$

The value of sigma controls the thresholding difference between the lower and upper threshold. A higher value of sigma indicates that the threshold is wide whereas a lower values of sigma indicates tighter threshold. As a rule of thumb we select sigma value of 0.33 as it generally gives us good results.

In addition to the above thresholding technique, we have implemented the threshold based on Otsu's threshold.^[4] The results of these methods are compared in section 3.3

3.2.2 Structured Edge detection

Structured forest is a state of the art algorithm that is used to compute the edges on a given image. This edge detector considers a patch around the pixel and determines the likelihood of the pixel being an edge point or not. Edge patches are classified into sketch tokens using random forest classifiers. Generally, a set of sketch tokens represent a variety of local edge structures such as straight lines, parallel lines, T junctions, Y junctions, curves and so on. Then, structured learning is utilized to address problems associated with training a mapping function where the input or output space may be complex.

Initially the random forest is trained by taking a patch from the input and ground truth images and forming a structured output of whether a certain combination should be classified as an edge or not. After the random forest is trained by an appreciable amount of samples, it can be tested. In the testing phase, only the test image is given for

segmentation and due to the speed of random forests, it computes the segmented output. A simple flowchart is shown in figure 3.1.

A random forest is an ensemble approach. They run on the divide and conquer paradigm that helps to improve their performance. The main idea of ensemble approach is to put a set of weak learners to form a strong learner. The basic element of a random forest is a decision tree. The decision trees are weak learners that are put together to form a random forest.

A decision tree classifies an input that belongs to space x as an output that belongs to a space y . 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. Each node in the decision tree has a binary split function with some parameters. This split function decides whether the test sample should progress towards the right child node or the left child node. Often, the split function is complex.

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.

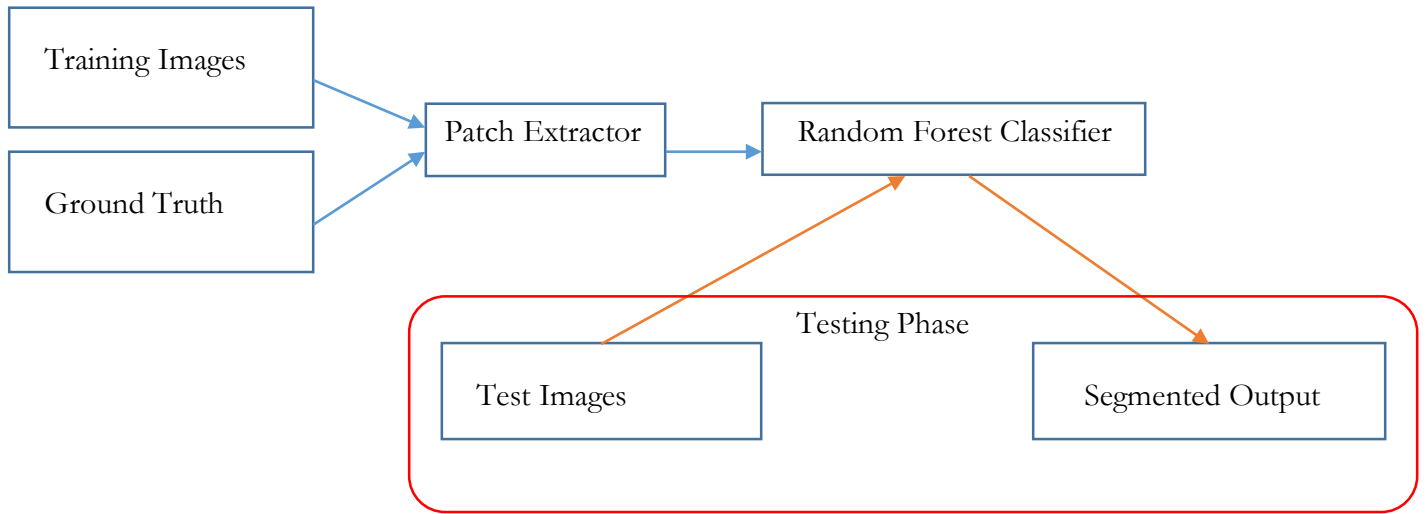


Figure 3.1: Structure Edge detection algorithm

3.2.3 Performance evaluation

To evaluate the performance of different edge detectors, we have used the online sources code. We first obtain the binary image from the edge detectors. For this problem we have taken the output from SE and Canny. Since the output of these are not binary, we convert this to a binary image by taking a threshold.

For Canny edge detector, we took a threshold of 0.5 for all images whereas different thresholds were tried on with the SE detector. After obtaining the binary image, we passed this image to an evaluator function with different ground truths.

The function returned the precision and recall of the edge detected image. We then calculate the F score from precision and recall:

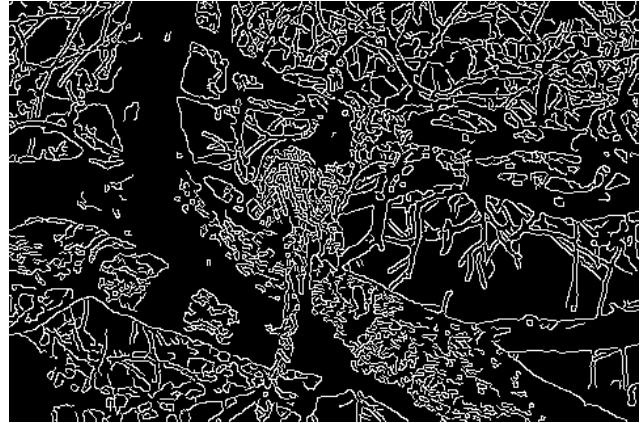
$$F_{score} = \frac{2*Precision*Recall}{Precision+Recall} \text{ -----(Equation 3.1)}$$

The calculated performance metrics are tabulated in the next section.

3.3 Experiment Results

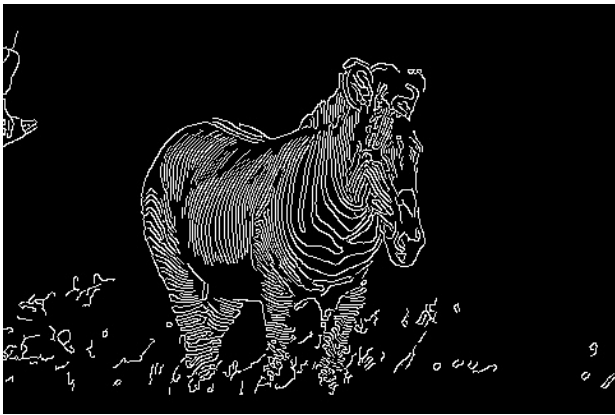


(a) Zebra (168,85)



(a) Jaguar (123,62)

Figure 3.2: Canny Edge detection with auto thresholding



(a) Zebra (114,57)



(a) Jaguar (106,53)

Figure 3.3: Canny Edge detection with Otsu's threshold

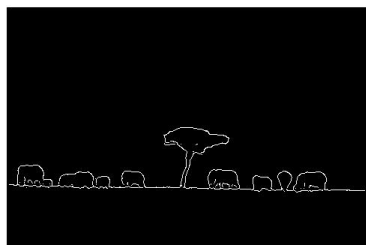


(a) Zebra (150,50)

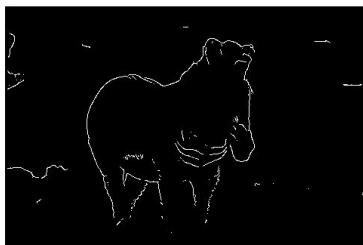


(a) Jaguar (150,50)

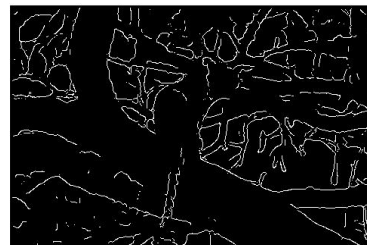
Figure 3.4: Canny Edge detection



(a) Elephant (Thresh = 0.25)



(b) Zebra (Thresh = 0.2)



(c) Jaguar (Thresh = 0.15)

Figure 3.5: Structure Edge detection

| Image | Recall | Precision | F1 Score |
|-----------|--------|-----------|----------|
| Zebra_gt1 | 0.31 | 0.88 | 0.46 |
| Zebra_gt2 | 0.19 | 0.91 | 0.32 |
| Zebra_gt3 | 0.31 | 0.86 | 0.46 |
| Zebra_gt4 | 0.38 | 0.87 | 0.53 |
| Zebra_gt5 | 0.31 | 0.89 | 0.46 |
| Mean: | 0.3 | 0.88 | 0.44 |

Table 3.1: Mean Precision and Recall for Zebra using SE

| Image | Recall | Precision | F1 Score |
|------------|--------|-----------|----------|
| Jaguar_gt1 | 0.36 | 0.74 | 0.48 |
| Jaguar_gt2 | 0.45 | 0.37 | 0.41 |
| Jaguar_gt3 | 0.21 | 0.98 | 0.34 |
| Jaguar_gt4 | 0.26 | 0.88 | 0.40 |
| Jaguar_gt5 | 0.51 | 0.55 | 0.53 |
| Jaguar_gt6 | 0.4 | 0.38 | 0.39 |
| Mean: | 0.36 | 0.65 | 0.42 |

Table 3.2: Mean Precision and Recall for Jaguar using SE

| Image | Recall | Precision | F1 Score |
|-----------|--------|-----------|----------|
| Zebra_gt1 | 0.5 | 0.12 | 0.19 |
| Zebra_gt2 | 0.36 | 0.14 | 0.2 |
| Zebra_gt3 | 0.57 | 0.13 | 0.21 |
| Zebra_gt4 | 0.64 | 0.12 | 0.2 |
| Zebra_gt5 | 0.56 | 0.13 | 0.21 |
| Mean: | 0.52 | 0.12 | 0.20 |

Table 3.3: Mean Precision and Recall for Zebra using Canny

| Image | Recall | Precision | F1 Score |
|------------|--------|-----------|----------|
| Jaguar_gt1 | 0.98 | 0.19 | 0.32 |
| Jaguar_gt2 | 0.98 | 0.07 | 0.14 |
| Jaguar_gt3 | 0.98 | 0.07 | 0.14 |
| Jaguar_gt4 | 0.98 | 0.07 | 0.14 |
| Jaguar_gt5 | 0.95 | 0.42 | 0.59 |
| Jaguar_gt6 | 0.93 | 0.08 | 0.15 |
| Mean: | 0.96 | 0.15 | 0.25 |

Table 3.4: Mean Precision and Recall for Jaguar using Canny

3.4 Discussion

In figures 3.1, 3.2 and 3.3, we can see the resulting edge maps from canny by applying different thresholding methods. The automatic thresholding does a very good job at detecting the edges of the object. If we observe Fig 3.1 closely, we can find the object in the zebra image where as we cannot find the object in the Jaguar image. This is because the jaguar is camouflaged with the background. It is very difficult to obtain a good edge map for the Jaguar image. Since the threshold values relies on the image statistics, it tends to perform well.

Additionally, the edge map generated from Otsu's threshold in figure 3.2 is similar to that of automatic thresholding. Again the Otsu's global threshold is derived from the minimizing the intra-class variance and maximizing the inter-class variance. This purely depends on the distribution of image gray levels.

It is important to note that Canny edge detector cannot differentiate if the edge is from the object or texture of an object. Changing the parameters of the Canny detector, one cannot avoid the edge detected inside the object. Canny edge detector doesn't posses this layer of intelligence that prevents it from detecting uninformative edges inside the object.

The gradient magnitude with values above the upper threshold are considered as strong edges. These edges are detected by the Canny edge detector. The gradient magnitudes that are less than the lower thresholds are rejected where as the magnitudes between the lower and upper thresholds are taken only if they come in contact with a strong edge. So, by setting appropriate thresholds, we can filter out noisy edges and retain weak edges that are assumed to form a closed surface.

Structured Edge performs really when in edge detection tasks. It has an added layer of intelligence that enables it to disregard unimportant edges even though it has a very high gradient change. This method can smartly detect textures and disregard its information. Due to its training, it performs much better than the Canny edge detector. This is because Canny edge detector is not intelligent to detect textures. Table 3.1 to 3.4 also displays the resulting evaluation metrics applied on various ground truths. The SE outperforms the Canny edge detector.

Tables 3.1 to 3.4 shows the resulting recall, precision and recall. We can clearly see the difference in the performance of Canny Edge detector and Structure Edge detector. As discussed before, Canny doesn't have the ability to discard noisy edges introduced due to texture.

Intuitively, it is easier to get a higher F score on Zebra image. Visually, this is due to the fact that this image is way simpler to segment than the Jaguar image. Since the jaguar is camouflaged into its background, it is difficult to segment it with high precision. This will happen sometime in the future when computers can segment images based on semantics.

The F measure or the F1 score is a measure of test accuracy in statistics. It value is directly effected by both precision and recall. If any one of these metrics is low, then it results in a bad F1 score. Additionally, it is interesting to note that there exists an inverse relation between precision and recall. If recall increases, then precision gets low and vice versa. To maximize the product of the two which follows an inverse relation, we need to have them to be equal.

References :

- [1] Ali Ghodsi, “Dimensionality Reduction A short tutorial”, Department of Stats, University of Waterloo, Canada, 2006.
- [2] David G. Lowe. “Distinctive Image features from Scale Invariant Key points”, University of British Columbia, Vancouver, Canada.
- [3] P M Panchal, S R Panchal, S K Shah, “A comparison of SIFT and SURF”, IJIRCCE, Vol 1, Issue 2, April 2013.
- [4] Mei Fang, GuangXue Yue, QingCang Yu, “The study of an application of Otsu’s method in Canny Operator”, ISIP’09, China.