EE 569 Homework #3
Report

Akash Mukesh Joshi
USC ID : 4703642421

Submission Date : 6th November 2016

Homework #3

Problem 1: Texture Analysis and Segmentation

(a) Texture Classification

I. Abstract and Motivation

Earlier in the 1990s image content retrieval was done on basis of image color, shapes and texture. Image texture analysis was used to analyze images of varying texture to understand texture segmentation within the same image. This kind of image data content analysis is called texture classification. Texture classification is a method of applying different kind of masking filters to images to understand the way a texture is embedded within an image. Image texture gives us spatial information of colors in an image. Image texture was later being used for object classification as different objects within an image have different texture. Objects within an image can get segmented using different filtering methods. These filtering methods can be approached using edge detection, co-occurrence matrix, laws texture energy measures and or autocorrelation and power spectrum.

Texture classification is a dual logic method of image segmentation and machine learning for classification. Machine learning is learning algorithm that analyzes a data set and then uses this data to train a model. This model is then used on a set of unknown data set that can be then classified into a number of different classes or clusters such that we can obtain image segmentation. Machine learning plays a important role in texture classification where the image captured through a camera has various information in it and machine learning can be used to understand different aspects of the image that is captured. There are various different kinds of machine learning algorithms that are used in texture classification such as k means clustering, support vector machines, k nearest neighbors, mixture model classification, etc. Some of these image filtering and classification techniques are discussed in the problem solving assignments below.

II. Approach and Procedures

The approach for this problem is done in both c++ and MATLAB. The image filtering algorithm is run in c++ and the output is exported to a MATLAB environment where the dataset is applied to a classification algorithm to perform texture classification.

To go ahead with algorithm we need to understand the concept of law filters. The law filters are a set filters made by K. I. Laws and have been used for many diverse application. The first step is to define these law filters to generate a set of 5 by 5 law filters which will then be applied as convolution kernel to the digital filter to calculate image pixel energy values.

In this problem we have been given 12 images. These images are divided into 4 types of textures rock, grass, weave and sand. This means that each texture has 3 images such that the classification model can train on it.

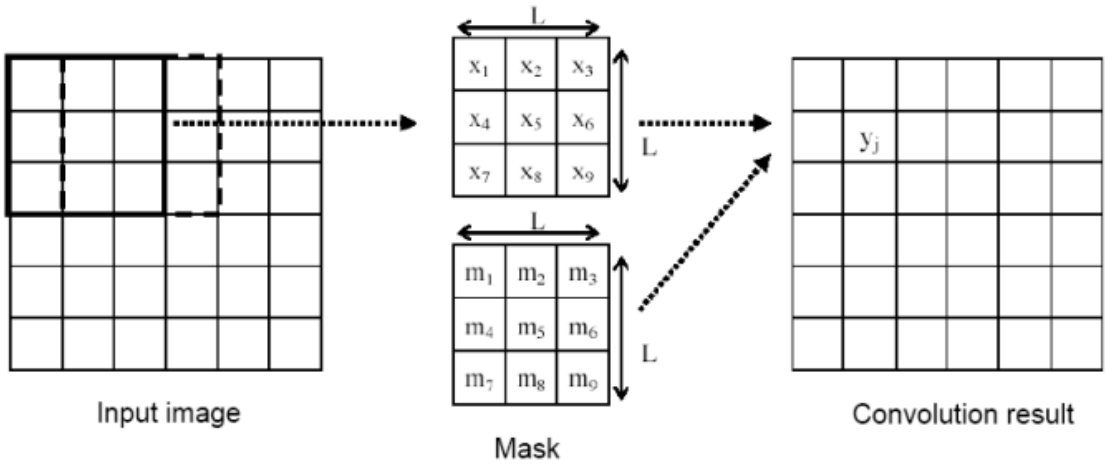
- The first step is create a library function to generate the 25 law filters using the 5 filters provided in the problem statement. These 25 law filters will be used for feature extraction from each pixel leading to a 25-D feature vector for each of the 12 images.
- This is done by initializing the 5 law filters and performing a cross multiplication between each element of the law filter to generate a 5 by 5 matrix as shown below:

- • **1D Masks are "multiplied" to construct 2D masks:**
  mask E5L5 is the "product" of E5 and L5 –

$$\underset{\text{E5}}{\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix}} \times \underset{\text{L5}}{\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}} = \underset{\text{E5L5}}{\begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}}$$

is include this library file – ***law_filter.h*** in every code where we need to use the law filters for texture analysis.

- Now we input the 12 texture images into a 12 – dimensional image matrix. Storing it in a singular matrix allow easy access to all the images using a single for loop.
- Now we subtract the global mean from the image. This is done using a 5 by 5 mask. The mask is a all ones 5 by 5 matrix with each value divided by 25.
- This mask is applied to every pixel in the image using a N+2 padding method. The mean value is then subtracted from the center pixel value to form a global zero mean image which is stored in an variable called ***IMean***.
- The mean was calculated by making a mean function which does all the calculations.
- The next step in the algorithm is to now apply all the law filters to each image. This means that we will have 25 energy values for a image forming a 25-D feature vector. There are 12 images thus we will form 25 by 12 matrix to store the energy data.
- To perform this we have made a energy function. Here the ***IMean*** image data is sent pixel wise and the it is multiplied with a law filter.
- The value is then added to a energy variable and this is done for all the pixels and the energy values are added to the same variable. This means that the energy value for a particular image and particular filter gives a single energy value. This is shown in the image below.

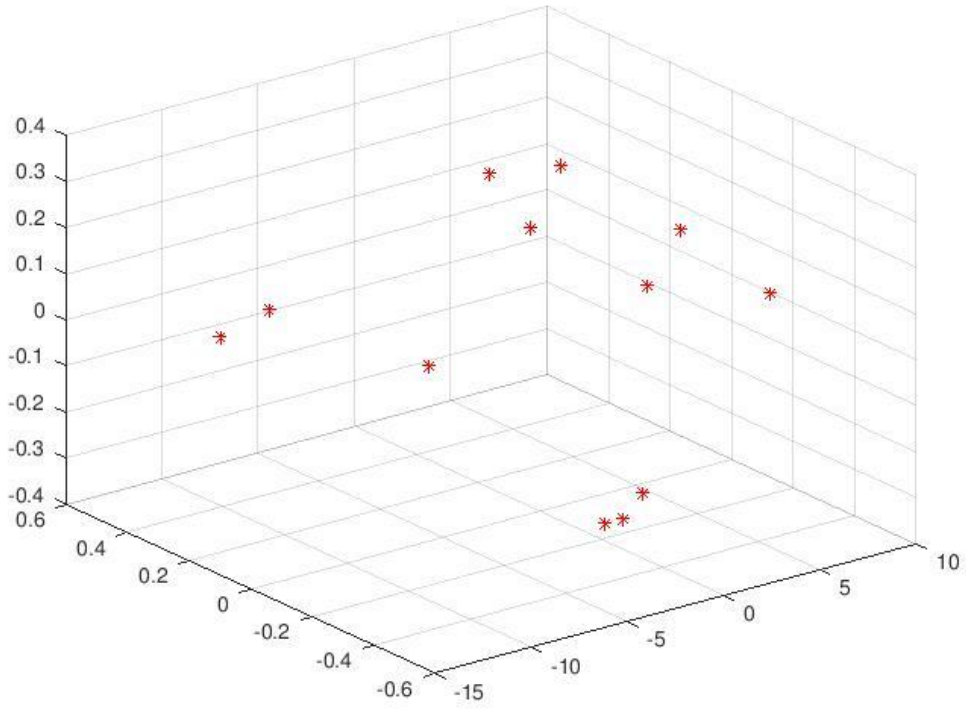


- The convolution result

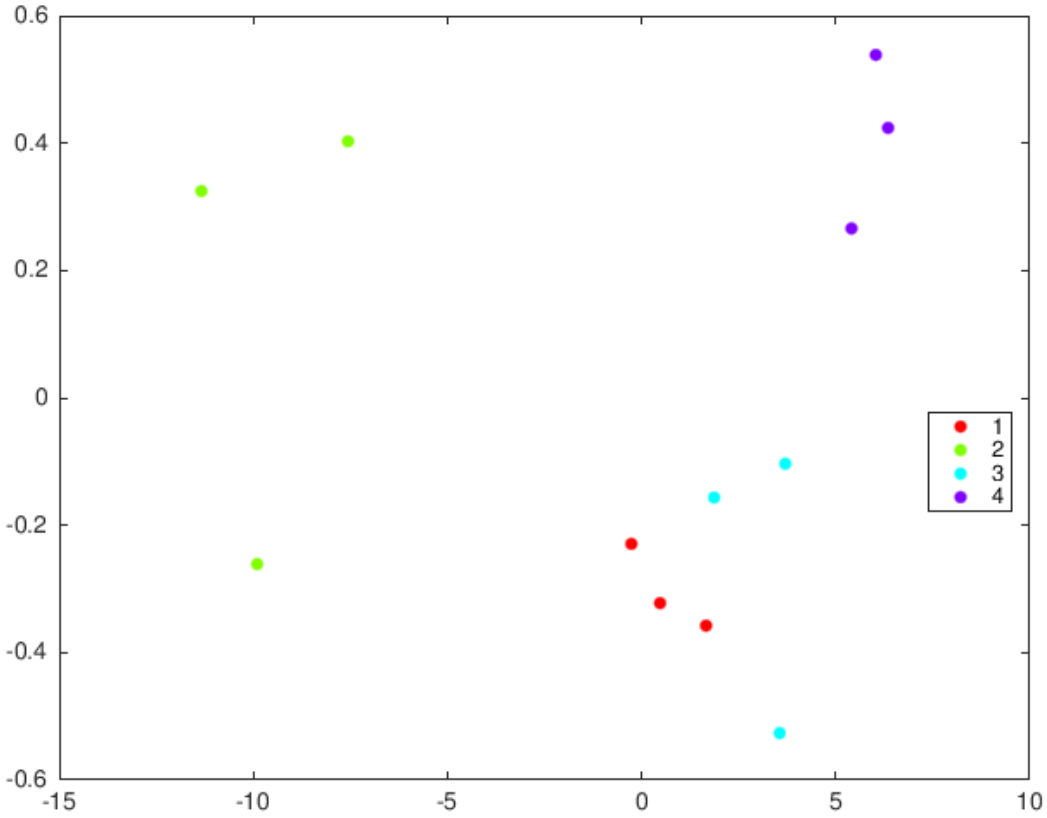
is the energy value we were talking about and it is calculated as shown below:

Mask convolution. $Pixels = N = L \times L$, $y_i = \Sigma_{i=1}^{N} x_i \times m_i$.

- The next is to normalize these energy matrix of 25 by 12. This is done using the 25 values and finding the minimum and maximum value in the array and normalizing the values between 0 and 1.
- Once the feature vectors are normalized they are saved as a .csv file to be accessed in the MATLAB environment to do further processing of PCA and KMeans clustering texture classification.
- Now we need to move over to MATLAB environment to and import the .csv file using the ***csvread*** function.
- The next step is to apply to the Principal Component Analysis on the 25-D feature vector. The PCA will transform the 25 by 12 input feature vector into a 12 by 12 feature vector. No we need to find the most important features.
- The PCA function outputs a score 12 by 11 matrix. This contains which of the features will be relevant in texture classification. This done by plotting a bar graph.
- Looking at the bar graph below we can see the first 2 principal components are the most important features. We have been advised to use the first 3 principal components.
- The first 3 features are then stored from the ***score*** matrix in a variable called ***fv_pca*** which were computed from a 25-D feature vector called ***fv_norm.***
- The next step is to apply k-means algorithm to both the feature vector and see the clustering output.
- Now we plot the 3-D PCA feature vector using the scatter plot function ***scatter3*** in MATLAB. The scatter plot in shown in the image below:



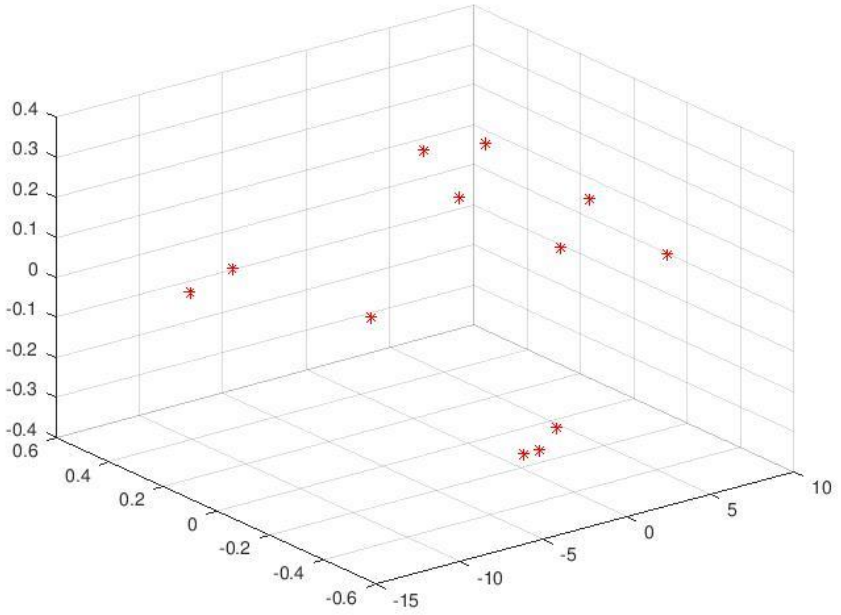
- From the above scatter plot we can see the 4 clusters that are formed using ***fv_pca.***
- The next step is to call the ***kmeans*** MATLAB function and pass the PCA feature vector and also the number of clusters required. Additional parameters can be provided if required.
- The ***kmeans*** function outputs the cluster assignment for each image and 12 such cluster assignment is done from 1 to 4.
- The four clusters are formed and labeled as shown below in the result of kmeans clustering.



- The next is to apply kmeans clustering to the entire 25-D feature vector. This is done is same fashion as explained above but now we send the ***fv_norm*** feature vector to the ***kmeans*** function.
- The output of both the kmeans clustering is display on the command window in MATLAB.

III. Results

The results of the kmeans clustering on the 25D feature vector and 3D PCA feature vector are shown in the images below.



3D Feature Vector in Feature Space

```
clusters =

     3     3
     4     1
     1     3
     3     3
     3     3
     1     3
     2     2
     2     4
     4     1
     2     5
     1     3
     4     1
```

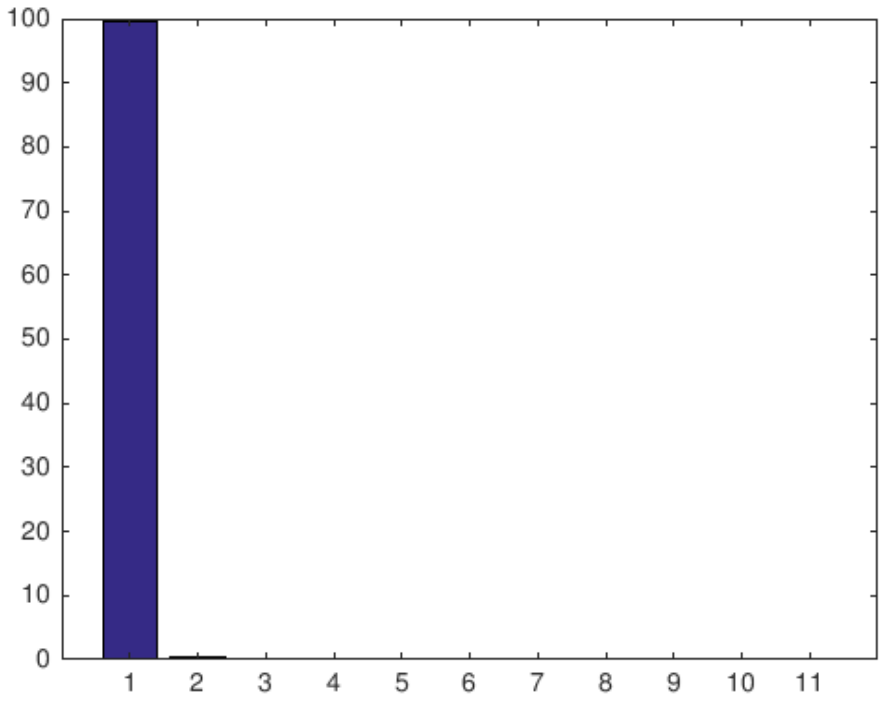
K-Means Clustering Output from MATLAB Command Window. Column 1 is output of 3D PCA Feature Vector and Column 2 is output of 25D Feature Vector

IV. Discussion

From the above images and the algorithm discussed we can see the output generated in the MATLAB environment. From the above result it is clear that the PCA algorithm has worked much better than the normal 25D feature vector. The clustering of images could not happen properly but when the feature vector was transformed into a new feature space it helped in forming 4 clustering sectors as we can see from the above 3D scatter plot.

Then a primary initial centroid were chosen and the data was then given to kmeans clustering function. This helped in better formation of the clusters in comparison to the 25D feature vector kmeans clustering.

The PCA feature vector worked well because the PCA algorithm worked well to form a new feature vector in new space where are first 3 components of the feature vector carried the most information that is it had the highest principal components as shown below:



From the above bar graph it is clear that the first 2 principal components provided the most information possible and this clear from the clustering output in the images above.

(b) Texture Segmentation

Problem 2: Salient Point Descriptors and Image Matching

(a) Extraction and Description of Salient Points

I. Abstract and Motivation

Humans are able to detect a single object when viewed from different angles. This is possible only because we are able to identify certain features of an object and helping us understand how the object would look from different view points. The same can be applied to recognizing facial features where a person remembers certain facial features of a person and that is how one is able to identify one person from another.

Looking at images now we can say that the same concept applies of feature extraction and image matching. These properties of an image is what helps us in image recognition. Object features or salient points in an image are the important key extracting points that provide us with descriptive information of different part of an image. Now this is done by looking at specific patterns or specific features that are unique and can be easily understood. Now once we have food a set of features in an image we need to now find the same set of features in another image such that it helps us in deciding if an image is similar to an another image.

This kind of salient point descriptors and image matching can be done using two methods called as SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features). These methods of feature descriptor extraction will be discussed in the section below.

II. Approach and Procedures

To perform SIFT and SURF analysis on the images provided in the problem we need to use OpenCV libraries since they have precoded functions for both SIFT and SURF analysis. OpenCV is an open source computer vision library which we can include as header files. The main goal of this problem is to code a SIFT and SURF image descriptor search program using inbuilt OpenCV libraries.

- The first step is to read the two images provided in the problem statement using the ***imread*** function of OpenCV.
- The next step is to create two pointers namely ***detector_surf*** and ***detector_sift.***
- Now we create two matrices which will store the output of the SIFT and SURF analysis.
- The next step is to use a inbuilt OpenCV function called ***drawKeypoints*** which finds the salient features and displays in on the image as an overlay.
- These values are stored in the two matrices for both type of analysis.
- The final stage is to convert these matrices to images and store them using ***imwrite*** function.

III. Results

We are provided with two images. The SIFT and SURF outputs for both the images are shown below:

bus.jpg

bus_sift.jpg

bus_surf.jpg

SIFT and SURF analysis output of bus.jpg image

jeep.jpg

jeep_sift.jpg

jeep_surf.jpg

SIFT and SURF analysis output of bus.jpg image

IV. Discussion

- The two SIFT and SURF outputs are shown above and it can clearly seen that both algorithms give different keypoint descriptors
- The SIFT works using a 4 step algorithm which involves scale-space extrema detection, keypoint localization, orientation assignment and keypoint descriptors.
- In comparison to SURF the SIFT works much slower and hence SURF has become more and more popular.
- The SURF algorithm uses box filtering technique which makes it much faster that SIFT algorithm. But the SIFT algorithm does approximation of Laplacian of Gaussian (LOG) using difference of gaussian method in comparison to box filtering.
- The SURF algorithm uses wavelets responses in the vertical and horizontal direction for orientation assignment as compared to bin covering logic for SIFT.
- In short the SURF adds a lot of features to increase the speed of operation. It has been seen that a SURF algorithm is typically thrice as fast as the SIFT algorithm.

(b) Image Matching

I. Abstract and Motivation

Now that we have discussed how keypoint descriptors work now we can look at how we can use these descriptors to perform image matching. Image matching is a technique where two similar images can be matched with each other based on the features that were extracted using one of the above mentioned methods of keypoint descriptors.

We are going to use a Brute Force algorithm to preform image matching between two images. This algorithm works by taking descriptors form a set and matching it with all the other features in the second set formed by the second image using distance calculation.

II. Approach and Procedures

- The first step is to read two user provided images using the ***imread*** function in OpenCV library.
- The next step is to create two pointers namely ***detector_surf*** and ***detector_sift.***
- Now we create two matrices which will store the output of the SIFT and SURF analysis and they will be stored in individual matrices called ***descriptors_1*** and ***descriptors_2***.
- The next step is to use a inbuilt OpenCV function called ***drawKeypoints*** which finds the salient features and displays in on the image as an overlay.
- These values are stored in the two matrices for both type of analysis.
- The next is to create a Brute Force Matcher and use the ***matcher.match*** OpenCV function to store matches within a vector called ***matches***.
- A thresholding algorithm was used to get good matches between the two images so that we don't show all the unnecessary matches that have low descriptor values.
- A distance of thrice the minimum distance and twice the minimum distance algorithm were used to get good matches for SIFT and SURF algorithms respectively.
- Now we use a ***drawMatches*** OpenCV function and send the ***keypoints***, the two input images and the matches which then results in an output image which stores the output in a matrix called ***img_matches***.
- This matrix is saved as a .jpg file within the code itself.

III. Results

The image matching is done 3 pairs of images. The 3 pairs are ***rav4_1*** & ***rav4_2*** , ***rav4_1*** & ***bus*** and, ***rav4_1*** & ***jeep***. The output of image matching of these pairs of images is shown in the images below:

rav4_2.jpg

rav4_1 .jpg

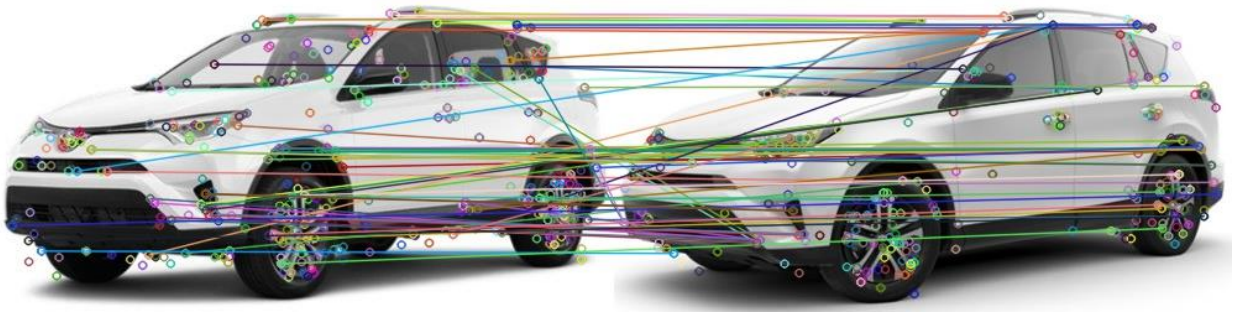
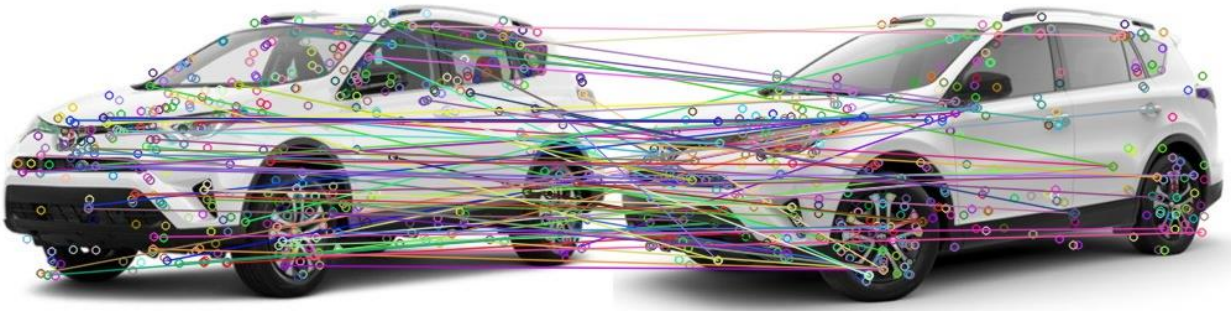


Image Matching using SIFT algorithm

Image Matching using SURF algorithm



bus.jpg

rav4_1.jpg

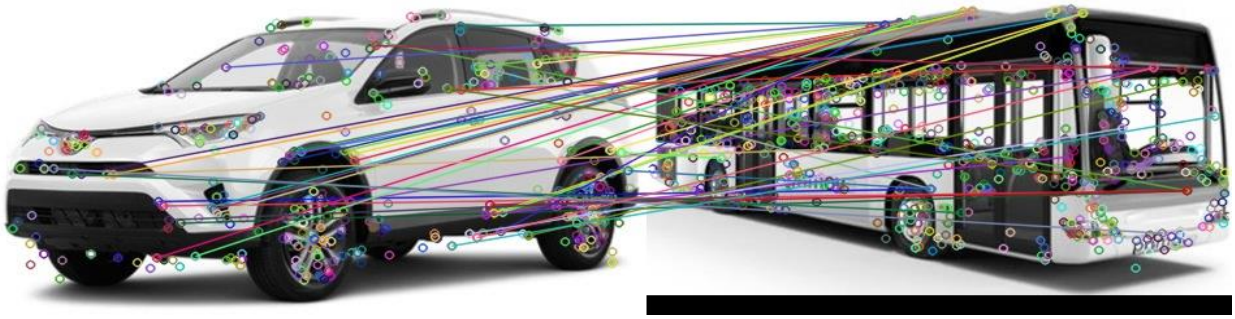
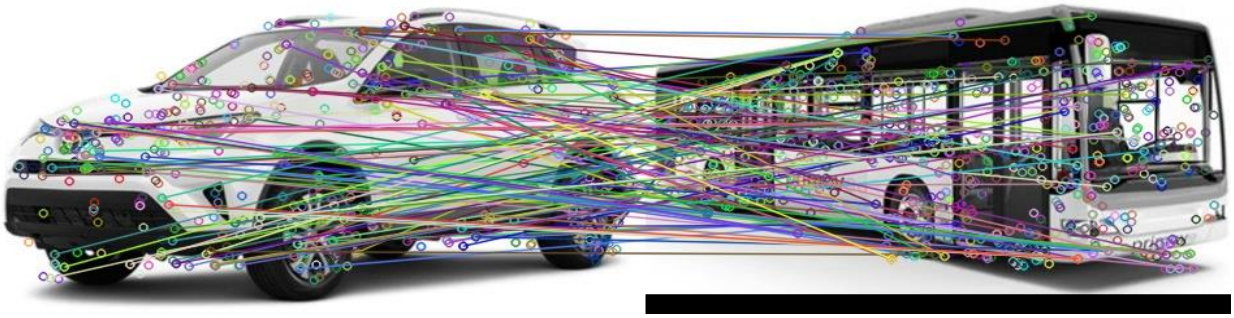


Image Matching using SIFT algorithm

Image Matching using SURF algorithm



jeep.jpg

rav4_1 .jpg

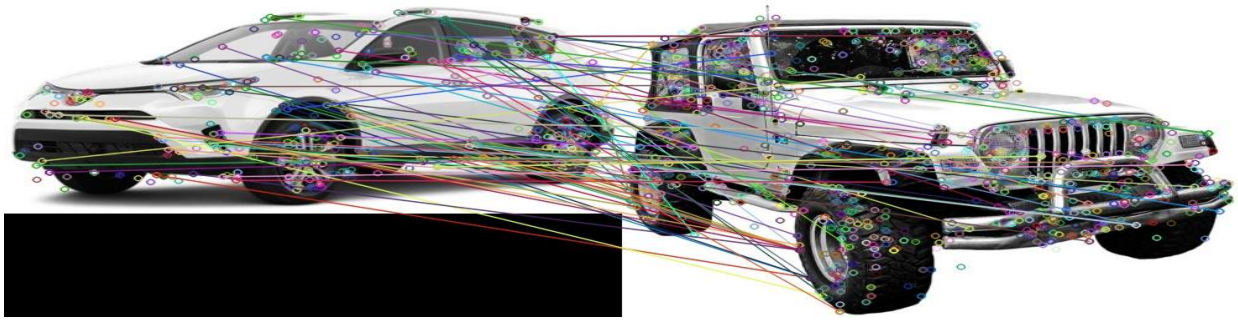
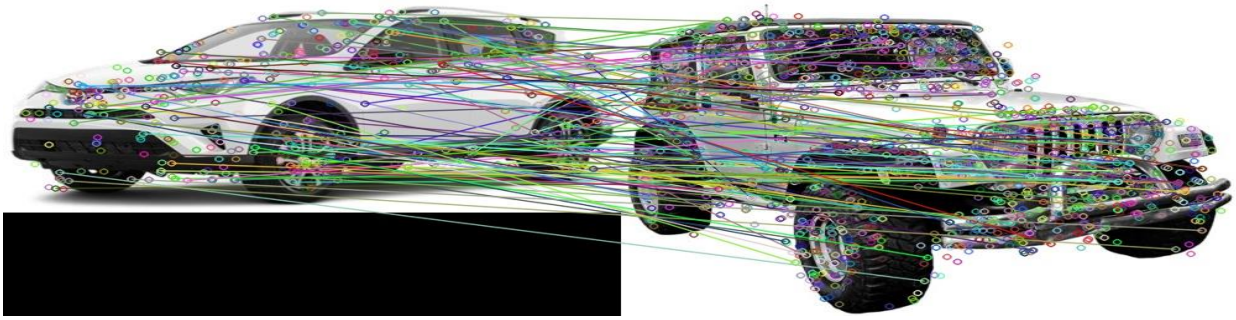


Image Matching using SIFT algorithm

Image Matching using SURF algorithm



IV. Discussion

- From the above result we can see that we have run a good matches algorithm where we use thresholding algorithm.
- The image matching works on all 3 pairs of images. This shows that the SIFT and SURF algorithms run on image segmentation.
- This means that small sections of an image can match with small sections of another image even if the image are not similar.
- Now we know that image matching would work on dissimilar images too even if the images are dissimilar is shape and contour.
- This shows that image matching checks for similarity in edges, shapes, contours, etc.
- We cannot say that image matching has failed but it wouldn't work if were to used a horse image and car image.

Problem 3 : Edge Detection

I. Abstract and Motivation

In an image we have many objects that are present. The reason we are able to detect these objects separately is because the human brain is capable of understanding one object from another. This happens because the brain is able to form a edge or boundary around an object so as to confine it within an boundary and identify it with predefined objects stored in our brain.

Edge detection also works well with images which have distinctive objects in the image. This is possible because gradient change in image color space such that we can detect it as an edge. Edge detection has many applications as it can be used in image segmentation, data or feature extraction. Edge detection is used in the medical field in X-Rays analysis too to find cracks in bones by analysis change in bone density grayscale values. It is also used in autonomous driving systems where we image frames from a live video feed are captured and analyzed to check for edges which will later be used for object classification. This object classification will then help in driving the autonomous vehicle.

Hence we can say that edge detection is a vital process in image processing. We will discuss two such edge detection techniques below namely know as Canny Edge Detection and Structured Edge Random Forest Edge Detection. These edge detection algorithms will be run using OpenCV and MATLAB image processing toolbox.

***Please note** : We will not discuss "**Abstract and** **Motivation**" in the further part of this section.*

(a) Canny Edge Detector

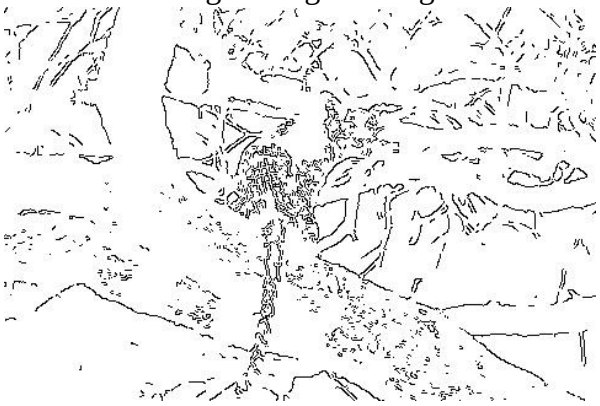
II. Approach and Procedure

- The first step is to input both the given raw images of zebra and jaguar into the two different variables.
- Now we need to change the color image into grayscale image such that we it can be used by the canny edge detector algorithm provided in the OpenCV library.
- The next step to create 3 matrices of zeros of the same size as the input images.
- Now the image data is stored in a matrix called ***M*.**
- The next step is to call the Canny OpenCV function where we send the matrix M, a low threshold and a high threshold for edge detection.
- The output of the Canny function stores its the matrix ***N***
- This matrix N is then inverted and the output is displayed and saved in .jpg file.
- This process is again carried out for the second input image that we have provided.

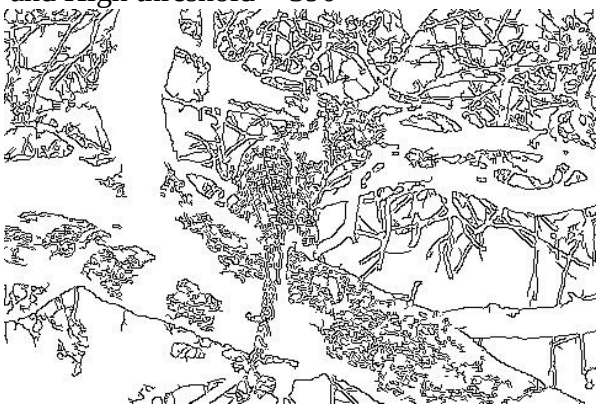
III. Results

Original Zebra Image

Original Jaguar Image



Low threshold = 312 and High threshold = 390



Low threshold = 200 and High threshold = 400

Low threshold = 500 and High threshold = 600

IV. Discussion

- From the above results it is clear that different thresholding values give us different edge detection. Edge detection does not work very well using the Canny Edge detector using varied thresholding values.
- For the Zebra image we are able outline the body of Zebra but the Jaguar is camouflaged with the trees and it gets difficult in getting the object outline.
- We can now discuss how thresholding and image content work hand in hand is getting perfect edge detection. Many factors play an important role in edge detection like texture, colorspace of the image, image contours, smoothness of an image and weak object boundaries.
- Varying texture in an image will help in image edge detection and improve accuracy of the edge generated.
- Varying color space increases the contrast of the image and this helps in improved edge detection. To improve edge detection a high contrast image is ideal.

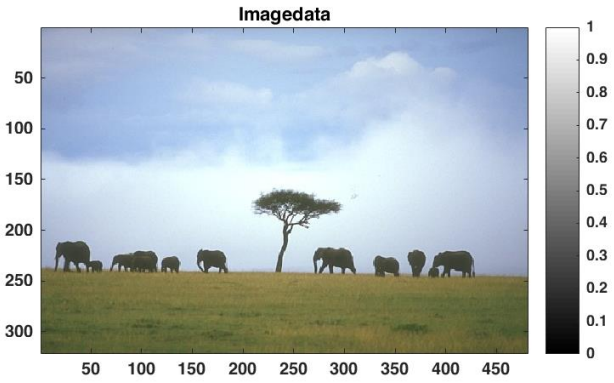
(b) and (c) Structured Edge Detection and Performance Evaluation

II. Approach and Procedure

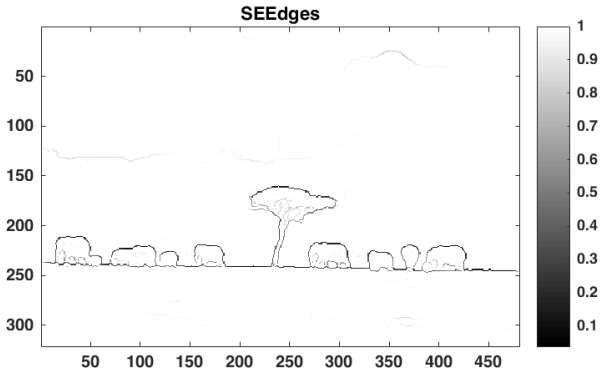
We will be performing part b and part c of the problem statement 3 together both in a Matlab environment.

- The first step is take the input image given by the user.
- Once we know the image we then input all the ground truth values given in the problem statement.
- We have made two functions to read RGB image and grayscale images.
- The next step is to assign a threshold depending on the input image provided by the user. This threshold will affect the F-measure during performance evaluation.
- Now we use a predefined code to create a training model using training dataset of images. To perform this have installed toolboxes to perform image processing. We get an output called ***model*** as the trained model.
- Now we call edgeDetect function where we send the image and the trained model. Then we invert the output of the edge detection function.
- The next step is to perform binarization and this is where we use the threshold values that we have initialized above. This will convert the image into a binary image of 1s and 0s.
- Now we create a structure of containing images of ground truths that are provided in the problem statement.
- The next step is to perform evaluation ***edgesEvalImg*** where we send the binary image and also the structures containing the ground truth.
- Now we calculate the recall and precision which creates an array of recall and precision. Here we remove all zero values and NAN values that we formed and then take a mean of all the values in the both arrays respectively.
- Using the mean recall and mean precision we calculate the F Measure of the edge detection output with the ground truth images.

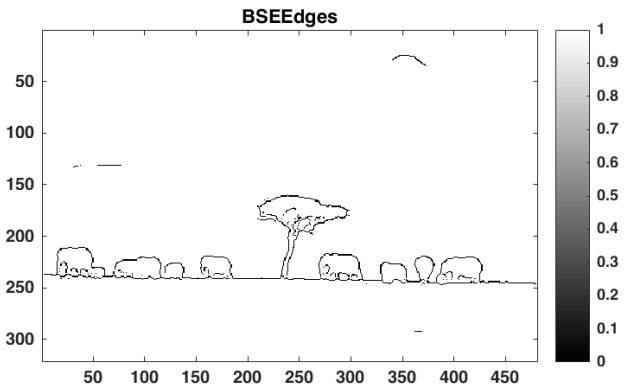
III. Results



Original Elephant Image



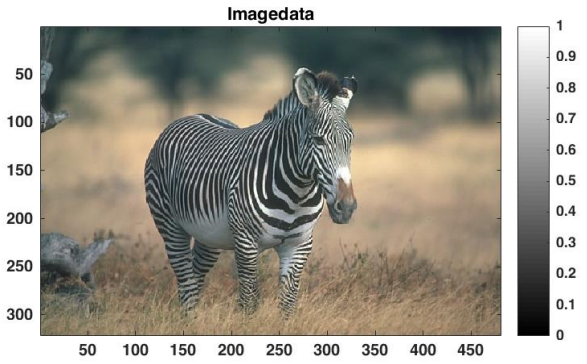
Structured Edge Output



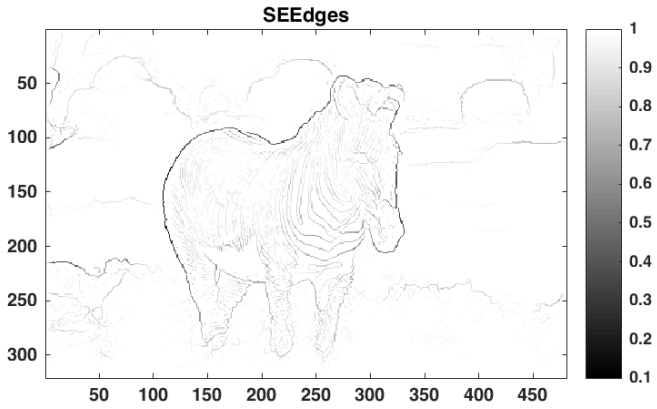
Binary Output

```
Input Image Filename as "Filename" : Elephant
    Retrieving Image Elephant.raw ...
    Retrieving Image Elephant_gt1.raw ...
    Retrieving Image Elephant_gt2.raw ...
    Retrieving Image Elephant_gt3.raw ...
    Retrieving Image Elephant_gt4.raw ...
    Retrieving Image Elephant_gt5.raw ...
    Retrieving Image Elephant_gt6.raw ...
Elapsed time is 0.460720 seconds.
Elapsed time is 0.201817 seconds.
Starting Performance Evaluation...
Performance Evaluation Done

F_Measure =

    0.9058
```

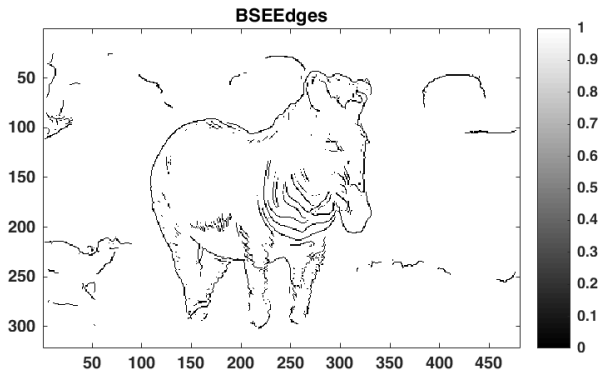
F-Measure Result



Original Zebra Image



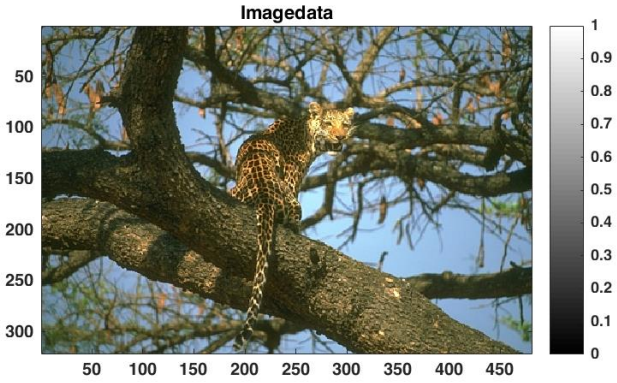
Structured Edge Output



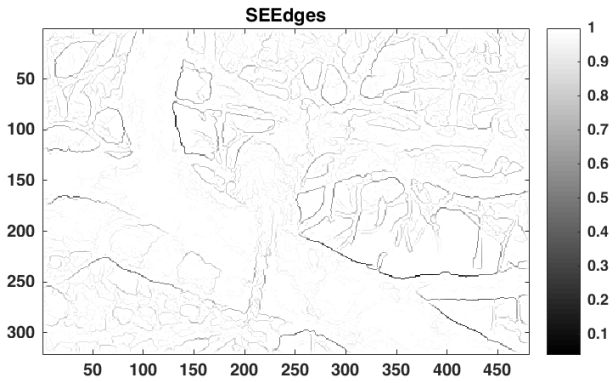
Binary Output

```
Input Image Filename as "Filename" : Zebra
    Retrieving Image Zebra.raw ...
    Retrieving Image Zebra_gt1.raw ...
    Retrieving Image Zebra_gt2.raw ...
    Retrieving Image Zebra_gt3.raw ...
    Retrieving Image Zebra_gt4.raw ...
    Retrieving Image Zebra_gt5.raw ...
Elapsed time is 0.453087 seconds.
Elapsed time is 0.270484 seconds.
Starting Performance Evaluation...
Performance Evaluation Done

F_Measure =

    0.5954
```

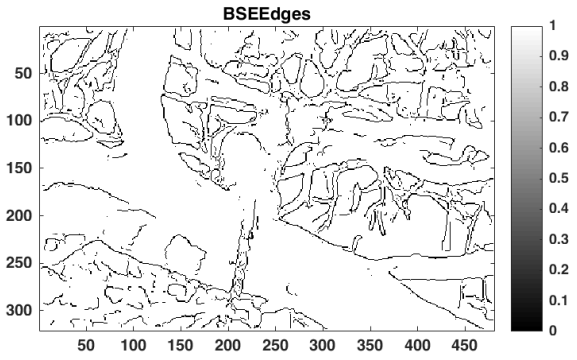
F-Measure Result



Original Jaguar Image



Structured Edge Output



Binary Output

```
Input Image Filename as "Filename" : Jaguar
    Retrieving Image Jaguar.raw ...
    Retrieving Image Jaguar_gt1.raw ...
    Retrieving Image Jaguar_gt2.raw ...
    Retrieving Image Jaguar_gt3.raw ...
    Retrieving Image Jaguar_gt4.raw ...
    Retrieving Image Jaguar_gt5.raw ...
    Retrieving Image Jaguar_gt6.raw ...
Elapsed time is 0.454610 seconds.
Elapsed time is 0.345996 seconds.
Starting Performance Evaluation...
Performance Evaluation Done

F_Measure =

    0.8778
```

F-Measure Result

IV. Discussion

- Looking at the output images we can see that edge detection has worked really well for the 3 images provided.
- The elephant image binary output gives us the best F Measure score
- The F Measure score for the jaguar score is really high due to high false positive and true negative.
- The Zebra gives us a lower F Measure score but the output of the image is better as we are able to identify the zebra as an object.
- The thresholding is chosen in such a way that precision and recall are calculated properly. A high precision and a low recall will give a low F Measure and hence the thresholding needs to be chosen such that we get a high F Measure.

References

[1] David G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, 60(2), 91-110, 2004.
[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", *Computer Vision and Image Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008
[3] Canny, John. "A computational approach to edge detection." Pattern Analysis and Machine Intelligence, IEEE Transactions on 6 (1986): 679-698.
[4] Dollár, Piotr, and C. Lawrence Zitnick. "Structured forests for fast edge detection." Computer Vision (ICCV), 2013 IEEE International Conference on. IEEE, 2013.
[5] Arbelaez, Pablo, et al. "Contour detection and hierarchical image segmentation." Pattern Analysis and Machine Intelligence, IEEE Transactions on 33.5 (2011): 898-916.
[6] [Online] http://images.google.com/
[7] [Online] http://sipi.usc.edu/database/
[8] [Online] https://en.wikipedia.org/wiki/CMYK_color_model
[9] [Online] https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect12.pdf
[10] [Online] http://courses.cs.washington.edu/courses/cse576/book/ch7.pdf
[11] [Online] http://www.macs.hw.ac.uk/texturelab/files/publications/phds_mscs/MJC/ch5.pdf