Texture and Pothole Detection for Mobility Assistant for the Visually Impaired (MAVI)

by

Durgesh (2012CS50286)

A thesis submitted in partial fulfillment
for the degree of

MASTER OF TECHNOLOGY

in

Computer Science and Engineering IIT Delhi

Under the Guidance of

Porf. M. Balakrishnan

Dr. Chetan Arora

August 2017

Declaration of Authorship

This is to declare that this thesis titled, **Texture and Pothole Detection for MAVI** being submitted by **Durgesh** for award of **Master of Technology in Computer Science and Engineering** is an authentic work carried out by him under my guidance and supervision at the **Department of Computer Science and Engineering**.

Prof. M. Balakrishnan
Department of Computer Science and Engineering
Indian Institute of Technology Delhi

Dr. Chetan Arora
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology Delhi

Abstract

Pothole and surface detection is one of the important task for guidance and safety of visually challenged people. Warning beforehand can be very helpful for the safety of visually impaired people. In recent years support vector machines (SVM) are showing excellent performance in classification problems. This thesis address Pothole detection and its area estimation using image segmentation and spectral clustering with the help of SVM on top of it. Ground plane estimation using depth sensor is used to eliminate unnecessary rough surface to be detected as potholes. For the Surface detection using their texture pattern I applied SVM using Segmentation based Fractal Texture Analysis (SFTA) feature extraction to classify different textures.

All the dataset are taken from IIT Delhi campus and nearby areas. The frames are taken by PivotHead camera and depth images are taken using Intel Depth sensor. In case of Surface texture detection the classes includes Pavement, Road and Other(any other surface). The classifier is trained with samples of 40x40 window size which has shown good performance and accuracy. In case of Potholes I've used window size of 80x80 which has shown good accuracy. All experiments and implementation are done using both MATLAB and OpenCV in C++.

Acknowledgements

I would like to articulate my deep gratitude to my supervisor Prof. M. Balakrishnan who has always been my motivation for carrying out the thesis. I am highly indebted to him for believing in me and for his constant supervision and valuable guidance during the course of thesis

I would also like to thank co-supervisor Dr. Chetan Arora for his valuable insight and assistance regarding subject of Computer Vision.

I also extent my thanks to Mr. Rajesh Kedia, Mr. Anupam Sobti and Mr. Saurabh Agrawal for having useful discussion and providing ideas, Mr. Som Dutt Sharma for providing me with all the lab equipment and support.

Durgesh

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

List of Figures vi

1 Prelude 1
- 1.1 Introduction . . . 1
- 1.2 Motivation and Objective . . . 2
- 1.3 Thesis Contribution . . . 2
- 1.4 Thesis Outline . . . 3

2 Deep Neural Network on ZedBoard 4
- 2.1 Introduction . . . 4
- 2.2 Cross Compilation of Caffe . . . 5
- 2.3 SqueezeNet Deep Compression on ZedBoard . . . 7

3 Texture Classification using SVM 8
- 3.1 Introduction . . . 8
- 3.2 SFTA and Dataset tuning for PivotHead . . . 10
  - 3.2.1 Dataset Tuning . . . 10
  - 3.2.2 SFTA Tuning . . . 10

4 Analysis - Time, Distance and Accuracy 11
- 4.1 Accuracy vs Window Size . . . 11
- 4.2 Accuracy vs Distance . . . 13
  - 4.2.1 40x40 window size . . . 13
  - 4.2.2 80x80 window size . . . 14
- 4.3 Time Analysis . . . 16
  - 4.3.1 40x40 window size . . . 16
  - 4.3.2 80x80 window size . . . 17

5 Pothole Detection **18**
5.1 Identifying Distressed Regions . . . 18
5.2 Shape Extraction . . . 20
5.3 Identification and Extraction of Candidate Regions . . . 21
5.4 Classifying Candidate Regions using SVM . . . 23

6 Ground Plane Estimation **25**
6.1 Method . . . 25
6.2 Integration with Pothole Detection . . . 27

7 Pothole Software Only Implementation **28**
7.1 Dataset . . . 28
7.2 MATLAB Implementation . . . 28
7.2.1 Accuracy . . . 29
7.3 OpenCV Implementation . . . 29
7.3.1 Accuracy . . . 29
7.3.2 Performance . . . 30
7.4 Cross Compilation on ZedBoard . . . 30
7.4.1 Description of ZedBoard . . . 30
7.4.2 Cross Compilation of OpenCV and Xillinux on zedboard . . . 30
7.4.3 Accuracy . . . 30
7.4.4 Performance . . . 31
7.5 Profiling and Hardware Acceleration Hotspots . . . 31
7.5.1 Texture Detection . . . 32
7.5.2 Pothole Detection . . . 33

8 Conclusion and Future Work **34**
8.1 Results & Conclusion . . . 34
8.1.1 Texture Detection . . . 34
8.1.2 Pothole Detection . . . 35
8.1.3 Ground Plane Detection . . . 36
8.2 Future Work . . . 37

Bibliography **38**

List of Figures

1.1 Overview of MAVI 1

3.1 Overview of Training SVM classifyer 9
3.2 Overview of Testing SVM classifier 9

4.1 Accuracy of 40x40 vs 80x80 window size 12
4.2 CDF of accuracy of 40x40 vs 80x80 window size 12
4.3 Accuracy vs distance for 40x40 window 13
4.4 Sample output of 40x40 window size 14
4.5 Accuracy vs distance for 80x80 window 15
4.6 Sample output of 80x80 window size 15
4.7 CDF 40x40 16
4.8 CDF 80x80 17

5.1 Process of forming 2 cropped images from original image. 19
5.2 Segmented image. 20
5.3 Resulting clustered image. 22
5.4 Pothole identification with seeds points. 23
5.5 Pothole identification with SVM. 24

6.1 Fitted curve vs Original data 26
6.2 Ground plane detection. White region is above or below ground plane. 27

7.1 Texture detection profiling 32
7.2 Pothole detection profiling 33

8.1 Texture Detection 34
8.2 Texture Detection 35
8.3 Pothole Detection 35
8.4 Accuracy vs distance for 40x40 window 36
8.5 Original Image 36
8.6 Ground Plane curve fitting 36
8.7 Ground Plane detected 36
8.8 Original Image 36
8.9 Ground Plane curve fitting 36
8.10 Ground Plane detected 36

Chapter 1

Prelude

1.1 Introduction

Mobility Assistant for the Visually Impaired (MAVI)

MAVI is an ambitious project aimed at enabling mobility for visually impaired individuals, specially in India. The three major problems we want to tackle with MAVI are Safty, Social Inclusion and Navigation. The overview of mavi system is shown in figure below

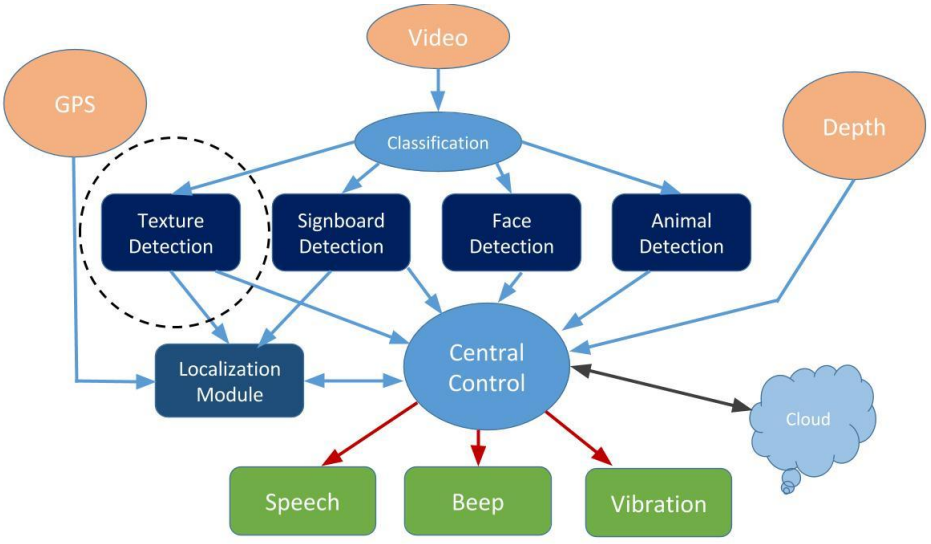


FIGURE 1.1: Overview of MAVI

1.2 Motivation and Objective

Eyes are one of the most important sense organ human body have to interact with their surrounding environment. Because of visual impairment, the ability for interacting with surrounding is very difficult and limited. i.e it become very difficult or almost impossible to walk, find places and social inclusion for visually challenged people. So, visual impairment is one of the severe type of disabilities a person must endure.

In India mostly visually impaired people have to rely on traditional cane. The cane is very limited in terms of providing interaction and independence to its users. Now days Computer Vision is one of the biggest and most active research area and it can be very helpful for the visually impaired people as their eyes. So we team MAVI decided to design prototype that consist of Texture Detection , Pothole Detection, Face Detection, Signboard Detection and location information. Objective of this thesis is to develop and Pothole and Texture detection module which aids the visually impaired in navigation.

1.3 Thesis Contribution

This thesis address problem of Texture and Pothole detection using texture patterns. This module is one of the four sensors of the MAVI. I've continued Yousoof's work (Last year Student). He Solved texture detection problem using SVM classifier. SFTA, a feature extraction algorithm based on fractal dimensions, is used as feature extraction algorithm and then SVM with linear kernel on top of it for classification of texture in different classes. Three classifiers for classifying classes Pavement, Road, grass and mud have been implemented by Yousoof.

For pothole detection I first explored different pothole detection algorithms that suits the application. After that based of accuracy, performance and available technology. Best approach I found which

suits our application is based on Image processing and spectral clustering which use no sophisticated equipment or perform computationally intensive tasks. This is a unsupervised vision-based approach which deploys image processing and spectral clustering for identification and rough estimation of potholes. Spectral clustering is used for identification of regions with histogram-based data from gray-scaled image. I found that this approach work well on ground but it also detect other objects which are not on the ground as potholes So, to avoid that I've first detected ground-plane using depth camera and applied pothole detection on top of it. First I implemented it using MATLAB then using OpenCV and verified the result with the MATLAB. Finally I implemented on ZedBoard.

1.4 Thesis Outline

The body of thesis is divided into eight chapters including this introduction. The remaining of thesis is organized as follows: Chapter 2 Describes about feasibility of neural network on ZedBoard. Chapter 3 describes texture detection using SVM (which is continued from last year) and my contribution to it. Chapter 4 berif about time, distance and accuracy analysis of texture detection algorithm on both PC and ZedBoard. Cahpter 5 describes ground plane detection using depth camera. Chapter 6 describes complete flow of pothole detection. Chapter 7 gives result of Pothole detection in MATLAB and OpenCV on Desktop platform and ZedBoard. Then Chapter 8 is conclusion of my work and possible future extension of my work

Chapter 2

Deep Neural Network on ZedBoard

2.1 Introduction

Neural Networks are one of the latest and growing technology in field of Machine learning and Artificial intelligence. Neural Networks are computing systems inspired by working of biological network in the brain. Neural network are found most useful in problems which are difficult to solve using traditional computational algorithms. Many classification problems in area of vision and image processing can be solved using neural network for example image recognition. A classification problem involves a training and testing data set with class labels (In case of supervised learning) or without labels. And given a training data set we compute a function from training set to class. Both supervised and unsupervised classification problems can be solved using neural networks. Thus neural network can be very useful in Face Detection, Texture and Pothole detection, Signboard detection and OCR and Animal detection modules of MAVI system. But neural network require to perform large amount of computation So, we wanted to test its performance on ZedBoard in order to find out if it can be any useful for us. We are more interested in memory consumption of neural network as ZedBoard has very limited memory available.

2.2 Cross Compilation of Caffe

Implementing Neural Network from scratch can be very time consuming task So, I first explored already available tools for neural network implementation. Caffe is one of the popular open source deep learning framework Developed by Berkeley AI Research. It supports many different types of architectures geard towards image classification and image segmentation. In order to implement neural network in caffe we require defination of network in prototex format(Simple text format). So, I decided to use caffe for feasibility testing of neural network on ZedBoard.

ZedBoard(Zynq Evaluation and Development Board) is an excellent development kit based on Zynq All Programmable SoC (AP SoC). ZedBoard is ARM processor based. So, In order to compile for ZedBoard first we have generated arm binaries on Intel process using cross compiler tool chain for ARM. Then transferred binaries on ZedBoard

Caffe and it's all dependencies provide CMake build system for building. CMake is cross-platform build system which use compiler-independent method and provide very easy way to handle dependencies. CMake separates build from source so several builds are possible from same source directory. This is very useful and easy to handle in case when we need to compile of different platforms(In our case for Intel and ARM).

After compiling all dependencies, next step is to link those dependencies to the caffe itself and generate final executable image. The linker can either put all dependencies entirely into executable image or just remember their path on system and include dynamically at runtime. For Caffe I needed to compile following dependencies for ARM:

- **C++ Boost library:** provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing

- **Opencv(Open Computer Vision)**
- **LevelDB:** Fast Key-Value storage library. Github
- **gflags:** C++ library that implements commandline flags processing. Github
- **glog:** C++ implementation of the Google logging module. Github
- **LMDB:** Lightning Memory-Mapped Database Github
- **Protobuf:** Protocol Buffers - Google's data interchange format. Github
- **Snappy:** A fast compressor/decompressor. Github
- **HDF5:** Data model, library, and file format for storing and managing data. HDF5

Dynamic Linking: In this kind of linking linker takes path of shared libraries(.so on Linux, .dll on Windows) at the time of compiling and link them while running the program by copying into RAM from storage and subsequently filling jump tables and relocation pointers. This is useful when more then one program is using libraries, we can reduce storage uses by linking to same shared library in storage. And any change in library don't require re-compilation of executable.

Static Linking: This is useful when we don't have high-level OS thus we can't have shared libraries to link in runtime. So, static libraries(.a in Linux) are copied into executable at the time of compilation of executable. In our case portability of executable on different platform become very easy as executable contain everyting that it needs to run. Size of executable will be more in this case due to attached libraries.

2.3 SqueezeNet Deep Compression on ZedBoard

I tested performance of 3 different networks on ZedBoard:

LeNet: LeNet is small convolution neural network for hand-written digit recognition. I used MNIST dataset for hand-writted digits. Caffe provide pre-trained model using MNIST dataset so, we just need to feed in network defination, pre-trained model and a input image to caffe and it will output class of that image. LetNet took only 8MB of memory and 0.154 sec. So, it was Feasible on ZedBoard. But LeNet is very small network.

ILSVRC: LeNet was feasible on ZedBoard but its very small network and not much suitable for over application So, I tested on large network. ImageNet Large Scale Visual Recognition Challenge(ILSVRC) is a competition based on ImageNet dataset for detection and classification of objects in scenes. ImageNet is very large visual database designed for use in visual object recognition softwares. ImageNet contain over 10 million annotated images with their classes. I used pre-trained caffe model of ILSVRC12 network. It took 931MB of memory on Desktop platform. So, It was not feasible on ZedBoard because ZedBoard has total of 500MB memory available.

SqueezeNet Deep Compression: SqueezedNet is optimized version of AlexNet(network based on ImageNet dataset) which contain 363x smaller parameters as of AlexNet with same accuracy. SqueezedNet took only 60MB of memory So, it is feasible on ZedBoard.

Chapter 3

Texture Classification using SVM

3.1 Introduction

Texture is defined as structural pattern of surfaces which is homogeneous in spite of fluctuations in brightness and color. Texture Classification using SVM can be divided into following three parts:

1. Texture feature extraction using SFTA
2. Training SVM classifier
3. Classification of unknown texture image

Our implementation classify texture into three main classes: Pavement, Road and Grass. Given Image we divide into windows of 80x80 then extract features of each window and pass to SVM for training and classification. MATLAB and OpenCV Implementation has shown following accuracy's for different classes:

Pavement: 90.74%

Road: 93.16%

Grass: 95.12%

Please refer to Yoosuf's thesis for more information

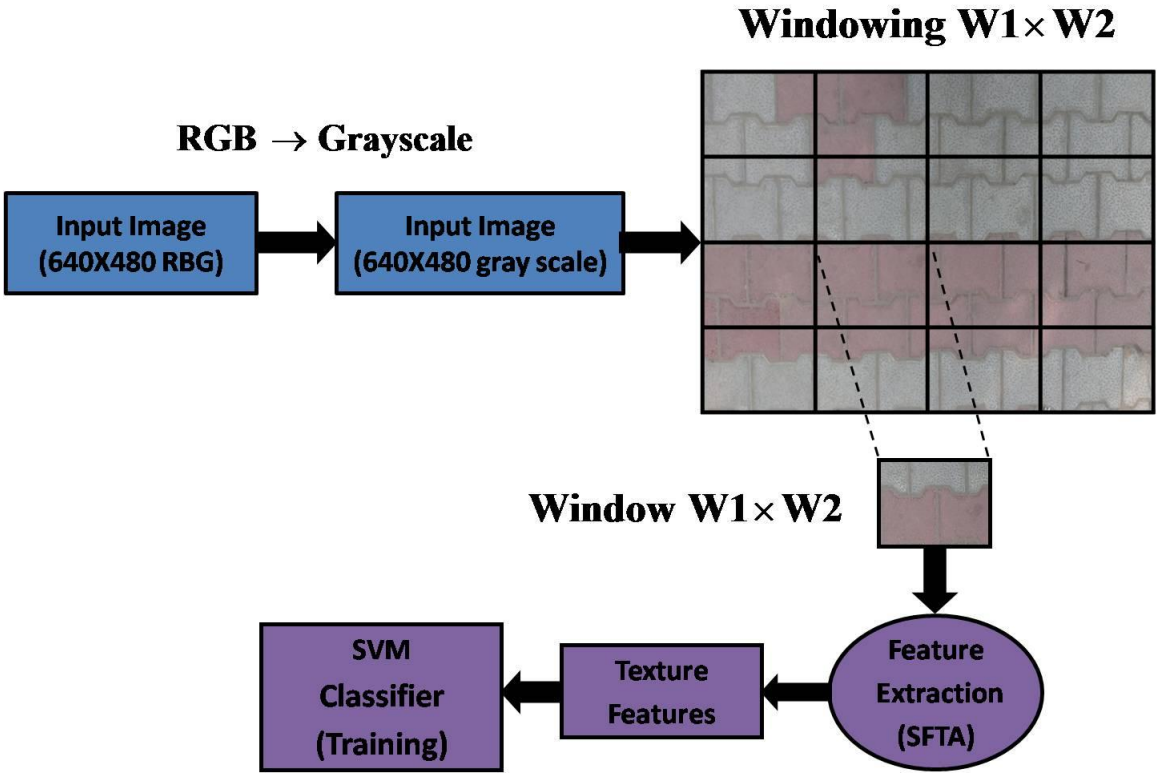


FIGURE 3.1: Overview of Training SVM classifyer

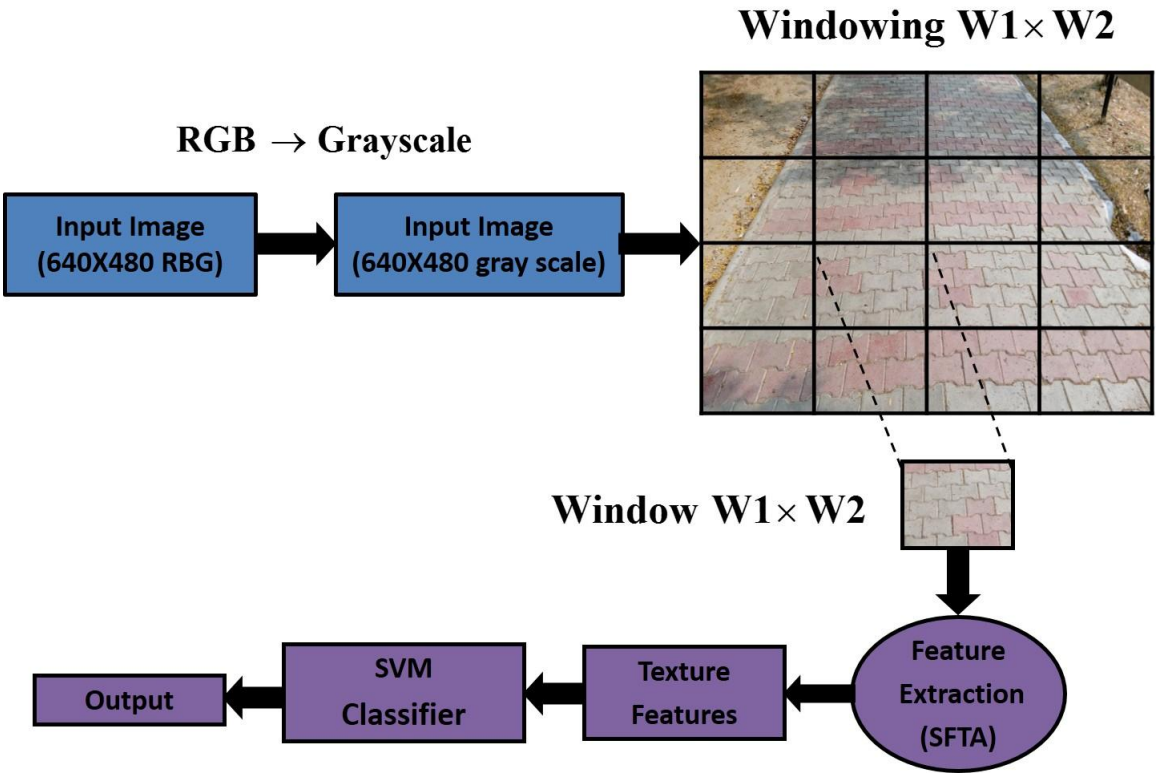


FIGURE 3.2: Overview of Testing SVM classifier

3.2 SFTA and Dataset tuning for PivotHead

Previously we were using chest mounted camera at 60* angle to vertical plane. But now we have changed setting to PivotHead camera which is almost 90* angle with vertical plane. Due to change in camera settings accuracy's of SVM dropped as it was trained with 60* camera pictures. So, I retrained SVM with dataset collected from PivotHead settings. As from 90* angle camera was able to capture very little details so, just retraining didn't gained same accuracy as 60* camera angle settings. I had to tune dataset and SFTA for achieving same accuracy's.

3.2.1 Dataset Tuning

Because of more angle with vertical plane of PivotHead camera it was able to capture less details. Due to which classification between Road and Grass was very difficult as their texture pattern is very similar to each other So, I had to exclude grass class and finally trained SVM for Pavement, Road and Other(which include anything other that Road and Pavement) classes. In case of Road and Pavement I had to exclude road image which contain more rough surface as it lead to miss-classification between road and pavement.

3.2.2 SFTA Tuning

SVM accuracy's depends largely on window size also. If we train the classifier with small window size then it won't be able to capture more details of texture pattern. On the other hand if we train with large window size then it won't be able to capture localization properly. With 60* camera angle 80x80 window size was most accurate. But in case of PivotHead due to large distance 80x80 lead very poor precision in localization and lead to inaccuracy.

Here I explored SVM accuracy's with different window size of 60x60, 40x40 and 20x20. In case of PivotHead 40x40 was giving best accuracy's.

Chapter 4

Analysis - Time, Distance and Accuracy

In order to run different modules in MAVI we have a scheduler which will help in schedule different modules. For that purpose scheduler require to know how much time will a given module take. So, we need statical analysis on our module for scheduling.

4.1 Accuracy vs Window Size

As we discuees in chapter 3 accuracy of SVM varies with window size. As we can see in figure below that in case of 40x40 peek of cuve is more towards 100% then 80x80. Which indicates that more greater number of frames has shown more accuracy in 40x40 window size then 80x80 window size.

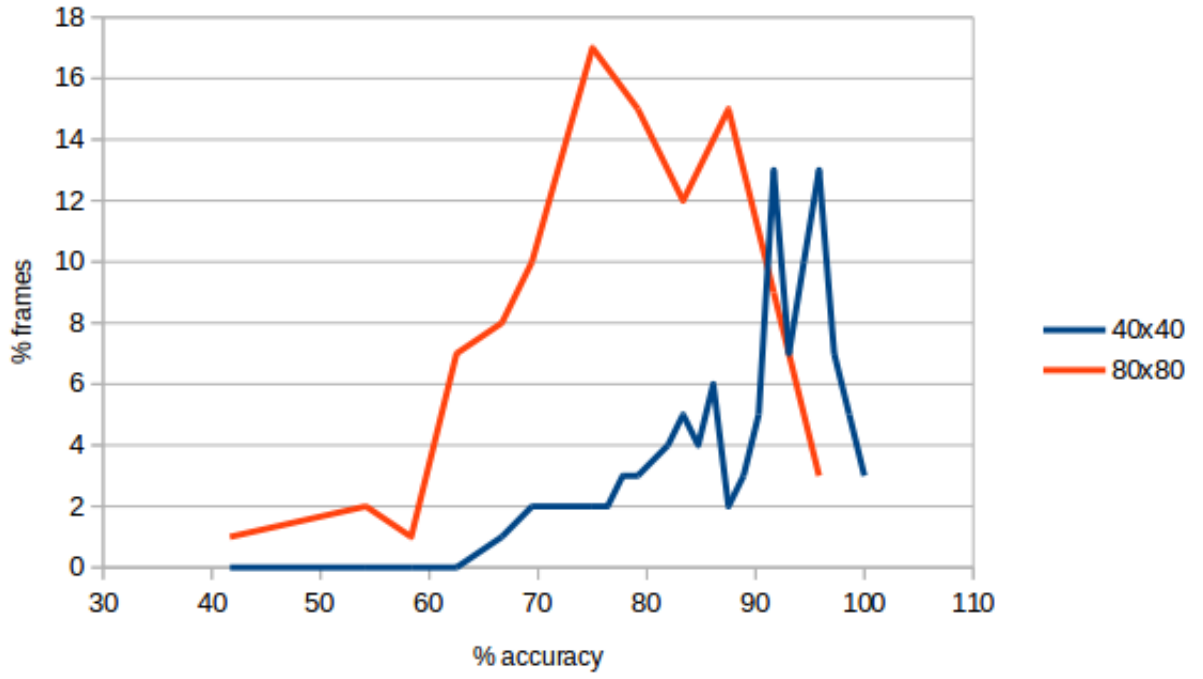


FIGURE 4.1: Accuracy of 40x40 vs 80x80 window size

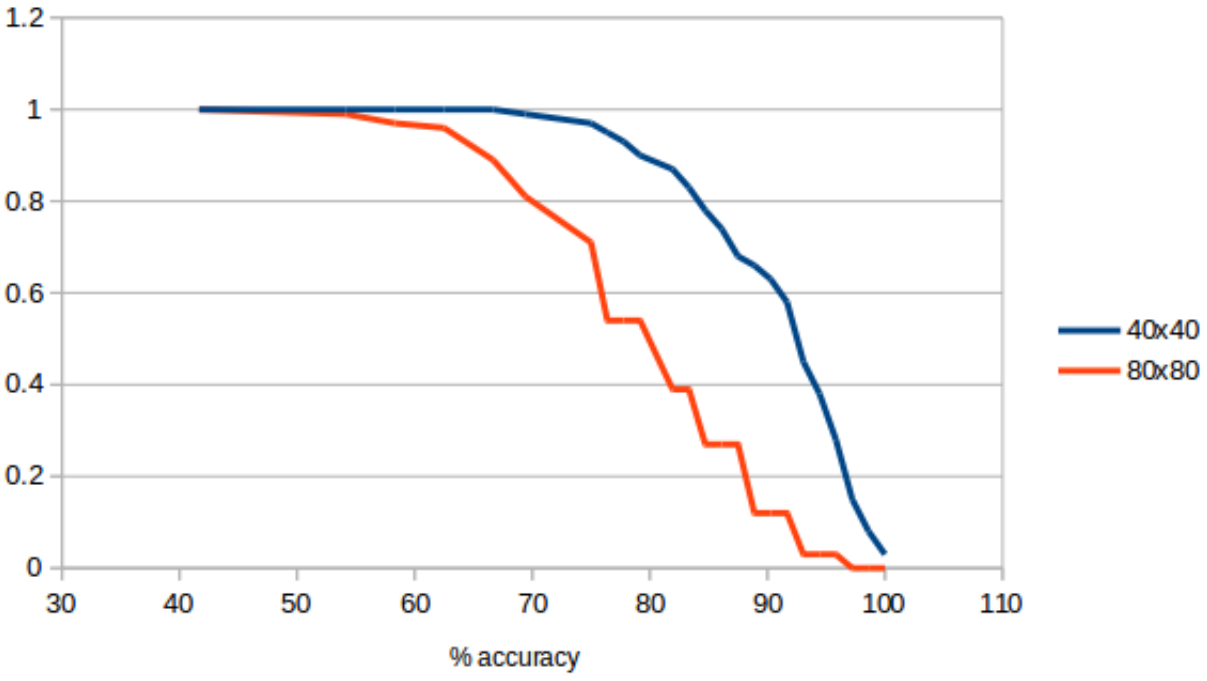


FIGURE 4.2: CDF of accuracy of 40x40 vs 80x80 window size

From CDF plot its more clear that in case of 40x40 more then 80% of the frames show more then 80% accuracy's. On the other hand only 40% of the frames show more then 80% accuracy's in case of 80x80 window size.

4.2 Accuracy vs Distance

4.2.1 40x40 window size

From figure we can see that 6th row show lowest accuracy because of inclusion of sky which is getting classified into road

Distance and avg accuracy's of rows from camera are as follows:

1 st Row is 3.2m away, Avg acc: 88.5%

2 nd Row is 3.6m away, Avg acc: 90.7%

3 rd Row is 4.5m away, Avg acc: 91%

4 th Row is 5.8m away, Avg acc: 94.8%

5 th Row is 7.5m away, Avg acc: 90.4%

6 th Row is 11.5m away, Avg acc: 83%

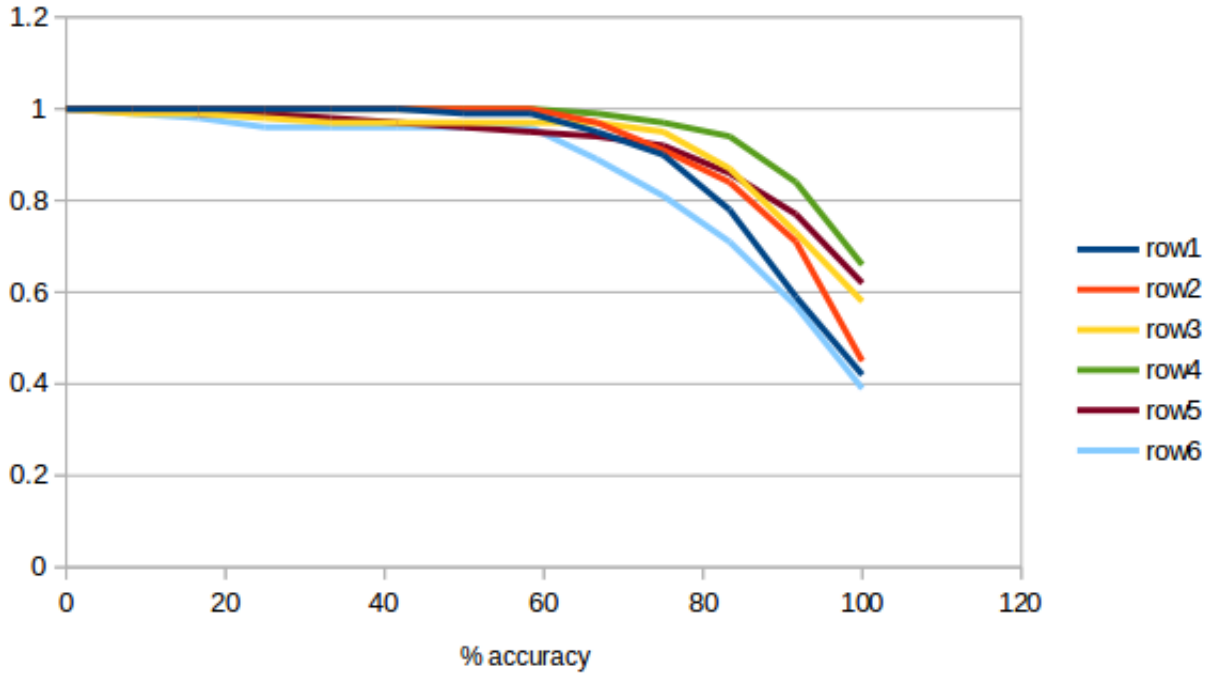


FIGURE 4.3: Accuracy vs distance for 40x40 window

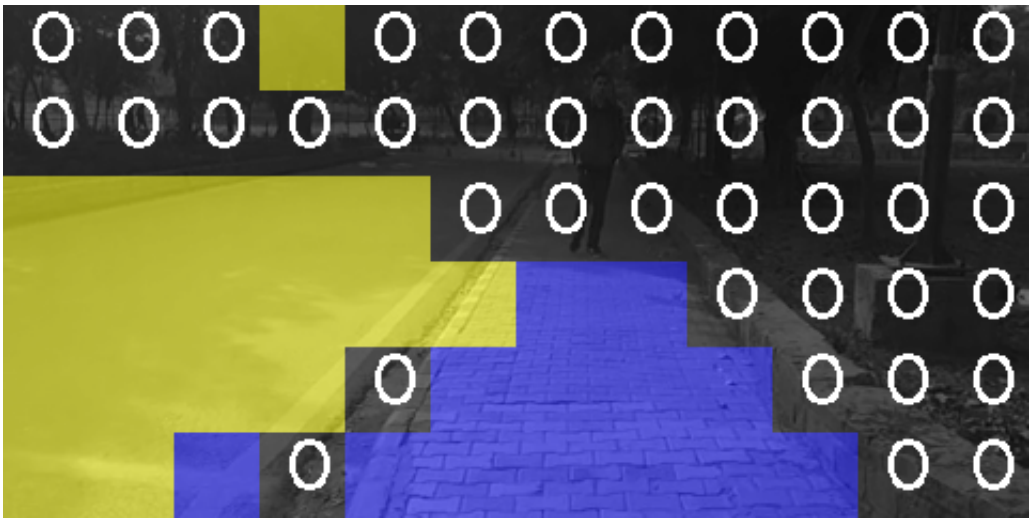


FIGURE 4.4: Sample output of 40x40 window size

Here we can see that middle rows(3rd and 4th) are showing maximum accuracy because middle rows mostly consist of blocks of Other class and SVM can separate Other class from Road or Pavement better because texture of Other class is in general very different form Road and Pavement. But it will miss-classify between Road and Pavement more because they have similar texture. In 1st and 2nd rows mostly there is Road and Pavement that's why its accuracy is less then middle rows due to more miss-classification between Road and Pavement.

4.2.2 80x80 window size

Distance and avg accuracys of rows from camera are as follows:

1 st Row is 3.4m away, Avg acc: 66.1%

2 nd Row is 5m away, Avg acc: 66.4%

3 rd Row is 9.5m away, Avg acc: 78%

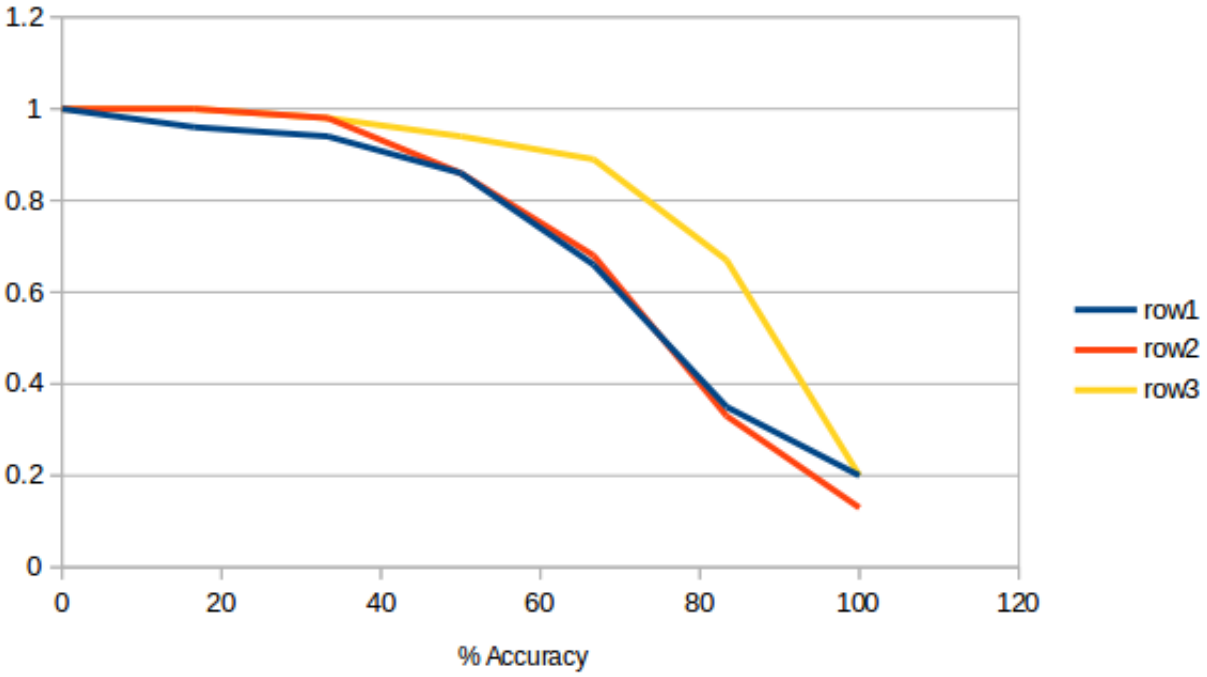


FIGURE 4.5: Accuracy vs distance for 80x80 window

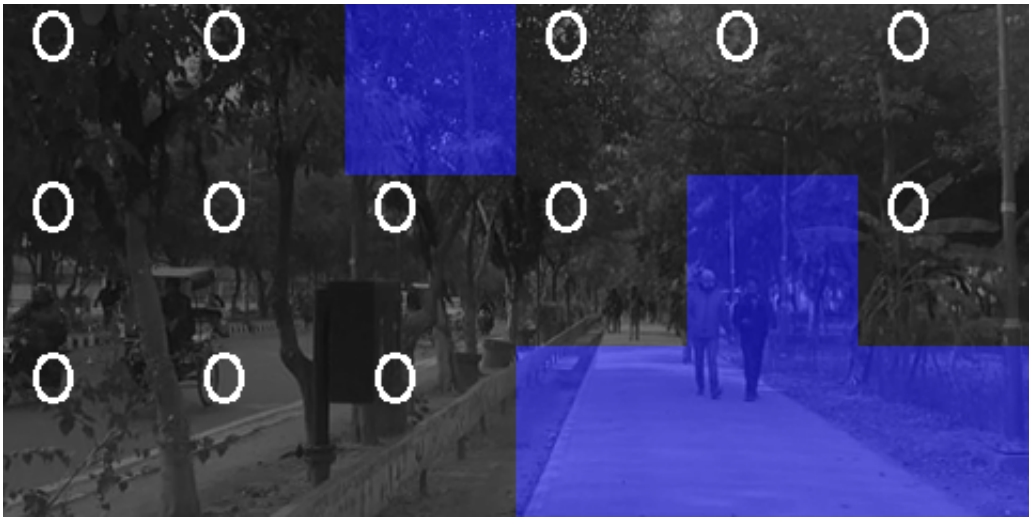


FIGURE 4.6: Sample output of 80x80 window size

Here 3rd row show maximum accuracy because in 3rd row most of the time a block contain texture of Other class entirely. But in case of 1st and 2nd row block may contain texture from multiply classes so it will show less accuracy.

4.3 Time Analysis

4.3.1 40x40 window size

Here 90% of the frames take 1.25 second to complete. So, it can be good time slot for scheduler to assign to this module

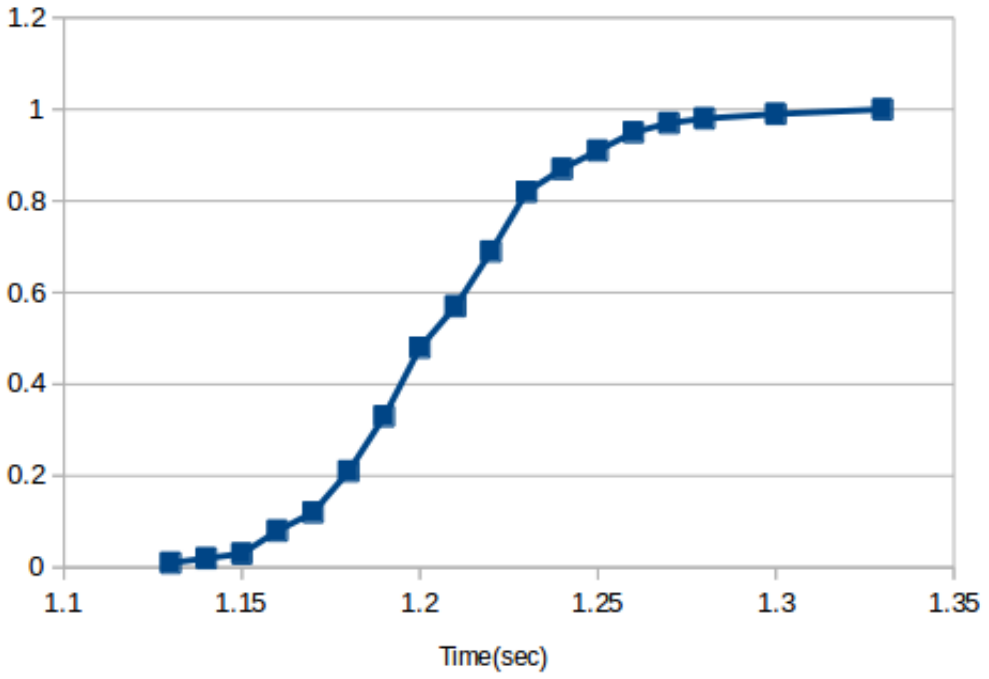


FIGURE 4.7: CDF 40x40

Here I observed that initial few frames take more time then most of the frames its due to initial cache worm-up.

4.3.2 80x80 window size

Here 90% of the frames take 1.4 second to complete. So, it can be good time slot for scheduler to assign to this module

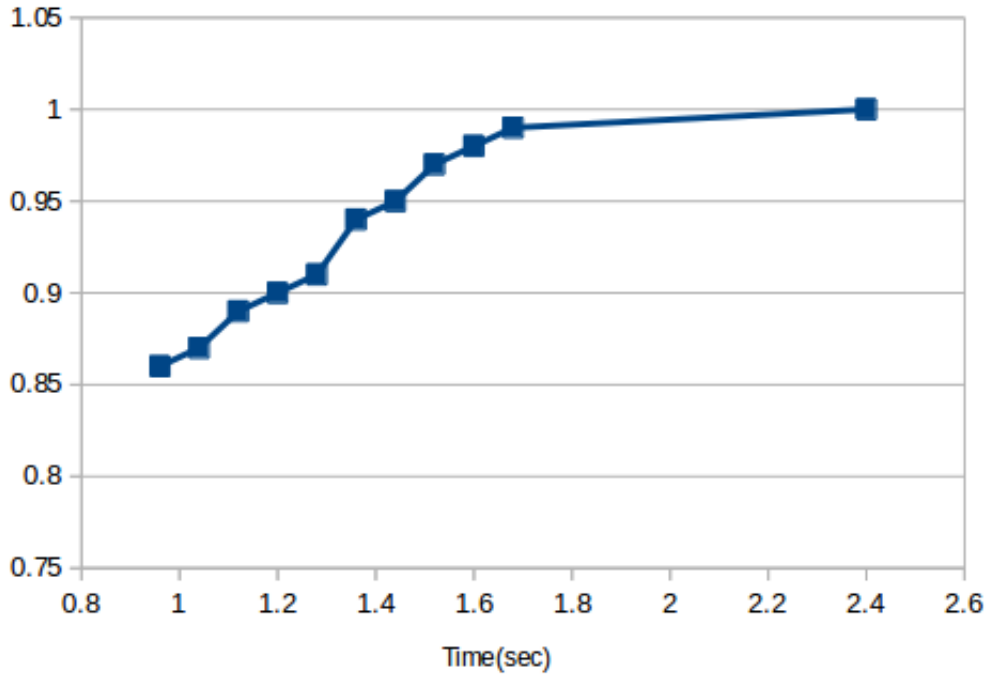


FIGURE 4.8: CDF 80x80

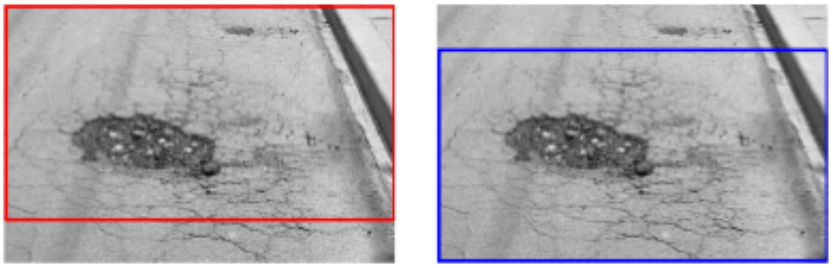
Chapter 5

Pothole Detection

Pothole detection is one of the important task for guidance and safety of visually challenged people. There are many methods are available for detection and estimation of potholes which use sophisticated equipment and impose computationally intensive tasks. As we want to reduce cost of MAVI. So, I chosen simple unsupervised vision-based method, which does not require expensive equipment. Spectral clustering is used for identification of regions with histogram-based data from gray-scaled image. Based on these results, we identify potholes and estimate their surface.

5.1 Identifying Distressed Regions

Our technique based on assumption that pothole will be rough area and will be heaving more distressed region then surrounding surface. So, First step would be to detect rough region and then extract exact area of pothole. In our case I am using image segmentation to detect distressed region. Image segmentation is largely used method to detect relevant information from digital image, and it is first step of image analysis and pattern recognition. There are many ways to perform image segmentation including clustering, threshloding etc. In our case we are using histogram based thresholding method for image segmentation. Here we will set threshold T such that it will separate background pixels from foreground.



(a) The image is cropped for δ rows from bottom (marked red).

(b) The image is cropped for δ rows from top (marked blue).

FIGURE 5.1: Process of forming 2 cropped images from original image.

In this case I am using Otsu's thresholding to find threshold T which separates background and foreground pixels. This threshold T will be used to obtain 2 cropped images. From those 2 cropped images I obtain segmented binary image which will represent distressed regions, candidate regions for potholes. Process of obtaining 2 cropped images from original image is shown in figure below. In above figure δ will be calculated as follows

$$\delta = |T - \sum_{i=1}^{x}\sum_{j=1}^{y}(p_{ij})/xy| \times 2$$

here T is Osut's throshold, x and y are rows and columns of image and p is pixel value at the specific location. In case if δ is less then 10 we make it 16.

After forming 2 cropped images we have to obtain segmented image. Segmented image is achieved by subtracting both cropped images in following manner.

$$g(x,y) = \begin{cases} 1, & \text{if } ci_1(x,y) - ci_2(x,y) >= T1/4 \\ 0, & \text{if } ci_1(x,y) - ci_2(x,y) < T1/4 \end{cases}$$

Here T1 is calculated as $\frac{T+255}{2}$. and ci_1, ci_2 are cropped images. Figure 2 show result segmented image obtained from 5.2 cropped image.

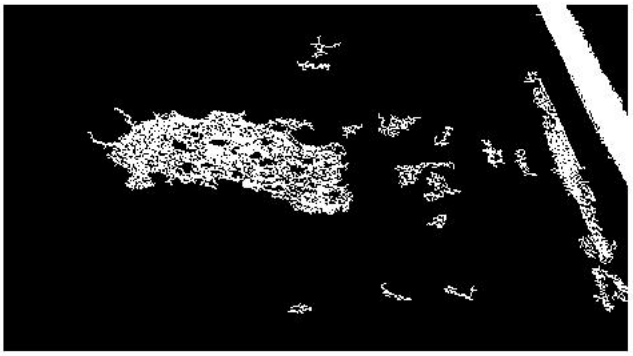


FIGURE 5.2: Segmented image.

5.2 Shape Extraction

As we can see image segmentation give us candidate region for pothole with some small and linear shapes. So, we remove linear shapes and shapes smaller then δ. To identifying and removing linear shapes I am using eccentricity which will tell us how much elliptical a shape is. Linear shapes will be heaving eccentricity near 0 and perfect circle will be having eccentricity of 1.

To extract exact shape of pothole I am using spectral clustering. Here traditional clustering algorithms are don't give very good result because they generally use very simple matrix to calculate distance between points and cluster them.

For our purpose I am using normalized spectral clustering algorithm. Spectral clustering algorithm will require a affinity matrix which will represent affinity between pixels as input and then perform clustering. Generally affinity matrix is defined as very similar as Gaussian kernel e^{-d^2/σ^2}. where d is Euclidean distance between pixels and σ is scalar factor. basis steps of algorithm are presented below To use spectral clustering to extract shape of pothole first calculate histogram $h \in Z^{256\times 2}$ of input image and perform spectral clustering on it. Number of clusters are calculated automatically by eigenvalues λ

$$\alpha = (\frac{1}{n-1}\sum_{i=1}^{n}(\lambda_{ii} - \frac{1}{n}\sum_{j=1}^{n}\lambda_{jj})^2)^{\frac{1}{2}}$$

Algorithm 1: Spectral clustering

Input: k : number of cluster we need to form

1 Calculate affinity matrix $S \in R^{n \times n}$ from input dataset $X = x_1, x_2, ..., x_n \in R$ defined as $s_{ij} = e^{\frac{-d^2(x_i, x_j)}{\sigma_2}}$ where d is Euclidean distance between x_i and x_j.

2 Compute degree matrix $D = diag(d_i)$ where $d_i = \sum_{j=1}^{n} s_{ij}$

3 Compute the normalized Laplacian matrix $L = D^{\frac{-1}{2}} \times L \times D^{\frac{1}{2}}$, where L is Laplacian matrix defined as L = D - S

4 perform eigen value decomposition as $L \times v = \lambda \times v$, where $v \in R^{n \times n}$ are eigen vectors and $\lambda \in R^{n \times n}$ is eigen values.

5 define new matrix $U \in R^{n \times k}$ from eigen vector matrix. $u_{ij} = v_{im}$ where i = 1,2,3,...,n, j = 1,2,3,...,k and m is k largest eigen, last k columns of v.

6 construct normalized matrix Y from U as follows $y_{ij} = \frac{u_{ij}}{(\sum_{l=1}^{k} u_{il}^2)^{\frac{1}{2}}}$ where i = 1,2,3,...,n;j,l = 1,2,3,...,k

7 cluster n points $y_i \in R^k$ i = 1,2,3,...,n with K-means clustering into k clusters.

After clustering histogram into k clusters we have to apply clustering

Algorithm 2: Finding number of clusters

Input: α;λ-eigenvalues

Output: k: Number of clusters

for $i <= n; i = i + 1$ **do**
if $\lambda_{ii} > \alpha$
k = k+1
end if
end for

to the original image. Figure 5.3 show final image after applying clustering result to original image. As you can see one color has more density in pothole region.

5.3 Identification and Extraction of Candidate Regions

After extraction of clustering results we can see that one color in pothole region have more intensity. Next step is to extracting pothole region for that we will select seed which will mark area for the

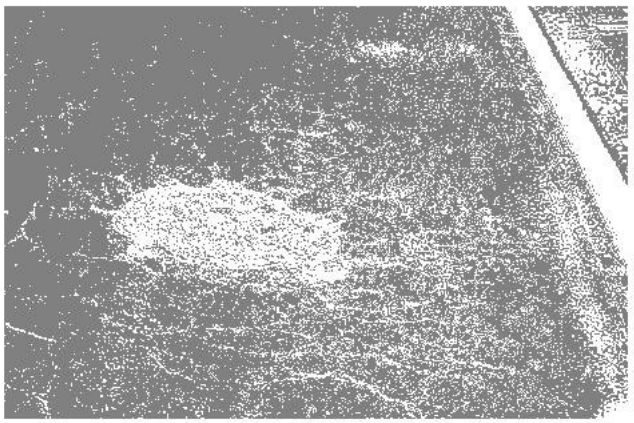


FIGURE 5.3: Resulting clustered image.

identification of pothole region. Algorithm below will explain seed selection procedure. Precision of pothole region is based on seed selection. More number of seed points increase algorithm processing time but give more precise pothole area. So, we have to select optimal number of seeds so that we can reduce algorithm processing time and have good precision. In our case I've selected every 50th point as seed which will provide good result and fast execution. We

Algorithm 3: Seed selection from segemented image for clustered image

Input: iImage: segmented image, cImage: clustered image

1 (rows, columns) : Find all the points where pixel value of iImage is 1

2 **for i ¡= size of number of columns of iImage; i = i+50 do**

k = k+1

row = rows(i)

col = columns(i)

seedPoints(i) = (row,col)

colors(k) = cImage(row,col)

end for

will first extract vertical area and then horizontal area with help of vertical. For vertical area extraction we first find top vertical point for each seed in clustered image which has same color value as that seed point. Also, all connecting points of same color. Find bottom point using same procedure. In this way we can define vertical region of pothole

FIGURE 5.4: Pothole identification with seeds points.

Next step is to find horizontal area of pothole. For all seeds connecting points between top and bottom point in previous step are used as new seed points for horizontal area extraction. In this setp, first find most left point for each seed in same way as above then right most point. In this way we've defined horinzontal area for pothole region.

Complete pothole area extraction by plotting vertical and horizontal line between top to bottom and left to right points. Result image will be as shown in figure 5.4

5.4 Classifying Candidate Regions using SVM

Our algorithm can accurately extract any rough surface and discard plain surface. But not all rough surface on the road is a pothole it can be other other objects like someone is walking or a car etc. So, it will identify them as pothole. To remove those false positive results we are using SVM on top of it. SVM is trained for separating pothole images from other images like people standing, cars and other common rough surfaces which are not potholes. For feature extraction I am using SFTA as it has shown good accuracy in texture detection. As both potholes and other rough surfaces will be heaving very similar texture so, we have to be very careful while selecting

FIGURE 5.5: Pothole identification with SVM.

dataset for training SVM as it can lead to decrease in accuracy. I've kept only 400 images in dataset with increase in size of training dataset its accuracy decrease. Result of complete pothole algorithm can be seen in figure 5.5. Here black rectangles are pothole which are discarded by SVM and white are approved by SVM so that will be final result. As we can see rough surface which is car is discarded by SVM and pothole has been approved.

Chapter 6

Ground Plane Estimation

Our pothole detection algorithm is based on assumption that pothole will be rough surface and other part of road will be smooth. But all rough surfaces are not potholes always due to which it will give false positive results. To remove those false positive we have used SVM on top of it. But texture of potholes and other rough regions are very similar which affects SVM's accuracy. To make further improvement in pothole detection I've implemented ground plane detection using Depth sensor. With ground plain detection we can discard all the false positive which are above ground. We are using Intel Depth sensor for depth detection. It will output a RGB image and a depth image. Pixel value of depth image will be directly proportional to depth of that point.

6.1 Method

In our case depth sensor will be located with fix angle(pitch angle) with vertical plane. The sensor pitch angle causes allocation of more pixels for the closer locations of scene then the further part. So that linear distance from the sensor is projected on the depth map as a rational function. From plot of depth values we can observe that depth value of any column on the ground plane is increasing exponentially from bottom to top as shown in figure 6.1.

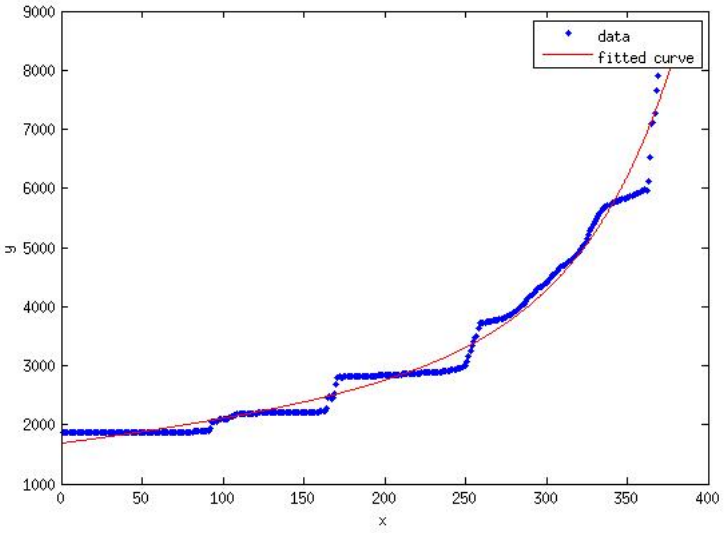


FIGURE 6.1: Fitted curve vs Original data

Thus we can fit a exponential curve to any vertical line of the depth map. The curve that fit best is sum of two exponential functions.

$$f(x) = ae^{bx} + ce^{dx}$$

where f(x) is pixel depth value and x is row number of that pixel in depth image. The coefficients a,b,c and d depends upon pitch angle, height and other parameters of depth sensor.

I am using least square fitting method to obtain values of those coefficient using a sample image taken with certain pitch angle with given height. Once we obtain coefficient from a sample image those coefficients will be same for all images taken with same settings of depth sensor. Then it is possible to obtain curve of ground plane.

To detect ground plane on new depth image we can simply compare depth of any given pixel with depth of that pixel obtained from ground plan curve. Any value which is more then ground plan is below ground and less then ground plane will lay above the ground. Hence we can compare absolute difference against a predefined threshold T and mark pixel on ground if its difference is less then T. In figure above x-axis represent row number of pixel from bottom and y-axis is depth. Blue points represent actual depth values of ground plane and red is curve fitted. We can see that pixel

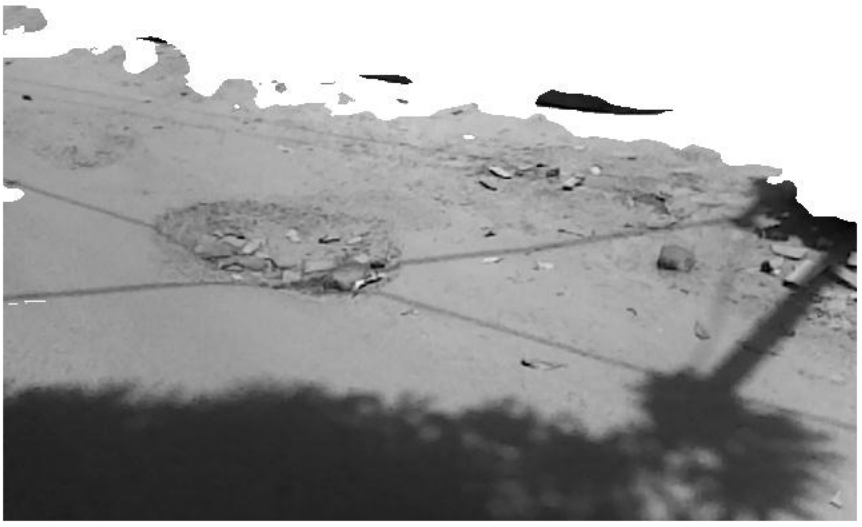


FIGURE 6.2: Ground plane detection. White region is above or below ground plane.

values of ground plan fix pretty good in above exponential function. Result of ground plane detection is shown in figure 6.2.

6.2 Integration with Pothole Detection

To improve accuracy of pothole detection we can remove those potholes which are above ground plane. We can compare each pixel of pothole region and if that lie above ground plane then discard it. As due to error in depth sensor sometime depth values can be inaccurate. So, for our purpose I will discard a pothole if its 50% of the region above ground. It will overcome error in sensor values and give good results.

Chapter 7

Pothole Software Only Implementation

7.1 Dataset

Dataset for pothole was taken from and around IIT. Those dataset include manhole, broken road, pothole on footpath etc.

For SVM I am re-sizing cropped pothole image to 80x80 window and extracting features using SFTA feature extraction for training and testing as we discussed in Chapter 3 for texture detection. As texture of pothole and other rough surfaces are very similar So, SVM is showing good accuracy when its trained on less number of images.

7.2 MATLAB Implementation

The algorithm is implemented in MATLAB on Intel Core i5-3770 CPU @ 3.40GHz x 4 with 4GB RAM. As discussed in previous section the classifier is trained with 176 samples. After that they were tested for 100 images. The implementation results are shown as follows:

7.2.1 Accuracy

As we are using SVM on top of pothole detection algorithm and both the algorithms have different accuracy's and final accuracy will be more then without SVM algorithm. Table 7.1 show different accuracy's.

Algorithm	**Number of Samples**	**Accuracy**
SVM	100	78%
Pothole Detection(Without SVM)	100	75%
Pothole Detection(With SVM)	100	77%

Here all testing were done on 100 images taken outside IIT Delhi

7.3 OpenCV Implementation

Open Source Computer Vision (OpenCV) version 3.1 provide interface for SVM but its result were poor then MATLAB's SVM so, I used libsvm which gave accuracy's same as MATLAB. The algorithm is implemented on Intel Core i5-4210U CPU @ 1.70GHz x 4 with 4GB RAM. Functions like removing small areas and eccentricity calculation are not available in standard OpenCV so, I used libopencvblobs and eigen.

7.3.1 Accuracy

Finally all the accuracy's are same as MATLAB as shown in table 7.3

Algorithm	**Number of Samples**	**Accuracy**
SVM	100	78%
Pothole Detection(Without SVM)	100	75%
Pothole Detection(With SVM)	100	77%

7.3.2 Performance

Time taken by different algorithms is showing in table 7.2

Algorithm	Number of Samples	Avg. Time(sec)	Max. Time(sec)	min. Time(sec)
SVM	100	0.2	0.2	0.2
Pothole Detection(Without SVM)	100	8.3	7.2	10.5
Pothole Detection(With SVM)	100	8.4	7.3	10.6

7.4 Cross Compilation on ZedBoard

7.4.1 Description of ZedBoard

ZedBoard (Zynq Evaluation and Development Board) is an excellent development kit based on Zynq All Programmable SoC (AP SoC). It is a collaboration of three vendors Xilinx (Zynq AP SoC), Digilent (board manufacturer) and Avnet (distributor). This product integrates Xilinx programmable logi(PL) and a feature rich dual-core ARMCortex A9 MPCore based processing system (PS) in a single device having high performance and low power process technology [13].

7.4.2 Cross Compilation of OpenCV and Xillinux on zedboard

We used Xillinux which is arm based linux distro for all of our testing on zedboard. In order to generate binaries for arm we first have to cross compile all the dependencies which include OpenCV, eigen2, libopencvblobs and clustering library. I've generated static library for all dependencies then statically linked to them. Once we generate binary we can just transfer it to zedboard sdcard and run it.

7.4.3 Accuracy

All the accuracy's are same as Desktop as shown in table 7.3

Algorithm	**Number of Samples**	**Accuracy**
SVM	100	78%
Pothole Detection(Without SVM)	100	75%
Pothole Detection(With SVM)	100	77%

7.4.4 Performance

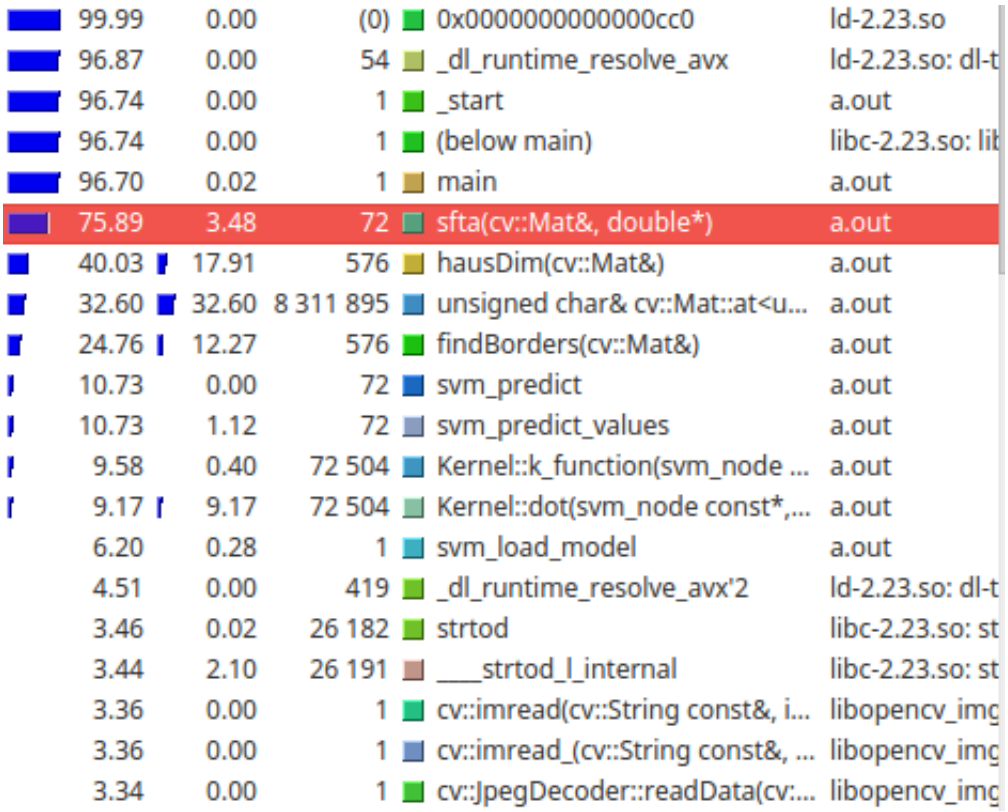
Time taken by different algorithms is showing in table 7.2

Algorithm	Number of Samples	Avg. Time(sec)	Max. Time(sec)	min. Time(sec)
SVM	100	1.8	1.8	1.8
Pothole Detection(Without SVM)	100	80	65	98
Pothole Detection(With SVM)	100	81	65	99

7.5 Profiling and Hardware Acceleration Hotspots

As we can see that time taken by pothole detection algorithm is more then 1min. This can't be much useful in real scenario because we need to give user warning before he reach to pothole. This time taken can be reduced by moving costly computation from software to hardware. For that purpose we have to first detect the part of algorithm which take most of the time by profiling. Profiling results of Texture and pothole detection are as follows:

7.5.1 Texture Detection

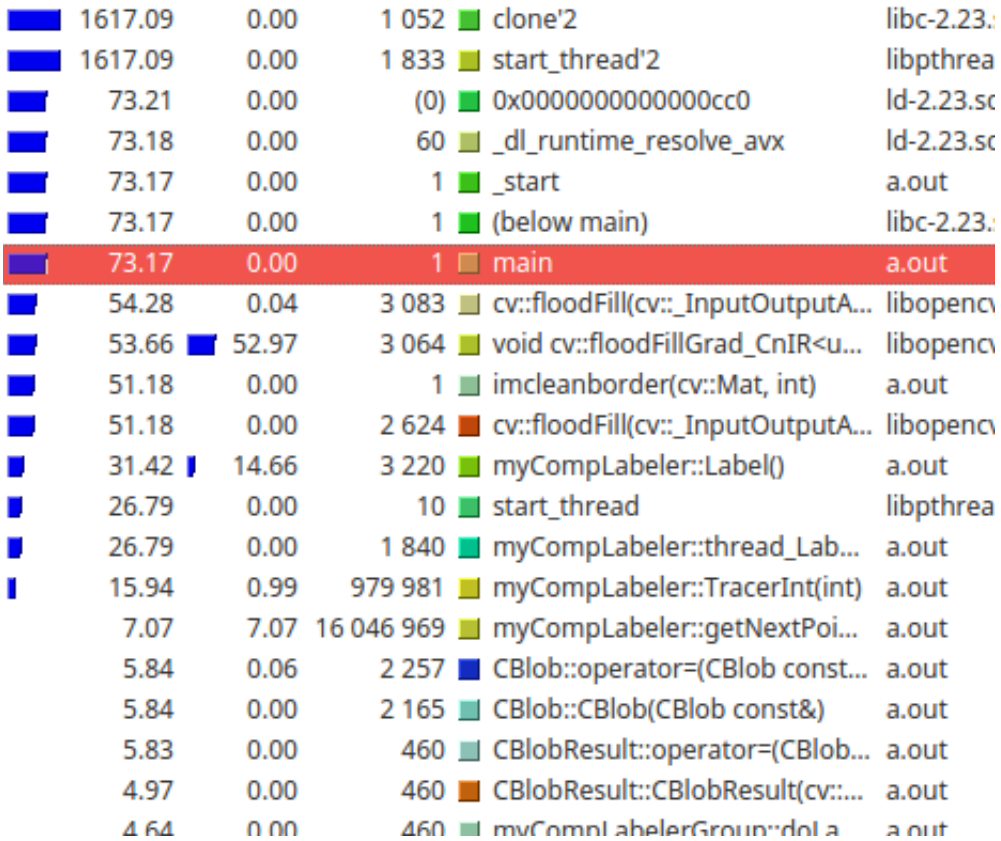


99.99	0.00	(0)	0x0000000000000cc0	ld-2.23.so
96.87	0.00	54	_dl_runtime_resolve_avx	ld-2.23.so: dl-t
96.74	0.00	1	_start	a.out
96.74	0.00	1	(below main)	libc-2.23.so: lib
96.70	0.02	1	main	a.out
75.89	3.48	72	sfta(cv::Mat&, double*)	a.out
40.03	17.91	576	hausDim(cv::Mat&)	a.out
32.60	32.60	8 311 895	unsigned char& cv::Mat::at<u...	a.out
24.76	12.27	576	findBorders(cv::Mat&)	a.out
10.73	0.00	72	svm_predict	a.out
10.73	1.12	72	svm_predict_values	a.out
9.58	0.40	72 504	Kernel::k_function(svm_node ...	a.out
9.17	9.17	72 504	Kernel::dot(svm_node const*,...	a.out
6.20	0.28	1	svm_load_model	a.out
4.51	0.00	419	_dl_runtime_resolve_avx'2	ld-2.23.so: dl-t
3.46	0.02	26 182	strtod	libc-2.23.so: st
3.44	2.10	26 191	___strtod_l_internal	libc-2.23.so: st
3.36	0.00	1	cv::imread(cv::String const&, i...	libopencv_img
3.36	0.00	1	cv::imread_(cv::String const&, ...	libopencv_img
3.34	0.00	1	cv::JpegDecoder::readData(cv:...	libopencv_img

FIGURE 7.1: Texture detection profiling

Here from above figure we can see that most of that time is taken by SFTA algorithm. In SFTA hausDim function is taking 40% of the time. So, it is a good candidate for hardware acceleration.

7.5.2 Pothole Detection



1617.09	0.00	1 052	clone'2	libc-2.23.
1617.09	0.00	1 833	start_thread'2	libpthrea
73.21	0.00	(0)	0x0000000000000cc0	ld-2.23.so
73.18	0.00	60	_dl_runtime_resolve_avx	ld-2.23.so
73.17	0.00	1	_start	a.out
73.17	0.00	1	(below main)	libc-2.23.
73.17	0.00	1	main	a.out
54.28	0.04	3 083	cv::floodFill(cv::_InputOutputA...	libopencv
53.66	52.97	3 064	void cv::floodFillGrad_CnIR<u...	libopencv
51.18	0.00	1	imcleanborder(cv::Mat, int)	a.out
51.18	0.00	2 624	cv::floodFill(cv::_InputOutputA...	libopencv
31.42	14.66	3 220	myCompLabeler::Label()	a.out
26.79	0.00	10	start_thread	libpthrea
26.79	0.00	1 840	myCompLabeler::thread_Lab...	a.out
15.94	0.99	979 981	myCompLabeler::TracerInt(int)	a.out
7.07	7.07	16 046 969	myCompLabeler::getNextPoi...	a.out
5.84	0.06	2 257	CBlob::operator=(CBlob const...	a.out
5.84	0.00	2 165	CBlob::CBlob(CBlob const&)	a.out
5.83	0.00	460	CBlobResult::operator=(CBlob...	a.out
4.97	0.00	460	CBlobResult::CBlobResult(cv::...	a.out
4.64	0.00	460	myCompLabelerGroup::doLa	a.out

FIGURE 7.2: Pothole detection profiling

In case of pothole detection floodFill is taking most of the time. its used for finding connected components. So it's a good candidate for hardware acceleration.

Chapter 8

Conclusion and Future Work

8.1 Results & Conclusion

8.1.1 Texture Detection

Here are some of the results on sample dataset for texture detection

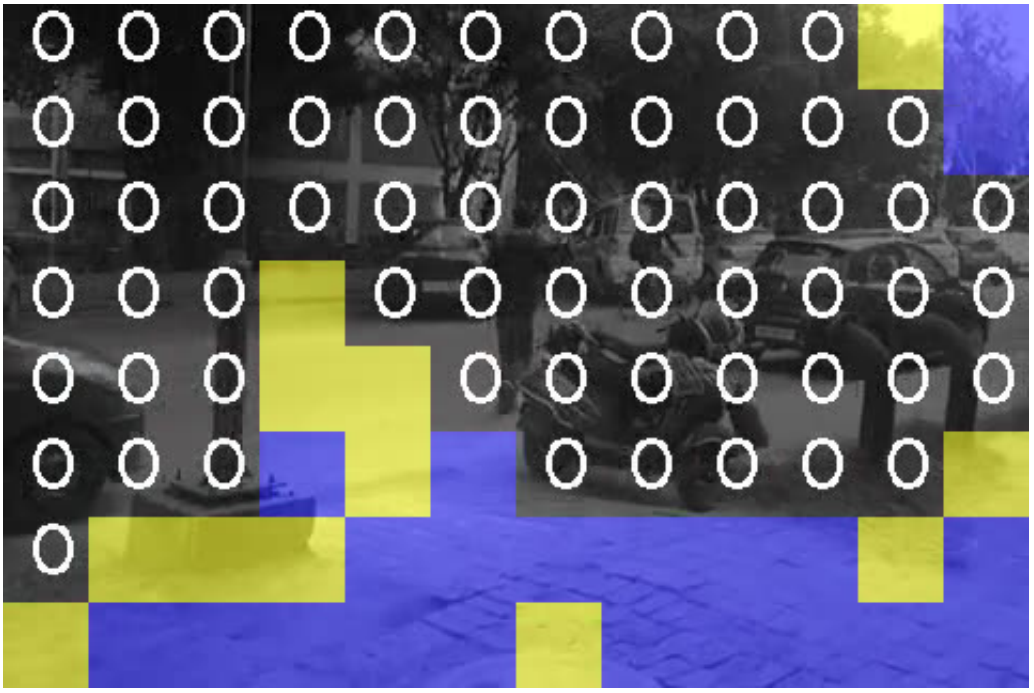


FIGURE 8.1: Texture Detection

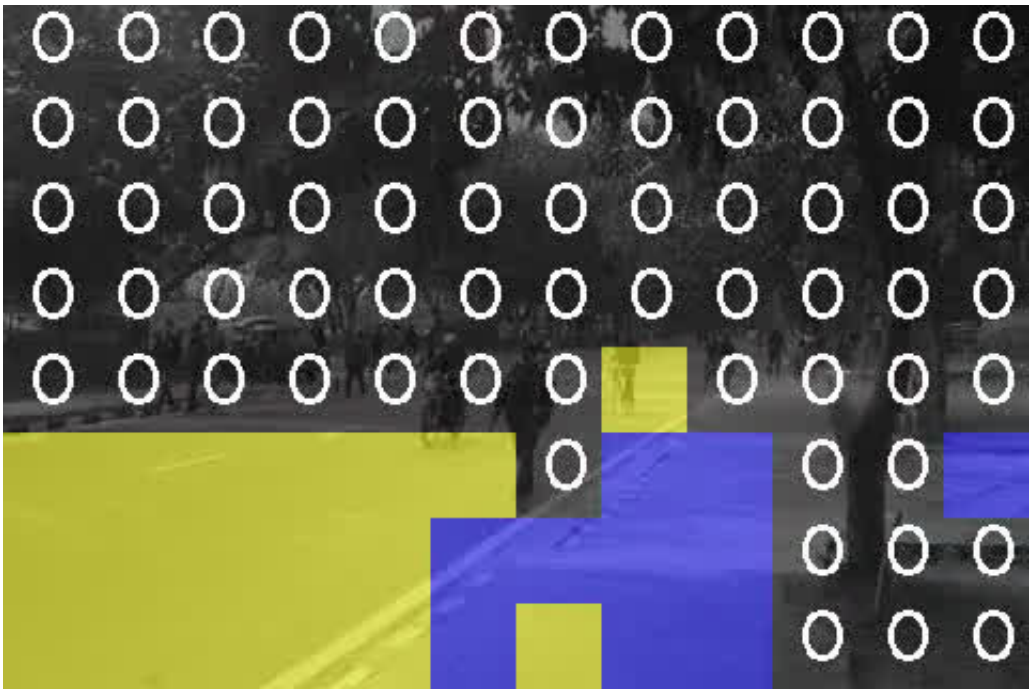


FIGURE 8.2: Texture Detection

8.1.2 Pothole Detection

FIGURE 8.3: Pothole Detection

FIGURE 8.4: Accuracy vs distance for 40x40 window

8.1.3 Ground Plane Detection

FIGURE 8.5: Original Image

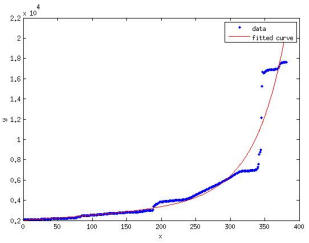


FIGURE 8.6: Ground Plane curve fitting

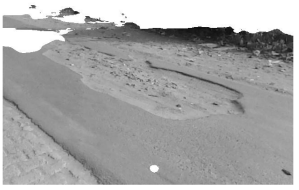


FIGURE 8.7: Ground Plane detected

FIGURE 8.8: Original Image

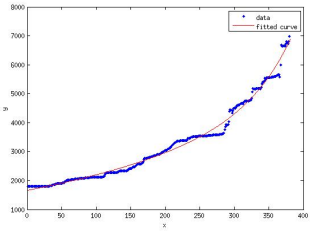


FIGURE 8.9: Ground Plane curve fitting

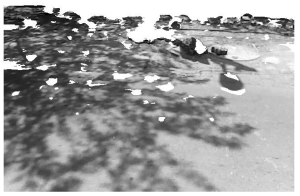


FIGURE 8.10: Ground Plane detected

8.2 Future Work

As Pothole detection is taking lots of time (more then 1 min) which is not much useful in real sinario and profiling results show the hotspots where there is possibility of increasing proformance by shifting some computation to hardware so, this could be possible future work.

References

[1] C.Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning.* 20(3):273297, 1995.

[2] Alceu Ferraz Costa, Gabriel Humpire-Mamani, and Agma Juci Machado Traina. *An Efficient Algorithm for Fractal Analysis of Textures*, 2012.

[3] Caffe: Yangqing Jia and Evan Shelhamer
http://caffe.berkeleyvision.org/tutorial/

[4] EMIR BUZA, SAMIR OMANOVIC, ALVIN HUSEINOVIC, *Pothole Detection with Image Processing and Spectral Clustering*, 2013

[5] Do gan Krcal and F. Boray Te, *Ground Plane Detection Using an RGB-D Sensor*, 2014

[6] Xilinx. Inc. UG585 Zynq - 7000 All Programmable SoC Technical Ref- erence Manual, 2015