

MOVING OBJECT TRACKING AND VELOCITY ESTIMATION USING BACKGROUND SUBTRACTION ALGORITHM

By

N.RAVALIKA 16SS1A0432

P.PRIYANKA 16SS1A0438

MD.RIYAZUDDIN 16SS1A0430

Under the Guidance of

Dr.Y.RAGHAVENDER RAO



Department of Electronics & Communication

Engineering

Jawaharlal Nehru Technological University Hyderabad

College of Engineering Sultanpur

PULKAL(M),SANGAREDDY(D)-502293 TELANGANA

2019-2020

MOVING OBJECT TRACKING AND VELOCITY ESTIMATION USING BACKGROUND SUBTRACTION ALGORITHM

PROJECT REPORT

**SUBMITTED IN THE PARTIAL FULFILMENTS OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF TECHNOLOGY**

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

BY

N.RAVALIKA (16SS1A0432)

P.PRIYANKA (16SS1A0438)

MD.RIYAZUDDIN (16SS1A0430)



Department of Electronics & Communication

Engineering

Jawaharlal Nehru Technological University Hyderabad

College of Engineering Sultanpur

PULKAL(M),SANGAREDDY(D)-502293 TELANGANA

2019-2020

JNTUH COLLEGE OF ENGINEERING SULTANPUR

PULKAL(MANDAL),SANGAREDDY(DISTRICT),TELANGANA



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING CERTIFICATE

This is to certify that the project work entitled MOVING OBJECT TRACKING AND VELOCITY ESTIMATION USING BACKGROUND SUBTRACTION ALGORITHM is a bonafied work carried out by N.Ravalika ,P.Priyanka ,MD.Riyazuddin-bearing Roll no. 16SS1A0432 ,16SS1A0438 ,16SS1A0430 in partial fulfillment of the requirements for the degree of BACHELOR OF TECHNOLOGY in ELECTRONICS AND COMMUNICATION ENGINEERING jawaharlal Nehru Technology University, Hyderabad during the academic year 2019-20.

The results embodied in this report have not been submitted to any other University or Institution for the award of any degree or diploma.

DR.Y.RAGHAVENDER RAO

Project guide

DR.Y.RAGHAVENDER RAO

Head of the Department

ACKNOWLEDGEMENTS

We take this opportunity to thank our Principal Dr B. BALU NAIK sir who was kind enough and encouraged us throughout our project. We would like to thank him to make atmosphere around us easy to work. We would also like to take this privilege to thank our Head of the department Dr Y. RAGHAVENDER RAO sir who made the atmosphere so easy to work. We shall always be indebted to them. We would like to thank him for his encouragement, constant inspiration and support. Their invaluable guidance and advices helped us in course of our project. We also thank our project guiders Sri V.RAJANESH sir for being patient and enthusiastic at the same time. It has been a great privilege to work under him. We would also like to take this opportunity to thank all other respected teachers of that particular department for their valuable time and suggestions which made us complete our project within scheduled time and also thank them to show right path at the time of necessity.

By

N.RAVALIKA 16SS1A0432

P.PRIYANKA 16SS1A0438

MD.RIYAZUDDIN 16SS1A0430

Abstract:

Currently, both the market and the academic communities have required applications based on image and video processing with several real-time constraints. On the other hand, detection of moving objects is a very important task in mobile robotics and surveillance applications. The project proposes an efficient motion detection and object velocity determination based on background subtraction using dynamic threshold and morphological process. These methods are used effectively for object detection and most of previous methods depend on the assumption that the background is static over short time periods. In dynamic threshold based object detection, morphological process and filtering also used effectively for unwanted pixel removal from the background. Then object is detected in a sequence of frames with respect to the frame rate that the video is recorded. A simulation result proves that the proposed method is effective for background subtraction for object detection compared to several competitive methods proposed in the literature and determination. The method is able to identify moving persons, track them and provide unique tag for the tracked persons. The background subtraction algorithm can also be used to detect multiple objects. The algorithms developed can also be used for other applications (real time, object classification, etc.).

Keywords: Object tracking, Frame separation, Background Subtraction Algorithm, Object detection

INDEX

| CONTENTS | page no |
|---|--------------|
| CERTIFICATE..... | (i) |
| ACKNOWLEDGEMENT..... | (ii) |
| ABSTRACT..... | (iii) |
| LIST OF FIGURES..... | (vi) |
| | |
| CHAPTER 1 : INTRODUCTION | 1-3 |
| 1.1 Objective | |
| CHAPTER 2 : LITERATURE REVIEW | 4-7 |
| CHAPTER 3: DIGITAL IMAGE PROCESSING | 8-31 |
| 3.1 Basics of Image preprocessing | |
| 3.2 Image Segmentation | |
| 3.3 Edge Detection | |
| 3.4 Segmentation using Thresholding | |
| 3.5 Basic Global Thresholding Technique | |
| 3.6 Basic Adaptive Thresholding Technique | |
| 3.7 Optical Thresholding Techniques | |
| 3.8 Region based Segmentation | |
| 3.8.1 Segmentation by region growing | |
| 3.8.2 Segmentation by Morphological watershed | |
| CHAPTER 4 : EXISTING METHODS | 32-37 |
| 4.1 Background Subtraction methods | |
| 4.1.1 Background Modeling | |
| CHAPTER 5 : PROPOSED METHOD | 38-45 |
| 5.1 Block diagram of the proposed method | |
| 5.1.1 Input video | |
| 5.1.2 Gray scale conversion | |
| 5.1.3 Detection of object | |
| 5.1.4 Bounding box | |
| 5.1.5 Calculating speed of vehicle | |
| 5.1.6 Detailed information of vehicles | |
| | |
| CHAPTER 6 : SIMULATION RESULTS | 46-50 |
| 6.1 Fig-Overall structure of gui for proposed methodology | |
| 6.2 Fig-Selected input video for proposed work | |
| 6.3 Fig-Detected vehicles from input video using Background subtraction | |
| 6.4 Fig-Speed calculation for vehicles from Detected boundary boxes | |

6.5 Fig-Total count for Detected vehicles and volume calculation

CHAPTER 7 : CONCLUSION

51

7.1 Conclusion and Future scope

REFERENCES..... 52-53

APPENDIX A.....

APPENDIX B.....

APPENDIX C.....

LIST OF FIGURES

| | Pg no |
|--|--------------|
| Fig 3.2-Image segmentation | 10 |
| Fig 3.2.1- Line masks | 11 |
| Fig 3.3-Different edge profiles | 13 |
| Fig 3.3.1-Homogeneity operator | 14 |
| Fig 3.3.2-Homogeneity operator | 15 |
| Fig 3.3.3-Cross section of LoG with various s | 19 |
| Fig 3.3.4-LoG vs DoG function | 20 |
| Fig 3.5.1 Original image | 23 |
| Fig 3.5.2 Image histogram | 23 |
| Fig 3.5.3 Result of global thresholding with T midway b/w max and min gray levels | 23 |
| Fig 3.6.3 Result of segmentation with threshold estimated by iteration | 24 |
| Fig 3.7.1 Gray level probability density function of 2 regions in an img | 25 |
| Fig 3.8-Img tree slit merge | 27 |
| Fig 3.8.1-Start of growing of a region | 28 |
| Fig 3.8.2-Growing process after few iterations | 29 |
| Fig 3.8.2.1-Two stages of flooding | 30 |
| Fig 4.1-Classification of Background subtraction | 32 |
| Fig 5.1-Block diag of proposed method | 38 |
| Fig 5.1.2-Color image and Gray Scale image | 39 |
| Fig 5.1.4-Bounding box | 43 |
| Fig 6.1 -Overall structure of gui for proposed methodology | 46 |
| Fig 6.2 -Selected input video for proposed work | 47 |
| Fig 6.3 -Detected vehicles from input video using Background Subtraction | 48 |
| Fig 6.4 -Speed calculation for vehicles from Detected boundary boxes | 49 |
| Fig 6.5 -Total count for Detected vehicles and volume calculation | 50 |

CHAPTER-1

INTRODUCTION

1.1 Objective

The main objective is to detect moving object, calculation of distance and velocity of the moving object. Extraction of objects using the features is known as object detection. Every object has a specific feature based on its dimensions. Applying feature extraction algorithm, the object in each frame can be pointed out. We propose an implementation of image subtraction algorithm to detect a moving object. Detection of object motion has been implemented and the velocity is been calculated. The algorithm is processed with matlab software and we calculated the distance, frame per time, velocity. This can be used in speed measurements in traffic signal monitoring system. Background modeling is used to adapt the change in the dynamic environment. It is very challenging for every background subtraction based method. Traditional method utilizes temporal differencing or optic flow based method.

Video change detection or Background Subtraction (BS) is one of the most widely studied topics in computer vision. It is a basic pre-processing step in video processing and therefore has numerous applications including video surveillance, traffic monitoring, human detection, gesture recognition, etc. Typically, a BS process produces a foreground (FG) binary mask given an input image and a background (BG) model. BS is a difficult problem because of the diversity in background scenes and the changes originated from the camera itself. Scene variations can be in many forms such as, to name just a few, dynamic background, illumination changes, intermittent object motion, shadows, highlights, camouflage as well as a multitude of environmental conditions like rain, snow, and change in sunlight. Likewise, the changes linked to camera can be due to auto-iris, camera jitter, sensor noise and pan-tilt-zoom.

No single technique exists that is able to simultaneously handle all key challenges and produce satisfactory results. In this paper, we propose a BS system that is robust against various challenges associated with real world videos. The proposed approach uses a Background Model Bank (BMB) that comprises of multiple Background (BG) models of the scene. To separate foreground pixels from changing background pixels caused by scene variations or camera itself, we apply Mega-Pixel (MP) based spatial denoising to pixel level probability estimates on different color spaces to obtain multiple Foreground (FG) masks. They are then combined to produce a final output FG mask.

The major contribution of this paper is a universal background subtraction system called Multimode Background Subtraction (MBS) with following major innovations: Background Model Bank (BMB), model update mechanism, MP-based spatial denoising of pixel-based probability estimates, fusion of multiple binary masks, and use of multiple color spaces for BS process. Preliminary results of using our system to handle illumination changes and camera movements were presented in [1] and [2] respectively.

Improvements upon these prior works include:

- A detailed analysis of the fusion of appropriate color spaces for BS,
- A novel model update mechanism, and
- A novel MP-based spatial denoising and
- A dynamic model selection scheme that significantly reduces the number of parameters and improve computational speed.

BS is well-researched topics in computer vision, therefore, we demonstrate the performance of MBS by providing a comprehensive comparison with 15 other state-of-the-art BS algorithms on a set of publicly-available challenging sequences across

different categories, totalling to 56 video sets. To avoid bias in our evaluations, we have adopted the same sets of metrics as recommended by the CDnet 2014. The

extensive evaluation of our system demonstrates better foreground segmentation and superiority of our system in comparison with existing state-of-the-art approaches.

CHAPTER 2

LITERATURE REVIEW

In (Ghobadi et al., 2008) the foreground of 2D/3D videos is extracted simply by defining a volume of interest and this is used for hand tracking as well as gesture recognition. Harville et al. applied the standard approach of background modeling by Gaussian mixtures, see e.g. (Stauffer and Grimson, 1999), to color and depth videos in (Harville et al., 2001). They are using full sized depth images so that there is no need to handle the different resolutions. In (Bianchi et al., 2009) a rather simple approach to foreground segmentation for 2D/3D videos that is based on region growing and refrains from modeling the background is evaluated, whereas in (Leens et al., 2009) a simple pixel-based background modeling method called ViBe is used for color and depth dimensions separately and the resulting foreground masks are fused with the help of binary image operations such as erosion and dilation. A more elaborate method of fusing color and depth is bilateral filtering, which is used e.g. in (Crabb et al., 2008). Here the preliminary foreground is produced by a dividing plane in space and a bilateral filter is applied to gain the final results.

The method is demonstrated on depth augmented alpha matting, which is also the focus of the paper (Wang et al., 2007). In (Schuon et al., 2008) the ability of bilateral filtering to deal with geometric objects is demonstrated and in (Chan et al., 2008) a variant designed to handle noise and invalid measurements is presented. The problem of fusing the depth and color dimensions and handling their different nature is also discussed in the course of depth upscaling. To that end a cost function or volume is defined in (Yang et al., 2007), that describes the cost of in theory all possible refinements of the depth for a color pixel. Again a bilateral filter is applied to this volume and after sub-pixel refinement a proposed depth is gained. The optimization is performed iteratively to achieve the final depth map. The incorporation of a second view is also discussed.

In (Bartczak and Koch, 2009) a similar method using multiply views was presented. An approach working with one color image and multiple depth images is described in (Rajagopalan et al., 2008). Here the data fusion is formulated in a statistical manner and modeled using Markov Random Fields on which an energy minimization method is applied. Another advanced method to combine depth and color information was introduced in (Lindner et al., 2008). It is based on edge preserving biquadratic upscaling and performs a special treatment of invalid depth measurements by refinements of the depth for a color pixel. Again a bilateral filter is applied to this volume and after sub-pixel refinement a proposed depth is gained. The optimization is performed iteratively to achieve the final depth map. The incorporation of a second view is also discussed.

In (Bartczak and Koch, 2009) a similar method using multiply views was presented. An approach working with one color image and multiple depth images is described in (Rajagopalan et al., 2008). Here the data fusion is formulated in a statistical manner and modeled using Markov Random Fields on which an energy minimization method is applied. Another advanced method to combine depth and color information was introduced in (Lindner et al., 2008). It is based on edge preserving biquadratic upscaling and performs a special treatment of invalid depth measurements.

R.H. Evangelio presents splitting Gaussians in mixture model (SGMM) for background extraction. Gaussian mixture models extensively used in the domain of surveillance. Due to low memory requirement this model used in the real time application. Split and merge algorithm provides the solution if main mode stretches and that causes weaker distribution problem. SGMM define criteria of selection of modes for the case of background subtraction. SGMM provides better background models in terms of low processing time and low memory requirements; therefore it is appealing in surveillance domain.

L. Maddalena and A. Pestrosino present Self Organizing Background subtraction (SOBS) for detection of moving object based on neural background model. Such model generate self-organizing model automatically without prior knowledge about involved pattern. This adaptive model background extraction with scene containing gradual illumination variation, moving backgrounds and camouflage can include into moving object with background model shadows cast and achieves detection of different types of video taken by stationary camera. The introduction of spatial coherence into the background model update procedures leads to the so-called SC-SOBS algorithm that gives further robustness against false detection. L. Maddalena and A. Pestrosino discuss extensive experimental results of SOBS and SC-SOBS based on change detection challenges.

A. Morde, X. Ma, S. Guler [3] discusses background model for change detection. Change detection or foreground and background segmentation, has been extensively used in image processing and computer vision, as it is fundamental step for extracting motion information from video frames. Chybyshchev probability inequality based background model present a robust real time background/foreground segmentation technique. Such model supported with peripheral and recurrent motion detectors. The system uses detection of moving object shadows, and feedback from higher level object tracking and object classification to refine the further segmentation accuracy. In this method present experimental result on wide range of test videos demonstrate the presented method with high performance with camera jitter, dynamic backgrounds, and thermal video as well as cast shadows.

Pixel based adaptive segmenter (PBAS) is one of the technique for detecting moving object in the video frame using background segmentation with feedback. Martin Hofmann, Philipp Tiefenbacher and Gerhard Rigoll discuss the novel method for detection of object i.e for foreground segmentation. This adaptive segmentation technique follows a nonparametric background modeling paradigm and the background is designed by recently observed pixel history

. The decision threshold plays an important role in pixel

based adaptive segmentation for taking foreground decision. In this method learning model is used to update background of the object. The learning parameter introduces dynamic controllers for each of dynamic per pixel state variables. Pixel based adaptive segmenter is state of the art methods.

Dirichlet process Gaussian mixture model is probabilistic model which assumes that all data points generated from a mixture of a finite number of Gaussian distribution having unknown parameter. There are different classes to execute a gaussian mixture model which corresponds to different strategies. Dirichlet process is a probability distribution whose domain is a set of probability distribution. This process is used in Bayesian inference to describe prior knowledge about distribution of random variables, according to this formulation random variables are distributed based on one or another particular distribution.

CHAPTER 3

DIGITAL IMAGE PROCESSING

3.1 Basic of Image Preprocessing :

A. Noise

Noise is any entity which is not of benefit to the purpose of image processing. The influence of noises on the image signal amplitude and phase is complexity. So how to smooth out noise and keep the details of image is the major tasks of the image filtering.

B. Noise Filter

We use the median filter in this paper. Median filter is a non-linear method for removing noise. Its basic idea is to use the median of the neighborhood pixel gray value instead of the gray value of pixel point. For the odd elements, the median refers to the size of the middle value after sorting; For even numbered elements, the median refers to the average size of the two middle values after sorting . Median filter as a result of this method is not dependent on the neighborhood with a lot of difference between typical values, which can remove impulse noise, salt and pepper noise at the same time retain the image edge details. In general the use of a median filters contain odd numbered points of the sliding window. Specific methods is determining a first odd-numbered pixel window W . Each pixels in window line by the size of the gray value, and use the location of the gray value between the image $f(x, y)$ gray value as a substitute for enhanced images $g(x, y)$,as follows: $g(x, y) = \text{Med}\{f(x - k, y - l), (k, l) \in W\}$ (1) W is the window size which is selected.

3.2 Image Segmentation :

In the Images research and application, Images are often only interested in certain parts. These parts are often referred to as goals or foreground (as other parts of the background). In order to identify and analyze the target in the image, we need to isolate them from the image.

The Image segmentation refers to the image is divided into regions, each with characteristics and to extract the target of interest in processes. The image segmentation used in this paper is threshold segmentation. To put it simply, the threshold of the gray-scale image segmentation is to identify a range in the image of the gray-scale threshold, and then all image pixels gray values are compared with the threshold and according to the results to the corresponding pixel is divided into two categories: the foreground of, or background.

$$g(x, y) = \begin{cases} 1 & f(x, y) > T \\ 0 & f(x, y) \leq T \end{cases}$$

The simplest case, the image after the single-threshold segmentation can be defined as: Threshold segmentation has two main steps:

- 1) Determine the threshold T .
- 2) Pixel value will be compared with the threshold value T .

In the above steps to determine the threshold value is the most critical step in partition. In the threshold selection, there is a best threshold based on different goals of image segmentation.

If we can determine an appropriate threshold, we can correct the image for segmentation.

Image Segmentation

Segmentation is a process of that divides the images into its regions or objects that have similar features or characteristics.

Some examples of image segmentation are

1. In automated inspection of electronic assemblies, presence or absence of specific objects can be determined by analyzing images.

2. Analyzing aerial photos to classify terrain into forests, water bodies etc.
3. Analyzing MRI and X-ray images in medicine for classify the body organs.

Some figures which show segmentation are



Fig 3.2 Image segmentation

Segmentation has no single standard procedure and it is very difficult in non-trivial images. The extent to which segmentation is carried out depends on the problem Specification. Segmentation algorithms are based on two properties of intensity values- discontinuity and similarity. First category is to partition an image based on the abrupt changes in the intensity and the second method is to partition the image into regions that are similar according to a set of predefined criteria.

In this report some of the methods for determining the discontinuity will be discussed and also other segmentation methods will be attempted. Three basic techniques for detecting the gray level discontinuities in a digital images points, lines and edges.


The other segmentation technique is the thresholding. It is based on the fact that different types of functions can be classified by using a range functions applied to the intensity value of image pixels. The main assumption of this technique is that different objects will have distinct frequency distribution and can be discriminated on the basis of the mean and standard deviation of each distribution. Segmentation on the third property is region processing. In this method an attempt is made to partition or group regions according to common image properties.

A very brief introduction to morphological segmentation will also be given. This method combines most of the positive attributes of the other image segmentation methods.

Segmentation using discontinuities :

Several techniques for detecting the three basic gray level discontinuities in a digital image are points, lines and edges. The most common way to look for discontinuities is by spatial filtering methods.

Point detection idea is to isolate a point which has gray level significantly different from its background.



| | | |
|-------|-------|-------|
| w_1 | w_2 | w_3 |
| w_4 | w_5 | w_6 |
| w_7 | w_8 | w_9 |

$w_1=w_2=w_3=w_4=w_6=w_7=w_8=w_9=-1$, $w_5 = 8$.

Response is $R = w_1z_1+w_2z_2+...+w_9z_9$, where z is the gray level of the pixel.

Based on the response calculated from the above equation we can find out the points desired. Line detection is next level of complexity to point detection and the lines could be vertical, horizontal or at ± 45 degree angle.

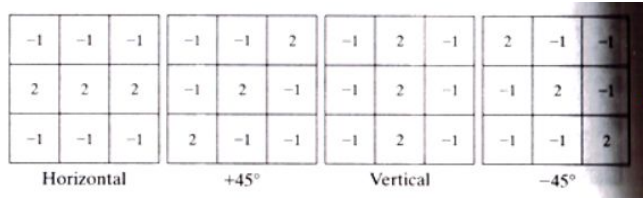


Fig 3.2 line masks

Responses are calculated for each of the mask above and based on the value we can detect if the lines and their orientation.

3.3 Edge Detection :

The edge is a regarded as the boundary between two objects (two dissimilar regions) or perhaps a boundary between light and shadow falling on a single surface.

To find the differences in pixel values between regions can be computed by considering gradients.

The edges of an image hold much information in that image. The edges tell where objects are, their shape and size, and something about their texture. An edge is where the intensity of an image moves from a low value to a high value or vice versa.

There are numerous applications for edge detection, which is often used for various special effects. Digital artists use it to create dazzling image outlines. The output of an edge detector can be added back to an original image to enhance the edges. Edge detection is often the first step in image segmentation. Image segmentation, a field of image analysis, is used to group pixels into regions to determine an image's composition. A common example of image segmentation is the "magic wand" tool in photo editing software. This tool allows the user to select a pixel in an image. The software then draws a border around the pixels of similar value. The user may select a pixel in a sky region and the magic wand would draw a border around the complete sky region in the image. The user may then edit the color of the sky without worrying about altering the color of the mountains or whatever else may be in the image.

Edge detection is also used in image registration. Image registration aligns two images that may have been acquired at separate times or from different sensors.

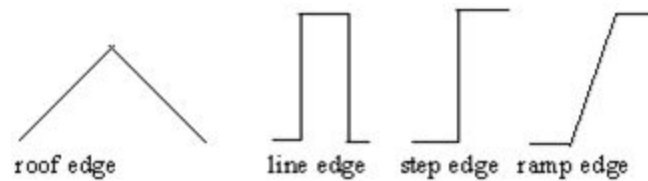


Figure.3.3 Different edge profiles.

There is an infinite number of edge orientations, widths and shapes (Figure e1). Some edges are straight while others are curved with varying radii. There are many edge detection techniques to go with all these edges, each having its own strengths. Some edge detectors may work well in one application and perform poorly in others. Sometimes it takes experimentation to determine what the best edge detection technique for an application is.

The simplest and quickest edge detectors determine the maximum value from a series of pixel subtractions. The homogeneity operator subtracts each 8 surrounding pixels from the center pixel of a 3 x 3 window as in Figure e2. The output of the operator is the maximum of the absolute value of each difference.

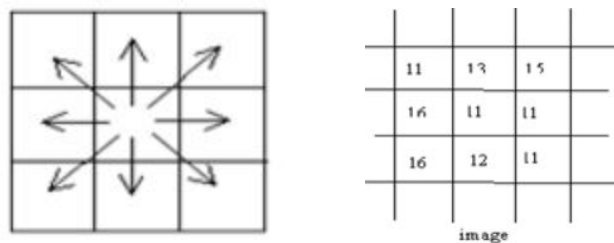


Fig 3.3.1 Homogeneity operator

$$\text{new pixel} = \text{maximum} \left\{ \frac{1}{2} 11-11\frac{1}{2}, \frac{1}{2} 11-13\frac{1}{2}, \frac{1}{2} 11-15\frac{1}{2}, \frac{1}{2} 11-16\frac{1}{2}, \frac{1}{2} 11-11\frac{1}{2}, \right. \\ \left. \frac{1}{2} 11-16\frac{1}{2}, \frac{1}{2} 11-12\frac{1}{2}, \frac{1}{2} 11-11\frac{1}{2} \right\} = 5$$

Figure.2 How the homogeneity operator works.

Similar to the homogeneity operator is the difference edge detector. It operates more quickly because it requires four subtractions per pixel as opposed to the eight needed by the homogeneity operator. The subtractions are upper left - lower right, middle left - middle right, lower left - upper right, and top middle - bottom middle (Figure e3).

| | | | |
|----|----|----|--|
| | | | |
| 11 | 13 | 15 | |
| 16 | 11 | 11 | |
| 16 | 12 | 11 | |
| | | | |

image

Fig 3.3.2 Homogeneity operator

New pixel = maximum $\{ \frac{1}{2} 11-11\frac{1}{2}, \frac{1}{2} 13-12\frac{1}{2}, \frac{1}{2} 15-16\frac{1}{2}, \frac{1}{2} 11-16\frac{1}{2} \} = 5$

Figure e3 how the difference operator works.

First order derivative for edge detection

If we are looking for any horizontal edges it would seem sensible to calculate the difference between one pixel value and the next pixel value, either up or down from the first (called the crack difference), i.e. assuming top left origin

$$H_c = y_difference(x, y) = value(x, y) - value(x, y+1)$$

In effect this is equivalent to convolving the image with a 2 x 1 template

Likewise

$$H_r = X_difference(x, y) = value(x, y) - value(x-1, y)$$

uses the template

$$\begin{matrix} -1 & 1 \end{matrix}$$

H_c and H_r are column and row detectors. Occasionally it is useful to plot both $X_difference$ and $Y_difference$, combining them to create the gradient magnitude (i.e. the strength of the edge).

Combining them by simply adding them could mean two edges canceling each other out (one positive, one negative), so it is better to sum absolute values (ignoring the sign) or sum the squares of them and then, possibly, take the square root of the result.

It is also to divide the Y_difference by the X_difference and identify a gradient direction (the angle of the edge between the regions)

$$\text{gradient_direction} = \tan^{-1} \left\{ \frac{Y_difference(x, y)}{X_difference(x, y)} \right\}$$

The amplitude can be determine by computing the sum vector of HcandHr

$$H(x, y) = \sqrt{H_r^2(x, y) + H_c^2(x, y)}$$

Sometimes for computational simplicity, the magnitude is computed as

$$H(x, y) = |H_r(x, y)| + |H_c(x, y)|$$

The edge orientation can be found by

$$\theta = \tan^{-1} \frac{H_c(x, y)}{H_r(x, y)}$$

In real image, the lines are rarely so well defined, more often the change between regions is gradual and noisy.

The following image represents a typical read edge. A large template is needed to average at the gradient over a number of pixels, rather than looking at two only

```

0 0 0 0 0 0 2 0 3 3
0 0 0 1 0 0 0 2 4 2
0 0 2 0 3 4 3 3 2 3
0 0 1 3 3 4 3 3 3 3
0 1 0 4 3 3 2 4 3 2
0 0 1 2 3 3 4 4 4 3

```

Sobel edge detection

The Sobel operator is more sensitive to diagonal edges than vertical and horizontal edges.

The Sobel 3 x 3 templates are normally given as

X-direction

```
-1 -2 -1
0  0  0
1  2  1
```

Y-direction

```
-1 0 1
-2 0 2
-1 0 1
```

Original image

```
0 0 0 0 0 0 2 0 3 3
0 0 0 1 0 0 0 2 4 2
0 0 2 0 2 4 3 3 2 3
0 0 1 3 3 4 3 3 3 3
0 1 0 4 3 3 2 4 3 2
0 0 1 2 3 3 4 4 4 3
```

absA + absB

```
4  6  4 10 14 12 14 4
6  8 10 20 16 12  6 0
4 10 14 10  2  4  2 4
2 12 12  2  2  4  8 8
```

Threshold at 12

```
0 0 0 0 1 1 1 1
2 0 0 1 1 1 0 0
0 0 1 0 0 0 0 0
0 1 1 0 0  0 0
```

Other first order operation

The Roberts operator has a smaller effective area than the other mask, making it more susceptible to noise.

$$H_r = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad H_c = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The Prewit operator is more sensitive to vertical and horizontal edges than diagonal edges.

$$H_r = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad H_c = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

The Frei-Chen mask

$$H_r = \begin{bmatrix} 0 & 0 & -1 \\ \sqrt{2} & 0 & \sqrt{2} \\ 0 & 0 & -1 \end{bmatrix} \quad H_c = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

In many applications, edge width is not a concern. In others, such as machine vision, it is a great concern. The gradient operators discussed above produce a large response across an area where an edge is present. This is especially true for slowly ramping edges. Ideally, an edge detector should indicate any edges at the center of an edge. This is referred to as localization. If an edge detector creates an image map with edges several pixels wide, it is difficult to locate the centers of the edges. It becomes necessary to employ a process called thinning to reduce the edge width to one pixel. Second order derivative edge detectors provide better edge localization.

Example. In an image such as

```

1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9

```

The basic Sobel vertical edge operator (as described above) will yield a value right across the image. For example if

```

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

```

is used then the results is

```

8 8 8 8 8 8 8
8 8 8 8 8 8 8
8 8 8 8 8 8 8

```

Implementing the same template on this "all eight image" would yield

```

0 0 0 0 0 0 0

```

This is not unlike the differentiation operator to a straight line, e.g. if $y = 3x - 2$.

$$\frac{dy}{dx} = 3 \quad \text{and} \quad \frac{d^2y}{dx^2}$$

Once we have gradient, if the gradient is then differentiated and the result is zero, it shows that the original line was straight. Images often come with a gray level "trend" on them, i.e. one side of a region is lighter than the other, but there is no "edge" to be discovered in the region, the shading is even, indicating a light source that is stronger at one end, or a gradual color change over the surface. Another advantage of second order derivative operators is that the edge contours detected are closed curves. This is very important in image segmentation. Also, there is no response to areas of smooth linear variations in intensity.

The Laplacian is a good example of a second order derivative operator. It is distinguished from the other operators because it is omnidirectional. It will highlight edges in all directions. The Laplacian operator will produce sharper edges than most other techniques. These highlights include both positive and negative intensity slopes.

The edge Laplacian of an image can be found by convolving with masks such as

```

0  -1  0      -1  -1  -1
-1  4  -1      -1  8  -1
0  -1  0      -1  -1  -1

```

or

The Laplacian set of operators is widely used. Since it effectively removes the general gradient of lighting or coloring from an image it only discovers and enhances much more discrete changes than, for example, the Sobel operator.

window across the image determining the maximum and minimum values within that window. If the difference between the maximum and minimum value exceed the predetermined threshold, an edge is present. Notice the larger number of edges with the smaller threshold. Also notice that the widths of all the edges are one pixel wide.

A second order derivative edge detector that is less susceptible to noise is the Laplacian of Gaussian (LoG). The LoG edge detector performs Gaussian smoothing before application of the Laplacian. Both operations can be performed by convolving with a mask of the form

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

Where, x, y present row and column of an image, s is a value of dispersion that controls the effective spread. Due to its shape, the function is also called the Mexican hat filter. Figure e4 shows the cross section of the LoG edge operator with different values of s. The wider the function, the wider the edge that will be detected. A narrow function will detect sharp edges and more detail.

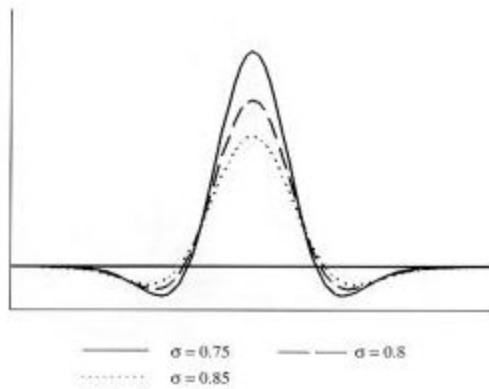


Figure 3.3.3. Cross section of LoG with various s.

The greater the value of s, the wider the convolution mask necessary. The first zero crossing of the LoG function is at . The width of the positive center lobe is twice that. To have a convolution mask that contains the nonzero values of the LoG function requires a

width three times the width of the positive center lobe (8.49s). Edge detection based on the Gaussian smoothing function reduces the noise in an image. That will reduce the number of false edges detected and also detects wider edges.

Most edge detector masks are seldom greater than 7 x 7. Due to the shape of the LoG operator, it requires much larger mask sizes. The initial work in developing the LoG operator was done with a mask size of 35 x 35.

Because of the large computation requirements of the LoG operator, the Difference of Gaussians (DoG) operator can be used as an approximation to the LoG. The DoG can be shown as

$$DoG(x, y) = \frac{e^{-\left(\frac{x^2+y^2}{2\sigma_1^2}\right)}}{2\pi\sigma_1^2} - \frac{e^{-\left(\frac{x^2+y^2}{2\sigma_2^2}\right)}}{2\pi\sigma_2^2}$$

The DoG operator is performed by convolving an image with a mask that is the result of subtracting two Gaussian masks with different σ values. The ratio $\sigma_1/\sigma_2 = 1.6$ results in a good approximation of the LoG. Figure e5 compares a LoG function ($\sigma = 12.35$) with a DoG function ($\sigma_1 = 10, \sigma_2 = 16$).

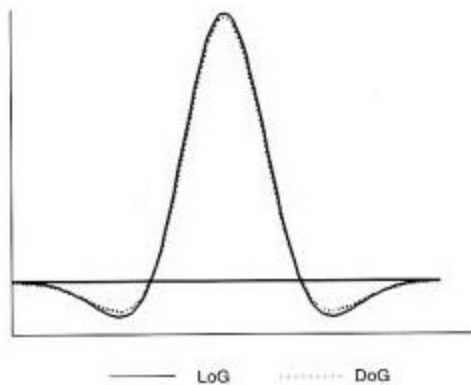


Figure.3.3.4 LoG vs. DoG functions.

One advantage of the DoG is the ability to specify the width of edges to detect by varying the values of σ_1 and σ_2 . Here are a couple of sample masks. The 9 x 9 mask will detect wider edges than the 7x7 mask.

For 7x7 mask, try

| | | | | | | |
|----|----|----|----|----|----|----|
| 0 | 0 | -1 | -1 | -1 | 0 | 0 |
| 0 | -2 | -3 | -3 | -3 | -2 | 0 |
| -1 | -3 | 5 | 5 | 5 | -3 | -1 |
| -1 | -3 | 5 | 16 | 5 | -3 | -1 |
| -1 | -3 | 5 | 5 | 5 | -3 | -1 |
| 0 | -2 | -3 | -3 | -3 | -2 | 0 |
| 0 | 0 | -1 | -1 | -1 | 0 | 0 |

For 9 x 9 mask, try

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| 0 | 0 | 0 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -2 | -3 | -3 | -3 | -3 | -2 | -2 | 0 |
| 0 | -3 | -2 | -1 | -1 | -1 | -3 | -3 | 0 |
| -1 | -3 | -1 | 9 | 9 | 9 | -1 | -3 | -1 |
| -1 | -3 | -1 | 9 | 19 | 9 | -1 | -3 | -1 |
| -1 | -3 | -1 | 9 | 9 | 9 | -1 | -3 | -1 |
| 0 | -3 | -2 | -1 | -1 | -1 | -3 | -3 | 0 |
| 0 | -2 | -3 | -3 | -3 | -3 | -2 | -2 | 0 |
| 0 | 0 | 0 | -1 | -1 | -1 | 0 | 0 | 0 |

3.4 Segmentation using thresholding :

Thresholding is based on the assumption that the histogram is has two dominant modes, like for example light objects and an dark background. The method to extract the objects will be to select a threshold $F(x,y)= T$ such that it separates the two modes. Depending on the kind of problem to be solved we could also have multilevel thresholding. Based on the region of thresholding we could have global thresholding and local thresholding. Where global thresholding is considering the function for the entire image and local thresholding involving only a certain region. In addition to the above mentioned techniques that if the thresholding function T depends on the spatial coordinates then it is known as the dynamic or adaptive thresholding.

Let us consider a simple example to explain thresholding.

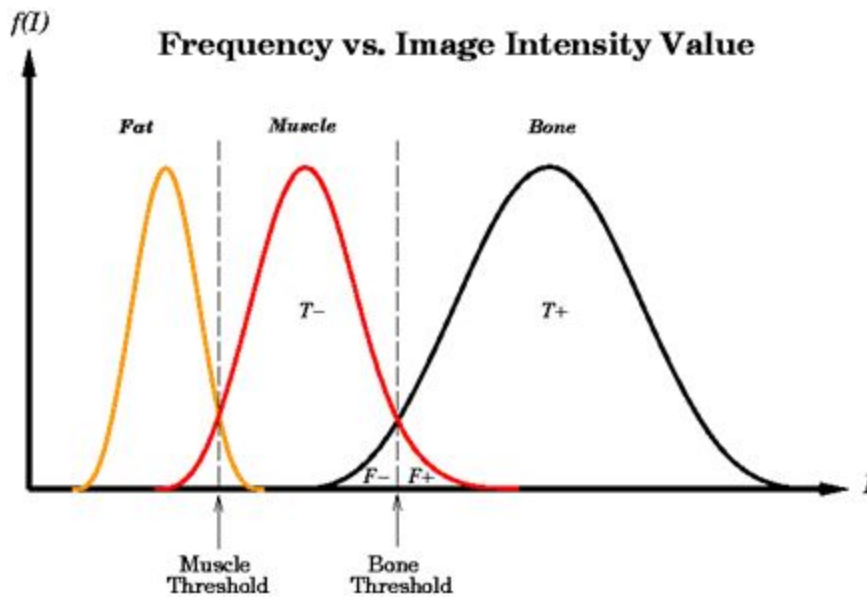


Figure 3.4: For hypothetical frequency distribution of intensity values for fat ,muscel and bone

A hypothetical frequency distribution $f(I)$ of intensity values $I(x,y)$ for fat, muscle and bone, in a CT image. Low intensity values correspond to fat tissues, whereas high intensity values correspond to bone. Intermediate intensity values correspond to muscle tissue. $F+$ and $F-$ refer to the false positives and false negatives; $T+$ and $T-$ refer to the true positives and true negatives.

3.5 Basic global thresholding technique:

In this technique the entire image is scanned by pixel after pixel and hey is labeled as object or the background, depending on whether the gray level is greater or lesser than the thresholding function T . The success depends on how well the histogram is constructed. It is very successful in controlled environments, and finds its applications primarily in the industrial inspection area.

The algorithm for global thresholding can be summarized in a few steps.

Select an initial estimate for T .

- 2) Segment the image using T . This will produce two groups of pixels. $G1$ consisting of all pixels with gray level values $>T$ and $G2$ consisting of pixels with values $\leq T$.
- 3) Compute the average gray level values $mean1$ and $mean2$ for the pixels in regions $G1$ and $G2$.
- 4) Compute a new threshold value $T = (1/2)(mean1 + mean2)$.
- 5) Repeat steps 2 through 4 until difference in T in successive iterations is smaller than a predefined parameter $T0$.

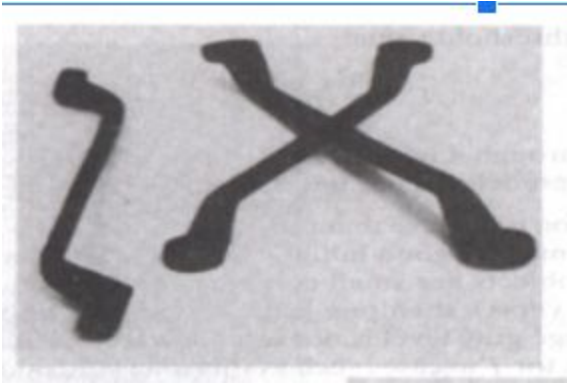


Fig 3.5.1: original img

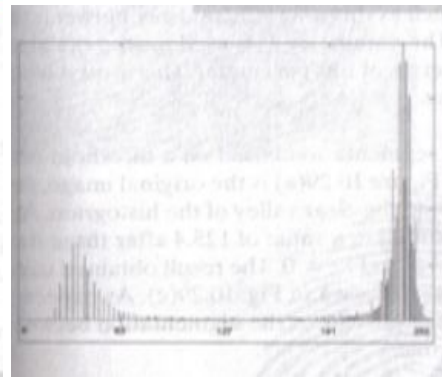


Fig 3.5.2: Img Histogram

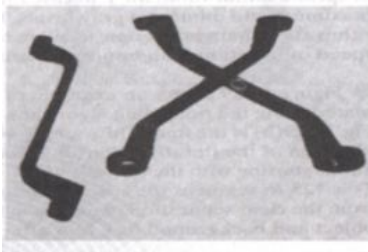


Fig 3.5.3: Result of global thresholding with T midway between max and min gray levels

3.6 Basic adaptive thresholding technique:

Images having uneven illumination make it difficult to segment using the histogram. In this case we have to divide the image in many sub images and then come up with different threshold to segment each sub image. The key issues are how to divide the image into sub images and utilize a different threshold to segment each sub image.

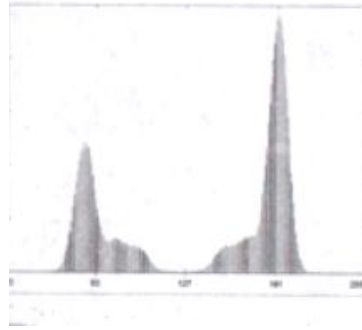


Fig 3.6.1 original img *Fig3.6.2:* Image histogram *Fig3.6.3:* result after segmentation
 With threshold estimated by iteration

The major drawback to threshold-based approaches is that they often lack the sensitivity and specificity needed for accurate classification.

3.7 Optimal thresholding technique

In the above two sections we described what global and adaptive thresholding mean. Below we illustrate how to obtain minimum segmentation error.

Let us consider an image with 2 principle gray levels regions. Let z denote the gray level values. Values as random quantities and their histogram may be considered an estimate of probability $P(z)$.

Overall density function is the sum or mixture of two densities, one of them is for the light and other is for the dark region.

The total probability density function is the $P(z) = P_1 p_1(z) + P_2 p_2(z)$, Where P_1 and P_2 are the probabilities of the pixel (random).

$$P_1 + P_2 = 1$$

The overall error of probability is $E(T) = P_2 E_1(T) + P_1 E_2(T)$, where E_1 and E_2 are the probability of occurrence of object or background pixels.

We need to find the threshold value of the error $E(T)$, so by differentiating E w.r.t T we obtain $P_1 p_1(T) = P_2 p_2(T)$.

So we can use Gaussian probability density functions and obtain the value of T .

$$T = (\mu_1 + \mu_2)/2 + (\sigma^2/(\mu_1 - \mu_2)) * \ln(P_2/P_1).$$

Where μ and σ^2 are the mean and variance of the Gaussian function for the object of a class. The other method for finding the minimum error is finding the mean square error, to estimate the gray-level PDF of an image from image histogram.

$$Ems = (1/n) * (\sum (p(z_i) - h(z_i))^2) \text{ for } i = 1 \text{ to } n.$$

Where n is the number of points in the histogram. The important assumption is that either one of the objects or both are considered. Probability of classifying the objects and background is classified erroneously.

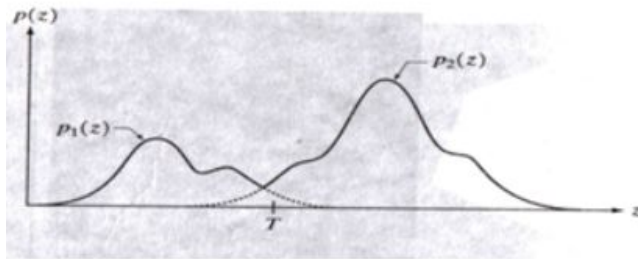


Fig 3.7.1 Grey level probability density functions of 2 regions in an image

3.8 Region based segmentation:

We have seen two techniques so far. One dealing with the gray level value and other with the thresholds. In this section we will concentrate on regions of the image.

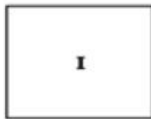
Formulation of the regions:

An entire image is divided into sub regions and they must be in accordance to some rules such as

1. Union of sub regions is the region
2. All are connected in some predefined sense.
3. No to be same, disjoint
4. Properties must be satisfied by the pixels in a segmented region $P(R_i)=\text{true}$ if all pixels have same gray level.
5. Two sub regions should have different sense of predicate.

Segmentation by region splitting and merging:

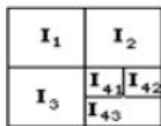
The basic idea of splitting is, as the name implies, to break the image into many disjoint regions which are coherent within themselves. Take into consideration the entire image and then group the pixels in a region if they satisfy some kind of similarity constraint. This is like a divide and conquers method. Merging is a process used when after the split the adjacent regions merge if necessary. Algorithms of this nature are called split and merge algorithms. consider the example of the split and merge process.



*Fig 3.8.1.*Whole image



Fig 3.8.2 First split



*Fig 3.8.3.*Second Split

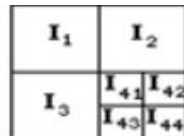


Fig 3.8.4 Merge

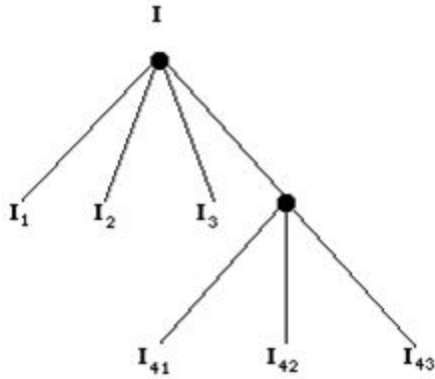


Fig 3.8: Image tree split –merge.

3.8.1 Segmentation by region growing :

Region growing approach is the opposite of split and merges.

1. An initial set of small area are iteratively merged based on similarity of constraints.
2. Start by choosing an arbitrary pixel and compared with the neighboring pixel.
3. Region is grown from the seed pixel by adding in neighboring pixels that are similar, increasing the size of the region.
- 4 When the growth of one region stops we simply choose another seed pixel which does not yet belong to any region and start again.
- 5 This whole process is continued until all pixels belong to some region.
- 6 A bottom up method.

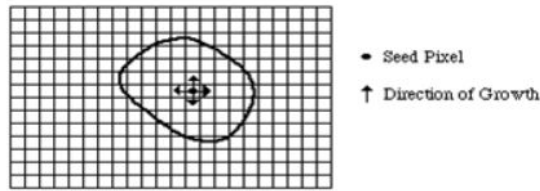


Fig 3.8.1 Start of growing a region

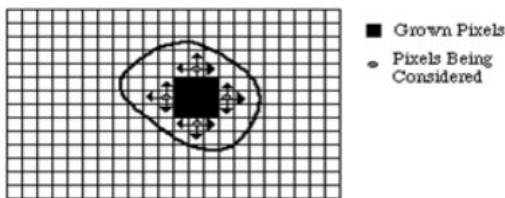


Fig 3.8.2 Growing process after few iterations

Some of the undesirable effects of the region growing are,

- Current region dominates the growth process -- ambiguities around edges of adjacent regions may not be resolved correctly.
- Different choices of seeds may give different segmentation results.
- Problems can occur if the (arbitrarily chosen) seed point lies on an edge.

However starting with a particular seed pixel and letting this region grow completely before trying other seeds biases the segmentation in favor of the regions which are segmented first. To counter the above problems, simultaneous region growing techniques have been developed.

- Similarities of neighboring regions are taken into account in the growing process.
- No single region is allowed to completely dominate the proceedings.
- A number of regions are allowed to grow at the same time.
 - o similar regions will gradually coalesce into expanding regions.
- Control of these methods may be quite complicated but efficient methods have been developed.
- Easy and efficient to implement on parallel computers.

3.8.2 Segmentation by Morphological watersheds

This method combines the positive aspects of many of the methods discussed earlier. The basic idea to embody the objects in “watersheds” and the objects are segmented. Below only the basics of this method is illustrated without going into greater details.

The concept of watersheds

It is the idea of visualizing an image in 3D. 2 spatial versus gray levels. So all points in such a topology are either

1. Belonging to regional minimum.
2. All with certain to a single minimum.
3. equal to two points where more than one minimum

A particular region is called watershed if it is a region minimum satisfying certain conditions.

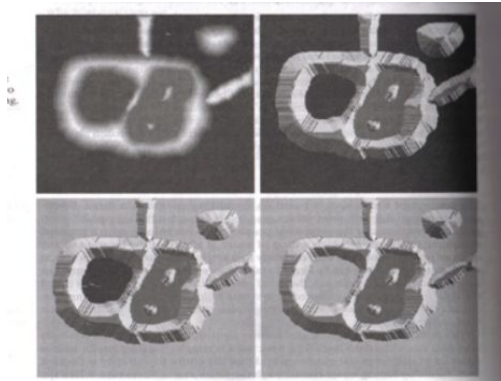


Fig 3.8.2.1 Two stages of flooding

Watershed lines:

Simple if we have a hole and water is poured at a constant rate. The level of water rises and fills the region uniformly. When the regions are about to merge with the remaining regions we build dams. Dams are boundaries. The idea is more clearly illustrated with the help of diagrams. The heights of the structures are proportional to the gray level intensity. Also the entire structure is enclosed by the height of the dam greatest of the dam height. In the last figure we can see that the water almost fills the dams out, until the highest level of the gray level in the images researched.

The final dam corresponds to the watershed lines which are the desired segmentation result. The principle applications of the method are in the extraction of uniform objects from the background. Regions are characterized by small variations in gray levels, have small gradient values. So it is applied to the gradient than the image. Region with minimum correlated with the small value of gradient corresponding to the objects of interest.

Use of Motion in segmentation

Motion of objects can be very important tool to exploit when the background detail is irrelevant. This technique is very common in sensing applications.

Let us consider two image frames at time t_1 and t_2 , $f(x,y,t_1)$ and $f(x,y,t_2)$ and compare them pixel to pixel. One method to compare is to take the difference of the pixels

$$D12(x,y) = \begin{cases} 1 & \text{if } |f(x,y,t_1) - f(x,y,t_2)| > T, \\ 0 & \text{otherwise.} \end{cases}$$

Where, T is a threshold value.

This threshold is to signify that only when there is an appreciable change in the gray level, the pixels are considered to be different.

In dynamic image processing the D12 has value set to 1 when the pixels are different; to signify the objects are in motion.

CHAPTER 4

EXISTING METHODS

4.1 Background subtraction methods:

The common problem of all computer-vision systems is to separate people from a background scene (determine the foreground and the background). Many methods are proposed to resolve this problem. Background subtraction is used in several applications to detect the moving objects in a scene as in multimedia, video surveillance and optical motion capture. Background subtraction consist of steps, like Background modeling, Background initialization, Background maintenance and Foreground detection shown in Figure

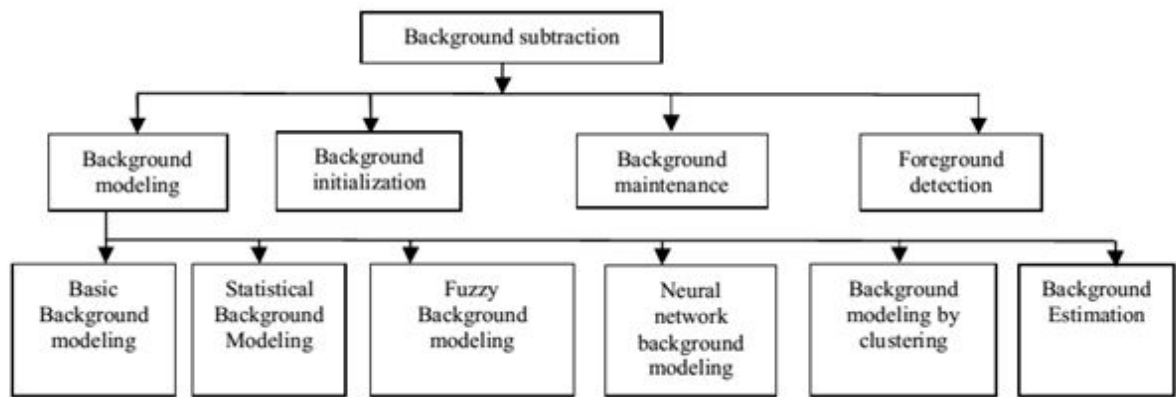


Figure 2.1 Classification of Background Subtraction

4.1.1 Background Modeling

Background modeling describes the kind of model used to represent the background. The simplest way to model the background is to acquire a background image which does not include any moving object. In some environments, the background is not available, and can always be changed under critical situations, like illumination changes, objects being introduced or removed from the scene.

In literature, there are many proposed background modeling algorithms. This is mainly because no single algorithm is able to cope with all the challenges in this area. There are several problems that a good background subtraction algorithm must resolve. First, it must be robust against changes in illumination. Second, it should avoid detecting non-stationary background objects, such as swaying leaves, grass, rain, snow, and shadows cast by moving objects. Finally, the background model should be developed such that it should react quickly to changes in the background, such as the starting and stopping of vehicles.

Background modeling techniques could be classified into two broad categories: Non-Predictive Modeling and Predictive Modeling. The former tries to model the scene as a time series, and creates a dynamic model at each pixel to consider the incoming input, using the past observations, and utilizes the magnitude of deviation between the actual observation and the predicted value, to categorize the pixels as part of the foreground or background. The latter neglects the order of the input observations, and develops a statistical (probabilistic) model, such as the probability density function at each pixel.

Background adaptation (Cheung and Kamath, 2004) techniques could also be categorized as non-recursive and recursive. A non-recursive technique estimates the background based on a sliding-window approach. The observed video frames are stored in a buffer, considering the existing pixel variations in the buffer the background image will be estimated. Since in practice the buffer size is fixed as time passes and more video frames come along the initial frames of the buffer are discarded which makes these techniques adaptive to scene changes depending on their buffer size. However, in the case of adapting to slow moving objects or coping with transient stops of certain objects in the scene the non-recursive techniques require large amount of memory for storing the appropriate buffer. With a fixed buffer size this problem can partially be solved by reducing the frame rate as they are stored.

On the contrary the recursive techniques instead of maintaining a buffer to estimate the background they try to update the background model recursively using either a single or multiple model(s) as each input frame is observed. The very first input frames are capable to leave an effect on new input video frames which makes the algorithm adapt with periodical motions such as flickering, shaking leaves, etc. Recursive methods need less storage in comparison with non-recursive methods but possible errors stay visible for longer time in the background model. The majority of schemes use exponential weighting or forgetting factors to determine the proportion of contribution of past observations. Many of the pixel based probability density function methods in the literature are of the mixture-of-Gaussians (MOG) variety; each pixel's intensity is described by a mixture of K Gaussian distributions where K is usually a small number. Over the years, increasingly complex pixel-level algorithms have been proposed. Among these, by far the most popular is the Gaussian Mixture Model (GMM) (Stauffer and Grimson 1999, 2000). This model consists of modeling the distribution of the values observed over time at each pixel by a weighted mixture of gaussians.

This background pixel model is able to cope with the multimodal nature of many practical situations and leads to good results when repetitive background motions, such as tree leaves or branches, are encountered. A particle swarm optimization method (White and Shah 2007) is proposed to automatically determine the parameters of the GMM algorithm. Varcheie et al (2010) combining a GMM model with a region-based algorithm based on color histograms and texture information. In the experiments, the method outperforms the original GMM algorithm.

However, the technique has a considerable computational cost as they only manage to process seven frames of 640×480 pixels per second with an Intel Xeon 5150 processor. The downside of the GMM algorithm resides in its strong assumptions that the background is more frequently visible than the foreground and

that its variance is significantly lower. None of this is valid for every time window.

Furthermore, if high- and low-frequency changes are present in the background, its sensitivity cannot be accurately tuned and the model may adapt to the targets themselves or miss the detection of some high speed targets, (Elgammal et al 2000), the estimation of the parameters of the model can become problematic in real-world noisy environments. This often leaves one with no other choice than to use a fixed variance in a hardware implementation.

Finally, it should be noted that the statistical relevance of a gaussian model is debatable that natural images exhibit non-gaussian statistics (Srivastava et al 2003). To avoid the difficult question of finding an appropriate shape for the probability density function, some authors have turned their attention to non-parametric methods to model background distributions. One of the strengths of non-parametric kernel density estimation methods is their ability to circumvent a part of the delicate parameter estimation step due to the fact that they rely on pixel values observed in the past. For each pixel, these methods build a histogram of background values by accumulating a set of real values sampled from the pixel's recent history. These methods then estimate the probability density function with this histogram to determine whether or not a pixel value of the current frame belongs to the background.

Non-parametric kernel density estimation methods can provide fast responses to high-frequency events in the background by directly including newly observed values in the pixel model. However, the ability of these methods to successfully handle concomitant events evolving at various speeds is questionable since they update their pixel models in a first-in first-out manner. This has led some authors to represent background values with two series of values or models: a short term model and a long term model (Monari and Pasqual 2007).

While this can be a convenient solution for some situations, it leaves open the question of how to determine the proper time interval. In practical terms, handling two models increases the difficulty of fine-tuning the values of the underlying parameters, it improves the overall detection quality significantly.

In the codebook algorithm (Kim, et al 2004, 2005) each pixel is represented by a codebook, which is a compressed form of background model for a long image sequence. Each codebook is composed of codeword comprising colors transformed by an innovative color distortion metric. An improved codebook incorporating the spatial and temporal context of each pixel has been proposed (Wu and Peng 2009). Codebooks are believed to be able to capture background motion over a long period of time with a limited amount of memory. However, one should note that the proposed codebook update mechanism does not allow the creation of new codewords and this can be problematic if permanent structural changes occur in the background. Instead of choosing a particular form of background density model, (Wang and Suter 2006, 2007) use the notion of “consensus”. They keep a cache of a given number of last observed background values for each pixel and classify a new value as background if it matches most of the values stored in the pixel’s model. One might expect that such an approach would avoid the issues related to deviations from an arbitrarily assumed density model, but since values of pixel models are replaced according to a first-in first-out update policy.

The W4 model presented (Haritaoglu et al 2000) is a rather simple but nevertheless effective method. It uses three values to represent each pixel in the background image: the minimum and maximum intensity values, and the maximum intensity difference between consecutive images of the training sequence. A small improvement made (Jacques et al 2005) to the W4 model together with the incorporation of a technique for shadow detection and removal. 19 An on-line K-means approach is taken (Stauffer and Grimson, 2000).

In this approach, each incoming pixel is matched to a cluster if it is within 2.5 standard deviations from the cluster mean.

The parameters for that cluster are then updated with the new observation. Repeated similar pixels will drive the weight of their cluster up and simultaneously reduce the cluster variance, indicating a higher selectivity. If no match is found, the observation becomes a new cluster with an initially low weight and wide variance. Background estimation is formulated (Cohen 2005) as an optimal labeling problem in which each pixel of the background image is labeled with a frame number, indicating which colour from the past must be copied.

The author's proposed algorithm produces a background image, which is constructed by copying areas from the input frames. Impressive results are shown for static backgrounds but the method is not designed to cope with objects moving slowly in the background, as its outcome is a single static background frame. Inspired (Mahadevan and Vasconcelos 2010) by the biological mechanism of motion-based perceptual grouping a spatio-temporal saliency algorithm proposed is applicable to scenes with highly dynamic backgrounds, which can be used to perform background subtraction. Comparisons of their algorithm with other state-of-the-art techniques show that their algorithm reduces the average error rate, but at a cost of a prohibitive processing time (several seconds per frame), which makes it unsuitable for real-time applications. Pixel-based background subtraction techniques compensate for the lack of spatial consistency by a constant updating of their model parameters.

All these methods have the following steps: Background Modeling, Background Initialization, Background Maintenance and Foreground Detection. Developing a background subtraction method, researchers must propose each step and decide the features size (pixel, a block or a cluster) and type (color, edge, stereo, motion and texture) in relation to the serious situations they want to handle different kinds

CHAPTER 5

PROPOSED METHOD

5.1 Block diag of proposed method :

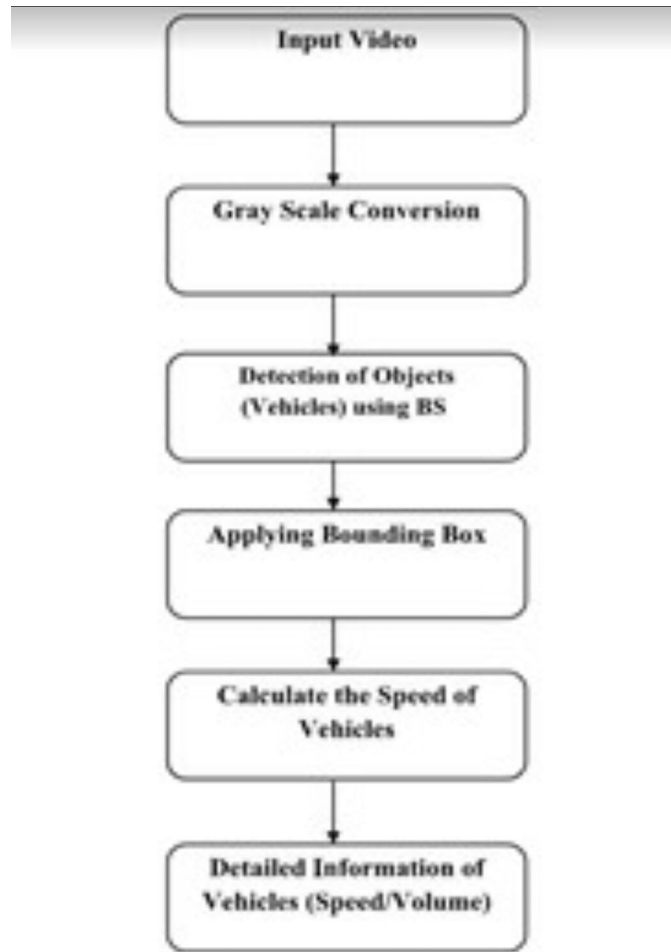


Fig.5.1 Proposed Work Block Diagram

5.1.1 Input Video :

Selection of the input video is a major task for any video processing application. We took video in Ameerpet to SR Nagar location from Hyderabad. Our videos taken from mobile device are working properly for proposed work.

5.1.2 Gray scale conversion :

Average method is the simplest one. You just have to take the average of three colors. Since its an RGB image, so it means that you have add r with g with b and then divide it by 3 to get your desired grayscale image.

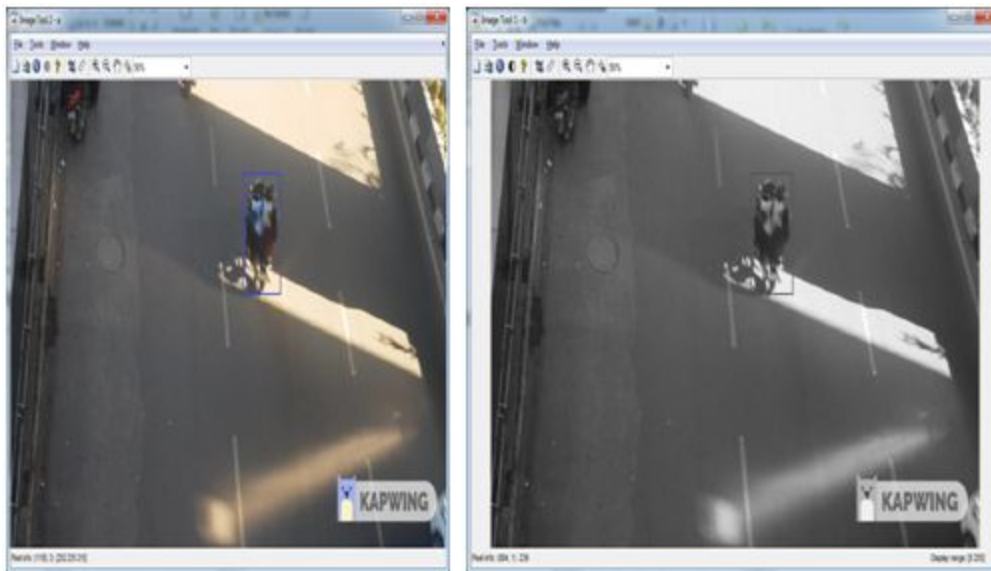


Fig.5.1.2 Color image(RGB image) and gray scale image

An RGB image can be viewed as three images (a red scale image, a green scale image and a blue scale image) stacked on top of each other. In MATLAB, an RGB image is basically an $M \times N \times 3$ array of colour pixel,

where each colour pixel is a triplet which corresponds to red, blue and green colour component of RGB image at a specified spatial location. Similarly, A Grayscale image can be viewed as a single layered image. In MATLAB, a grayscale image is basically $M \times N$ array whose values have been scaled to represent intensities.

Algorithm for conversion:

1. Read RGB colour image into MATLAB environment
2. Extract Red, blue and green colour components from RGB image into 3 different 2-D matrices
3. Create a new matrix with the same number of rows and columns as RGB image, containing all zeros.
4. Convert each RGB pixel values at location (i, j) to grayscale values by forming a weighted sum of the Red, Green, and Blue colour components and assign it to corresponding location (i, j) in new matrix

5.1.3 Detection of Object(Vehicles)

Vehicle Detection and Tracking using Background Subtraction

Background modeling is used to adapt the change in the dynamic environment. It is very challenging for every background subtraction based method. Traditional method utilizes temporal differencing or optic flow based method. The temporal differencing method utilizes two or more consecutive frames to extract moving regions. Optical flow uses the characteristics of flow vectors of the moving objects over time. If the background contents are not visible for long time, all these methods will fail to generate accurate background model. These methods are vulnerable and prone to false detection, if the temporal changes are generated by noise or illumination change due to weather condition.

Considering the robustness, suitability and reduced data access rate, current research has focused on the benefit of edges structure in processing sequence images. But edges show shape and size variation within frames due to the changes in illumination and noise. Moreover, the variations for different edges are not the same.

Without considering this variation from the environment, detectors output cannot be reliable. To solve this problem, the edge is represented as segment that allows incorporating knowledge to every edge segment about its motion, shape, position, and size variation. And thus, it helps to model the environment using the statistics of all edge segments. Thus, in this proposed method, edges are treated differently, i.e., segments having small motion variation statistic will be matched with a high threshold, while a low threshold may be used for those having large motion variation.

Background subtraction is particularly a commonly used technique for motion segmentation in static scenes. It attempts to detect moving regions by subtracting the current image pixel-by-pixel from a reference background image that is created by averaging images over time in an initialized period. The pixels for which the difference is above the threshold are classified as foreground. After creating foreground pixel map, some morphological post processing operations such as erosion, dilation and closing are performed to reduce the effects of noise and enhance the detected regions.

The reference background is updated with new images over time to adapt to dynamic scene changes. Background subtraction technique often relies on deriving the probabilistic model of the background. When a new image is captured, the difference between the image and background model is computed for moving object detection. Unfortunately, the derivation of the model is complex and computationally expensive. Although alternative approaches not based on probabilistic modeling of the background image exist, they are very specific to applications.

Therefore, most of existing approaches for moving object detection are computationally heavy and subject to large delays, adversely affecting the performance of real-time surveillance. Background subtraction is a computational vision process of extracting foreground objects in a particular scene. A foreground object can be described as an object of attention, which helps in reducing the amount of data to be processed as well as provide important information to the task under consideration. Often, the foreground object can be thought of as a coherently moving object in a scene. We must emphasize the word coherent here because if a person is walking in front of moving leaves, the person forms the foreground object while leaves though having motion associated with them are considered background due to its repetitive behavior. In some cases, distance of the moving object also forms a basis for it to be considered a background, e.g if in a scene one person is close to the camera while there is a person far away in background, in this case the nearby person is considered as foreground while the person far away is ignored due to its small size and the lack of information that it provides. Identifying moving objects from a video sequence is a fundamental and critical task in many computer-vision applications. A common approach is to perform background subtraction, which identifies moving objects from the portion of video frame that differs from the background.

Background subtraction is a class of techniques for segmenting out objects of interest in a scene for applications such as surveillance. There are many challenges in developing a good background subtraction algorithm. First, it must be robust against changes in illumination. Second, it should avoid detecting non-stationary background objects and shadows cast by moving objects. A good background model should also react quickly to changes in background and adapt itself to accommodate changes occurring in the background such as moving of a stationary chair from one place to another. It should also have a good foreground detection rate and the processing time for background subtraction should be real-time.

Color Histograms- the most simple and intuitive way is to extract the features from various color channels of the images. This can be done by plotting the histograms of various color channels and then collecting the data from the bins of the histogram. These bins give us useful information about the image and are really helpful in extracting good features.

Well we are almost done at this moment! The pipeline works fantastic with the images but still there is one problem if you will run the pipeline on the images coming from a video stream. The final detected boxes will become very shaky and will not deliver a smooth experience; it may be possible that the box goes away in some frames. So what's the solution? The solution is pretty intuitive, store all the refined windows detected from the previous 15 frames and average out the rectangles in the current frame. Also you need to adjust threshold to a higher level now. Just by doing so the final bounding boxes appear less shaky and delivers a smooth flow. I tried my pipeline on the project video and results are somewhat like this.

5.1.4 Bounding Box:



Fig.5.1.4 Applied Bounding box for detected vehicle

Based on the linear assignment results, we keep two lists for unmatched detections and unmatched trackers, respectively. When a car enters into a frame and is first detected, it is not matched with any existing tracks, thus this particular detection is referred to as an unmatched detection, as shown in the following figure. In addition, any matching with an overlap less than `iou_thrd` signifies the existence of an untracked object. When a car leaves the frame, the previously established track has no more detection to associate with. In this scenario, the track is referred to as unmatched track. Thus, the tracker and the detection associated in the matching are added to the lists of unmatched trackers and unmatched detection, respectively.

5.1.5 Calculating speed of vehicle

The only thing that changes when you change the tire size is the distance traveled. So, for every revolution of the tire, you travel a distance equal to the circumference of the circle that is the tread of the tire. (Assuming the tire is not slipping).

$$\begin{array}{lcl} \text{Speed} & = & \frac{\text{Distance}}{\text{Time}} \\ \text{Distance} & = & \text{Speed} \times \text{Time} \\ \text{Time} & = & \frac{\text{Distance}}{\text{Speed}} \end{array}$$

Speed is the measure of how fast something is going at a certain time. If you've ever looked at a car's speed gauge while it's moving, you've seen speed being measured — the farther the needle goes, the higher the car's speed is. There are a few different ways to calculate speed depending on which types of information you have. For general purposes, the equation

Speed = distance/time (or $s = d/t$) is usually the easiest way to calculate speed.

5.1.6 Detailed Information of Vehicles (Speed/Volume)

Find the distance that an object has traveled.

The basic equation that most people use to figure out how fast something is going is very easy to use. The first thing you'll need to know is how far the object traveled. In other words, how far is its starting point from its ending point? This equation will be easier to understand with an example. Let's say that we are making a trip in a car to a theme park 100 miles away (about 161 kilometers). In the next few steps, we'll use this information to solve our equation.

Find the time that the object took to travel that distance. The next piece of information you'll need is how long the object took as it traveled. In other words, how long did it take to get from its starting point to its ending point?

Divide the distance by the time to find the speed. All you need are these two pieces of information to find your speed for the trip. The distance over the time will give you the object's speed. In our example, $100 \text{ miles} / 2 \text{ hours} = 50 \text{ miles/hour}$ (about 80 kilometers/hour).

CHAPTER 6

SIMULATION RESULTS

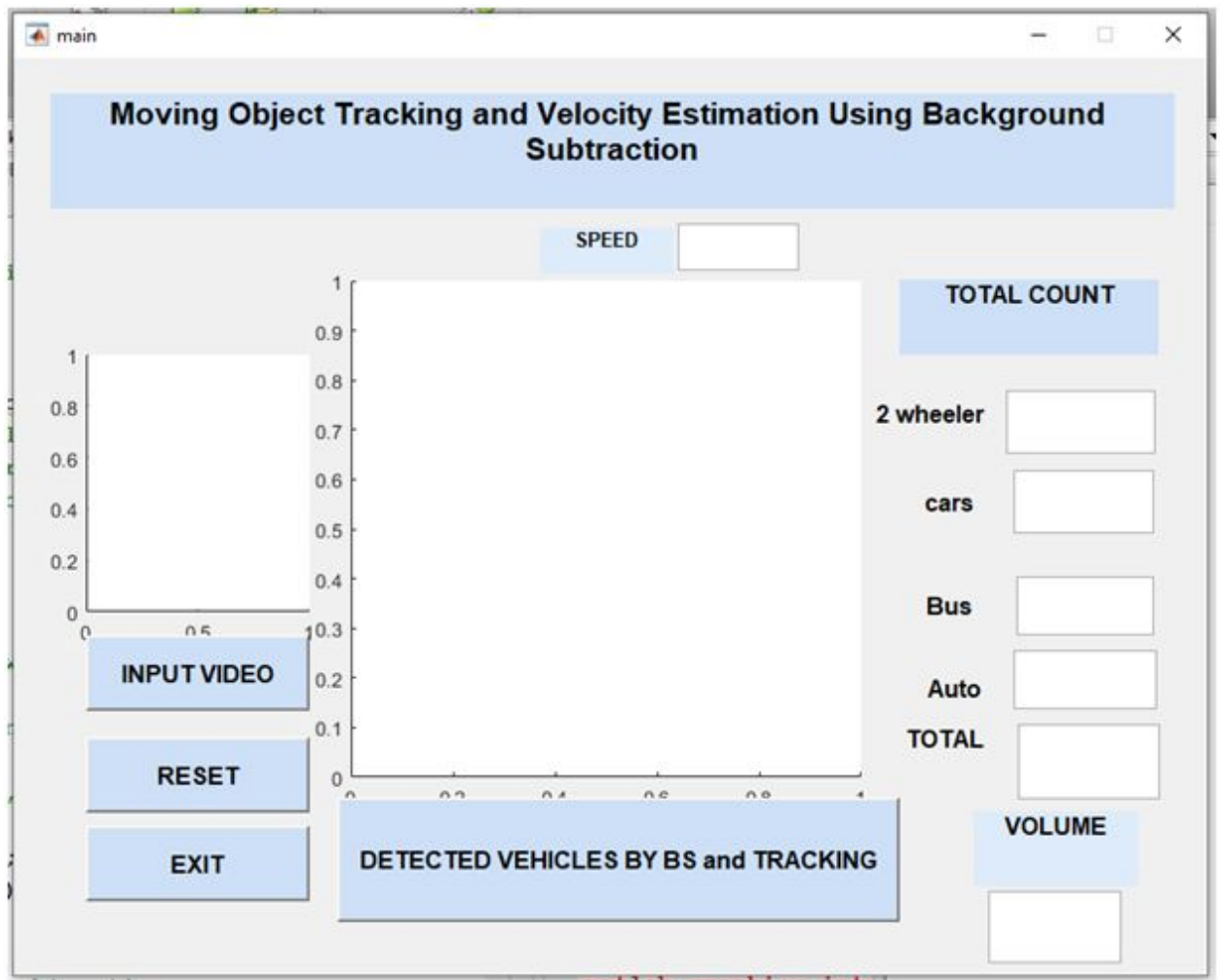


Fig.6.1 Overall Structure of GUI (Graphical User Interface) For Proposed Methodology

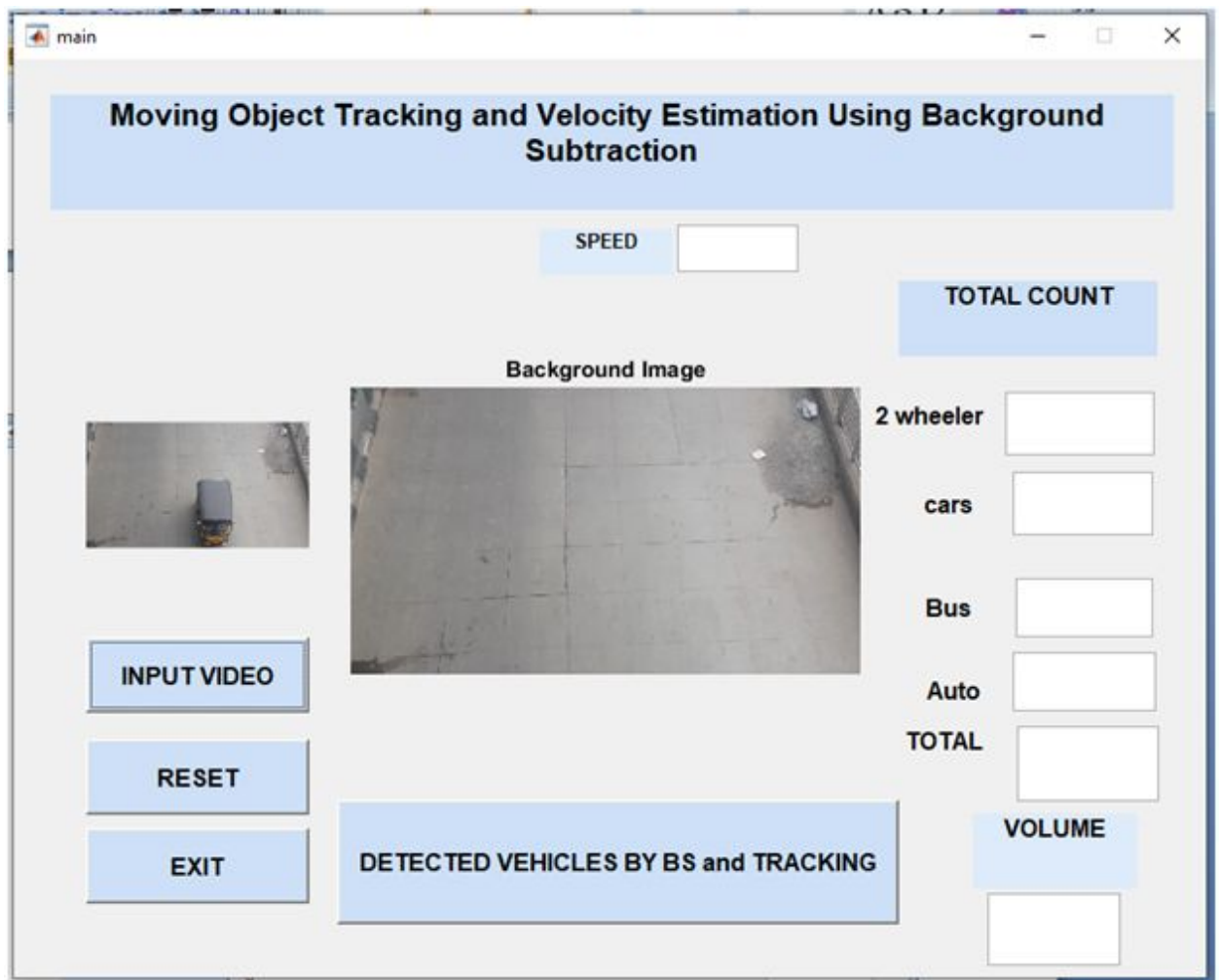


Fig.6.2 Selected Input Video for Proposed Work

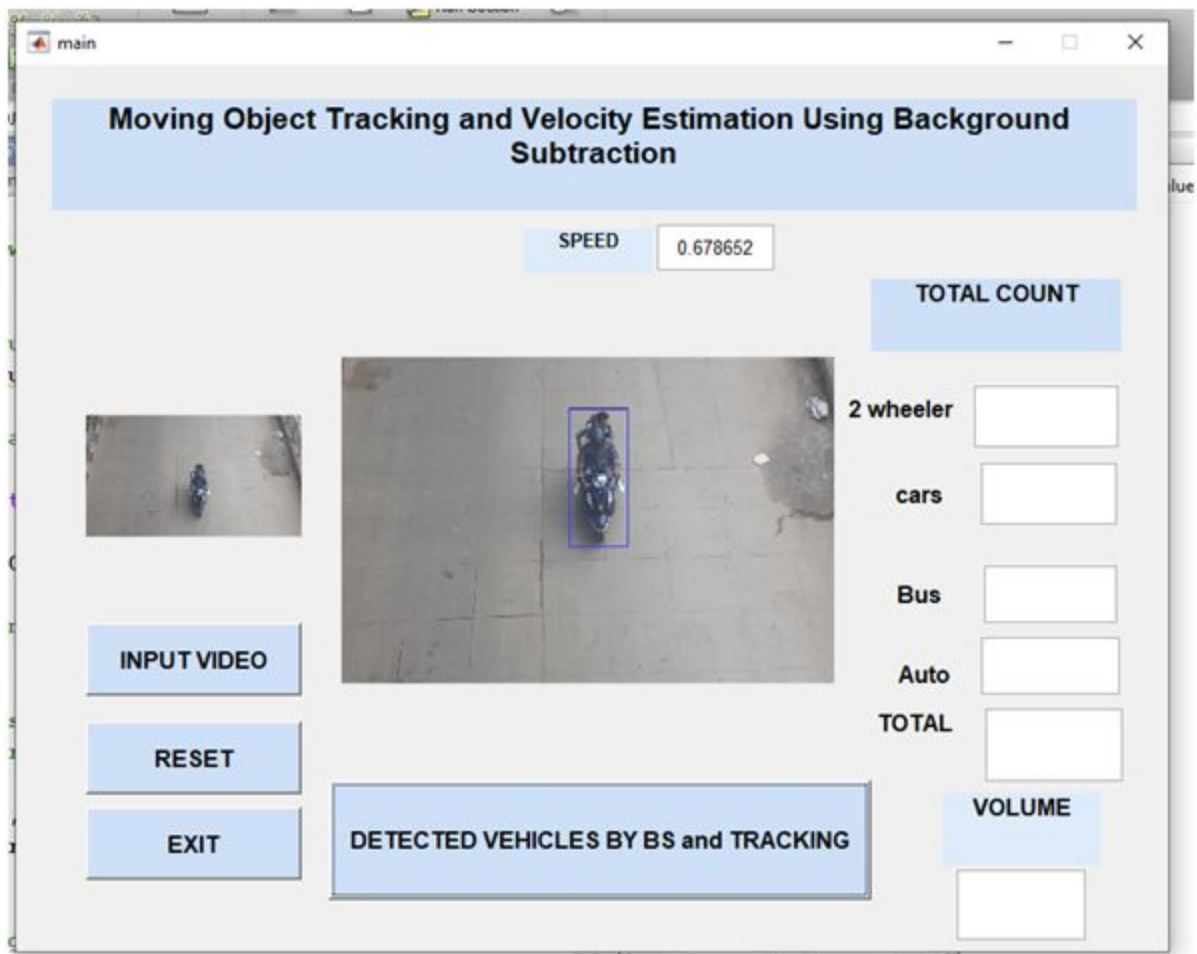


Fig.6.3 Detected Vehicles from the Input Video using background subtraction

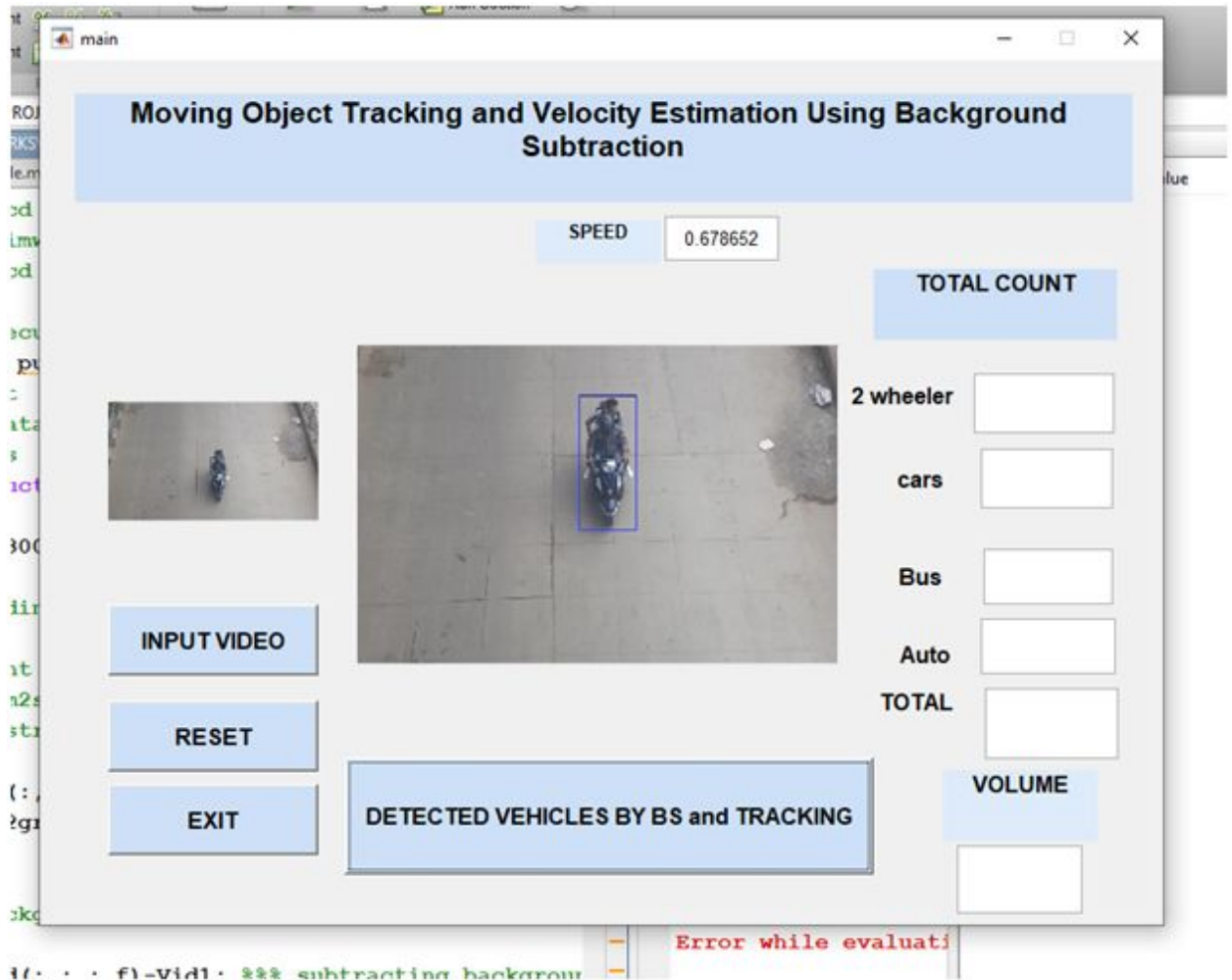


Fig.6.4 Speed Calculation for Vehicles from Detected Bounding Boxes

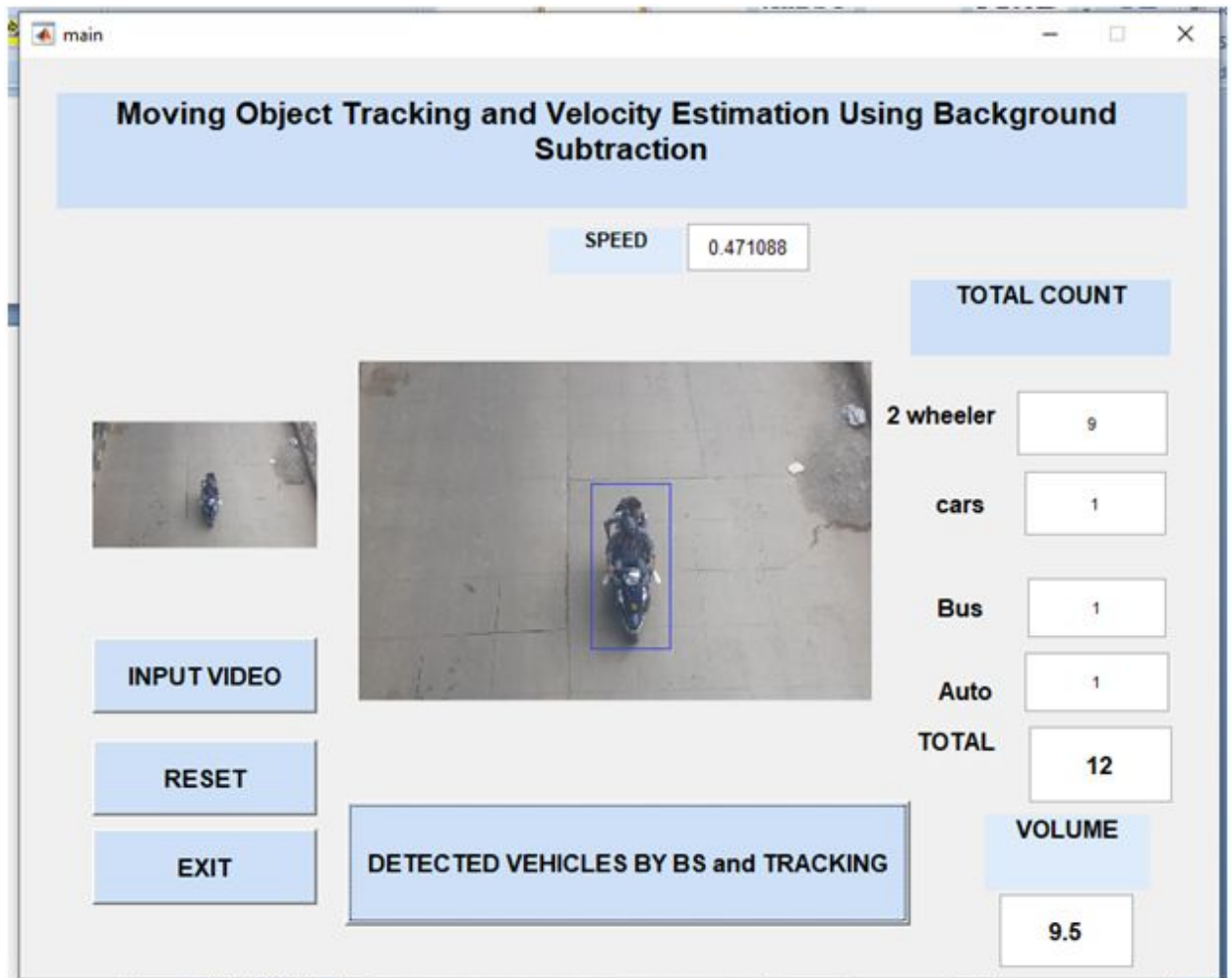


Fig.6.5 Total Count for Detected Vehicles and Volume Calculation

CHAPTER 7

CONCLUSION

7.1 Conclusion and future Scope

This paper illustrates the suitability of using this model to detect moving objects for the video surveillance based applications. The strength of our approach lies in the ability to separate background edge segments from moving edge segments. The velocity and speed of the moving object can also be calculated. In our future work, we will incorporate multi image detection and calculating speed for the multiple moving objects present in the scene for more sophisticated vision based applications like airport security, activity recognition, etc. This work can also be used along with other detection techniques to detect the kind of the object

REFERENCES

- [1] Speed examination of guests Accident headquartered on Two size Eccentric Collision. Xiaokun Miao.
- [2] Temporal shakiness and the examination of roadway accident information Fred Mannering.
- [3] ACCIDENT vehicle programmed DETECTION framework through picture PROCESSING innovation. Accomplished by methods for H.Takigawa and A.'ISuge.
- [4] Strange Incident Detection framework utilizing photo Processing innovative skill Produced with the guide of Hiromi Ikeda and Toshihiro Matsuo.
- [5] Ongoing programmed guests Accident acknowledgment using Developed by utilizing HFG Ayoub Al-Hamadi and Bernd Michaelis.
- [6] Genuine Time Detection and Reporting of car accident actualized by methods for ParagParmar and Ashok M. Sapkal.
- [7] Embedded unfaltering speed confine sign affirmation utilizing photo getting ready and AI programs executed by utilizing Samuel L. Gomes and Edson CavalcantiNeto.
- [8] Smartstreet Surveillance utilizing photograph Processing Developed by utilizing Gargi Desai and SagarJakharia.
- [9] A guests Accident Recording and Reporting model at Intersections. Made by utilizing Yong-Kul Ki and Dong-youthful Lee.
- [10] Halting car crashes with in-car elective genuinely strong systems - The result of disaster hotspot flag on driver direct.

- [11] VIDEO-set up car crash examination AT INTERSECTIONS using PARTIAL auto TRAJECTORIES. Made through Ömer Aköz and M. ElifKarsligil.
- [12] Creative and insightful established consistent vehicle crash area applied by means of Zuhui and Xieyaohua, Ma lu, Fu Jiansheng.
- [13] Result OF PEDESTRIANS UN-SIGNALIZED MID-BLOCK CROSSING ON VEHICULAR pace. Conveyed through B RaghuramKadali.
- [14] Results of Vehicular Lanes on Pedestrian opening Acceptance propensities. Made by utilizing B Raghu ramKadalia, Dr. P Vedagiri.
- [15] Assessment of the operational impacts of u-flip advancement. Finished by methods for Pan Liu.
- [16] The results of Crosswalks on traffic coast. Made by methods for Victor L. Knoop.
- [17] An assessment of modern upgrades to vehicle security. Made by methods for Glassbrenner, Donna, Ph.D.

APPENDIX -A

1. What Is MATLAB?

MATLAB® is a propelled language for specific enrolling. It consolidates count, portrayal, and programming in a simple to-utilize where disarranges and courses of action are imparted in like manner numerical documentation.

2. Typical uses of MATLAB

- Calculation as well as Math
- Algorithm advancement
- Data procurement
- Data examination, investigation, and perception

3. The main features of MATLAB

1. Advance computation for tip top numerical count, especially in the Field network variable based math
2. A gigantic combination of predefined numerical limits and the ability to portray one's very own abilities.
3. Two-and three dimensional plans for plotting and demonstrating data
4. A complete online help system
5. Powerful, cross section or vector arranged noteworthy level programming language for solitary applications.

6. Toolboxes available for dealing with front line issues in a couple of utilization zones.

4. MATRIX, VECTOR AND SCALAR:

- Three fundamental concepts in MATLAB, and in linear algebra, are scalars, vectors and matrices.
- A *scalar* is simply just a fancy word for a number (a single value).
- A *vector* is an ordered list of numbers (one-dimensional). In MATLAB they can be represented as a row-vector or a column-vector.
- A *matrix* is a rectangular array of numbers (multi-dimensional). In MATLAB, a two-dimensional matrix is defined by its number of rows and columns.

Mat lab uses variables that are defined to be matrices.

A matrix is a collection of numerical values that are organized into a specific configuration of rows and columns. The number of rows and columns can be any number.

A= [1 2 3 4

5 6 7 8];

A is for example, 2 rows and 4 columns define a 2 x 4 matrix which has 8 elements

In total.

A scalar is represented by a 1×1 matrix in Mat lab:
`a=1;`

A vector of n elements can be represented by an $n \times 1$ matrix, in which case it is called a column vector, or a vector can be represented by a $1 \times n$ matrix, in which case it is called a row vector of n elements.

`x = [3.5, 33.22, 24.5];` `x` is a row vector or 1×3 matrix

`X 1 = [2` `x1` is column vector or 4×1 matrix
5
3
-1];

The matrix name can be any group of letters and numbers up to 19, but always beginning with a letter.

Mat lab is "case sensitive", that is, it treats the name 'C' and 'c' as two different variables.

Similarly, 'MID' and 'mid' are treated as two different variables.

5. ALGEBRIC OPERATIONS IN MATLAB:

Scalar Calculations:

| | |
|---|--|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Right division (a/b means $a \div b$) |
| \ | left division (a\b means $b \div a$) |
| ^ | Exponentiation |

For example $3*4$ executed in 'mat lab' gives an=12

$4/5$ gives an=0.8

Array products: Recall that addition and subtraction of matrices involved addition or subtraction of the individual elements of the matrices. Sometimes it is desired to simply multiply or divide each element of a matrix by the corresponding element of another matrix 'array operations'.

Array or element-by-element operations are executed when the operator is preceded by a '.' (Period):

$a.*b$ multiplies each element of a by the respective element of b

$A./b$ divides each element of a by the respective element of b

$a.\backslash b$ divides each element of b by the respective element of a

A. \wedge b raise each element of a by the respective b element

6. WHY USE MATLAB:

Advantages:

- Handles vector and matrices very excellent
- quick plotting and evaluation huge
- documentation (type 'help') plenty of nice capabilities:
- FFT, fuzzy logic, neural nets, numerical integration, OpenGL •
- Drawbacks: gradual compared to C or Java
- **Data Classes:**

In spite of the way that we work with entire numbers sorts out the estimations of pixels themselves are not restricted to be entire numbers in MATLAB. Table above once-over various data classes reinforced by MATLAB and IPT are addressing pixels regards. The underlying eight areas in the table are suggests as numeric data classes. The ninth entry is the burn class and, as showed up, the last area is insinuated as predictable data class. Each and every numeric computation in MATLAB are done in twofold sums, so this is in like manner a progressive data class involvement with picture dealing with applications. Class unit 8 in like manner is experienced as frequently as could be expected under the circumstances, especially when scrutinizing data from stores devices, as 8 piece pictures are most fundamental depictions found for all intents and purposes.

These two data classes, classes reasonable, and, to a lesser degree, class unit 16 include the fundamental data classes on which we focus. Many pit works in any case reinforce all of the data classes recorded in table. Data class twofold requires 8 bytes to address a number uint8 and into 8 require one byte each, uint16 and int16 requires 2bytes and unit 32.

Name Description

| | |
|---------|--|
| Double | Double _ precision, floating_ point numbers the Approximate. |
| Unit8 | unsigned 8_bit integers in the range [0,255] (1byte per Element). |
| Unit16 | unsigned 16_bit integers in the range [0, 65535] (2byte Per element). |
| Unit 32 | unsigned 32_bit integers in the range [0, 4294967295] (4 bytes per element). |
| Int8 | signed 8_bit integers in the range[-128,127] (1 byte per element) |
| Into 16 | signed 16_byte integers in the range [32768, 32767] (2 bytes per element). |
| Into 32 | Signed 32_byte integers in the range [-2147483648, 2147483647] (4 byte per element). |

Single single _precision floating _point numbers with values

In the approximate range (4 bytes per elements).

Char characters (2 bytes per elements).

Logical values are 0 to 1 (1byte per element).

Into 32 and single, required 4 bytes each. The burn information class holds characters in Unicode portrayal. A character string is simply a 1*n exhibit of characters coherent cluster contains just the qualities 0 to 1, with every component being put away in memory utilizing capacity consistent or by utilizing social administrators.

7. Applications and Advantages

1. Avoid capacity of corruption and power for sound sign
2. Protection of scholarly advanced works
3. Content saving tasks in the transmission channel ought to be endured to encourage adaptable applications.
4. Low computational intricacy both at the transmitter and beneficiary side is significant continuously media preparing which can be conceivable
5. Localized vindictive change can be effectively distinguished for the ensured computerized information

APPENDIX-B

INTRODUCTION TO DIGITAL IMAGE PROCESSING

6.1 What is DIP?

An image may be defined as a two-dimensional function $f(x, y)$, where x & y are spatial coordinates, & the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point. When x , y & the amplitude values of f are all finite discrete quantities, we call the image a digital image. The field of DIP refers to processing digital image by means of digital computer. Digital image is composed of a finite number of elements, each of which has a particular location & value. The elements are called pixels

Gray scale image:

A grayscale image is a function $I(x, y)$ of the two spatial coordinates of the image plane.

$I(x, y)$ is the intensity of the image at the point (x, y) on the image plane.

$I(x, y)$ takes non-negative values assume the image is bounded by a rectangle $[0, a] \times [0, b]$: $I: [0, a] \times [0, b] \rightarrow [0, \text{info})$

Color image:

It can be represented by three functions, $R(x, y)$ for red, $G(x, y)$ for green and $B(x, y)$ for blue.

An image may be continuous with respect to the x and y coordinates and also in amplitude. Converting such an image to digital form requires that the coordinates as well as the amplitude to be digitized. Digitizing the coordinate's values is called sampling. Digitizing the amplitude values is called quantization.

6.3 Coordinate convention:

The result of sampling and quantization is a matrix of real numbers. We use two principal ways to represent digital images. Assume that an image $f(x, y)$ is sampled so that the resulting image has M rows and N columns. We say that the image is of size M X N. The values of the coordinates (xylem) are discrete quantities. For notational clarity and convenience, we use integer values for these discrete coordinates. In many image processing books, the image origin is defined to be at $(xylem)=(0,0)$. The next coordinate values along the first row of the image are $(xylem)=(0,1)$. It is important to keep in mind that the notation $(0,1)$ is used to signify the second sample along the first row. It does not mean that these are the actual values of physical coordinates when the image was sampled. Following figure shows the coordinate convention. Note that x ranges from 0 to M-1 and y from 0 to N-1 in integer increments.

6.4 Image as Matrices:

The preceding discussion leads to the following representation for a digitized image function:

$$\begin{array}{ccccccc}
 f(0, 0) & f(0, 1) & & \dots & & & f(0, N-1) \\
 f(1, 0) & f(1, 1) & & \dots & & & f(1, N-1) \\
 \vdots & \vdots & & \vdots & & \vdots & \vdots \\
 f(xylem) = & & & & & &
 \end{array}$$

$$f(M-1, 0) \ f(M-1, 1) \ \dots\dots\dots f(M-1, N-1)$$

The right side of this equation is a digital image by definition. Each element of this array is called an image element, picture element, pixel or pel. The terms image and pixel are used throughout the rest of our discussions to denote a digital image and its elements.

A digital image can be represented naturally as a MATLAB matrix:

$$f(1, 1) \ f(1, 2) \ \dots\dots\dots f(1, N)$$

$$f(2, 1) \ f(2, 2) \ \dots\dots\dots f(2, N)$$

$$f(M, 1) \ f(M, 2) \ \dots\dots\dots f(M, N)$$

Where $f(1, 1) = f(0, 0)$ (note the use of a monospace font to denote MATLAB quantities). Clearly the two representations are identical, except for the shift in origin. The notation $f(p, q)$ denotes the element located in row p and the column q . For example $f(6, 2)$ is the element in the sixth row and second column of the matrix f . typically we use the letters M and N respectively to denote the number of rows and columns in a matrix. A $1 \times N$ matrix is called a row vector whereas an $M \times 1$ matrix is called a column vector. A 1×1 matrix is a scalar.

Matrices in MATLAB are stored in variables with names such as A , a , RGB , $real$ array and so on. Variables must begin with a letter and contain only letters, numerals and underscores. As noted in the previous paragraph, all MATLAB quantities are written using monospace characters. We use conventional Roman, italic notation such as $f(x, y)$, for mathematical expressions

6.5 Reading Images:

Images are read into the MATLAB environment using function `imread` whose syntax is

```
imread('filename')
```

| Format name | Description | recognized extension |
|--------------------|--------------------------------|-----------------------------|
| TIFF | Tagged Image File Format | .tif, .tiff |
| JPEG | Joint Photograph Experts Group | .jpg, .jpeg |
| GIF | Graphics Interchange Format | .gif |
| BMP | Windows Bitmap | .bmp |
| PNG | Portable Network Graphics | .png |
| XWD | X Window Dump | .xwd |

Here filename is a string containing the complete of the image file(including any applicable extension).For example the command line

```
>> f = imread('8.jpg');
```

reads the JPEG (above table) image chestxray into image array f. Note the use of single quotes (') to delimit the string filename. The semicolon at the end of a command line is used by MATLAB for suppressing output. If a semicolon is not included.MATLAB displays the results of the operation(s) specified in that line. The prompt symbol(>>) designates the beginning of a command line, as it appears in the MATLAB command window.

When as in the preceding command line no path is included in filename, `imread` reads the file from the current directory and if that fails it tries to find the file in the

MATLAB search path. The simplest way to read an image from a specified directory is to include a full or relative path to that directory in filename.

For example,

```
>> f = imread( 'D:\myimages\chestxray.jpg');
```

reads the image from a folder called my images on the D: drive, whereas

```
>> f = imread( ' . \ myimages\chestxray .jpg');
```

reads the image from the my images subdirectory of the current of the current working directory. The current directory window on the MATLAB desktop toolbar displays MATLAB's current working directory and provides a simple, manual way to change it. Above table lists some of the most of the popular image/graphics formats supported by imread and imwrite.

Function size gives the row and column dimensions of an image:

```
>>size (f)
```

```
ans = 1024 * 1024
```

This function is particularly useful in programming when used in the following form to determine automatically the size of an image:

```
>>[M,N]=size(f);
```

This syntax returns the number of rows(M) and columns(N) in the image.

The whole function displays additional information about an array. For instance ,the statement

```
>>whos f
```

Gives

| Name | size | Bytes | Class |
|------|-----------|---------|-------------|
| F | 1024*1024 | 1048576 | unit8 array |

Grand total is 1048576 elements using 1048576 bytes

The unit8 entry shown refers to one of several MATLAB data classes. A semicolon at the end of a whose line has no effect ,so normally one is not used.

6.6 Displaying Images:

Images are displayed on the MATLAB desktop using function imshow, which has the basic syntax:

`imshow(f,g)`

Where f is an image array, and g is the number of intensity levels used to display it. If g is omitted ,it defaults to 256 levels .using the syntax

`Imshow (f, {low high})`

Displays as black all values less than or equal to low and as white all values greater than or equal to high. The values in between are displayed as intermediate intensity values using the default number of levels .Finally the syntax

`Imshow(f,[])`

Sets variable low to the minimum value of array f and high to its maximum value. This form of imshow is useful for displaying images that have a low dynamic range or that have positive and negative values.

Function `pixval` is used frequently to display the intensity values of individual pixels interactively. This function displays a cursor overlaid on an image. As the cursor is moved over the image with the mouse the coordinates of the cursor position and the corresponding intensity values are shown on a display that appears below the figure window. When working with color images, the coordinates as well as the red, green and blue components are displayed. If the left button on the mouse is clicked and then held pressed, `pixval` displays the Euclidean distance between the initial and current cursor locations.

The syntax form of interest here is `Pixval` which shows the cursor on the last image displayed. Clicking the X button on the cursor window turns it off.

The following statements read from disk an image called `rose_512.tif` extract basic information about the image and display it using `imshow` :

```
>>f=imread('rose_512.tif');
```

```
>>whos f
```

| Name | Size | Bytes | Class |
|------|---------|--------|-------------|
| F | 512*512 | 262144 | unit8 array |

Grand total is 262144 elements using 262144 bytes

```
>>imshow(f)
```

A semicolon at the end of an `imshow` line has no effect, so normally one is not used. If another image, `g`, is displayed using `imshow`, MATLAB replaces the image in the screen with the new image. To keep the first image and output a second image, we use function `figure` as follows:

```
>>figure ,imshow(g)
```

Using the statement

```
>>imshow(f),figure ,imshow(g) displays both images.
```

Note that more than one command can be written on a line ,as long as different commands are properly delimited by commas or semicolons. As mentioned earlier, a semicolon is used whenever it is desired to suppress screen outputs from a command line.

Suppose that we have just read an image `h` and find that using `imshow` produces the image. It is clear that this image has a low dynamic range, which can be remedied for display purposes by using the statement.

```
>>imshow(h,[ ])
```

6.7 WRITING IMAGES:

Images are written to disk using function `imwrite`, which has the following basic syntax:

```
Imwrite (f,'filename')
```

With this syntax, the string contained in `filename` must include a recognized file format extension .Alternatively, the desired format can be specified explicitly with a third input argument.

```
>>imwrite(f,'patient10_run1','tif')
```

Or alternatively

For example the following command writes `f` to a TIFF file named `patient10_run1`:

```
>>imwrite(f,'patient10_run1.tif')
```

If `filename` contains no path information, then `imwrite` saves the file in the current working directory.

The `imwrite` function can have other parameters depending on the file format selected. Most of the work in the following deals either with JPEG or TIFF images, so we focus attention here on these two formats.

More general `imwrite` syntax applicable only to JPEG images is

```
imwrite(f,'filename.jpg','quality',q)
```

where `q` is an integer between 0 and 100 (the lower number the higher the degradation due to JPEG compression).

For example, for `q=25` the applicable syntax is

```
>>imwrite(f,'bubbles25.jpg','quality',25)
```

The image for `q=15` has false contouring that is barely visible, but this effect becomes quite pronounced for `q=5` and `q=0`. Thus, an expectable solution with some margin for error is to compress the images with `q=25`. In order to get an idea of the compression achieved and to obtain other image file details, we can use function `imfinfo` which has syntax.

`Imfinfo filename`

Here `filename` is the complete file name of the image stored in disk.

For example,

```
>>imfinfo bubbles25.jpg
```

outputs the following information (note that some fields contain no information in this case):

Filename: 'bubbles25.jpg'

FileModDate: '04-jan-2003 12:31:26'

FileSize: 13849

Format: 'jpg'

Format Version: ' '

Width: 714

Height: 682

Bit Depth: 8

Color Depth: 'grayscale'

Format Signature: ' '

Comment: { }

Where file size is in bytes. The number of bytes in the original image is corrupted simply by multiplying width by height by bit depth and dividing the result by 8. The result is 486948. Dividing this file size gives the compression ratio: $(486948/13849)=35.16$. This compression ratio was achieved. While maintaining image quality consistent with the requirements of the appearance. In addition to the obvious advantages in storage space, this reduction allows the transmission of approximately 35 times the amount of un compressed data per unit time.

The information fields displayed by `imfinfo` can be captured in to a so called structure variable that can be for subsequent computations. Using the receding an example and assigning the name `K` to the structure variable.

We use the syntax `>>K=imfinfo('bubbles25.jpg');`

To store in to variable K all the information generated by command `imfinfo`, the information generated by `imfinfo` is appended to the structure variable by means of fields, separated from K by a dot. For example, the image height and width are now stored in structure fields `K.Height` and `K.Width`.

As an illustration, consider the following use of structure variable K to compute the compression ratio for `bubbles25.jpg`:

```
>> K=imfinfo('bubbles25.jpg');
```

```
>> image_bytes = K.Width* K.Height* K.Bit Depth /8;
```

```
>> Compressed_bytes = K.FileSize;
```

```
>> Compression_ratio=35.162
```

```
>>imwrite(f,'sf.tif','compression','none','resolution',.....[300 300])
```

the values of the vector `[columns rows]` were determined by multiplying 200 dpi by the ratio $2.25/1.5$, which gives 30 dpi. Rather than do the computation manually, we could write

```
>>res=round(200*2.25/1.5);
```

```
>>imwrite(f,'sf.tif','compression','none','resolution',res)
```

APPENDIX-C

```
function varargout = main(varargin)

% MAIN MATLAB code for main.fig

%     MAIN, by itself, creates a new MAIN or raises the existing
%     singleton*.

%
%     H = MAIN returns the handle to a new MAIN or the handle to
%     the existing singleton*.

%
%     MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
%     function named CALLBACK in MAIN.M with the given input arguments.

%
%     MAIN('Property','Value',...) creates a new MAIN or raises the existing
%     singleton*. Starting from the left, property value pairs are % applied to the
%     GUI before main_OpeningFcn gets called. An unrecognized property
%     name or invalid value makes property application stop. All inputs are
%     passed to main_OpeningFcn via varargin.

%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
%     one
%     instance to run (singleton)".

%

% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 16-Nov-2019 12:12:01
```



```

        % Begin initialization code - DO NOT EDIT

gui_Singleton = 1;

gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @main_OpeningFcn, ...
    'gui_OutputFcn', @main_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT


%--- Executes just before main is made visible. function main_OpeningFcn(hObject,
eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

```

```

% varargin command line arguments to main (see VARARGIN)

% Choose default command line output for main

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

% UIWAIT makes main wait for user response (see UIRESUME)

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line. function varargout =
main_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

% --- Executes on button press in pushbutton1. function pushbutton1_Callback(hObject,
eventdata, handles)

```

```

% hObject handle to pushbutton1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)


obj = VideoReader('real1_20sec.mp4');

vid = read(obj,[1 320]);

frames = obj.NumberOfFrames; %Read the Total number of frames and dispalyed in
command window

ST='.jpg';

%%ex)readind and writing the 100 frames

%if u want to change the values means replace upto 1:776

Vid1=vid(:,:,1);

imshow(Vid1),title('Background Image')


for x = 1:300

% Sx=num2str(x);

% Strc=strcat(Sx,ST);

pause(0.01)

    Vid=vid(:,:,x);

    axes(handles.axes1),

    imshow(Vid)

%          cd frames

```

```
%          imwrite(Vid,Strc);
```

```
%          cd ..
```

```
end
```

```
% --- Executes on button press in pushbutton2. function pushbutton2_Callback(hObject,  
 eventdata, handles)
```

```
% hObject handle to pushbutton2 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
obj = VideoReader('real1_20sec.mp4');
```

```
vid = read(obj,[1 300]);
```

```
frames = obj.NumberOfFrames;%Read the Total number of frames and dispalyed in  
command window
```

```
ST='.jpg';
```

```
%%ex)readind and writing the 100 frames
```

```
%if u want to change the values means replace upto 1:776
```

```
for x = 1:300
```

```
% Sx=num2str(x);
```

```
% Strc=strcat(Sx,ST);
```

```
pause(0.01)
```

```
    Vid=vid(:,:,x);
```

```
    Vid=rgb2gray(Vid);
```

```
    axes(handles.axes2), imshow(Vid)
```

```

%          cd frames
%          imwrite(Vid,Strc);
%          cd ..
end

```

% --- Executes on button press in pushbutton3.

```
function pushbutton3_Callback(hObject, eventdata, handles)
```

```
% hObject handle to pushbutton3 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA) load struct
```

```
for f=1:300
```

```
break
```

```
%%ex)readind and writing the 100 frames
```

```
%if u want to change the values means replace upto 1:776
```

```
% Sx=num2str(x);
```

```
% Strc=strcat(Sx,ST);
```

```
    Vid=vid(:, :, f);
```

```
    Vid=rgb2gray(Vid);
```

```
%%%%%%%% background subtraction
```

```
Vid=vid(:,:,f)-Vid1; %%%% subtracting background
```

```
[L,numOfvehicles(k-startAt+1)] = bwlabel(Vid,8);
```

```
s = regionprops(L, 'area'); sc = regionprops(L, 'centroid');
```

```
fgh = [];
```

```
cnt = 1;
```

```
for j = 1 : size(s,1)
```

```
    if s(j).Area > AREA_THR & ( (sc(j).Centroid(2) < 82) )& ( (sc(j).Centroid(2)>75))
```

```
        fgh(j) = 1;
```

```
        cent = sc(j).Centroid;
```

```
        L(find(L == j)) = cnt;
```

```
        cnt = cnt + 1;
```

```
        else
```

```
            fgh(j) = 0;
```

```
            cent = sc(j).
```

```
            Centroid; L(find(L == j)) = 0;
```

```
        end
```

```
end
```

```
L1 = L;
```

```
L1 = double(L1 > 0);
```

```
numOfvehicles1(k-startAt+1) = length(find(fgh));
```

```
boundBoxParams = princomp(regionprops(L, 'boundingBox'));
```

```
boundBoxParams_car = princomp(regionprops(L, 'boundingBox','car','r'));
```

```
boundBoxParams_2wheeler = princomp(regionprops(L, 'boundingBox','2wheeler','b')); boundBoxParams_auto = princomp(regionprops(L,
```

```

'boundingBox','auto','y')); boundBoxParams_bus = princomp(regionprops(L,
'boundingBox','bus/truck','g'));          fnumvehicles(k-startAt+1)      =
numOfvehicles1(k-startAt+1);

maask=uint8(zeros(Nr,Nc));

```

%labeling moving object

```

for nop = 1 : numOfvehicles1(k-startAt+1)

    cc = round(boundBoxParams(nop).BoundingBox);

    plot(cc(1):(cc(1)+cc(3)-1),cc(2),'r-');

    plot(cc(1):(cc(1)+cc(3)-1),cc(2)+cc(4),'r-');

    plot(cc(1), cc(2):(cc(2)+cc(4)-1),'r-');

    plot(cc(1)+cc(3), cc(2):(cc(2)+cc(4)-1),'r-');

    aspectRatio(1,nop) = cc(3)/cc(4);

    aask(cc(2):cc(2)+cc(4)-1,cc(1):cc(1)+cc(3)-1)=uint8(ones(cc(4),cc(3)));

    Lvec = sum(L1(cc(2):(cc(2)+cc(4)-1),cc(1):(cc(1)+cc(3)-1)),1);

end

plot([50 310],[180 120],'m');

plot([105 230],[145 120],'m');

title(strcat('count : ',num2str(length(track_veh))));

hold off;

s = regionprops(L, 'area');

sc = regionprops(L, 'centroid');

k=[track_veh s.Area];

```

```
speed1=(s.Area)./toc;
```

```
N2w=length( boundBoxParams_2wheeler);
```

```
Nc=length( boundBoxParams_car);
```

```
Nb=length( boundBoxParams_bus);
```

```
Na=length( boundBoxParams_auto);
```

```
end
```

```
k={a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18 a19 a20 a21 a22 a23  
a24 a25 a26 a27 a28 a29 a30 a31 a32 a33 a34 a35 a36 a37 a38 a39 a40 a41 a42 a43 a44  
a45 a46 a47 a48 a49 a50 a51 a52 a53 a54 a55 a56 a57 a58 a59 a60 a61 a62 a63 a64 a65  
a66 a67 a68 a69 a70 a71 a72 a73 a74 a75 a76 a77 a78 a79 a80 a81 a82 a83 a84 a85 a86  
a87 a88 a89 a90 a91 a92 a93 a94 a95 a96 a97 a98 a99 a100 a101 a102 a103 a104 a105  
a106 a107 a108 a109 a110 a111 a112 a113 a114 a115 a116 a117 a118 a119 a120 a121  
a122 a123 a124 a125 a126 a127 a128 a129 a130 a131 a132 a133 a134 a135 a136 a137  
a138 a139 a140 a141 a142 a143 a144 a145 a146 a147 a148 a149 a150 a151 a152 a153  
a154 a155 a156 a157 a158 a159 a160 a161 a162 a163 a164 a165 a166 a167 a168 a169  
a170 a171 a172 a173 a174 a175 a176 a177 a178 a179 a180 a181 a182 a183 a184 a185  
a186 a187 a188 a189 a190 a191 a192 a193 a194 a195 a196 a197 a198 a199 a200 a201  
a202 a203 a204 a205 a206 a207 a208 a209 a210 a211 a212 a213 a214 a215 a216 a217  
a218 a219 a220 a221 a222 a223 a224 a225 a226 a227 a228 a229 a230 a231 a232 a233  
a234 a235 a236 a237 a238 a239 a240 a241 a242 a243 a244 a245 a246 a247 a248 a249  
a250 a251 a252 a253 a254 a255 a256 a257 a258 a259 a260 a261 a262 a263 a264 a265  
a266 a267 a268 a269 a270 a271 a272 a273 a274 a275 a276 a277 a278 a279 a280 a281  
a282 a283 a284 a285 a286 a287 a288 a289 a290 a291 a292 a293 a294 a295 a296 a297  
a298 a299 a300};
```

```
sp=speed(k);
```

```
if (sp<=7)
```

```
    msgbox('Low Speed')
```

```
elseif (7<=sp<=11)
```



```

        msgbox('Average Speed')
elseif (sp>11)
        msgbox('OverSpeed')
end
for f=1:300
        pause(0.01)
        axes(handles.axes3), imshow(k{f})
        set(handles.edit6,'string',sp(f));
        % %   end
end

```

```

VOLUME=0.5*N2w+1*Nc+3*Nb+1*Na;
set(handles.edit1,'string',N2w)
set(handles.edit2,'string',Nc)
set(handles.edit3,'string',Nb)
set(handles.edit4,'string',Na)
total=N2w+Nc+Nb+Na;
set(handles.edit5,'string',total)
set(handles.edit7,'string',VOLUME)

```

```

function edit1_Callback(hObject, eventdata, handles)
% hObject handle to edit1 (see GCBO)

```

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text

% str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties. function
edit1_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),

get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in pushbutton4.

function pushbutton4_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton4 (see GCBO)

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)  
axes(handles.axes1),imshow([255])
```

```
% axes(handles.axes2),imshow([255])
```

```
axes(handles.axes3),imshow([255])
```

```
set(handles.edit1,'string','')
```

```
set(handles.edit2,'string','')
```

```
set(handles.edit3,'string','')
```

```
set(handles.edit4,'string','')
```

```
set(handles.edit5,'string','')
```

```
set(handles.edit6,'string','')
```

```
set(handles.edit7,'string','')
```

```
% --- Executes on button press in pushbutton5.
```

```
function pushbutton5_Callback(hObject, eventdata, handles)
```

```
% hObject handle to pushbutton5 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA) close
```

```
function edit2_Callback(hObject, eventdata, handles)
```

```
% hObject handle to edit2 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```

% Hints: get(hObject,'String') returns contents of edit2 as text
% str2double(get(hObject,'String')) returns contents of edit2 as a double

% --- Executes during object creation, after setting all properties.

function edit2_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

%     See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)

% hObject handle to edit3 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

```

```
% Hints: get(hObject,'String') returns contents of edit3 as text
% str2double(get(hObject,'String')) returns contents of edit3 as a double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function edit3_CreateFcn(hObject, eventdata, handles)
```

```
% hObject handle to edit3 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
%      See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
```

```
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function edit4_Callback(hObject, eventdata, handles)
```

```
% hObject handle to edit4 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit4 as text
```

```
%      str2double(get(hObject,'String')) returns contents of edit4 as a double
```

```

% --- Executes during object creation, after setting all properties.

function edit4_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit4 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit5_Callback(hObject, eventdata, handles)

% hObject handle to edit5 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit5 as text
%      str2double(get(hObject,'String')) returns contents of edit5 as a double

```

```

% --- Executes during object creation, after setting all properties.

function edit5_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit5 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called


% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit6_Callback(hObject, eventdata, handles)

% hObject handle to edit6 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of edit6 as text
%       str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.

function edit6_CreateFcn(hObject, eventdata, handles)

```

```
% hObject handle to edit6 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles empty - handles not created until after all CreateFcns called
```

Hint: edit controls usually have a white background on Windows.

```
%      See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
```

```
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function edit7_Callback(hObject, eventdata, handles)
```

```
% hObject handle to edit7 (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit7 as text
```

```
%      str2double(get(hObject,'String')) returns contents of edit7 as a double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function edit7_CreateFcn(hObject, eventdata, handles)
```

```
% hObject handle to edit7 (see GCBO)
```


% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

If ispc &&isequal(get(hObject,'BackgroundColor'),

get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end