A Summer Internship Project Report on

Vehicle Detection and Counting using Matlab

submitted to Dept. of ECE

By

Dama Rupesh Babu(R151737)

In partial fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the guidance of

Mr.Venkateshwarulu Naik (Assistant Professor)

Rajiv Gandhi University of Knowledge Technologies-AP

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Internship Project entitled "**Vehicle Detection and Counting using Matlab"** submitted by **D.RUPESH BABU(R151737).** Under the guidance of **Mr.VENKATESHWARULU NAIK** in partial fulfilment of the academic requirements for the award of degree of **Bachelor of Technology** in **Electronics and Communication Engineering** by **RGUKT,R K VALLEY** during the academic year 2019-2020.

Internal Guide

Mr.Venkateshwarulu Naik,

Assistant Professor,

Dept. of ECE,RGUKT,

RK Valley.

Head of the Department

Ms. G.Lakshmi Shireesha,

Assistant Professor,

Dept. of ECE,RGUKT,

RK Valley.

DECLARATION

I **Dama Rupesh Babu** hereby declare that this report entitled **"Vehicle Detection and Counting using Matlab"** submitted by me under the guidance and supervision of **Mr.Venkateshwarulu Naik** is a bonafide work. I also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma.

Date: 21-11-2019 D.Rupesh Babu(R151737)

Place:R K Valley

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Mr.Venkateshwarulu Naik** , our project internal guide for giving valuable suggestions and shown keen interest throughout the progress of our project.I am grateful to , HOD ECE, for providing excellent computing facilities and congenial atmosphere for progressing with my project. At the outset, I would like to thank Rajiv Gandhi University of Knowledge Technologies, R.K Valley for providing all the necessary resources for the successful completion of our project.

With sincere regards,

D.Rupesh Babu(R151737)

Contents:

1.Abstract:

Vehicle Detection and Counting using Matlab can give accurate results about the traffic density which can help in monitoring and managing the traffic. It has more advancements than using hardware devices like sensors as they easily get manipulated by external factors like temperature and climatic conditions. Various image processing techniques like segmentation, filtering, background subtraction, morphological operations, blob detection, etc. are used and the final count of the number of vehicles passed through the path of choice will be displayed on image using MATLAB.

2.Introduction:

Traffic management and information system mostly depend on sensors for estimating traffic count. Now-a-days magnetic loop detectors are used to count vehicles. Vision based systems have a number of advantages. Cameras are much less disruptive to install than magnetic sensors . The output from the camera based system can give information about the traffic density on a particular road and can warn about traffic if it arises. This project is mostly aimed at counting and classifying vehicles at highway from data obtained from the camera. In case of heavy traffic in a lane the vehicles can be redirected and this would maintain a better traffic flow. The vehicles passing in each lane can be counted and classified based on the area of pixels obtained from images of vehicles. This method is inexpensive and requires little or almost no human intervention.

The frames in a video are subjected to adaptive background subtraction against an background image. The result of background subtraction gives us an image with only the vehicles in it which in turn can be counted with the blob detection approach, which gives us the vehicle count for a particular frame. For each blob that is obtained, the blob parameters (height, length, width, profile, area) are determined, which can be used to classify a vehicle.

3.Software Description:

3.1.Matlab:

MATLAB is a multi-paradigm programming and numerical computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs. MATLAB is a programming platform designed specifically for engineers and scientists. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.
The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation.
MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

4.Products Used:

4.1.Image Processing Toolbox:

Image Processing Toolbox provides a comprehensive set of reference-standard algorithms and workflow apps for image processing, analysis, visualization, and algorithm development. You can perform image segmentation, image enhancement, noise reduction, geometric transformations, image registration, and 3D image processing.

Image Processing Toolbox apps let you automate common image processing workflows. You can interactively segment image data, compare image registration techniques, and batch-process large data sets. Visualization functions and apps let you explore images, 3D volumes, and videos; adjust contrast; create histograms; and manipulate regions of interest (ROIs).

You can accelerate your algorithms by running them on multicore processors and GPUs. Many toolbox functions support C/C++ code generation for desktop prototyping and embedded vision system deployment.

Import images and video generated by a wide range of devices, including webcams, digital cameras, satellite and airborne sensors, medical imaging devices, microscopes, telescopes, and other scientific instruments. Support for a number of specialized image file formats. For medical images, it supports DICOM files.

Applications of Image Processing Toolbox are:

- Apps for Exploration and Discovery

Use apps to explore and discover various algorithmic approaches. With the Color Thresholder app, you can segment an image based on various color spaces. The Image Viewer app lets you interactively place and manipulate ROIs, including points, lines, rectangles, polygons, ellipses, and freehand shapes.

- Apps for Exploration and Discovery

  Use apps to explore and discover various algorithmic approaches. With the Color Thresholder app, you can segment an image based on various color spaces. The Image Viewer app lets you interactively place and manipulate ROIs, including points, lines, rectangles, polygons, ellipses, and freehand shapes.

- 3D Visualization

  Explore a 3D volume by using different visualization methods to explore the structure of the data. You can map the pixel intensity of a 3D volume to opacity to highlight a specific region within the volume.

- 3D Processing

  Use many 3D-specific functions in addition to ND functions that enable complete image processing workflows with 3D data.

- 3D Segmentation

  Use programmatic functions and interactive apps to perform 3D segmentation. You can use thresholding, active contours, semantic segmentation and other techniques to perform segmentation of 3D Data.

- Edge Detection

  Identify object boundaries in an image using pre-built algorithms. These algorithms include the Sobel, Prewitt, Roberts, Canny, and Laplacian of Gaussian methods.

- Image Region Analysis

  Calculate the properties of regions in images, such as area, centroid, and orientation. Use the Image Region Analysis App to automatically count, sort, and remove regions based on properties.

- Hough Transform, Statistical Functions, and Color Space Conversions

  Find line segments, line endpoints, and circles. Statistical functions let you analyze the characteristics of an image. Color-space conversion accurately represents color independently from devices.

- Edge Detection

  Identify object boundaries in an image using pre-built algorithms. These algorithms include the Sobel, Prewitt, Roberts, Canny, and Laplacian of Gaussian methods.

- Image Region Analysis

  Calculate the properties of regions in images, such as area, centroid, and orientation. Use the Image Region Analysis App to automatically count, sort, and remove regions based on properties.

- Hough Transform, Statistical Functions, and Color Space Conversions

  Find line segments, line endpoints, and circles. Statistical functions let you analyze the characteristics of an image. Color-space conversion accurately represents color independently from devices.

4.2.Computer Vision Toolbox:

Computer Vision Toolbox provides algorithms, functions, and apps for designing and testing computer vision, 3D vision, and video processing systems. You can perform object detection and tracking, as well as feature detection, extraction, and matching. For 3D vision, the toolbox supports single, stereo, and fisheye camera calibration; stereo vision; 3D reconstruction; and lidar and 3D point cloud processing. Computer vision apps automate ground truth labeling and camera calibration workflows.

You can train custom object detectors using deep learning and machine learning algorithms. For semantic segmentation you can use deep learning algorithms such as SegNet, U-Net, and DeepLab. Pretrained models let you detect faces, pedestrians, and other common objects.

You can accelerate your algorithms by running them on multicore processors and GPUs. Most toolbox algorithms support C/C++ code generation for

integrating with existing code, desktop prototyping, and embedded vision system deployment.

Some Applications of Computer Vision Tool box:

- Object Detection and Recognition
- Semantic Segmentation
- Ground Truth Labeling
- Single Camera Calibration

  Automate checkerboard detection and calibrate pinhole and fisheye cameras using the Camera Calibrator app.

- Stereo Camera Calibration

  Calibrate a stereo pair to compute depth and reconstruct 3D scenes.

- 3D Vision

  Structure from motion and visual odometry.

- Stereo Vision

  Estimate depth and reconstruct a 3D scene using a stereo camera pair.

- Feature Detection, Extraction, and Matching

  Detect, extract, and match interesting features such as blobs, edges, and corners across multiple images.

- Feature-Based Image Registration

  Match features across multiple images to estimate geometric transforms between images and register image sequences.

- Object Tracking

  Track object trajectories from frame to frame in video sequences.

- Motion Estimation

  Estimate motion between video frames using optical flow, block matching, and template matching.

5.Image Processing Techniques:

5.1.Background Subtraction:

Background subtraction is a method of obtaining the foreground mask from an image after eliminating the stationary background. We propose to use adaptive background subtraction for the same. Most researchers have abandoned nonadaptive methods of background subtraction because of the need for manual initialization. Without re-initialization, errors in the background accumulate over time, making this method useful only in highly-supervised, short-term tracking applications without significant changes in the scene. A standard method of adaptive background subtraction is averaging the images over time, creating a background approximation which is similar to the current static scene except where motion occurs. While this is effective in situations where objects move continuously and the background is visible a significant portion of the time, it is not robust to scenes with many moving objects particularly if they move slowly. It also cannot handle bimodal backgrounds, recovers slowly when the background is uncovered, and has a single, predetermined threshold for the entire scene.

5.2.Filtering:

Image filtering and enhancement is the process of adjusting images so that the results are more suitable for display or further image analysis. For example, you can remove noise, sharpen, or adjust the contrast of an image, making it easier to identify key features.

Some of the Techniques are:

- Contrast adjustment
- Morphological filtering
- Deblurring
- ROI-based processing

Contrast Adjustment can be done by:

- Contrast adjustment
- Histogram equalization
- Decorrelation stretching

5.3.Morphological Operations:

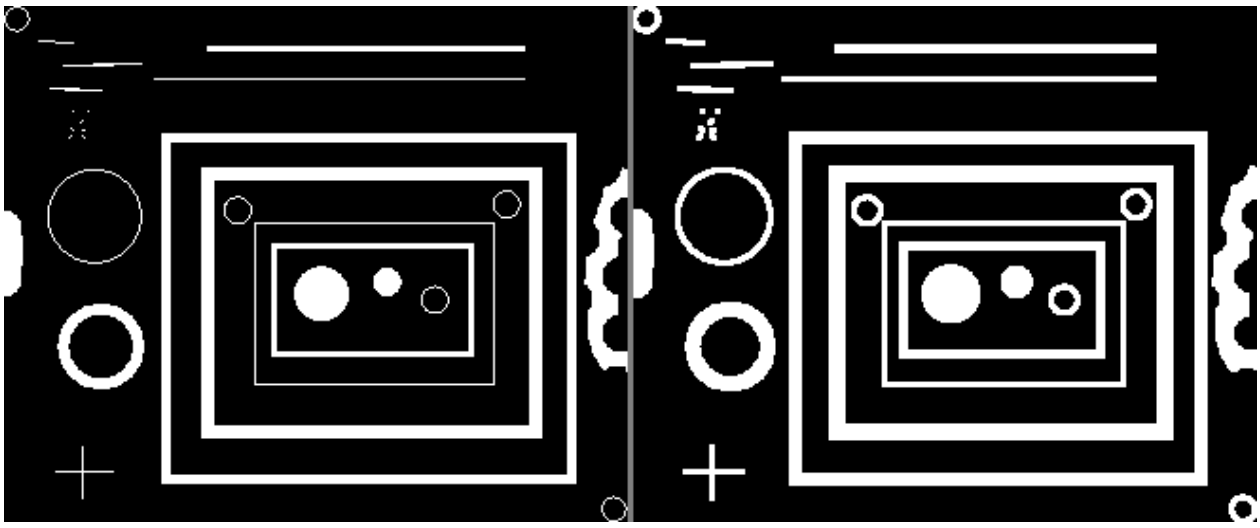
Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors.

Morphological Dilation and Erosion:

The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion. This table lists the rules for both dilation and erosion.

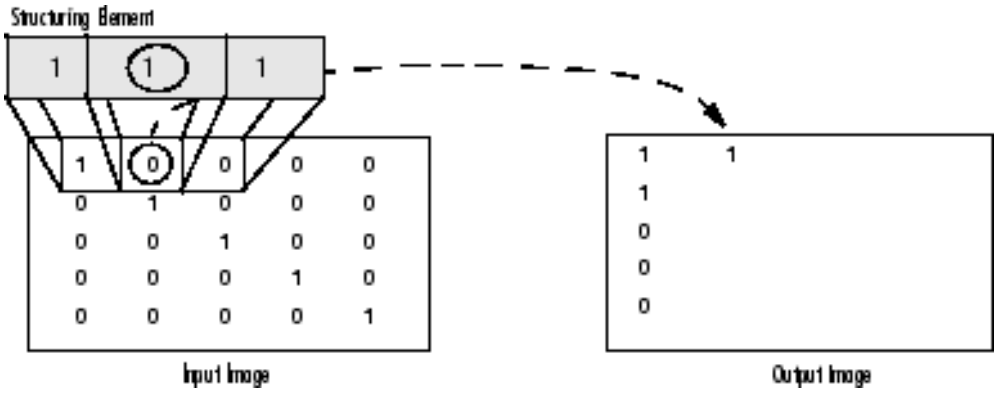
5.3.1.Dilation:

The value of the output pixel is the *maximum* value of all pixels in the neighborhood. In a binary image, a pixel is set to 1 if any of the neighboring pixels have the value 1.Morphological dilation makes objects more visible and fills in small holes in objects.



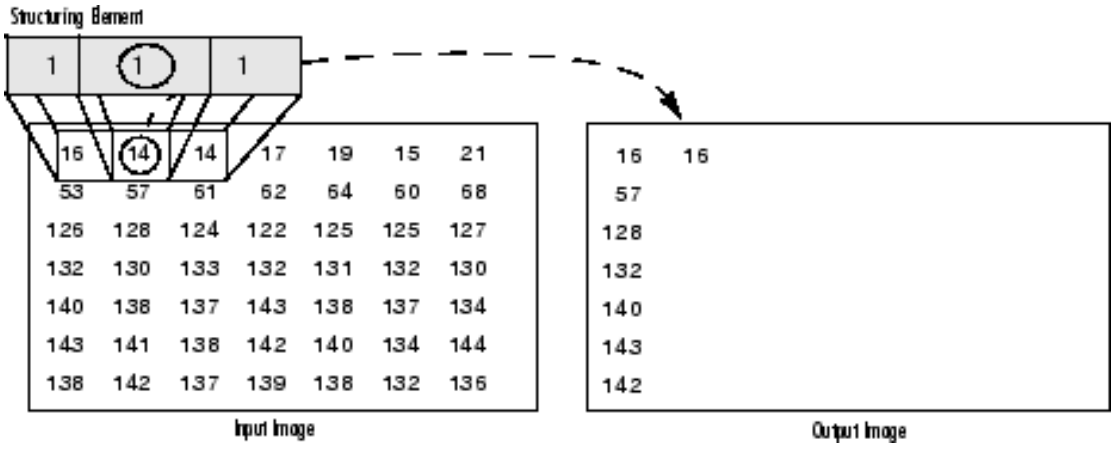
The following figure illustrates the dilation of a binary image. Note how the structuring element defines the neighborhood of the pixel of interest, which is circled. The dilation function applies the appropriate rule to the pixels in the neighborhood and assigns a value to the corresponding pixel in the output image. In the figure, the morphological dilation function sets the value of the output pixel to 1 because one of the elements in the neighborhood defined by the structuring element is on.

Morphological Dilation of a Binary Image



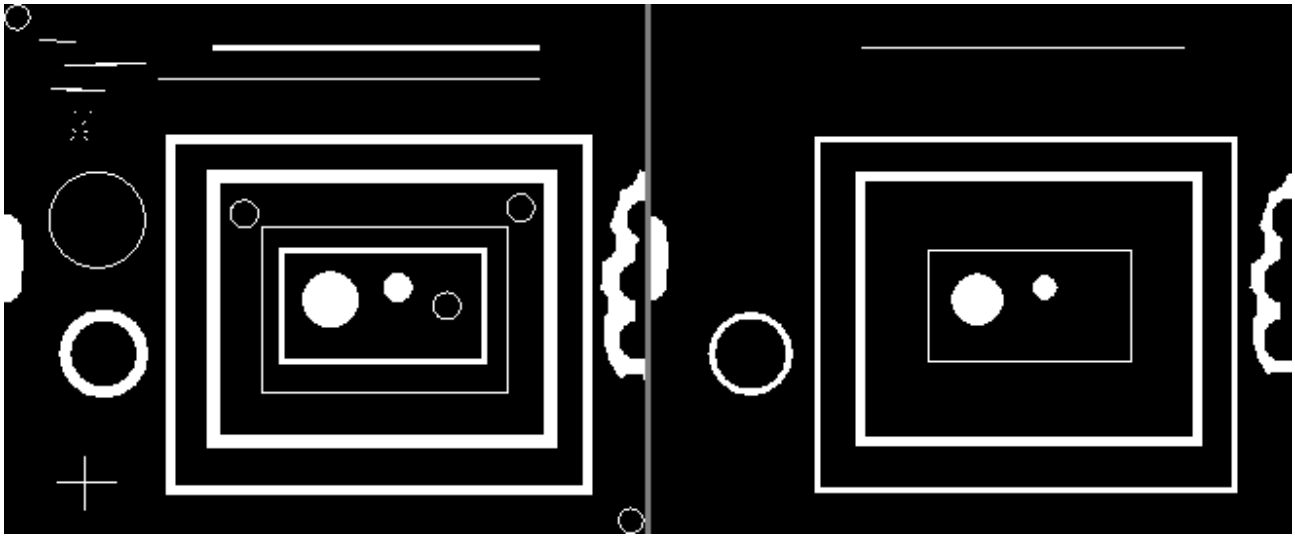
The following figure illustrates this processing for a grayscale image. The figure shows the processing of a particular pixel in the input image. Note how the function applies the rule to the input pixel's neighborhood and uses the highest value of all the pixels in the neighborhood as the value of the corresponding pixel in the output image.

Morphological Dilation of a Grayscale Image



5.3.2.Erosion:

The value of the output pixel is the *minimum* value of all pixels in the neighborhood. In a binary image, a pixel is set to 0 if any of the neighboring pixels have the value 0.Morphological erosion removes islands and small objects so that only substantive objects remain.



Structuring Elements:

An essential part of the morphological dilation and erosion operations is the structuring element used to probe the input image. A structuring element is a matrix that identifies the pixel in the image being processed and defines the neighborhood used in the processing of each pixel. You typically choose a structuring element the same size and shape as the objects you want to process in the input image. For example, to find lines in an image, create a linear structuring element.

There are two types of structuring elements: flat and nonflat. A flat structuring element is a binary valued neighborhood, either 2-D or multidimensional, in which the true pixels are included in the morphological computation, and the false pixels are not. The center pixel of the structuring element, called the origin, identifies the pixel in the image being processed. Use the strel function to create a flat structuring element. You can use flat structuring elements with both binary and grayscale images. The following figure illustrates a flat structuring element.

A nonflat structuring element is a matrix of type double that identifies the pixel in the image being processed and defines the neighborhood used in the processing of that pixel. A nonflat structuring element contains finite values used as additive offsets in the morphological computation. The center pixel of the matrix, called the origin, identifies the pixel in the image that is being processed. Pixels in the neighborhood with the value -Inf are not used in the computation. Use the offsetstrel function to create a nonflat structuring element. You can use nonflat structuring elements only with grayscale images.

To enhance performance, the strel and offsetstrel functions might break structuring elements into smaller pieces, a technique known as structuring element decomposition.

For example, dilation by an 11-by-11 square structuring element can be accomplished by dilating first with a 1-by-11 structuring element, and then with an 11-by-1 structuring element. This results in a theoretical speed improvement of a factor of 5.5, although in practice the actual speed improvement is somewhat less.

Structuring element decompositions used for the 'disk' and 'ball' shapes are approximations; all other decompositions are exact. Decomposition is not used with an arbitrary structuring element unless it is a flat structuring element whose neighborhood matrix is all 1's.

To see the sequence of structuring elements used in a decomposition, use the decompose method. Both strel objects and offsetstrel objects support decompose methods. The decompose method returns an array of the structuring elements that form the decomposition. For example, here are the structuring elements created in the decomposition of a diamond-shaped structuring element.

```
SE = strel('diamond',4)
```

An offsetstrel object represents a nonflat morphological *structuring element*, which is an essential part of morphological dilation and erosion operations.

A nonflat structuring element is a matrix that identifies the pixel in

- **Syntax**

  SE = offsetstrel(offset)

  SE = offsetstrel('ball',r,h)

  SE = offsetstrel('ball',r,h,n)

- **Description**

  SE = offsetstrel(offset) creates a nonflat structuring element with the additive offset specified in the matrix offset.

  SE = offsetstrel('ball',r,h) creates a nonflat, ball-shaped structuring element whose radius in the *x-y* plane is r and whose maximum offset height is h. For improved performance, offsetstrel approximates this shape by a sequence of eight nonflat line-shaped structuring elements.

  SE = offsetstrel('ball',r,h,n) creates a nonflat ball-shaped structuring element, where n specifies the number of nonflat, line-shaped structuring elements that offsetstrel uses to approximate the shape. Morphological operations using ball approximations run much faster when you specify a value for n greater than 0.

5.4.Blob Detection:

In Image processing, blob detection refers to modules that are aimed at detecting points and regions in the image that differ in properties like brightness or color compared to the surrounding. There are several motivations for studying and developing blob detectors. One main reason is to provide complementary information about regions, which is not obtained from edge detectors or corner detectors. It is used to obtain regions of interest for further processing. These regions could signal the presence of objects or parts of objects in the image domain with application to object recognition and/or object tracking.

Blob detection is usually done after color detection and noise reduction to finally find the required object from the image. A lot of unimportant blobs of the required color may be present in the picture and also we need the details of the blob-like the blob center, blob corners and all for which it is required

that we know the exact coordinates of all the pixels in which the required blob is present in.

vision.BlobAnalysis

Properties of connected regions are detected by using BlobAnalysis

Description

To compute statistics for connected regions in a binary image

To track a set of points:

1. Create the vision.BlobAnalysis object and set its properties.
2. Call the object with arguments, as if it were a function.

- **Syntax**

  Hblob = vision.BlobAnalysis

  Hblob = vision.BlobAnalysis(Name,Value)

- **Description**

  Hblob = vision.BlobAnalysis returns a blob analysis object, H, used to compute statistics for connected regions in a binary image.

  Hblob = vision.BlobAnalysis(Name,Value) sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, Hblob = vision.BlobAnalysis('AreaOutputPort',true)

- Properties

  Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them..

5.5.Counting and Representations:

- Inserting shapes in image or video:

  Syntax:

  RGB = insertShape(I,shape,position)

  RGB = insertShape(___,Name,Value)

  Description:

  RGB = insertShape(I,shape,position) returns a truecolor image with shape inserted. The input image, I, can be either a truecolor or grayscale image. You draw the shapes by overwriting pixel values.

  RGB = insertShape(___,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

- Inserting text in image or video:

  Syntax:

  RGB = insertText(I,position,text)

  RGB = insertText(I,position,numericValue)

  RGB = insertText(___,Name,Value)

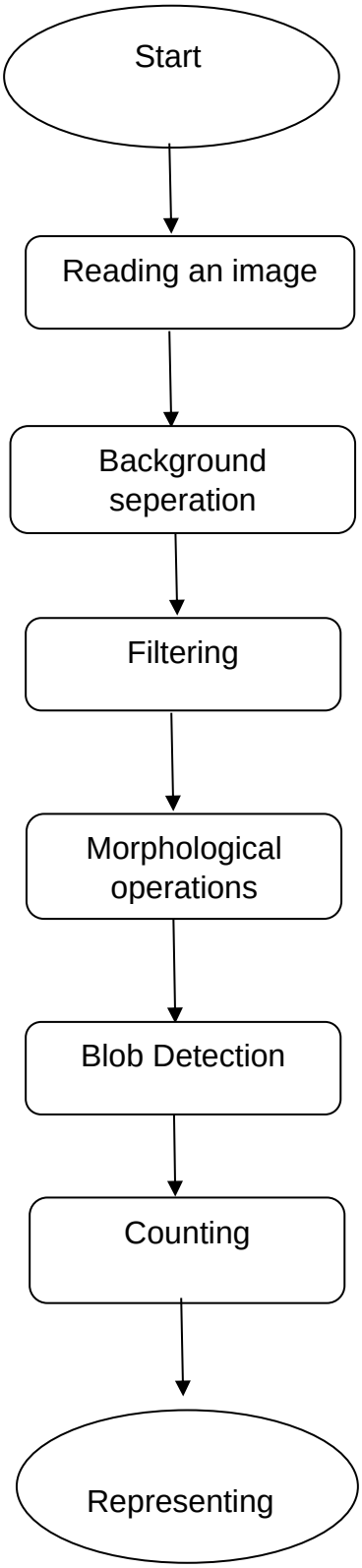
  Description:

  RGB = insertText(I,position,text) returns a truecolor image with text inserted. The input image, I, can be either a truecolor or grayscale image.

  RGB = insertText(I,position,numericValue) returns a truecolor image with numeric values inserted.

  RGB = insertText(___,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

6.Flow Chart:


Start
Reading an image
Background seperation
Filtering
Morphological operations
Blob Detection
Counting
Representing

7.MATLAB Code:

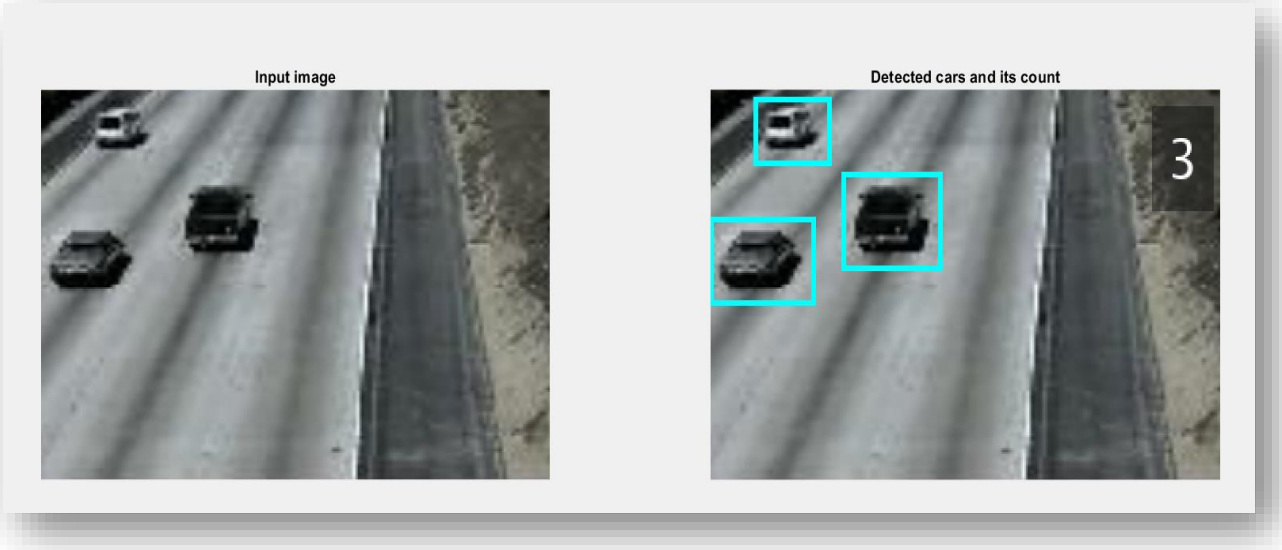
```
%Reading the images
road=imread('road2.png');
size_r=size(road);
carr=imread('car2.png');
size_c=size(carr);


%Background Seperation
back=ones(size_c);
back=im2uint8(back);

for i=1:size_r(1)
    for j=1:size_r(2)
        back(i,j,:)=road(i,j,:)-carr(i,j,:);
    end
end



%Converting RGB image to Grayscale
gry=ones(size_r(1),size_r(2));
gry=im2uint8(gry);


for i=1:size_r(1)
    for j=1:size_r(2)
        R=back(i,j,1);
        G=back(i,j,2);
        B=back(i,j,3);

%Intensity values : 0.2989 * R + 0.5870 * G + 0.1140 * B
        gry(i,j)=0.2989 * R + 0.5870 * G + 0.1140 * B;
    end
end


%Adjusting contrast
gry_adj=imadjust(gry);



%Morphological Operations: Erosion and Dilation
se = strel('square',5);
gry_erode= imerode(gry_adj,se);
```

```
se = strel('square',20);
gry_dilate= imdilate(gry_erode,se);


%Converting resulted image to Binary
gry_dilate=im2double(gry_dilate);
bin_img = im2bw(gry_dilate,0.4);


%Blob Analysis
blobAnalysis =
vision.BlobAnalysis('BoundingBoxOutputPort',
true,'AreaOutputPort', true, 'CentroidOutputPort',
true,'MinimumBlobArea', 150);
[area,centroid,bbox]=step(blobAnalysis, bin_img);


%Inserting shape on car
blob_img=insertShape(carr, 'Rectangle', bbox, 'Color',
'cyan','LineWidth',3);


%Representing no.of Cars
num_cars = size(bbox, 1);
res_img = insertText(blob_img,[size_r(2)-40 10],
num_cars,'FontSize',
30,'BoxColor','black','BoxOpacity',0.4,'TextColor','white
');


%Showing no.of Cars
figure;
subplot(1,2,1);
imshow(carr);
title('Input image');

subplot(1,2,2);
imshow(res_img);
title('Detected cars and its count');
```

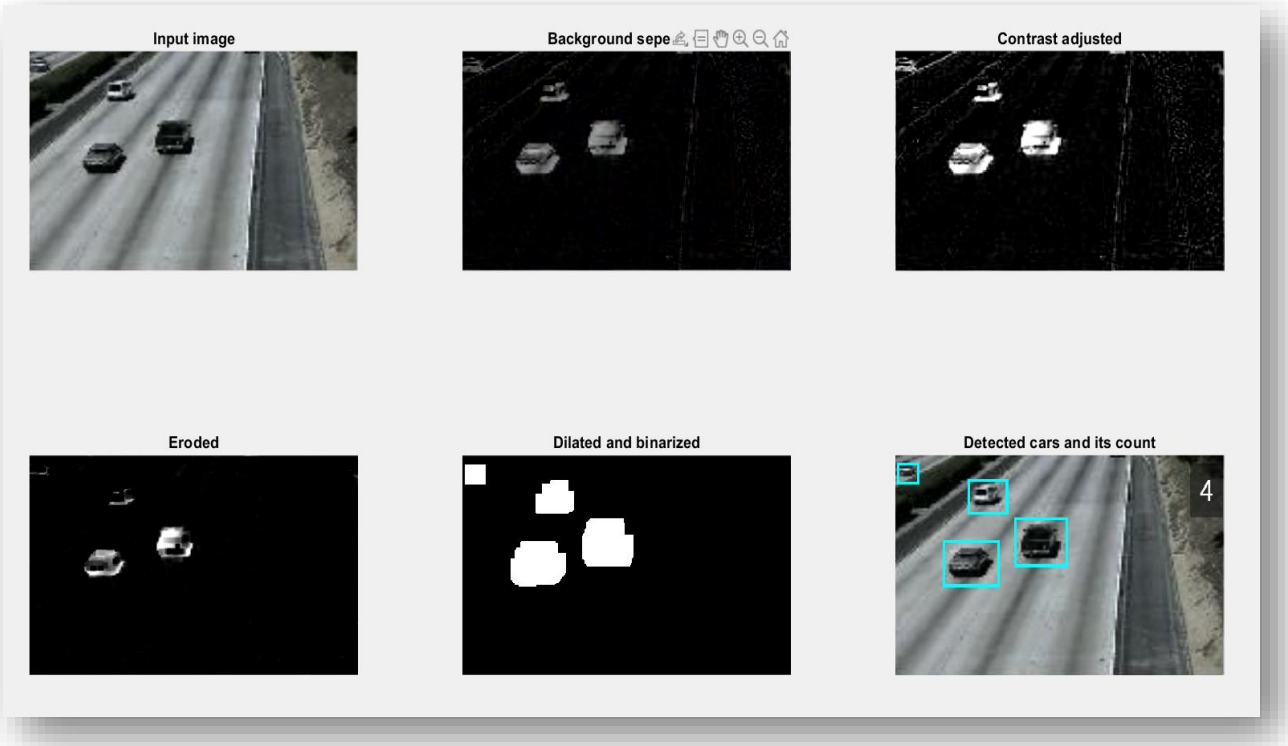
```
%Showing different image processing processes
figure;
subplot(2,3,1);
imshow(carr);
title('Input image');

subplot(2,3,2);
imshow(back);
title('Background seperated');

subplot(2,3,3);
imshow(gry_adj);
title('Contrast adjusted');

subplot(2,3,4);
imshow(gry_erode);
title('Eroded')

subplot(2,3,5);
imshow(gry_dilate);
title('Dilated');

subplot(2,3,6);
imshow(res_img);
title('Detected cars and its count');
```

8.Result:

Vehicles detected and counted:



Processes involved:



9.Conclusion:

This project produces multi domain outputs. It can count and classify vehicles on highways by the methods mentioned above and help with highway management, it can calculate traffic density on busy traffic roads for better monitoring. It is beneficial for better traffic management than the other alternatives which uses the sensors.

10.References:

1. Surendra Gupte, Osama Masoud, Robert F. K. Martin, and Nikolaos P. Papanikolopoulos, “Detection and Classification of Vehicles” IEEE Transactions on intelligent transportation systems, VOL. 3, NO. 1, March 2002.
2. S. Messelodi, C. M. Modena, and M. Zanin, “A computer vision system for the detection and classification of vehicles at urban road intersections,” Pattern Analysis & Applications, vol. 8, no. 1, pp. 17–31, 2005.
3. H. Samet and M. Tamminen (1988). "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees". IEEE Transactions on Pattern Analysis and Machine Intelligence (TIEEE Trans. Pattern Anal. Mach. Intell.)