CHAPTER 1: INTRODUCTION

In modern life we have to face with many problems one of which is traffic congestion becoming more serious day after day. It is said that the high tome of vehicles, the scanty infrastructure and the irrational distribution of the development are main reasons for augmented traffic jam. The major cause leading to traffic jam is the high number of vehicle which was caused by the population and the development of economy. To unravel this problem, the government should encourage people to use public transport or vehicles with small size such as bicycles or make tax on personal vehicles. Particularly, in some Asian countries such as Viet Nam, the local authorities passed law limiting to the number of vehicles for each family. The methods mentioned above is really efficient in fact. That the inadequate infrastructure cannot handle the issue of traffic is also a decisive reason. The public conveyance is available and its quality is very bad, mostly in the establishing countries. Besides, the highway and roads are incapable of meeting the requirement of increasing number of vehicle. Instead of working on roads to accommodate the growing traffic various techniques have been devised to control the traffic on roads like embedded controllers that are installed at the junction. These techniques are briefly described in next section.

1.1 Standard Traffic Control Systems:

1.1.1 Manual Controlling

Manual controlling the name instance it require man power to control the traffic. Depending on the countries and states the traffic polices are allotted for a required area or city to control traffic. The traffic polices will carry sign board, sign light and whistle to control the traffic. They will be instructed to wear specific uniforms in order to control the traffic.

1.1.2 Automatic Controlling

Automatic traffic light is controlled by timers and electrical sensors. In traffic light each phase a constant numerical value loaded in the timer. The lights are automatically getting ON and OFF depending on the timer value changes. While using electrical sensors it will capture the availability of the vehicle and signals on each phase, depending on the signal the lights automatically switch ON and OFF.

1.2 Drawbacks:

In the manual controlling system we need more man power. As we have poor strength of traffic police we cannot control traffic manually in all area of a city or town. So we need a better solution to control the traffic. On the other side, automatic traffic controlling a traffic light uses timer for every phase. Using electronic sensors is another way in order to detect vehicles, and produce signal that to this method the time is being wasted by a green light on an empty road. Traffic congestion also occurred while using the electronic sensors for controlling the traffic. All these drawbacks are supposed to be eliminated by using image processing.

1.3 Image Processing in Traffic Light Control

We propose a system for controlling the traffic light by image processing. The vehicles are detected by the system through images instead of using electronic sensors embedded in the pavement. A camera will be placed alongside the traffic light. It will capture image sequences. Image processing is a better technique to control the state change of the traffic light. It shows that it can decrease the traffic congestion and avoids the time being wasted by a green light on an empty road. It is also more reliable in estimating vehicle presence because it uses actual traffic images. It visualizes the practicality, so it functions much better than those systems that rely on the detection of the vehicles' metal content.

1.4 Introduction to Image Processing

Image Processing is a technique to enhance raw images received from cameras/sensors placed on space probes, aircrafts and satellites or pictures taken in normal day-today life for various applications. An Image is rectangular graphical object. Image processing involves issues related to image representation, compression techniques and various complex operations, which can be carried out on the image data. The operations that come under image processing are image enhancement operations such as sharpening, blurring, brightening, edge enhancement etc. Image processing is any form of signal processing for which the input is an image, such as photographs or frames of video; the output of image processing can be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. Image processing usually refers to digital image processing, but optical and analog image processing are also possible.

CHAPTER 2

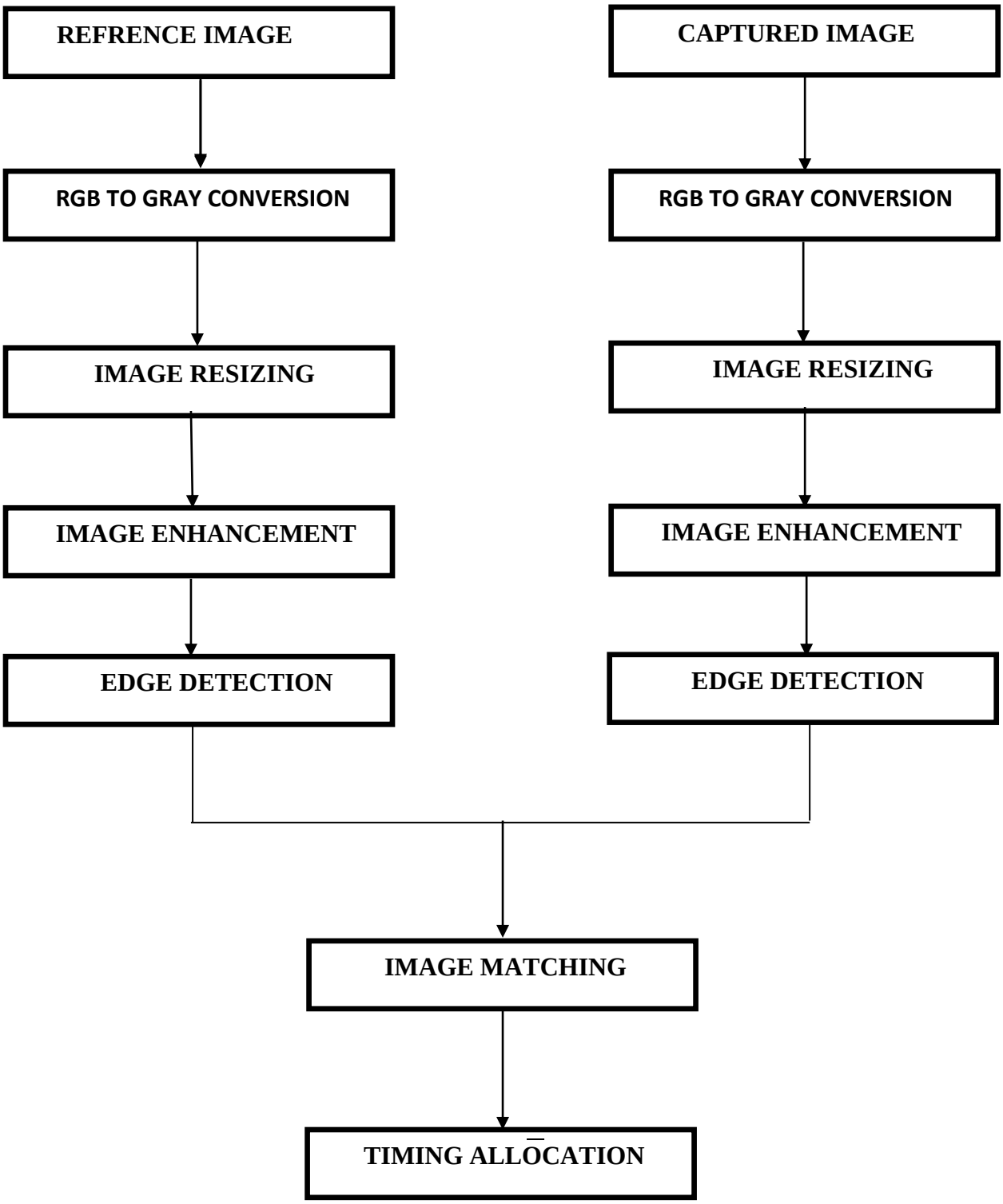
DETAILED DESCRIPTION OF OUR PROJECT

Many techniques have been developed in Image Processing during the last four to five decades. Most of the methods are developed for enhancing images obtained from unmanned space probes, spacecrafts and military reconnaissance flights. Image Processing systems are becoming widely popular due to easy availability of powerful personnel computers, large memory devices, graphics softwares and many more.

Image processing involves issues related to image representation, compression techniques and various complex operations, which can be carried out on the image data. The operations that come under image processing are image enhancement operations such as sharpening, blurring, brightening, edge enhancement .Traffic density of lanes is calculated using image processing which is done of images of lanes that are captured using digital camera. We have chosen image processing for calculation of traffic density as cameras are very much cheaper than other devises such as sensors.

Making use of the above mentioned virtues of image processing we propose a technique that can be used for traffic control. The block diagram of the proposed algorithm is given on next page.

2.1 BLOCK DIAGRAM



Block Diagram of "Traffic Control Using Image Processing" (proposed algorithm)

The Block diagram above gives an overview of how traffic will be controlled using image processing. Various boxes in Block diagram are explained below:

.

2.1.1 Image Acquisition:

Generally an image is a two-dimensional function f(x,y)(here x and y are plane coordinates).The amplitude of image at any point say f is called intensity of the image. It is also called the gray level of image at that point. We need to convert these x and y values to finite discrete values to form a digital image. The input image is a fundus taken from stare data base and drive data base. The image of the retina is taken for processing and to check the condition of the person. We need to convert the analog image to digital image to process it through digital computer. Each digital image composed of a finite elements and each finite element is called a pixel.

2.1.2 Formation of Image:

We have some conditions for forming an image f(x,y) as values of image are proportional to energy radiated by a physical source. So f(x,y) must be nonzero and finite.
i.e. **$0< f(x,y) < \infty$.**

2.1.3 Image Pre-Processing:

2.1.3.1 Image Resizing/Scaling:

Image scaling occurs in all digital photos at some stage whether this be in Bayer demosaicing or in photo enlargement. It happens anytime you resize your image from one pixel grid to another. Image resizing is necessary when you need to increase or decrease the total number of pixels. Even if the same image resize is performed, the result can vary significantly depending on the algorithm.

Images are resized because of number of reasons but one of them is very important in our project. Every camera has its resolution, so when a system is designed for some camera specifications it will not run correctly for any other camera depending on specification similarities. so it is necessary to make the resolution constant for the application and hence perform image resizing.

2.1.3.2 RGB to GRAY Conversion:

Humans perceive colour through wavelength-sensitive sensory cells called cones. There are three different varieties of cones, each has a different sensitivity to electromagnetic radiation (light) of different wavelength. One cone is mainly sensitive to green light, one to red light, and one to blue light. By emitting a restricted combination of these three colours (red, green and blue), and hence stimulate the three types of cones at will, we are able to generate almost any detectable colour. This is the reason behind why colour images are often stored as three separate image matrices; one storing the amount of red (R) in each pixel, one the amount of green (G) and one the amount of blue (B). We call such colour images as stored in an RGB format. In grayscale images, however, we do not differentiate how much we emit of different colours, we emit the same amount in every channel. We will be able to differentiate the total amount of emitted light for each pixel; little light gives dark pixels and much light is perceived as bright pixels. When converting an RGB image to grayscale, we have to consider the RGB values for each pixel and make as output a single value reflecting the brightness of that pixel. One of the approaches is to take the average of the contribution from each channel: (R+B+C)/3. However, since the perceived brightness is often dominated by the green component, a different, more "human-oriented", method is to consider a weighted average, e.g.: 0.3R + 0.59G + 0.11B.

2.1.4 Image Enhancement

Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further analysis. For example, we can eliminate noise, which will make it more easier to identify the key characteristics.

In poor contrast images, the adjacent characters merge during binarization. We have to reduce the spread of the characters before applying a threshold to the word image. Hence, we introduce "**POWER- LAW TRANSFORMATION**" which increases the contrast of the characters and helps in better segmentation. The basic form of power-law transformation is

$$s = cr^{\gamma},$$

where r and s are the input and output intensities, respectively; c and γ are positive constants. A variety of devices used for image capture, printing, and display respond according to a power-law. By convention, the exponent in the power-law equation is referred to as gamma. Hence, the process used to correct these power-law response phenomena is called gamma correction. Gamma correction is important, if displaying an image accurately on a computer screen is of concern. In our experimentation, γ is varied in the range of 1 to 5. If c is not equal to '1', then the dynamic range of the pixel values will be significantly affected by scaling. Thus, to avoid another stage of rescaling after power-law transformation, we fix the value of c = 1.With γ = 1, if the power-law transformed image is passed through binarization, there will be no change in the result compared to simple binarization. When γ > 1, there will be a change in the histogram plot, since there is an increase of samples in the bins towards the gray value of zero. Gamma correction is important if displaying an image accurately on computer screen is of concern.

2.1.5 Edge Detection:

Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more technically, has discontinuities or noise. The points at which image brightness alters sharply are typically organized into a set of curved line segments termed edges.

The same problem of detecting discontinuities in 1D signal is known as step detection and the problem of finding signal discontinuities over time is known as change detection. Edge detection is a basic tool in image processing, machine vision and computer envisage, particularly in the areas of feature reveal and feature extraction.

2.1.5.1 Edge detection techniques:

Different colours has different brightness values of particular colour. Green image has more bright than red and blue image or blue image is blurred image and red image is the high noise image.

Following are list of various edge-detection methods:-

- Sobel Edge Detection Technique
- Perwitt Edge Detection
- Roberts Edge Detection Technique
- Zerocross Threshold Edge Detection Technique
- Canny Edge Detection Technique

In our project we use "CANNY EDGE DETECTION TECHNIQUE" because of its various advantages over other edge detection techniques.

2.1.5.2 Canny Edge Detection

The Canny Edge Detector is one of the most commonly used image processing tools detecting edges in a very robust manner. It is a multi-step process, which can be implemented on the GPU as a sequence of filters. Canny edge detection technique is based on three basic objectives.

I. Low error rate:-

All edges should be found, and there should be no spurious responses. That is, the edges must be as close as possible to the true edges.

II. Edge point should be well localized:-

The edges located must be as close as possible to the true edges. That is , the distance between a point marked as an edge by the detector and the centre of the true edge should be minimum.

III. Single edge point response:-

The detector should return only one point for each true edge point. That is, the number of local maxima around the true edge should be minimum. This means that the detector should not identify multiple edge pixels where only a single edge point exist.

The essence of Canny's work was in expressing the preceding three criteria mathematically and then attempting to find optimal solution to these formulations, in general, it is difficult to find a close form solution that satisfies all the preceding objectives. However, using numerical optimization with 1-D step edges corrupted by additive while Gaussian noise led to the conclusion that a good approximation to the optimal step edge detector is the first derivative of Gaussian:

$$\frac{\mathbf{d}}{\mathbf{dx}}\mathbf{e}^{\frac{-\mathbf{x}^2}{2\sigma^2}} = \frac{-\mathbf{x}}{\sigma^2}\mathbf{e}^{\frac{-\mathbf{x}^2}{2\sigma^2}} \qquad \mathbf{(1)}$$

Generalizing this result to 2-D involves recognizing that the 1-D approach still applies in the direction of the edge normal. Because the direction of the normal is unknown beforehand, this would require applying the 1-D edge detector in all possible directions. This task can be approximated by first smoothing the image with circular 2-D Gaussian function, computing the gradient of the result, and then using the gradient magnitude and direction to estimate edge strength and direction at every point.

Let f(x,y) denote the input image and G(x,y) denote the Gaussian function:

$$\mathbf{G(x,y)} = \mathbf{e}^{-\frac{\mathbf{x}^2+\mathbf{y}^2}{2\sigma^2}} \qquad \mathbf{(2)}$$

We form a smoothed image, $f_s(x, y)$, by convolving G and f:

$$\mathbf{f_s(x,y)} = \mathbf{G(x,y)} * \mathbf{f(x,y)} \qquad \mathbf{(3)}$$

This operation is followed by computing the gradient and direction (angle)

$$\mathbf{M(x,y)} \quad = \quad \sqrt{\mathbf{g_x}^2 + \mathbf{g_y}^2} \qquad \mathbf{(4)}$$

And

$$\boldsymbol{\alpha(x,y)} \quad = \quad \mathbf{tan^{-1}}\frac{g_y}{g_x} \qquad \mathbf{(5)}$$

with $\mathbf{g_x} = \frac{\partial \mathbf{f_s}}{\partial \mathbf{x}}$ **and** $\mathbf{g_y} = \frac{\partial f_s}{\partial y}$ **.**

Equation (2) is implemented using an n x n Gaussian mask. Keep in mind that M(x, y) and α(x, y) are arrays of the same size as the image from which they are computed. Because it is generated using the gradient M(x, y) typically contains wide ridges around local maxima.

The next step is to thin those ridges. One approach is to use non maxima suppression. This can be done in several ways, but the essence of the approach is to specify a no. of discrete orientations of edge normal (gradient vector). For example, in 3x3 region we can define four orientation for an edge passing through the centre point of the region: horizontal, vertical, +45° and -45°.

The final operation is to threshold $g_N(x, y)$ to reduce false edge point. We do it by using a single threshold, in which all value below the threshold were set to 0. If we set the threshold too low, there will still be some false edge (called false positives). If the threshold id set too high, then actual valid edge points will be eliminated (false negatives). Canny's algorithm attempts to improve on this situation by using hysteresis threshold. We use two threshold,
A low threshold, T_L, and a high threshold, T_H. Canny suggested that the ratio of high to low threshold should be two or three to one.

We visualize the thresholding operation as creating two additional images

$$\mathbf{g_{NH}(x,y)} \quad = \quad \mathbf{g_N(x,y)} \geq \mathbf{T_H} \qquad \mathbf{(6)}$$

And

$$\mathbf{g_{NL}(x,y)} \quad = \quad \mathbf{g_N(x,y)} \geq \mathbf{T_L} \qquad \mathbf{(7)}$$

Where, initially, both $g_{NH}(x,y)$ *and* $g_{NL}(x,y)$ are set to 0. After thresholding, $g_{NH}(x,y)$ will have fewer nonzero pixels than $g_{NL}(x,y)$ in general, but all the nonzero pixels in $g_{NH}(x,y)$ will be contained in $g_{NL}(x,y)$ because the latter image is formed with lower threshold. We eliminate from $g_{NL}(x,y)$ all the nonzero from $g_{NH}(x,y)$ by letting

$$\mathbf{g_{NL}(x,y)} = \mathbf{g_{NL}(x,y)} - \mathbf{g_{NH}(x,y)} \qquad \mathbf{(8)}$$

The nonzero pixels in $g_{NL}(x,y)$ and $g_{NH}(x,y)$ may be viewed as being "strong". And "weak" edge pixels, respectively.

After the thresholding operations, all strong pixels in $g_{NH}(x,y)$ are assumed to be valid edge pixels and are so marked immediately. Depending on the value of T_H, the edges in $g_{NH}(x,y)$ typically have gaps.

Longer edges are formed using the following procedure

a. Locate the next unvisited pixel p in $g_{NH}(x,y)$.
b. Mark as valid edge pixels all the weak pixels in $g_{NL}(x,y)$ that are connected to p using say 8 connectivity.
c. If all nonzero pixels in $g_{NH}(x,y)$ have been visited go to step d. else return to step a.
d. Set to zero all pixels in $g_{NL}(x,y)$ that were not marked as valid edge pixels.

At the end of this procedure, the final image output by the Canny is formed by appending to $g_{NH}(x,y)$ all the nonzero pixels from $g_{Nl}(x,y)$.

We use two additional images, $g_{NH}(x, y)$ and $g_{Nl}(x, y)$, to simplify the discussion. In practice, hysteresis threshold can be implemented directly during nonmaxima suppression, and thresholding can be implemented directly on $g_{NL}(x, y)$ by forming a list of strong pixels and the weak pixels connected to them.

Summarizing, the Canny edge detection algorithm consist of the following basic steps;

i. Smooth the input image with Gaussian filter.
ii. Compute the gradient magnitude and angle images.
iii. Apply nonmaxima suppression to the gradient magnitude image.
iv. Use double thresholding and connectivity analysis to detect and link edges.

2.1.6 Image Matching:-

Recognition techniques based on matching represent each class by a prototype pattern vector. An unknown pattern is assigned to the class to which is closest in terms of predefined metric. The simplest approach is the minimum distance classifier, which, as its name implies, computes the (Euclidean) distance between the unknown and each of the prototype vectors. It chooses the smallest distance to make decision. There is another approach based on correlation, which can be formulated directly in terms of images and is quite intuitive.

We have used a totally different approach for image matching. Comparing a reference image with the real time image pixel by pixel. Though there are some disadvantages related to pixel based matching but it is one of the best techniques for the algorithm which is used in the project for decision making. Real image is stored in matric in memory and the real time image is also converted in the desired matric. For images to be same their pixel values in matrix must be same. This is the simplest fact used in pixel matching. If there is any mismatch in pixel value it adds on to the counter used to calculate number of pixel mismatches. Finally percentage of matching is expressed as

$$\%\text{match} = \frac{\text{No.of pixels matched sucessfully}}{\text{total no.of pixels}}$$

The block diagram of the proposed algorithm discussed above is implemented in MATLAB R2013a. So it is necessary to gain an insight of MATLAB.

2.2 Introduction to MATLAB

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research.

MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.

It has powerful built-in routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

There are various tools in Matlab that can be utilized for image processing, such as Simulink, GUI etc. Simulink contains various toolboxes and image processing toolbox is one such example. Simulink is used for simulation of various projects. GUI is another important tool in Matlab. It can be designed either by manual programming which is tedious task or by using guide. GUI is explained in next section.

2.3 GUI:

A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called components, which enable a user to perform interactive tasks. The user of the GUI does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user of a GUI need not understand the details of how the tasks are performed. GUI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders—just to name a few. GUIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other GUIs, and display data as tables or as plots. The following figure illustrates a simple GUI that you can easily build

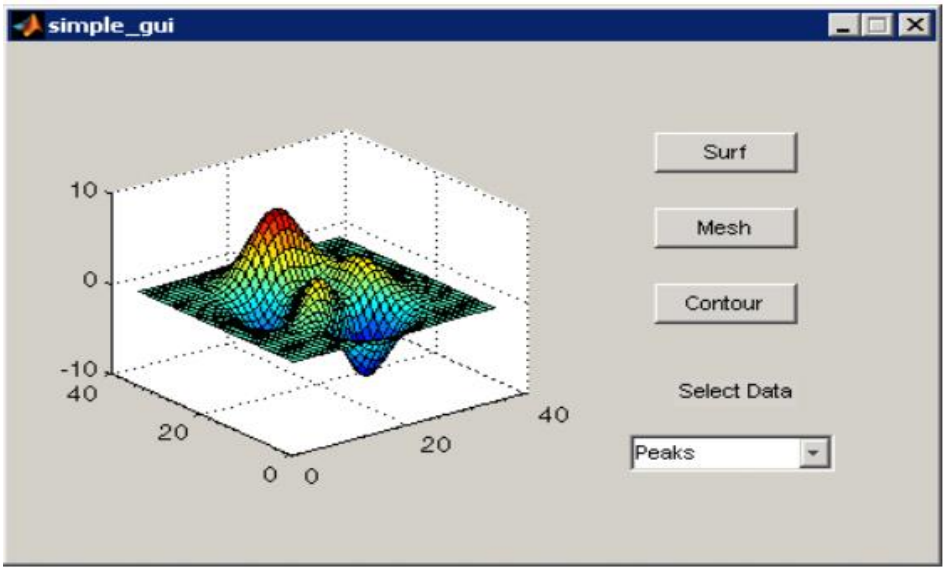


Fig 2.3 GUI

The GUI contains

•An axes component

•A pop-up menu listing three data sets that correspond to MATLAB

Functions: peaks, membrane, and sinc

•Astatic text component to label the pop-up menu

•Three buttons that provide different kinds of plots: surface, mesh, and contour

When you click a push button, the axes component displays the selected data set using the specified type of 3-D plot.

Typically, GUIs wait for an end user to manipulate a control, and then respond to each user action in turn. Each control, and the GUI itself, has one or more call-backs, named for the fact that they "call back" to MATLAB to ask it to do things. A particular user action, such as pressing a screen button, or passing the cursor over a component, triggers the execution of each call back. The GUI then responds to these events. You, as the GUI creator, write call-backs that define what the components do to handle events. This kind of programming is often referred to as event-driven programming. In event-driven programming, call back execution is asynchronous, that is, events external to the software trigger call back execution. In the case of MATLAB GUIs, most events are user interactions with the GUI, but the GUI can respond to other kinds of events as well, for example, the creation of a file or connecting a device to the computer.

You can code call-backs in two distinct ways:

•As MATLAB language functions stored in files

•As strings containing MATLAB expressions or commands (such as 'c = sqrt(a*a + b*b); 'or' print')Using functions stored in code files as call-backs is preferable to using strings, because functions have access to arguments and are more powerful and flexible. You cannot use

MATLAB scripts (sequences of statements stored in code files that do not define functions) as call-backs. Although you can provide a call back with certain data and make it do anything you want, you cannot control when call-backs execute. That is, when your GUI is being used, you have no control over the sequence of events that trigger particular call-backs or what other call-backs might still be running at those times. This distinguishes event-driven programming from other types of control flow, for example, processing sequential data files.

A MATLAB GUI is a figure window to which you add user-operated components. You can select, size, and position these components as you like using call-backs you can make the components do what you want when the user clicks or manipulates the components with keystrokes.

You can build MATLAB GUIs in two ways:

•Use GUIDE (GUI Development Environment), an interactive GUI construction kit.

This approach starts with a figure that you populate with components from within a graphic layout editor. GUIDE creates an associated code file containing call-backs for the GUI and its components. GUIDE saves both the figure (as a FIG-file) and the code file. Opening either one also opens the other to run the GUI.

•Create code files that generate GUIs as functions or scripts (programmatic GUI construction).

Using this approach, you create a code file that defines all component properties and behaviours. When a user executes the file, it creates a figure, populates it with components, and handles user interactions. Typically, the figure is not saved between sessions because the code in the file creates a new one each time it runs.

The code files of the two approaches look different. Programmatic GUI files are generally longer, because they explicitly define every property of the figure and its controls, as well as the call-backs. GUIDE GUIs define most of the properties within the figure itself. They store the definitions in its FIG-file rather than in its code file. The code file contains call-backs and other functions that initialize the GUI when it opens.

You can create a GUI with GUIDE and then modify it programmatically. However, you cannot create a GUI programmatically and then modify it with GUIDE. The GUI-building technique you choose depends on your experience, your preferences, and the kind of application you need the GUI to operate. This table outlines some possibilities.

CHAPTER 3: IMPLEMENTATION AND RESULTS

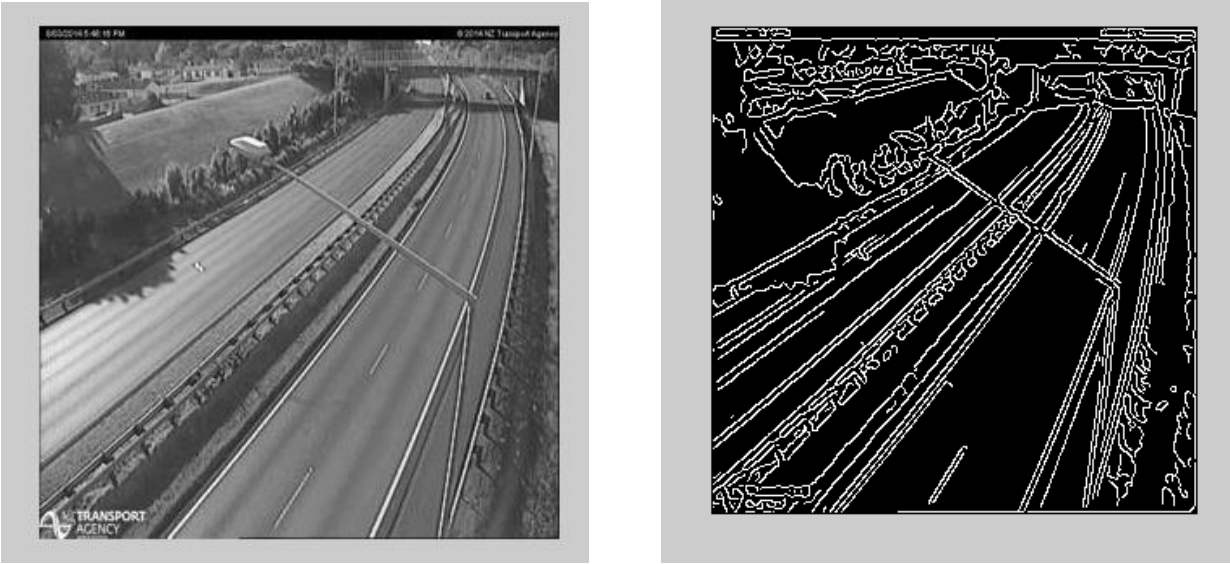
3.1 Implementation Algorithm

The block diagram of the project was discussed in previous chapter. The algorithm behind the block diagram consists of following steps

1. We have a reference image and the image to be matched is continuously captured using a camera that is installed at the junction.
2. The images are pre-processed in two steps as follows
   a. Images are rescaled to 300x300 pixels.
   b. Then the above rescaled images are converted from RGB to gray.
3. Edge detection of pre-processed images is carried out using Canny edge detection technique.
4. The output images of previous step are matched using pixel to pixel matching technique.
5. After matching the timing allocation is done depending on the percentage of matching as
   a. If the matching is between 0 to 30% - green light is on for 90 seconds.
   b. If the matching is between 30 to 50% - green light is on for 60 seconds.
   c. If the matching is between 50 to 70% - green light is on for 30 seconds.
   d. If the matching is between 70 to 90% - green light is on for 20 seconds.
   e. If the matching is between 90 to 100% - red light is on for 90 seconds.

The program written using MATLAB to implement the above algorithm is given in appendix however the output of each step and final results of the program are given in next sections.

3.2 Output Figures of individual steps:

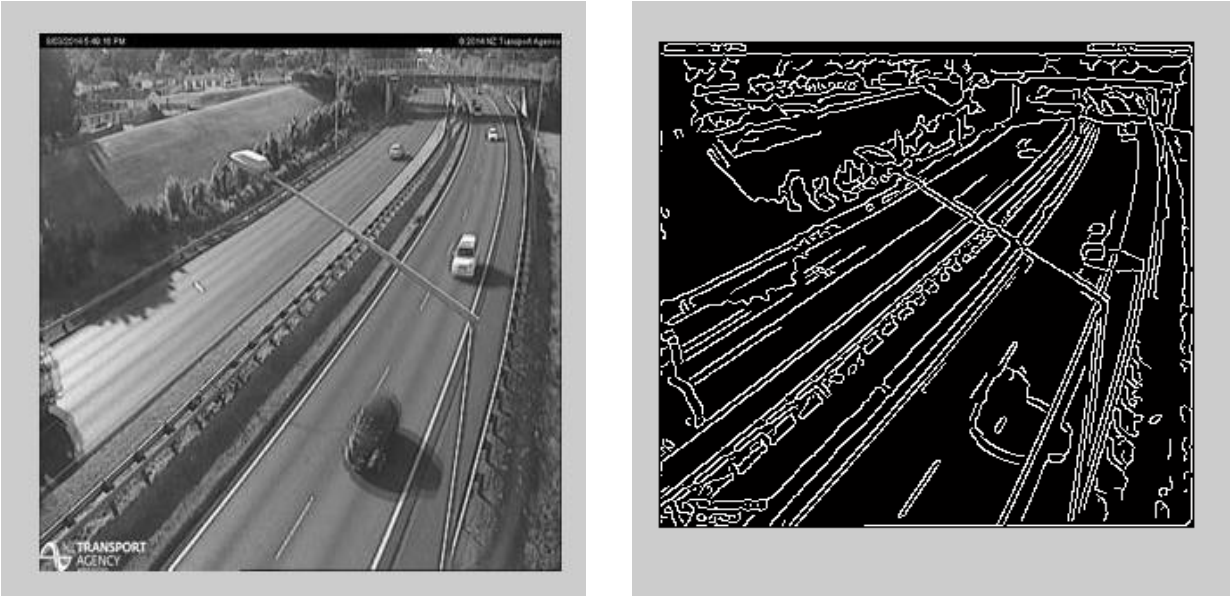


(a) (b)

Fig 3.2.1

(a): Gray reference image

(b): Edge detected output of reference image



(a) (b)

Fig 3.2.2

(a): Gray captured image for comparison

(b): Edge detected output of (a)

3.3 RESULTS

The output of GUI given below clearly indicates the implemented results of the algorithm designed.

3.3.1 Matching between 90-100%

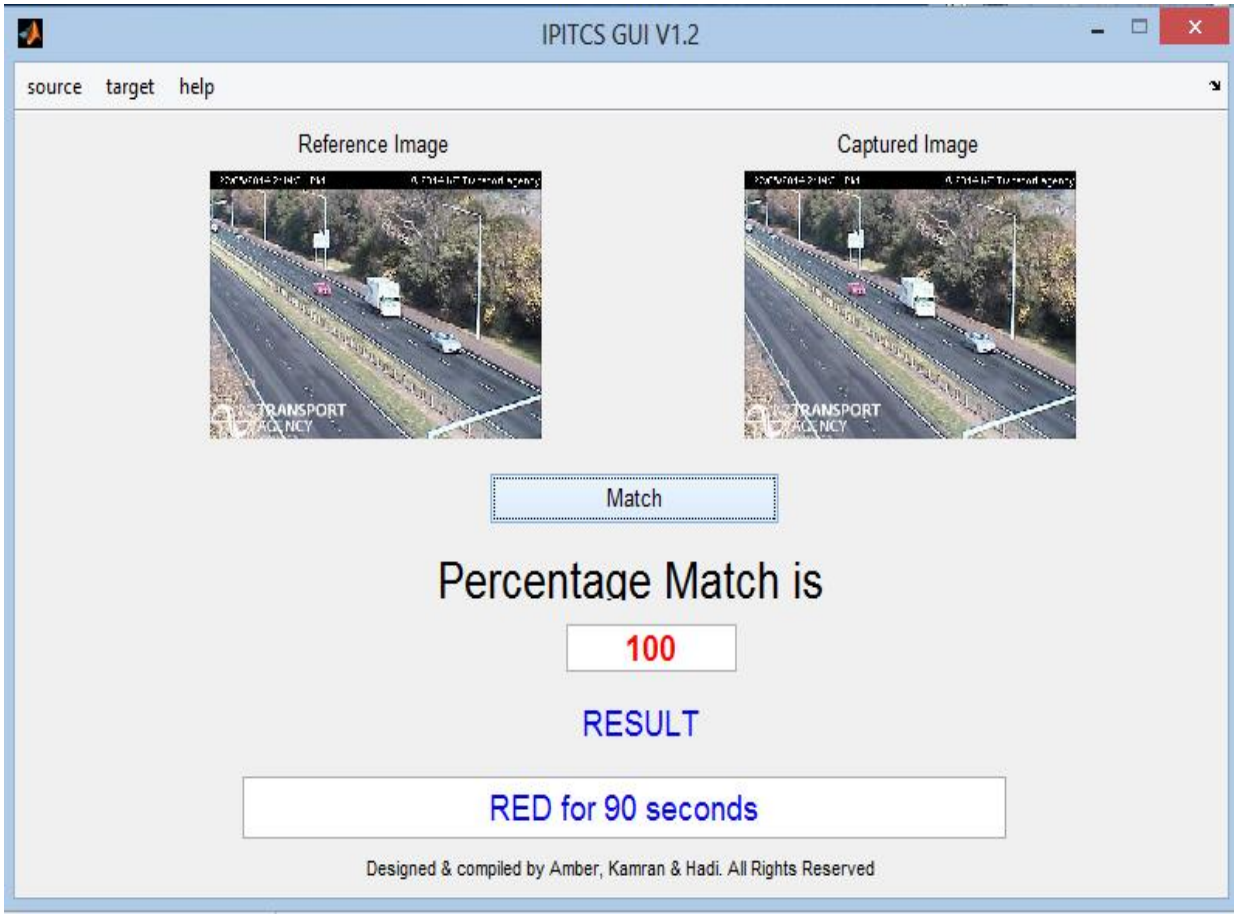


Fig 3.3.1

In the above figure both the images are same and captured image is a perfect match of reference image thereby showing 100% match and RED light is displayed for 90 seconds.

3.3.2 Matching between 70-90%

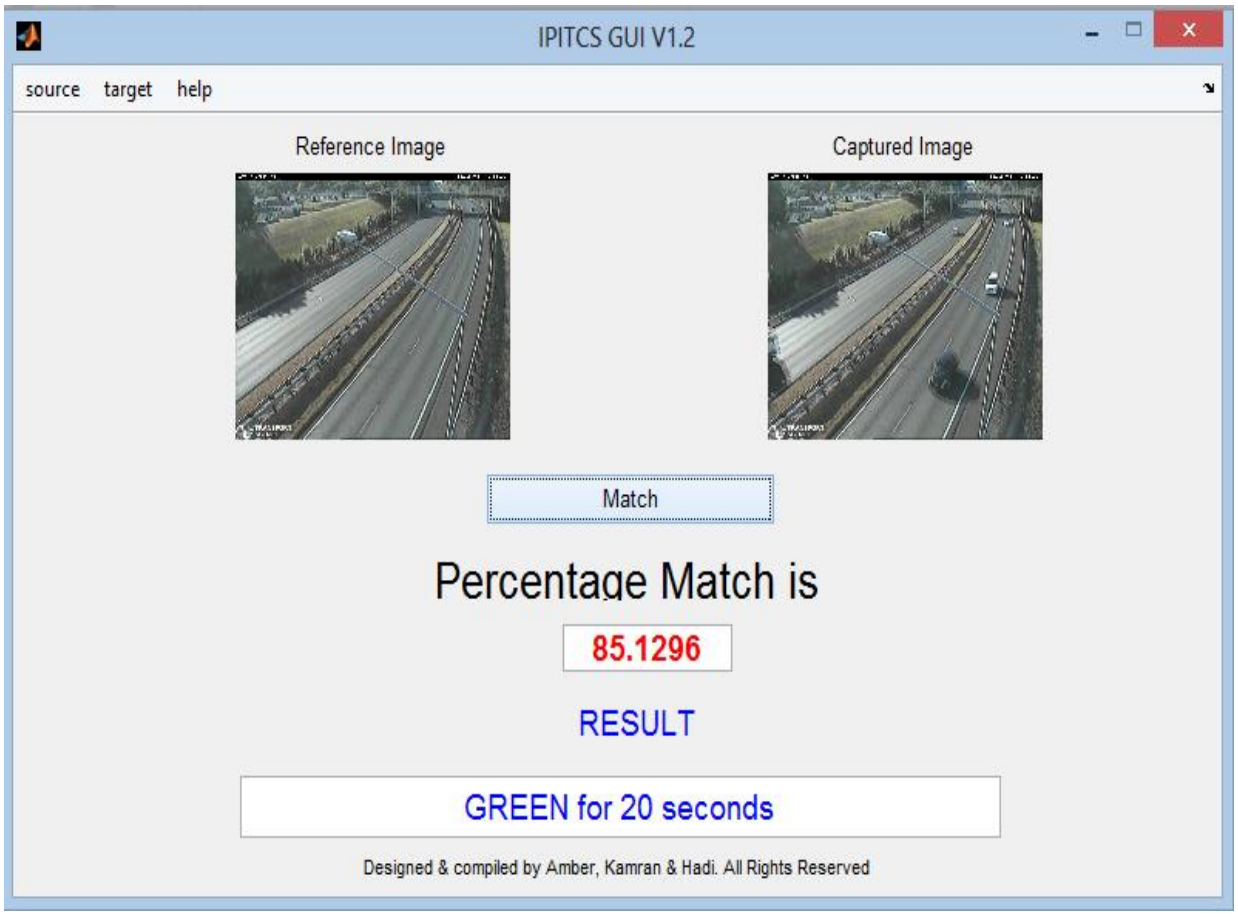


Fig 3.3.2 (a)

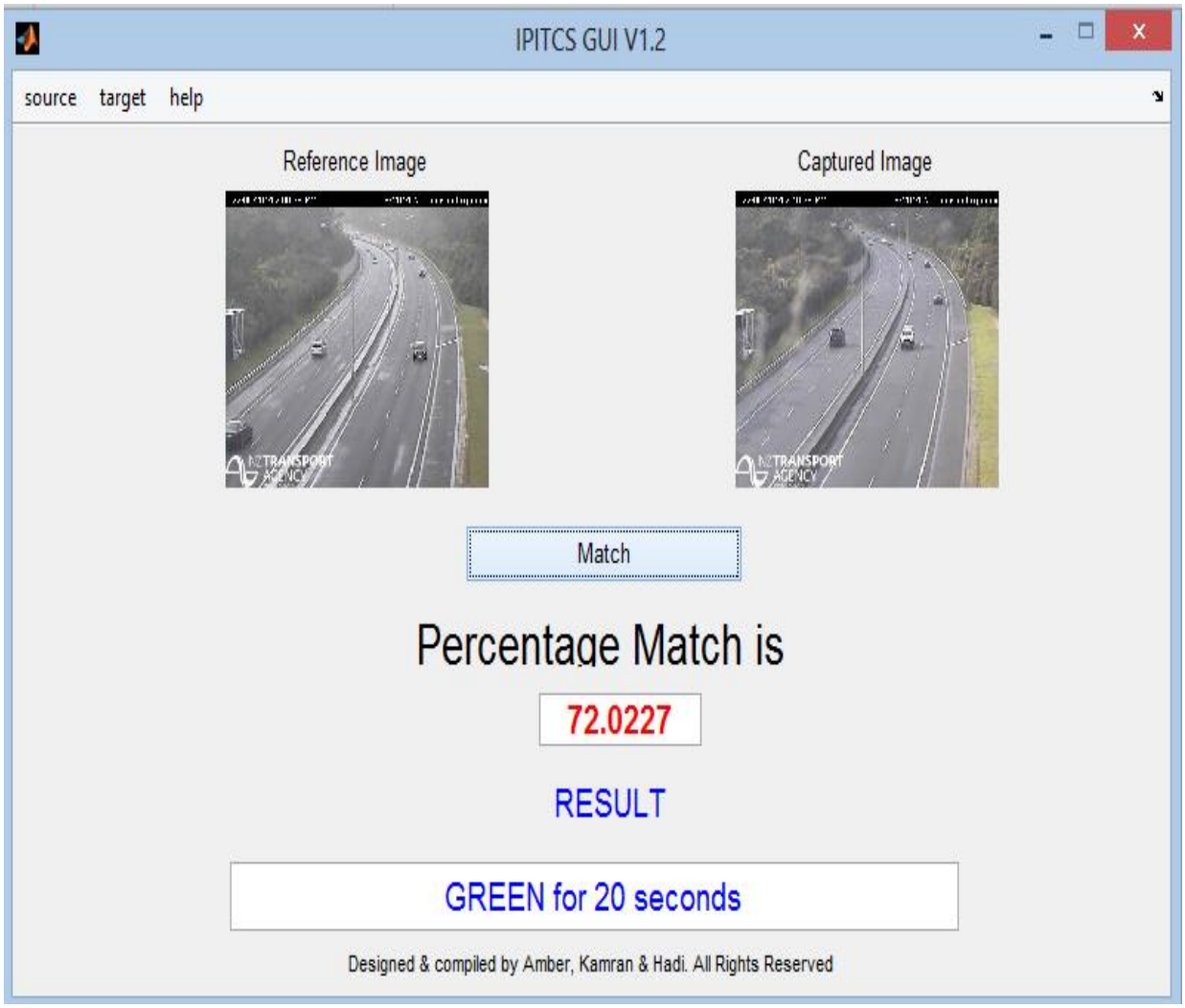

IPITCS GUI V1.2
source
target
help
Reference Image
Captured Image
Match
Percentage Match is
72.0227
RESULT
GREEN for 20 seconds
Designed & compiled by Amber, Kamran & Hadi. All Rights Reserved

Fig 3.3.2(b)

3.3.3 Matching between 50-70%

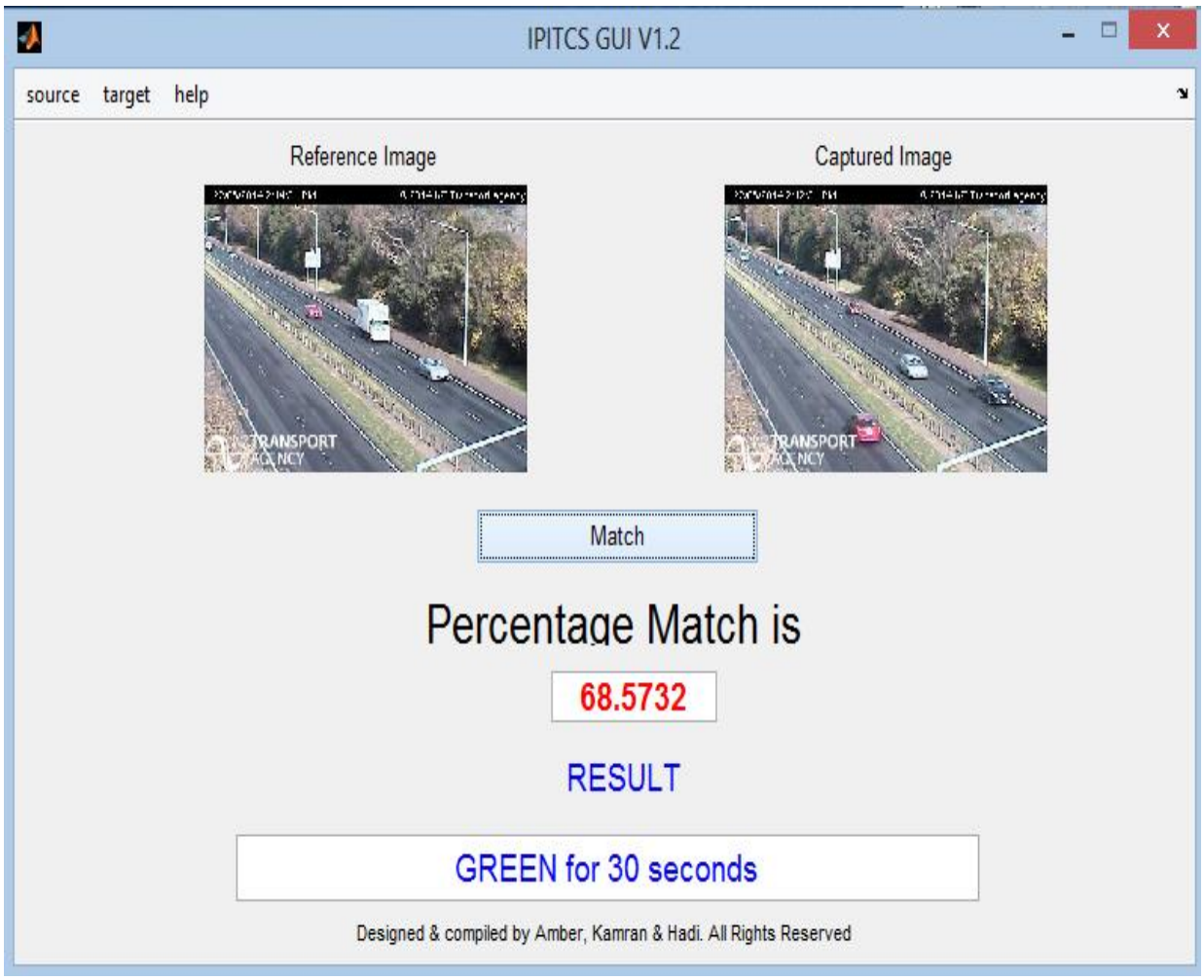


Fig 3.3.3(a)

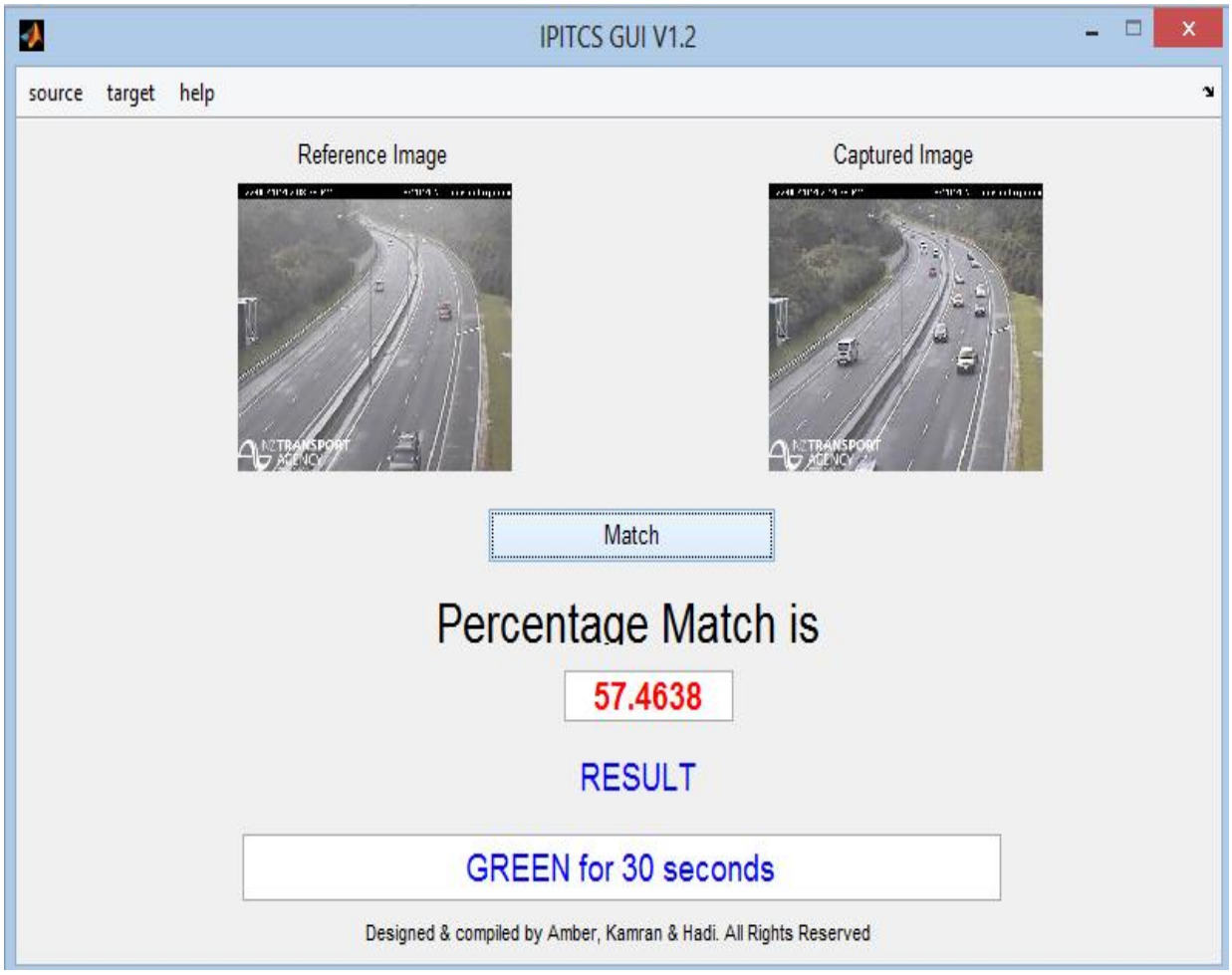

IPITCS GUI V1.2
source
target
help
Reference Image
Captured Image
Match
Percentage Match is
57.4638
RESULT
GREEN for 30 seconds
Designed & compiled by Amber, Kamran & Hadi. All Rights Reserved

Fig 3.3.3(b)

3.3.4 Matching between 30-50%

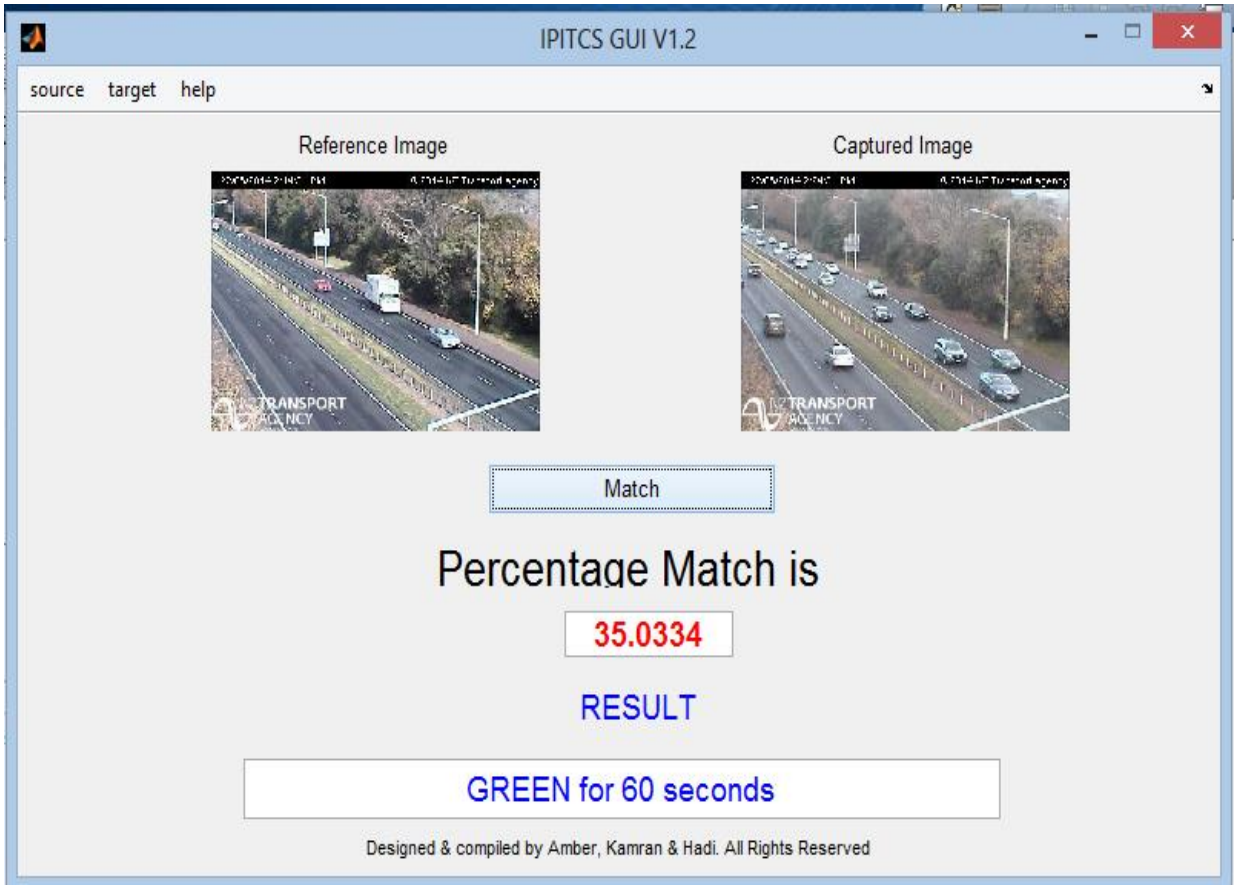


Fig 3.3.4

3.3.5 Matching less than 30%

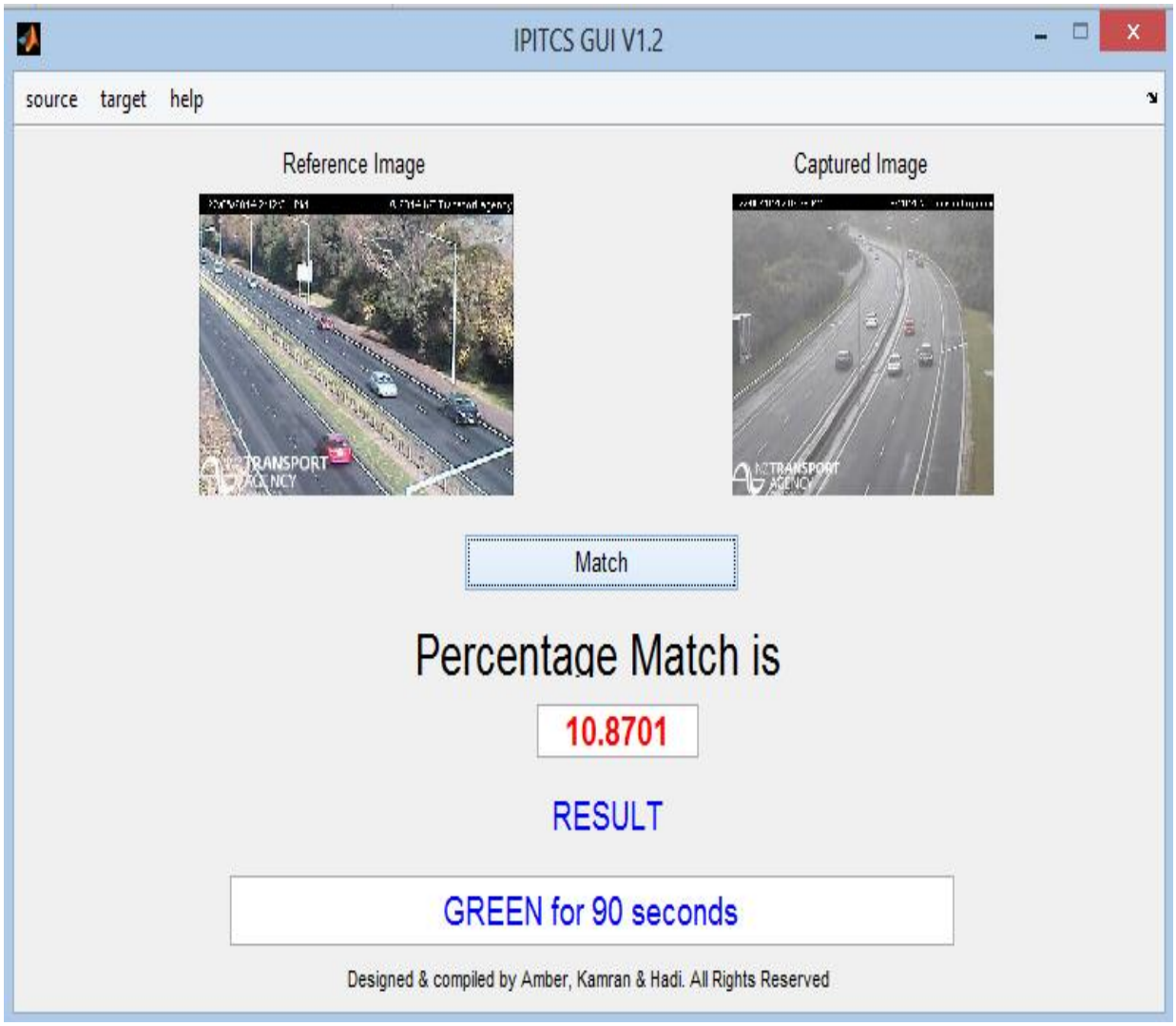


Fig 3.3.5

CHAPTER 4: CONCLUSION

4.1 Conclusion

"Traffic control using image processing" technique that we propose overcomes all the limitations of the earlier (in use) techniques used for controlling the traffic. Earlier in automatic traffic control use of timer had a drawback that the time is being wasted by green light on the empty. This technique avoids this problem. Upon comparison of various edge detection algorithms, it was inferred that Canny Edge Detector technique is the most efficient one. The project demonstrates that image processing is a far more efficient method of traffic control as compared to traditional techniques. The use of our technique removes the need for extra hardware such as sound sensors. The increased response time for these vehicles is crucial for the prevention of loss of life. Major advantage is the variation in signal time which control appropriate traffic density using Image matching. The accuracy in calculation of time due to single moving camera depends on the registration position while facing road every time. Output of GUI clearly indicated some expected results. It showed matching in almost every interval that were decided as boundaries like 10%, 35%, 68% etc.

4.2 Future Work

The focus shall be to implement the controller using DSP as it can avoid heavy investment in industrial control computer while obtaining improved computational power and optimized system structure. The hardware implementation would enable the project to be used in real-time practical conditions. In addition, we propose a system to identify the vehicles as they pass by, giving preference to emergency vehicles and assisting in surveillance on a large scale.

CHAPTER 5: REFERENCES

5.1 References

1. Digital image processing by Rafael C. Gonzalez and Richard E. Woods.

2. Ahmed S. Salama, Bahaa K. Saleh, Mohamad M. Eassa, "Intelligent Cross Road Traffic Management System (ICRTMS)," 2nd Int. Conf. on Computer Technology and Development, Cairo, Nov 2010, pp. 27-31.

3. B. Fazenda, H. Atmoko, F.Gu, L. Guan1 and A. Ball" Acoustic Based Safety Emergency Vehicle Detection for Intelligent Transport Systems" ICCAS-SICE, Fukuoka, Aug 2009, pp.4250-4255.

4. Z. Jinglei, L. Zhengguang, and T. Univ, "A vision-based road surveillance system using improved background subtraction and region growing approach," Eighth ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Qingdao, August 2007, pp. 819-822.

5. M. Siyal, and J. Ahmed, "A novel morphological edge detection and window based approach for real-time road data control and management," Fifth IEEE Int. Conf. on Information, Communications and Signal Processing, Bangkok, July 2005, pp. 324-328.

6. Y. Wu, F. Lian, and T. Chang, "Traffic monitoring and vehicle tracking using roadside camera," IEEE Int. Conf. on Robotics and Automation, Taipei, Oct 2006, pp. 4631– 4636.

7. Reulke, S. Bauer, T. D'oring, F. Meysel, "Traffic surveillance using multi-camera detection and multi-target tracking", Proceedings of Image and Vision Computing New Zealand 2007, pp. 175–180, Hamilton, New Zealand, December 2007.

8. www.nzta.govt.nz
live traffic webcams / NZ Transport Agency

9. http://www.mathworks.com Web

10. Traffic light control system simulation through vehicle detection using image processing

By Mac Michael B. Reyes and Dr Eliezer A. Albaccea

APPENDIX

MATLAB Program:

The project is simulated in matlab. GUI is designed using guide. The program used for designing of GUI is given below

```
function varargout = push(varargin)
% push M-source for push.fig
%      push, by itself, creates a new push or raises the existing
%      singleton*.
%      H = push returns the handle to a new push or the handle to
%      the existing singleton*.
%      push('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in push.M with the given input arguments.
%      push('Property','Value',) creates a new push or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before push_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to push_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
% Copyright 2002-2003 The MathWorks, Inc.
% Edit the above text to modify the response to help push
% Last Modified by GUIDE v2.5 05-Apr-2014 16:26:52
% Begin initialization code - DO NOT TARGET
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @push_OpeningFcn,...
```

```
                'gui_OutputFcn',  @push_OutputFcn,...
                'gui_LayoutFcn',  [] , ...
                'gui_Callback',   []);


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

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT TARGET

% --- Executes just before push is made visible.
function push_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 push (see VARARGIN)

% Choose default command line output for push
handles.pic='test1.jpg';

handles.output = hObject;
handles.pic='test1.jpg';

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes push wait for user response (see UIRESUME)
% uiwait(handles.figure1);

 % --- Outputs from this function are returned to the command line.
function varargout = push_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)

%set(handles.text1, 'String', 'push button pushed')
display('Initiating matching...');
display('Calibrating procedures...');
display('...   ...   ...');
k=handles.XX;
p=handles.YY;
[x,y,z] = size(k);
if(z==1)
   ;
else
```

```
k = rgb2gray(k);
end

[x,y,z] = size(p);
if(z==1)
   ;
else
   p = rgb2gray(p);
end

 %applying edge detection on first picture
%so that we obtain white and black points and edges of the objects present
%in the picture.
k1=imresize(k,[300 300]);
edge_det_k = edge(k1,'sobel');

%%applying edge detection on second picture
%so that we obtain white and black points and edges of the objects present
%in the picture.
p1=imresize(p,[300 300]);
edge_det_p = edge(p1,'sobel');

%definition of different variables to be used in the code below

%output variable if pictures have been matched.
OUTPUT_MESSAGE = ' RED for 90 seconds ';
OUTPUT_MESSAGE2 = ' GREEN for 20 seconds ';
OUTPUT_MESSAGE3 = ' GREEN for 30 seconds ';
OUTPUT_MESSAGE4 = ' GREEN for 60 seconds ';
OUTPUT_MESSAGE5 = ' GREEN for 90 seconds ';
%initialization of different variables used
```

```
matched_data = 0;
white_points = 0;
black_points = 0;
x=0;
y=0;
l=0;
m=0;

%for loop used for detecting black and white points in the picture.
for a = 1:1:300
   for b = 1:1:300
      if(edge_det_k(a,b)==1)
         white_points = white_points+1;
      else
         black_points = black_points+1;
      end
   end
end
display('matching...');
%for loop comparing the white (edge points) in the two pictures
for i = 1:1:300
   for j = 1:1:300
      if(edge_det_k(i,j)==1)&(edge_det_p(i,j)==1)
         matched_data = matched_data+1;
         else
            ;
      end
   end
end
```

```
%calculating percentage matching.
total_data = white_points;
total_matched_percentage = (matched_data/total_data)*100;
%handles.pp=total_matched_percentage;
%guidata(hObject,handles);
set(handles.result,'String',total_matched_percentage);
%outputting the result of the system.
if total_matched_percentage > 90          %can add flexability at this point by reducing the amount of matching.

   total_matched_percentage
   OUTPUT_MESSAGE
   display(OUTPUT_MESSAGE);
   set(handles.result1,'String',OUTPUT_MESSAGE);
elseif total_matched_percentage <= 90 & total_matched_percentage > 70
   total_matched_percentage
   OUTPUT_MESSAGE2
   display(OUTPUT_MESSAGE2);
   set(handles.result1,'String',OUTPUT_MESSAGE2);

elseif total_matched_percentage <= 70 & total_matched_percentage > 50
   total_matched_percentage
   OUTPUT_MESSAGE3
   display(OUTPUT_MESSAGE3);
   set(handles.result1,'String',OUTPUT_MESSAGE3);
elseif total_matched_percentage <= 50 & total_matched_percentage > 10
  total_matched_percentage
   OUTPUT_MESSAGE4
   display(OUTPUT_MESSAGE4);
   set(handles.result1,'String',OUTPUT_MESSAGE4);
```

```
else
   total_matched_percentage
   OUTPUT_MESSAGE5
   display(OUTPUT_MESSAGE5);
   set(handles.result1,'String',OUTPUT_MESSAGE5);
end

% --- Executes during object creation, after setting all properties.
function text1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% ------------------------------------------------------------------
function source_Callback(hObject, eventdata, handles)
% hObject    handle to source (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file = uigetfile('*.jpg');
   if ~isequal(file, 0)
      k=imread(file);
      handles.YY = k;
z=k;
guidata(hObject,handles);
%guidata(hObject,handles);
      subplot(2,2,1),imshow(k);title('Reference Image');

   end
```

```
% --------------------------------------------------------------------
function new_Callback(hObject, eventdata, handles)
% hObject    handle to new (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------
function target_Callback(hObject, eventdata, handles)
% hObject    handle to target (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file = uigetfile('*.jpg');
   if ~isequal(file, 0)
       p=imread(file);

     subplot(2,2,2),imshow(p);title('Captured Image');
      handles.XX = p;
guidata(hObject,handles);
   end

% --------------------------------------------------------------------
function Untitled_2_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_1 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function about_Callback(hObject, eventdata, handles)
% hObject    handle to about (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
msgbox('This script file is a part of our project Image Processing based Intelligent Traffic Control Systems (IPITCS) and has been developed by Amber Deep Singh, Abdul Hadi Parwana and Kamran Shahid Baig under the able guidance of Mrs. Farida Khurshid, Department of  Electronics and Communication, National Institue of Technology Srinagar.');
% --------------------------------------------------------------------
function Untitled_5_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Untitled_6_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function howto_Callback(hObject, eventdata, handles)
% hObject    handle to howto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
msgbox('Use images resized to 300*300 pixels');

% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% ------------------------------------------------------------------
function open_Callback(hObject, eventdata, handles)
% hObject    handle to open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

file = uigetfile('*.jpg');
   if ~isequal(file, 0)
      imshow(file);
      %open(source);
   end

function result_Callback(hObject, eventdata, handles)
% hObject    handle to result (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 result as text
%        str2double(get(hObject,'String')) returns contents of result as a double
```

```
% --- Executes during object creation, after setting all properties.
function result_CreateFcn(hObject, eventdata, handles)
% hObject    handle to result (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 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)

 % --- 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)

 function result1_Callback(hObject, eventdata, handles)
% hObject    handle to result1 (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 result1 as text
%        str2double(get(hObject,'String')) returns contents of result1 as a double
% --- Executes during object creation, after setting all properties.
function result1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to result1 (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 help_Callback(hObject, eventdata, handles)
% hObject    handle to help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes when figure1 is resized.
function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: delete(hObject) closes the figure
delete(hObject);
```