

# **Chapter 1**

## **Introduction**

## **1.1 About Autonomous Navigation System**

In the fourth century B.C., Aristotle who is popularly credited with the original concept of automation, wrote "if every instrument could accomplish its own work, obeying or anticipating the will of others i.e. if the shuttle could weave, and the pick touch the lyre, without a hand to guide them, chief workmen would not need servants ..." From the earliest implementations of clock based automatons in the early 14<sup>th</sup> century to "ASIMO" - one of the most advanced humanoid robot, the field of robotics has come a long way. The ever increasing applications of robots in all areas of modern life are expected to usher in a new era where these smart autonomous systems will have significant impact on how we live OUT daily lives. Hence, the need for the robots to understand human intents and desires - the unfulfilled part of Aristotle's vision, is becoming increasingly important.

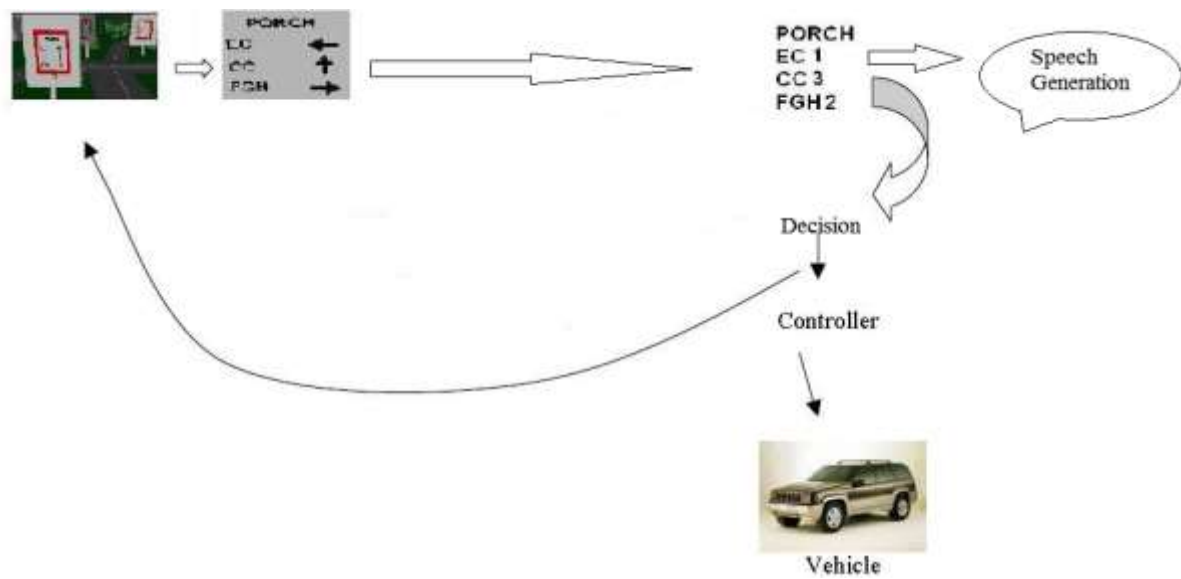
The fundamental aim of the project is to design a human interactive autonomous vehicle to navigate in an environment while regularly extracting relevant information and reacting as per this information. The vehicle can navigate to known and unknown environment autonomously, continually expanding and updating the map.

The project includes the design of the robot, the environment, the interface, the electronic hardware and the process control required to perform this task. The robot is a miniature prototype version of a four wheeled automobile. It is capable of moving flexibly in the environment with high precision. The environment is designed taking into consideration, the real life scenario.

The system design involves closed feedback loop between the hardware, the software and precise process performance goals. The camera, eye of the robot, captures the images of the road signs and sends them to the processing platform. It analyses the images for all kind of relevant information and respond appropriately.

As the vehicle cruise along the environment as per the information provided by the road signs, symbol recognition algorithm will play an important role in the project. Different neural sets have been studied and an optimum net has been chosen for this particular application.

Figure below precisely explains the procedure of execution of the system.



**Fig1.1** Flow Diagram of System Execution

## **1.2 Project Objectives**

Objective of the project is to develop a system for the following applications:-

1. Autonomous Navigation using Digital Image processing.
2. Making remote surveillance system.
3. Making image processing based text reader for visually challenged.

# **Chapter 2**

# **Software**

# **&**

# **Hardware**

# **Requirements**

## **2.1 Software Requirements**

- \* MATLAB R2009
- \* Microsoft Windows 7 (Robot, console)
- \* Team viewer 5.0
- \* Windows SAPI

## **2.2. Hardware Requirements**

### **2.2.1 Robot**

- \* AMD Sempron-2800, 2.8 Ghz Processor (or higher)
- \* Asus M2N68AM Plus Motherboard
- \* 2GB DDR2 667 Mhz RAM
- \* 80 GB HDD
- \* 550 watts power supply
- \* Drive mechanism (differential), four 60 RPM motors
- \* Two VGA Web-camera
- \* Wi-Fi LAN card
- \* Parallel port communication kit
- \* L-298 dual channel H-Bridge circuit
- \* Wireless Router

### **2.2.2 Console**

- \* Any computer with wireless network adapter and remote desktop capability

# **Chapter 3**

## **Literature Survey**

### **3.1 Literature survey**

One of the most vibrant fields of research in today's scenario is to equip machines with vision and ability of its analysis. One such task where, it is desired to automate things is vehicle navigation. Autonomous navigation systems can be developed with help of Digital Image Processing, aided by some sensors. The navigation systems can be further enhanced if they can be made capable of recognizing road signs and direction boards. On basis of information collected this way systems can be made to navigate autonomously without any external interference required. The recognition can be implemented by a no. ways, like using neural networks, fuzzy logic and direct comparison with database. Among these ways the performance, fidelity, and robustness of the system varies.

Another application of these systems can be for surveillance purposes. There are times when it is risky or not desired to send humans into battlefield for spying. In such cases robotic systems capable of being operated manually are used for surveillance. Then can also be equipped with weapons for self defense because they carry important intelligence information.



# **Chapter 4**

# **Requirement Analysis**

## 4.1 Problem Definition

There are three problem statements for this project which are as follows :-

### 1. Autonomous Navigation System (ANS)

The ANS is implemented with the help of Digital Image Processing. The system functions by analyzing the information captured from the environment and taking decisions based on it. It should be intelligent enough to understand and react to the environment and thus uses Neural Network based approach to recognize text and road signs.

### 2. Remote Surveillance System

The Robot is equipped with a no. of cameras and can be safely controlled over the internet or wireless network. Some of the advantages of the remote surveillance systems are:

- can enter environments that are dangerous to human life
- reduces the exposure risk of the operator
- can stay in the field for long durations, performing a precise, repetitive raster scan of a region, day-after-day, night-after-night in complete darkness, or, in fog, under computer control:
  - performing a geological survey
  - performing visual or thermal imaging of a region
  - measuring cell phone, radio, or, TV coverage over any terrain
- can be programmed to complete the mission autonomously even when contact with its **Ground Control Station** (GCS) is lost.

### 3. Image Reading System

The image reading device includes a unit for capturing input parameters from the environment. These parameters are the processed and resulting data is fed to an ANN. The output of the ANN is the converted into speech using speech API.

## 4.2 Feasibility Study

There are no. of systems which resemble in part or whole but not all the features this project.

Some of the examples of such systems are as follows:

### 1. Stanley

It is an autonomous vehicle created by Stanford University's Stanford Racing Team in cooperation with the Volkswagen Electronics Research Laboratory (ERL). It competed in, and won, the 2005 DARPA Grand Challenge, earning the Stanford Racing Team the 2 million dollar prize. To navigate, Stanley used five roof mounted Sick AG LIDAR units to build a 3-D map of the environment, supplementing the position sensing GPS system. An internal guidance system utilizing gyroscopes and accelerometers monitored the orientation of the vehicle and also served to supplement GPS and other sensor data. Additional guidance data was provided by a video camera used to observe driving conditions out to eighty meters (beyond the range of the LIDAR) and to ensure room enough for acceleration.



Fig4.1 Stanley



## 2. Daksh

It is an electrically powered and remotely controlled robot made by DRDO, used for locating, handling and destroying hazardous objects safely. It is a battery-operated robot on wheels and its primary role is to recover improvised explosive devices (IEDs). It locates IEDs with an X-ray machine, picks them up with a gripper-arm and defuses them with a jet of water. It has a shotgun, which can break open locked doors, and it can scan cars for explosives. Daksh can also climb staircases, negotiate steep slopes, navigate narrow corridors and tow vehicles. With a master control station (MCS), it can be remotely controlled over a range of 500 m in line of sight or within buildings.

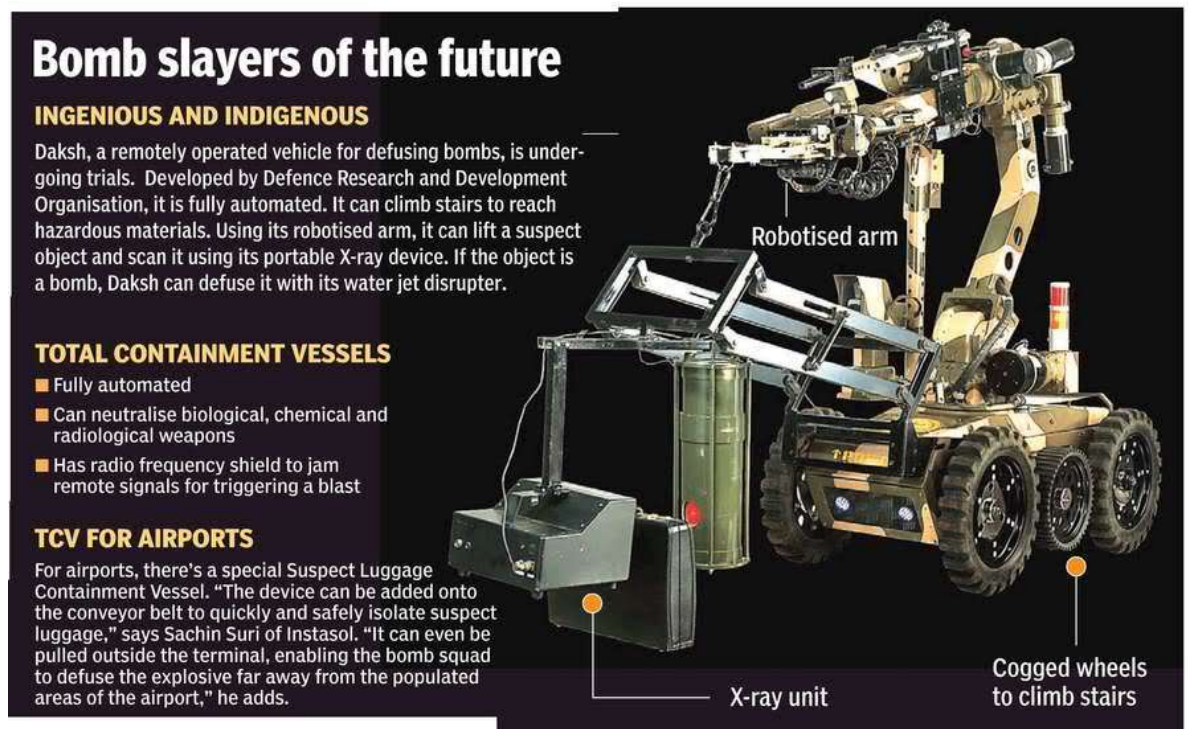


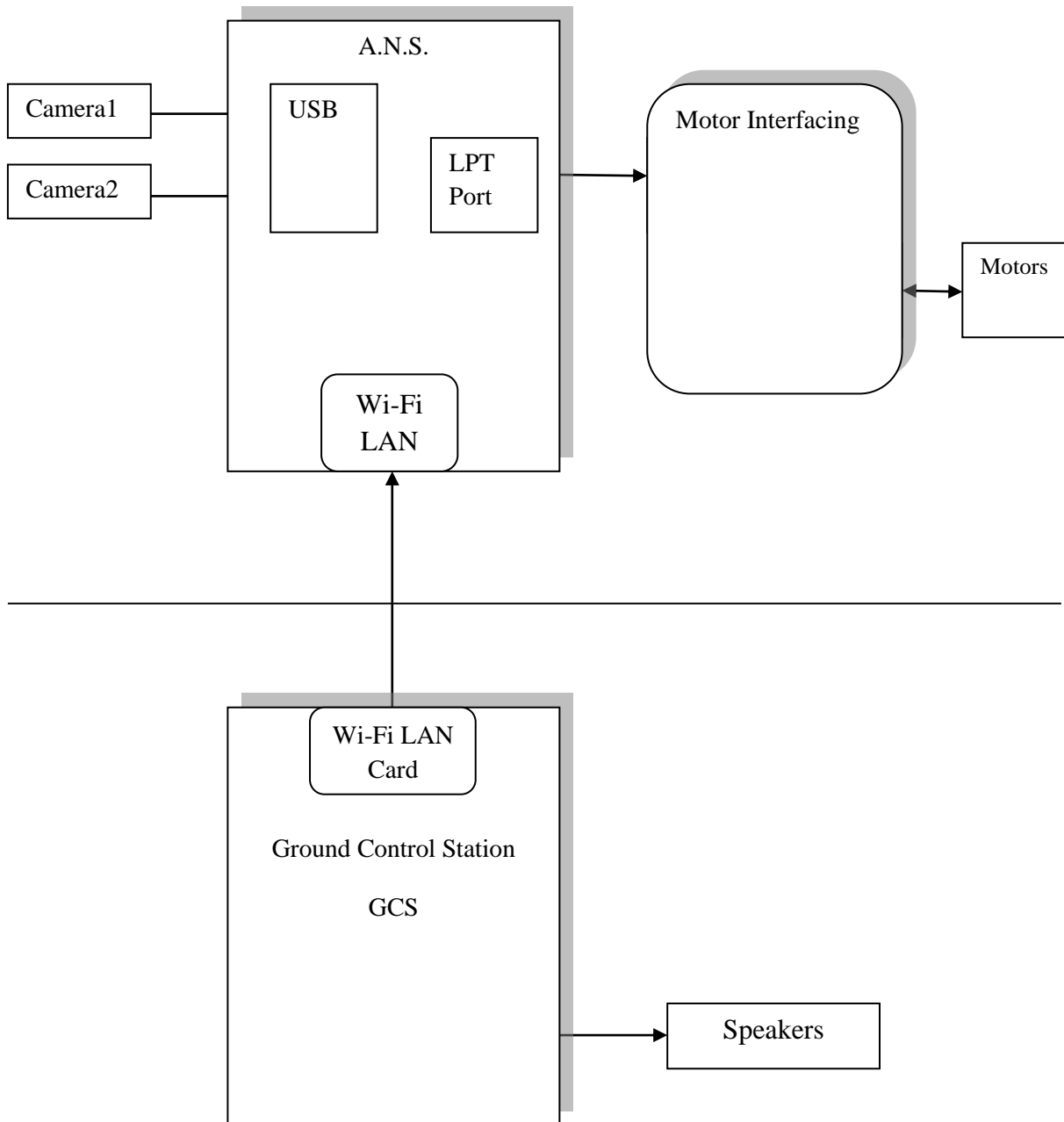
Fig 4.2 Daksh

# Chapter 5

## Design

## 5.1 Hardware Architecture

### 5.1.1 Block Diagram of Hardware Architecture



### 5.1.2 Working Principle

Image processing algorithms used in autonomous navigation system require enormous processing power that is why it is powered with AMD Sempron, 2.8 Ghz Processor. The robot is controlled with the help of Ground Station Controller in remote desktop configuration on wireless network 802.11. Two cameras are used as inputs, decision is made based on their output, and LPT port is triggered. The output of LPT port is fed to buffers which in turn act as input to motor drivers.

The robot is powered by an Switch Mode Power Supply AC/DC (right now being run by AC power supply). In order to increase the mobility DC SMPS can also be used.

The LPT Buffers are used to buffer the output of LPT port and raise the voltage level to a standard 5V supply. This is required because the output voltage of LPT port is not high enough to run the motor drivers. Also, the current draining capacity is low too.

The Motor driver used is L298 dual H-Bridge IC. The IC is able to provide a DC current of 2A per channel. The input output characteristics of L298 are as below:

Input1	Input2	Output
High	Low	Motor runs in one direction
Low	High	Motor runs in reverse direction
Low	Low	Motor is Free
High	High	Braking Action

### 5.1.3 Hardware Snapshots



Fig 5.2 Isotropic View Of Robot



Fig 5.3 Top View of Robot



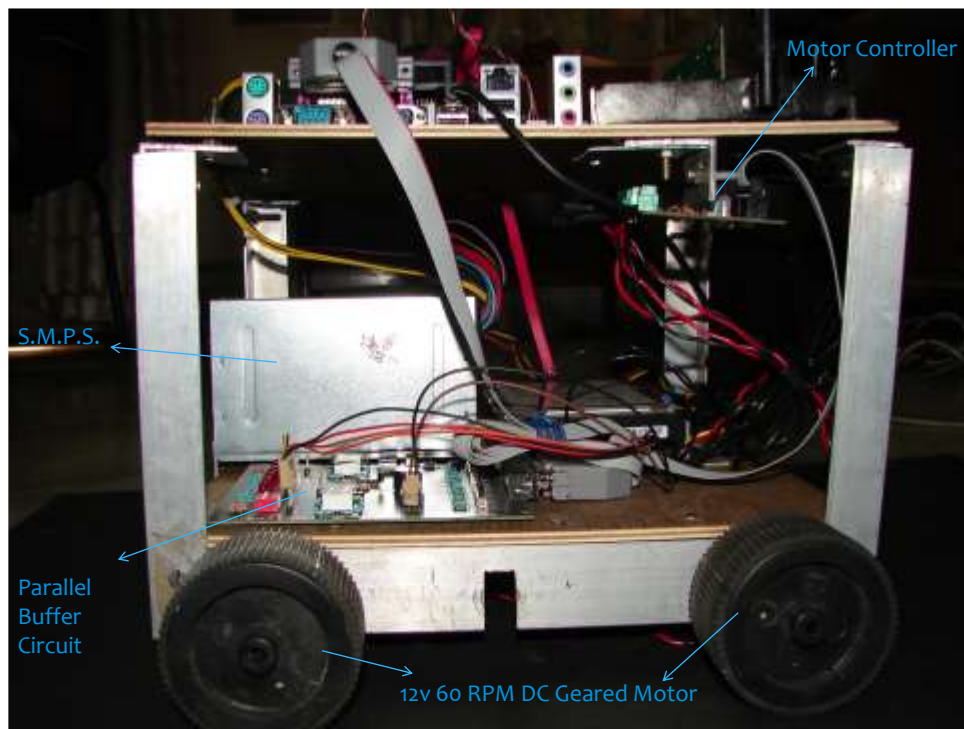


Fig 5.4 Side View of Robot

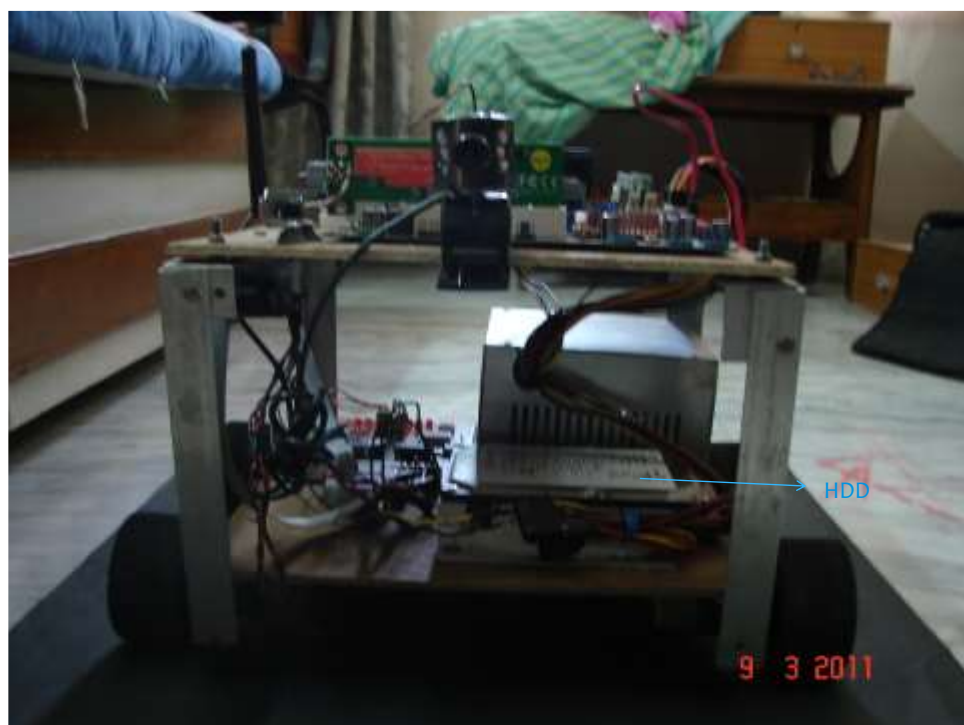


Fig 5.5 Front View of Robot

#### **5.1.4 Data Sheets**

- Buffer IC- HD74ls244
- Dual H-Bridge motor driver- L298
- Voltage Regulator – LM7805







## 5.2 Software Architecture

### 5.2.1 Block Diagram Of Software Architecture

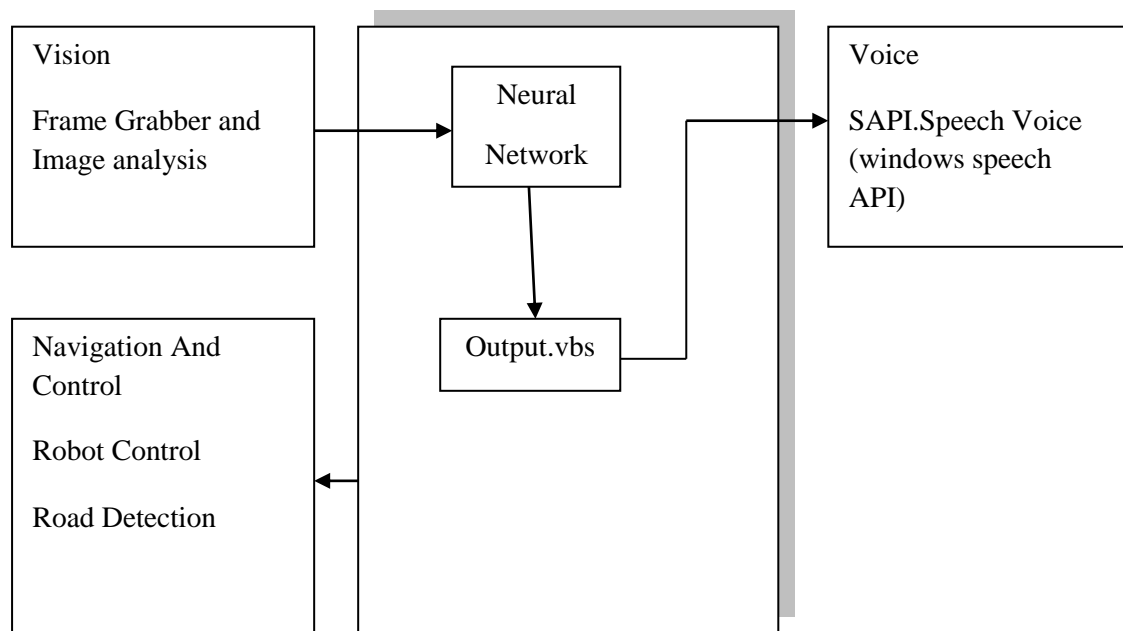


Fig 5.6 Block Diagram Of Software Architecture

# **Chapter 6**

## **Neural Networks and its Application**

## **6.1 Introduction**

### **6.1.1 What is Neural Network?**

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (ecogni) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the recognition. This is true of ANNs as well.

### **6.1.2 Historical Background**

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras.

Many important advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, important advances were made by relatively few people. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers, and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding.

The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. But the technology available at that time did not allow them to do too much.

### **6.1.3 Why use Neural Networks?**

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an “expert” in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer



“what if” questions.

Other advantages include:

1. **Adaptive learning:** An ability to learn how to do tasks based on the data given for training or initial experience.
2. **Self-Organization:** An ANN can create its own recognition or representation of the information it receives during learning time.
3. **Real Time Operation:** ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. **Fault Tolerance via Redundant Information Coding:** Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

## 6.2 Human and Artificial Neurons

### 6.2.1 How does Human Brain learns ?

Much is still unknown about how the brain trains itself to process information, so theories abound. In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*. The neuron sends out spikes of electrical activity through a long, thin strand known as an *axon*, which splits into thousands of branches. At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected recognition. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

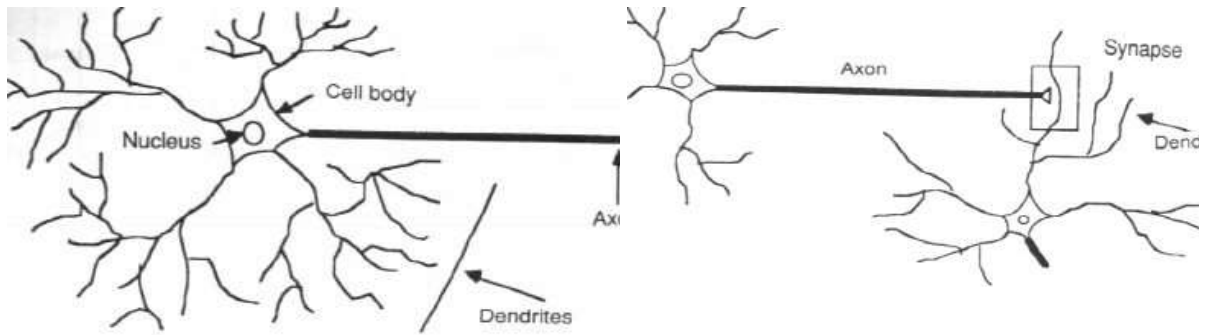


Fig 6.1 Components of a neuron

Fig 6.2 The synapse

### 6.2.2 From Human Neuron to Artificial Neurons

We conduct these neural networks by first trying to deduce the essential features of recognition and their interconnections. We then typically program a computer to simulate these features. However because our knowledge of recognition is incomplete and our computing power is limited, our models are necessarily gross recognition of real networks of recognition.

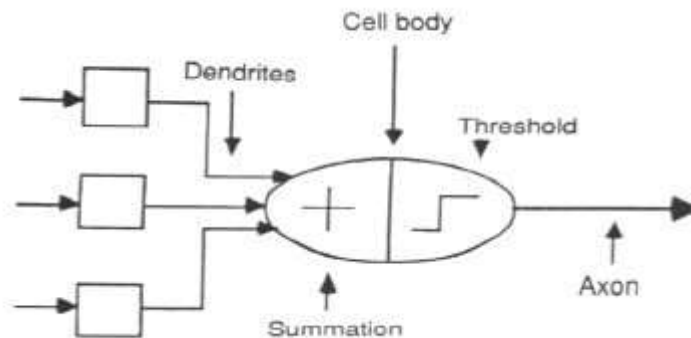


Fig 6.3 The neuron model

## 6.3 An Engineering Approach

### 6.3.1 A simple Neuron

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation: the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

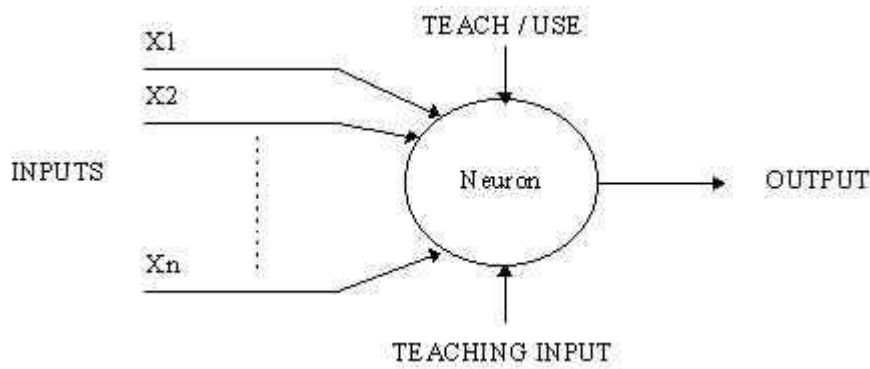


Fig 6.4 A simple neuron

### 6.3.2 Firing rules

The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained.

A simple firing rule can be implemented by using Hamming distance technique. The rule goes as follows:

Take a collection of training patterns for a node, some of which cause it to fire (the 1-taught set of patterns) and others which prevent it from doing so (the 0-taught set). Then the patterns not in the collection cause the node to fire if, on comparison, they have more input elements in common with the 'nearest' pattern in the 1-taught set than with the 'nearest' pattern in the 0-taught set. If there is a tie, then the pattern remains in the undefined state.

For example, a 3-input neuron is taught to output 1 when the input (X1, X2 and X3) is 111 or 101 and to output 0 when the input is 000 or 001. Then, before applying the firing rule, the truth table is;

X1:		0	0	0	0	1	1	1	1
X2:		0	0	1	1	0	0	1	1
X3:		0	1	0	1	0	1	0	1
OUT:		0	0	0/1	0/1	0/1	1	0/1	1

As an example of the way the firing rule is applied, take the pattern 010. It differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements and from 111 in 2 elements.

Therefore, the ‘nearest’ pattern is 000 which belongs in the 0-taught set. Thus the firing rule requires that the neuron should not fire when the input is 001. On the other hand, 011 is equally distant from two taught patterns that have different outputs and thus the output stays undefined (0/1).

By applying the firing in every column the following truth table is obtained;

X1:		0	0	0	0	1	1	1	1
X2:		0	0	1	1	0	0	1	1
X3:		0	1	0	1	0	1	0	1
OUT:		0	0	0	0/1	0/1	1	1	1

The difference between the two truth tables is called the recognition *of the neuron*. Therefore the firing rule gives the neuron a sense of similarity and enables it to respond ‘sensibly’ to patterns not seen during training.

### 6.3.3 Pattern Recognition -an example

An important application of neural networks is pattern recognition. Pattern recognition can be implemented by using a feed-forward (figure 1) neural network that has been trained accordingly. During training, the network is trained to associate outputs with input patterns. When the network is used, it identifies the input pattern and tries to output the associated output pattern. The power of neural networks comes to life when a pattern that has no output associated with it, is given as an input. In this case, the network gives the output that corresponds to a taught input pattern that is least different from the given pattern.

For Example:

The network of figure 1 is trained to recognition the patterns T and H. The associated patterns are all black and all white respectively as shown below.



If we represent black squares with 0 and white squares with 1 then the truth tables for the 3 neurons after recognition are;

### **Top neuron**

X11:		0	0	0	0	1	1	1	1
X12:		0	0	1	1	0	0	1	1
X13:		0	1	0	1	0	1	0	1
OUT:		0	0	1	1	0	0	1	1

### **Middle neuron**

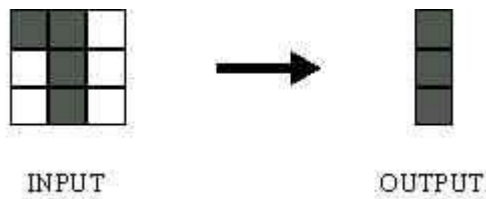
X21:		0	0	0	0	1	1	1	1
X22:		0	0	1	1	0	0	1	1
X23:		0	1	0	1	0	1	0	1
OUT:		1	0/1	1	0/1	0/1	0	0/1	0

### **Bottom neuron**

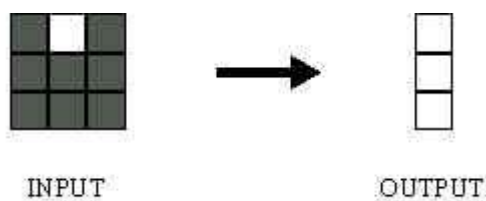
X21:		0	0	0	0	1	1	1	1
X22:		0	0	1	1	0	0	1	1
X23:		0	1	0	1	0	1	0	1

OUT:		1	0	1	1	0	0	1	0
------	--	---	---	---	---	---	---	---	---

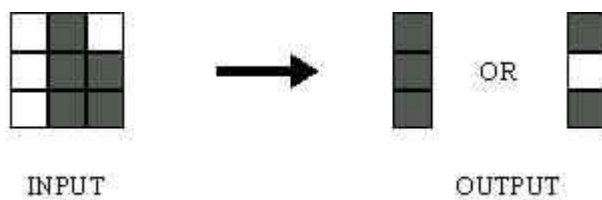
From the tables it can be seen the following associations can be extracted:



In this case, it is obvious that the output should be all blacks since the input pattern is almost the same as the 'T' pattern.



Here also, it is obvious that the output should be all whites since the input pattern is almost the same as the 'H' pattern.



Here, the top row is 2 errors away from the T and 3 from an H. So the top output is black. The middle row is 1 error away from both T and H so the output is random. The bottom row is 1 error away from T and 2 away from H. Therefore the output is black. The total output of the network is still in favor of the T shape.

### 6.3.4 A more complicated neuron.

The previous neuron doesn't do anything that conventional computers don't do already. A more sophisticated neuron (figure 2) is the McCulloch and Pitts model (MCP). The difference from the previous model is that the inputs are 'weighted'; the effect that each input has at decision making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a pre-set threshold value, the neuron fires. In any other case the neuron does not fire.

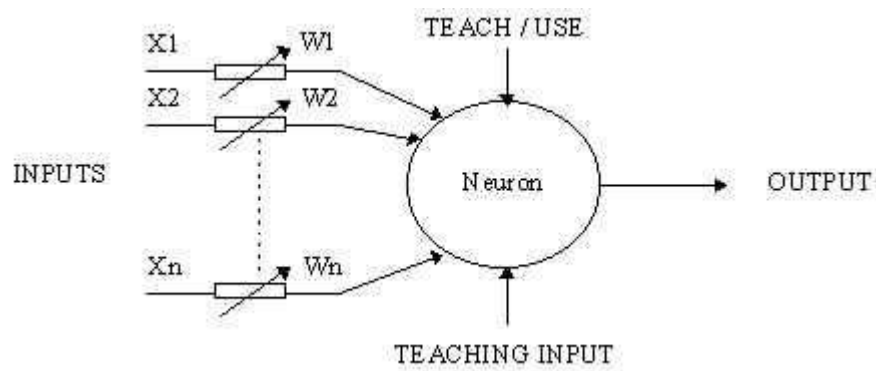


Fig 6.5. An MCP neuron

In mathematical terms, the neuron fires if and only if;

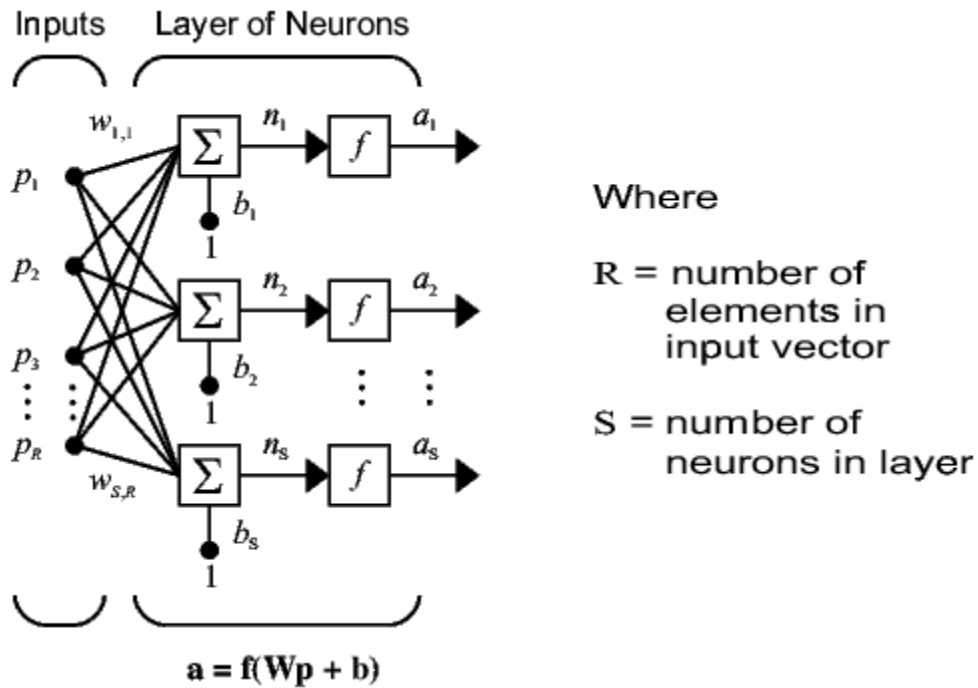
$$X1W1 + X2W2 + X3W3 + ... > T$$

The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to 'adapt'; the most used ones are the Delta rule and the back error propagation. The former is used in feed-forward networks and the latter in feedback networks.

## 6.4 Network Architectures

### 6.4.1 A Layer of Neurons

A one-layer network with  $R$  input elements and  $S$  neurons follows.



In this network, each element of the input vector  $\mathbf{p}$  is connected to each neuron input through the weight matrix  $\mathbf{W}$ . The  $i$ th neuron has a summer that gathers its weighted inputs and bias to form its own scalar output  $n(i)$ . The various  $n(i)$  taken together form an  $S$ -element net input vector  $\mathbf{n}$ . Finally, the neuron layer outputs form a column vector  $\mathbf{a}$ . The expression for  $\mathbf{a}$  is shown at the bottom of the figure.

Note that it is common for the number of inputs to a layer to be different from the number of neurons (i.e.,  $R$  is not necessarily equal to  $S$ ). A layer is not constrained to have the number of its inputs equal to the number of its neurons.

You can create a single (composite) layer of neurons having different transfer functions simply by putting two of the networks shown earlier in parallel. Both networks would have the same inputs, and each network would create some of the outputs.

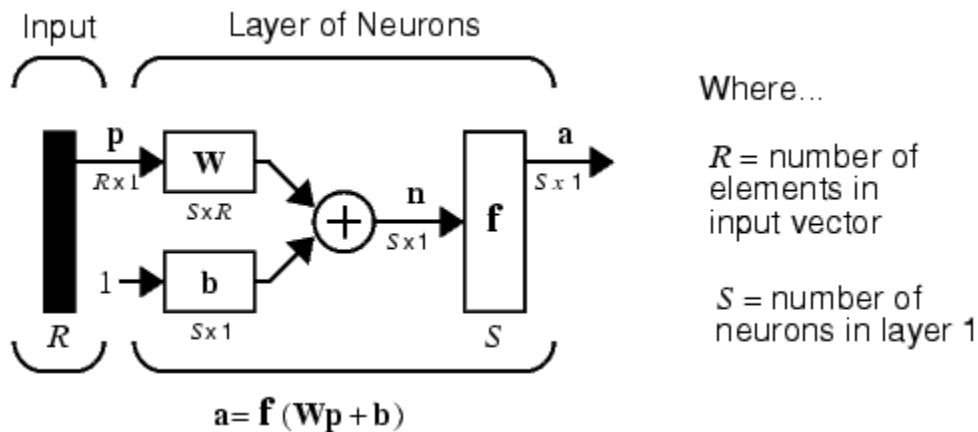
The input vector elements enter the network through the weight matrix  $\mathbf{W}$ .



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}$$

Note that the row indices on the elements of matrix  $\mathbf{W}$  indicate the destination neuron of the weight, and the column indices indicate which source is the input for that weight. Thus, the indices in  $w_{1,2}$  say that the strength of the signal *from* the second input element *to* the first (and only) neuron is  $w_{1,2}$ .

The  $S$  neuron  $R$  input one-layer network also can be drawn in abbreviated notation.

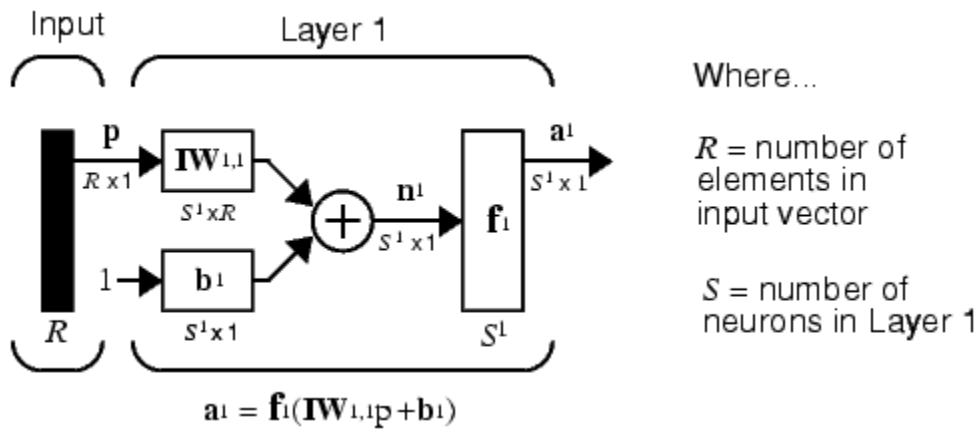


Here  $\mathbf{p}$  is an  $R$  length input vector,  $\mathbf{W}$  is an  $S \times R$  matrix, and  $\mathbf{a}$  and  $\mathbf{b}$  are  $S$  length vectors. As defined previously, the neuron layer includes the weight matrix, the multiplication operations, the bias vector  $\mathbf{b}$ , the summer, and the transfer function boxes.

### Inputs and Layers

To describe networks having multiple layers, the notation must be extended. Specifically, it needs to make a distinction between weight matrices that are connected to inputs and weight matrices that are connected between layers. It also needs to identify the source and destination for the weight matrices.

We will call weight matrices connected to inputs *input weights*; we will call weight matrices coming from layer outputs *layer weights*. Further, superscripts are used to identify the source (second index) and the destination (first index) for the various weights and other elements of the network. To illustrate, the one-layer multiple input network shown earlier is redrawn in abbreviated form below.

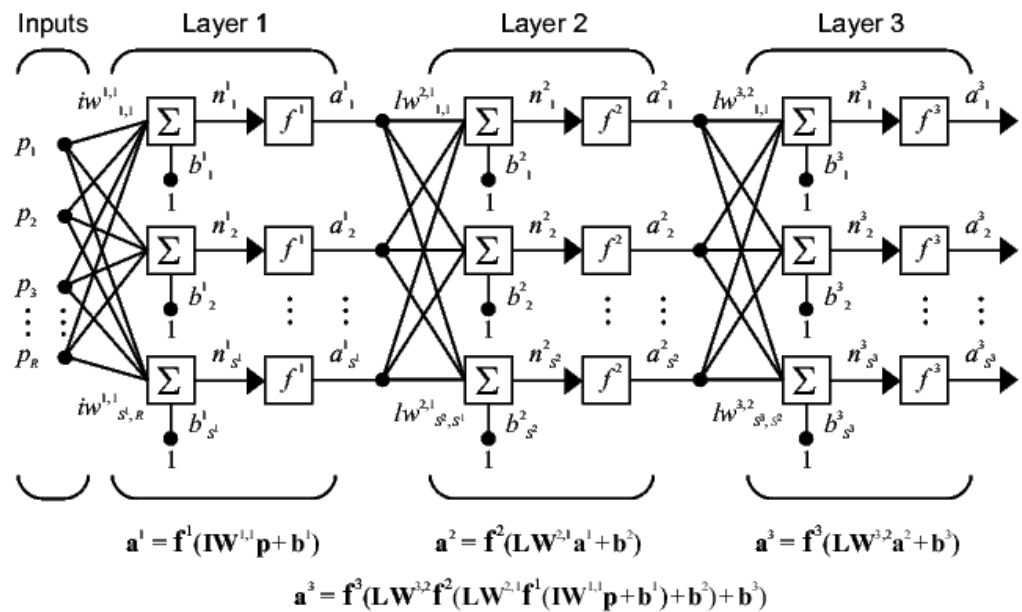


As you can see, the weight matrix connected to the input vector  $\mathbf{p}$  is labeled as an input weight matrix ( $\mathbf{IW}^{1,1}$ ) having a source 1 (second index) and a destination 1 (first index). Elements of layer 1, such as its bias, net input, and output have a superscript 1 to say that they are associated with the first layer.

Multiple Layers of Neurons uses layer weight ( $\mathbf{LW}$ ) matrices as well as input weight ( $\mathbf{IW}$ ) matrices.

## 6.4.2 Multiple Layers of Neurons

A network can have several layers. Each layer has a weight matrix  $\mathbf{W}$ , a bias vector  $\mathbf{b}$ , and an output vector  $\mathbf{a}$ . To distinguish between the weight matrices, output vectors, etc., for each of these layers in the figures, the number of the layer is appended as a superscript to the variable of interest. You can see the use of this layer notation in the three-layer network shown below, and in the equations at the bottom of the figure.

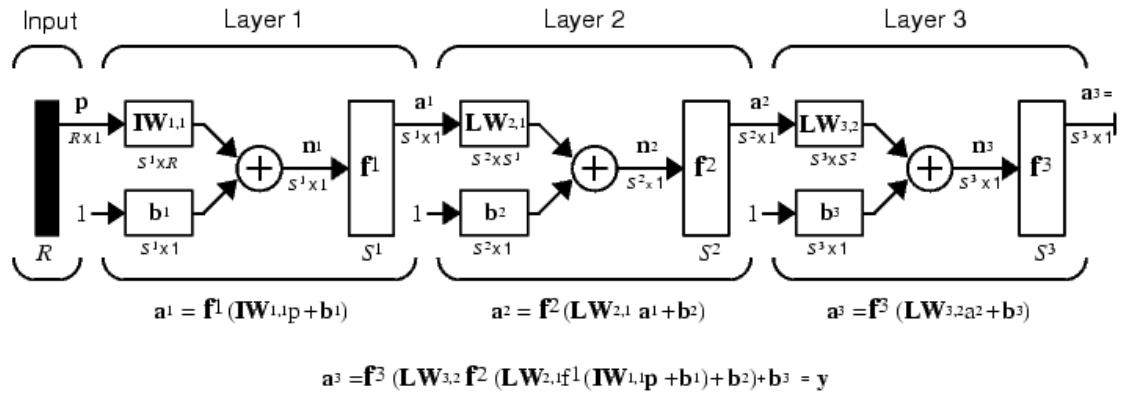


The network shown above has  $R^1$  inputs,  $S^1$  neurons in the first layer,  $S^2$  neurons in the second layer, etc. It is common for different layers to have different numbers of neurons. A constant input 1 is fed to the bias for each neuron.

Note that the outputs of each intermediate layer are the inputs to the following layer. Thus layer 2 can be analyzed as a one-layer network with  $S^1$  inputs,  $S^2$  neurons, and an  $S^2 \times S^1$  weight matrix  $W^2$ . The input to layer 2 is  $\mathbf{a}^1$ ; the output is  $\mathbf{a}^2$ . Now that all the vectors and matrices of layer 2 have been identified, it can be treated as a single-layer network on its own. This approach can be taken with any layer of the network.

The layers of a multilayer network play different roles. A layer that produces the network output is called an *output layer*. All other layers are called *hidden layers*. The three-layer network shown earlier has one output layer (layer 3) and two hidden layers (layer 1 and layer 2). Some authors refer to the inputs as a fourth layer. This toolbox does not use that designation.

The same three-layer network can also be drawn using abbreviated notation.



Multiple-layer networks are quite powerful. For instance, a network of two layers, where the first layer is sigmoid and the second layer is linear, can be trained to approximate any function (with a finite number of discontinuities) arbitrarily well. This kind of two-layer network is used extensively in Backpropagation.

Here it is assumed that the output of the third layer,  $\mathbf{a}^3$ , is the network output of interest, and this output is labelled as  $\mathbf{y}$ . This notation is used to specify the output of multilayer networks.

## 6.5 Applications of neural networks

The 1988 *DARPA Neural Network Study* [DARP88] lists various neural network applications, beginning in about 1984 with the adaptive channel equalizer. This device, which is an outstanding commercial success, is a single- neuron network used in long-distance telephone systems to stabilize voice signals. The DARPA report goes on to list other commercial applications, including a small word recognizer, a process monitor, a sonar classifier, and a risk analysis system.

Neural networks have been applied in many other fields since the DARPA report was written, as described in the next table.

Industry	Business Applications
Aerospace	High-performance aircraft autopilot, flight path simulation, aircraft control systems, autopilot enhancements, aircraft component simulation, and aircraft component fault detection
Automotive	Automobile automatic guidance system, and warranty activity analysis
Banking	Check and other document reading and credit application evaluation
Defence	Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, and signal/image identification
Electronics	Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, and nonlinear modelling
Entertainment	Animation, special effects, and market forecasting
Financial	Real estate appraisal, loan advising, mortgage screening, corporate bond rating, credit-line use analysis, credit card activity tracking, portfolio trading program, corporate financial analysis, and currency price prediction
Industrial	Prediction of industrial processes, such as the output gases of furnaces, replacing complex and costly equipment used for this purpose in the past

Insurance	Policy application evaluation and product optimization
Manufacturing	Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer-chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, and dynamic modeling of chemical process system
Medical	Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, and emergency-room test advisement
Oil and gas	Exploration
Robotics	Trajectory control, forklift robot, manipulator controllers, and vision systems
Speech	Speech recognition, speech compression, vowel classification, and text-to-speech synthesis
Securities	Market analysis, automatic bond rating, and stock trading advisory systems
Telecommunications	Image and data compression, automated information services, real-time translation of spoken language, and customer payment processing systems
Transportation	Truck brake diagnosis systems, vehicle scheduling, and routing systems

## 6.6 Implementation of Neural Networks in ANS using MATLAB

MATLAB's neural network toolbox basically performs three operations:

- **FITTING A FUNCTION :** Practically any function can be implemented using this operation. For this either the command-line functions or using inbuilt GUI function using the command *nftool*.
- **CLUSTERING DATA:** Clustering data is another excellent application for neural networks. This process involves grouping data by similarity. For example, you might perform:
  - Market segmentation by grouping people according to their buying patterns
  - Data mining by partitioning data into related subsets
  - Bioinformatic analysis by grouping genes with related expression patterns

For this we can directly use the GUI function *nctool*.

- **RECOGNIZING PATTERNS:** In addition to function fitting, neural networks are also good at recognizing patterns. For this we can directly use the GUI function *nprtool*.

Our project basically makes use of **recognizing pattern** operation and we will take up it in detail now.

### 6.6.1 Pattern Recognition

#### Defining a Problem

To define a pattern recognition problem, arrange a set of  $Q$  input vectors as columns in a matrix. Then arrange another set of  $Q$  target vectors so that they indicate the classes to which the input vectors are assigned. There are two approaches to creating the target vectors.

One approach can be used when there are only two classes; you set each scalar target value to either 1 or 0, indicating which class the corresponding input belongs to. For instance, you can define the exclusive-or classification problem as follows:

- $\text{inputs} = [0 \ 1 \ 0 \ 1; 0 \ 0 \ 1 \ 1];$
- $\text{targets} = [0 \ 1 \ 0 \ 1];$

Alternately, target vectors can have  $N$  elements, where for each target vector, one element is 1 and the others are 0. This defines a problem where inputs are to be classified into  $N$  different classes. For example, the following lines show how to define a classification problem that divides the corners of a 5-by-5-by-5 cube into three classes:

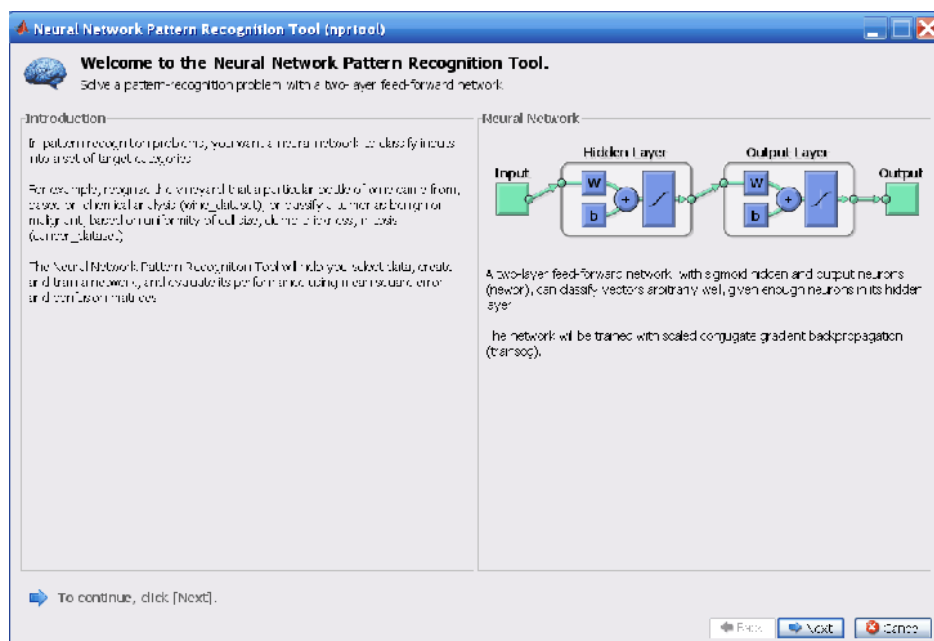
- The origin (the first input vector) in one class
- The corner farthest from the origin (the last input vector) in a second class
- All other points in a third class
  - `inputs = [0 0 0 0 5 5 5 5; 0 0 5 5 0 0 5 5; 0 5 0 5 0 5 0 5];`
  - `targets = [1 0 0 0 0 0 0 0; 0 1 1 1 1 1 1 0; 0 0 0 0 0 0 0 1];`

Classification problems involving only two classes can be represented using either format. The targets can consist of either scalar 1/0 elements or two-element vectors, with one element being 1 and the other element being 0.

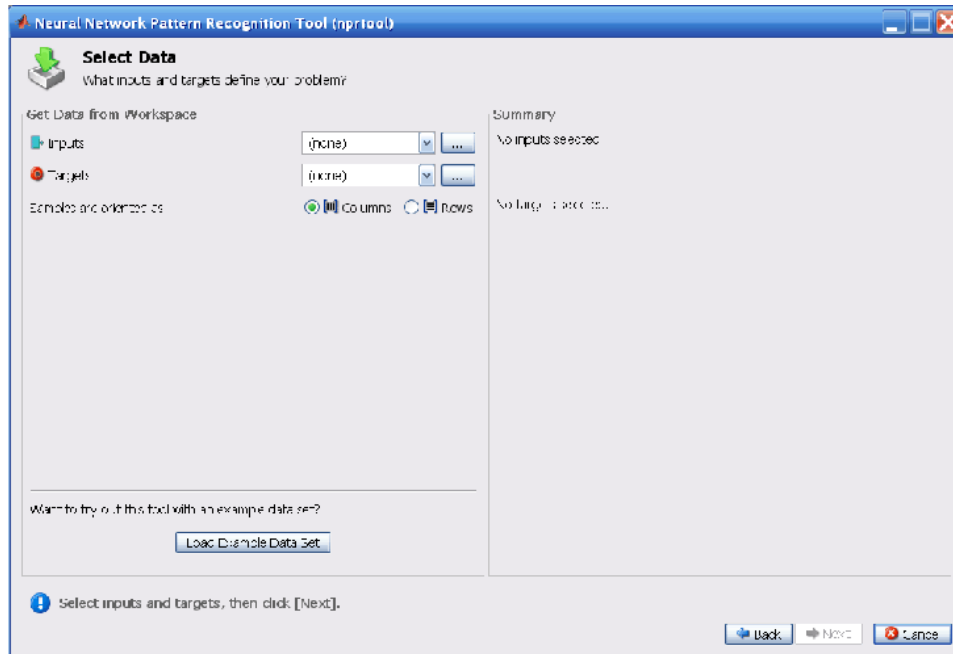
## 6.6.2 Using the Neural Network Pattern Recognition Tool GUI

1. Open the Neural Network Pattern Recognition Tool window with this command:

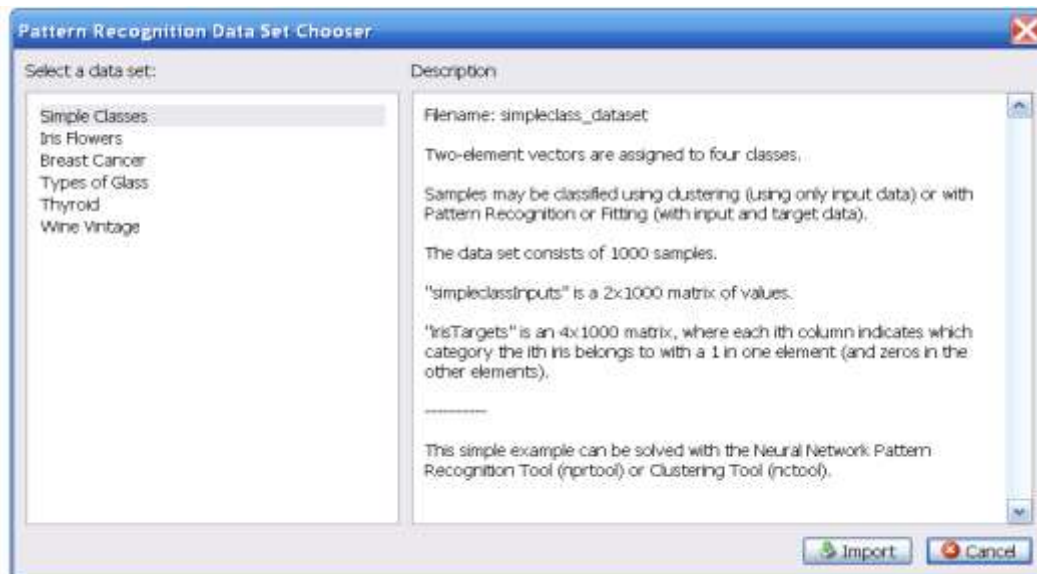
*nprtool*



2. Click **Next** to proceed. The Select Data window opens.

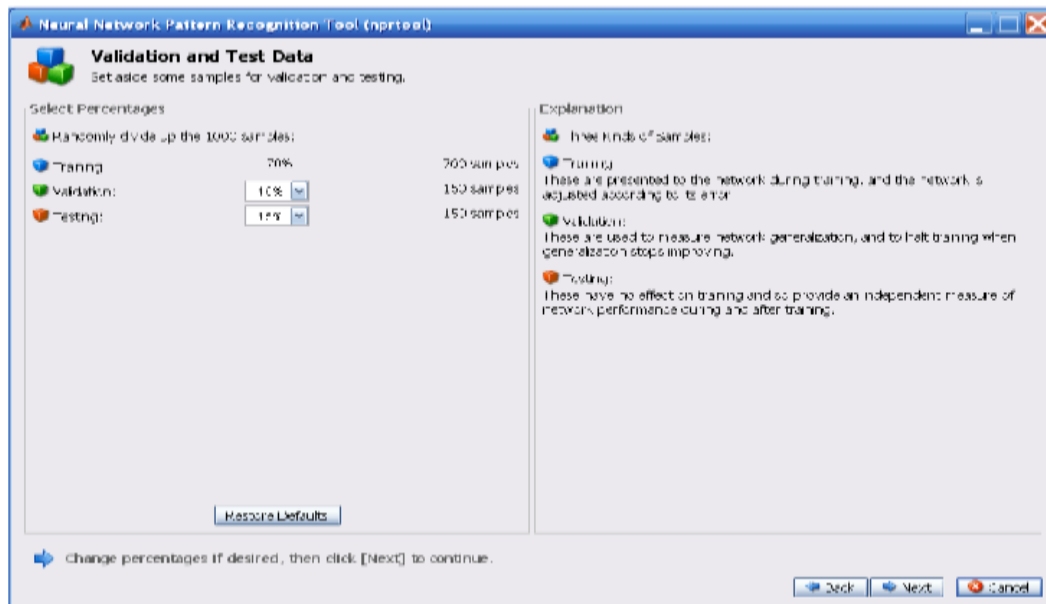


3. Click **Load Example Data Set**. The Pattern Recognition Data Set Chooser window opens.



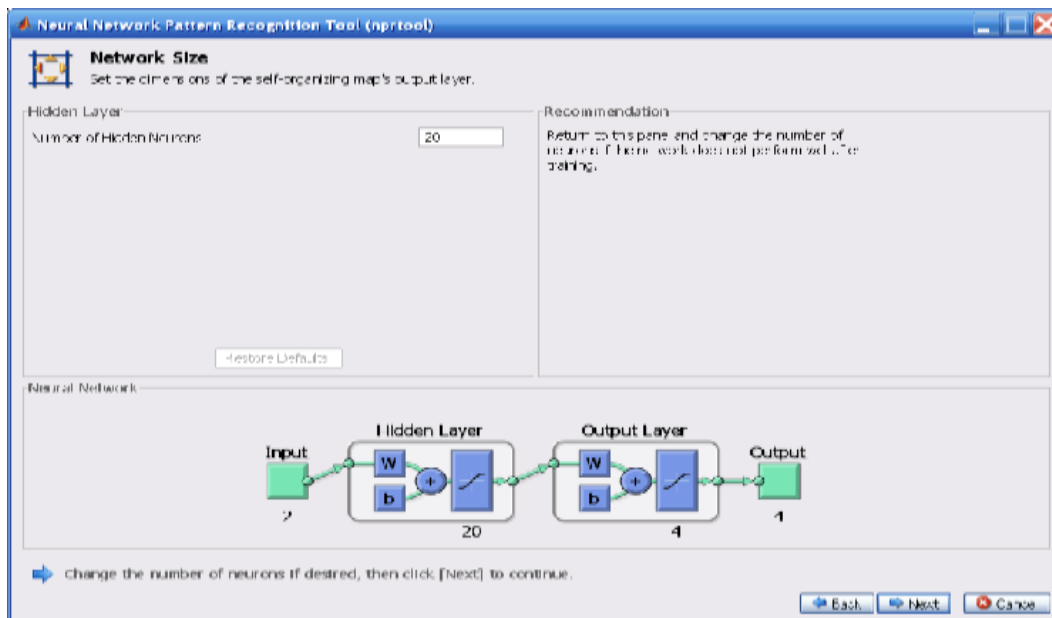
4. In this window, select **Simple Classes**, and click **Import**. You return to the Select Data window.
5. Click **Next** to continue to the Validate and Test Data window, shown in the following figure. Validation and test data sets are each set to 15% of the original data.



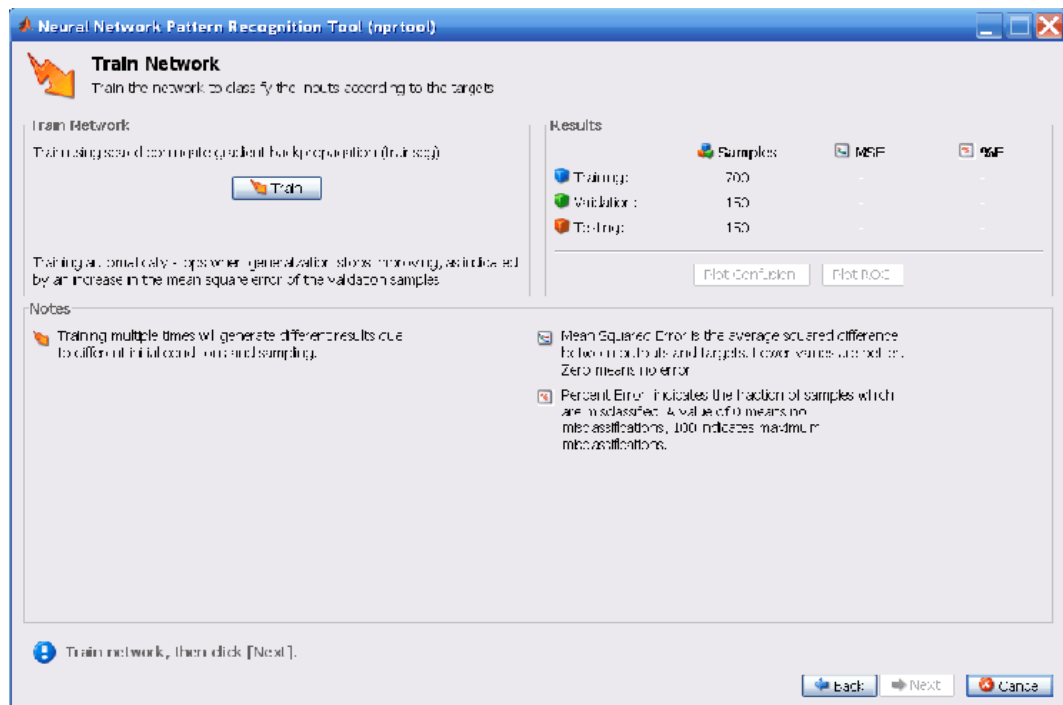


## 6. Click Next.

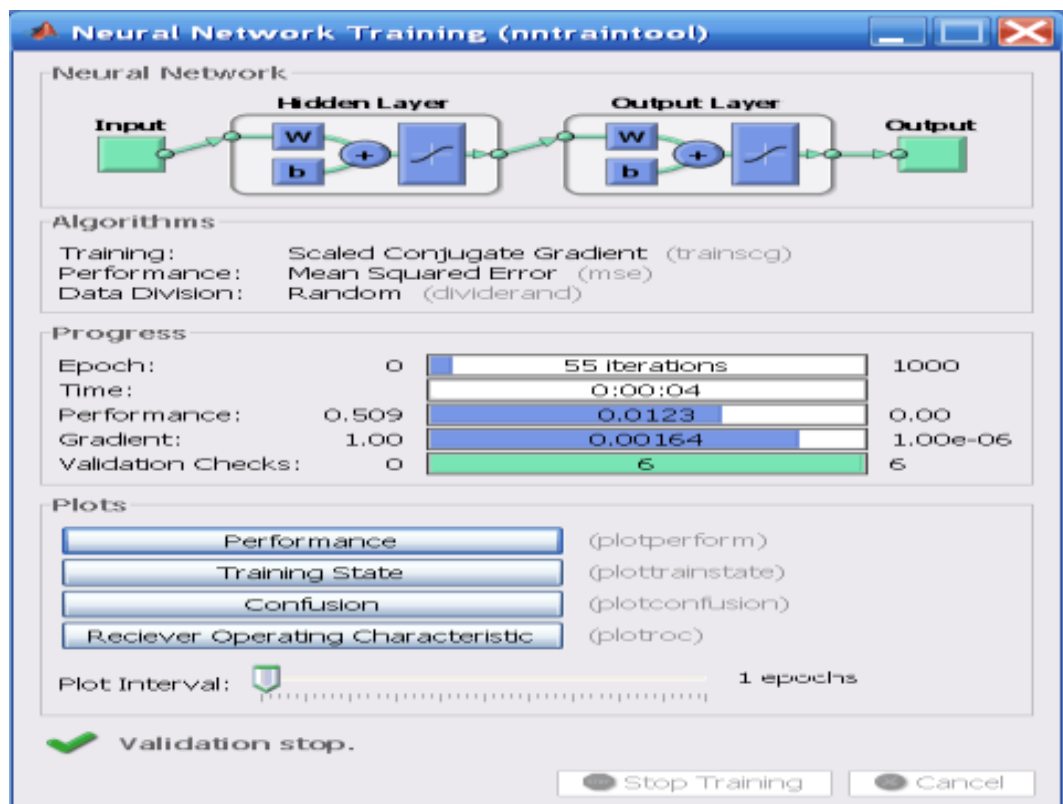
The number of hidden neurons is set to 20. You can change this in another run if you want. You might want to change this number if the network does not perform as well as you expect.



7. Click **Next**.

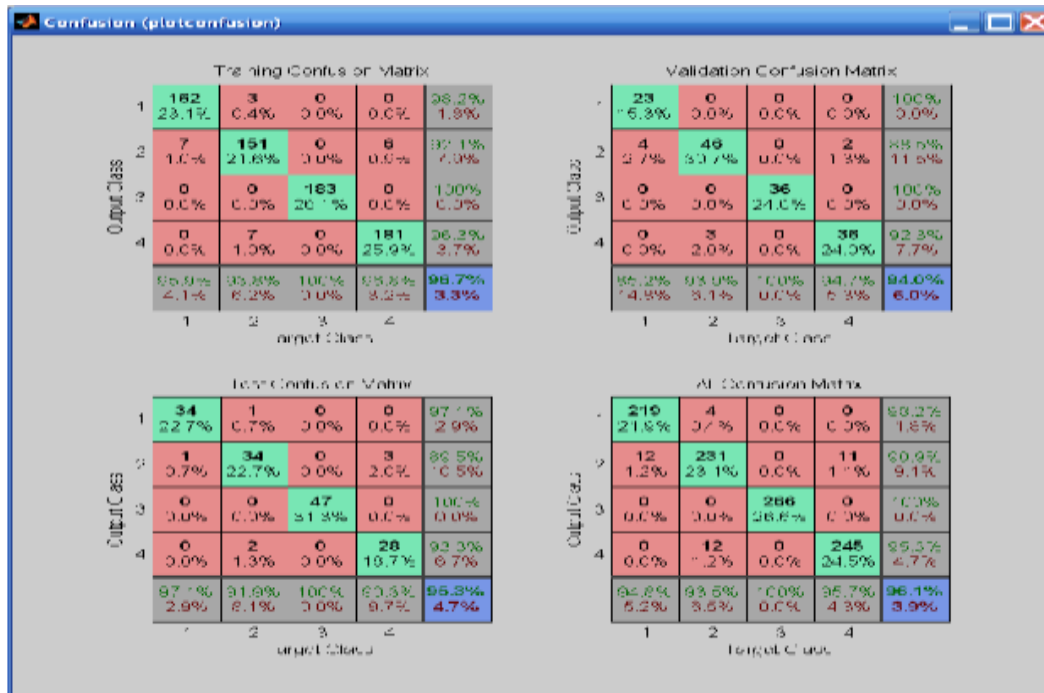


8. Click **Train**.

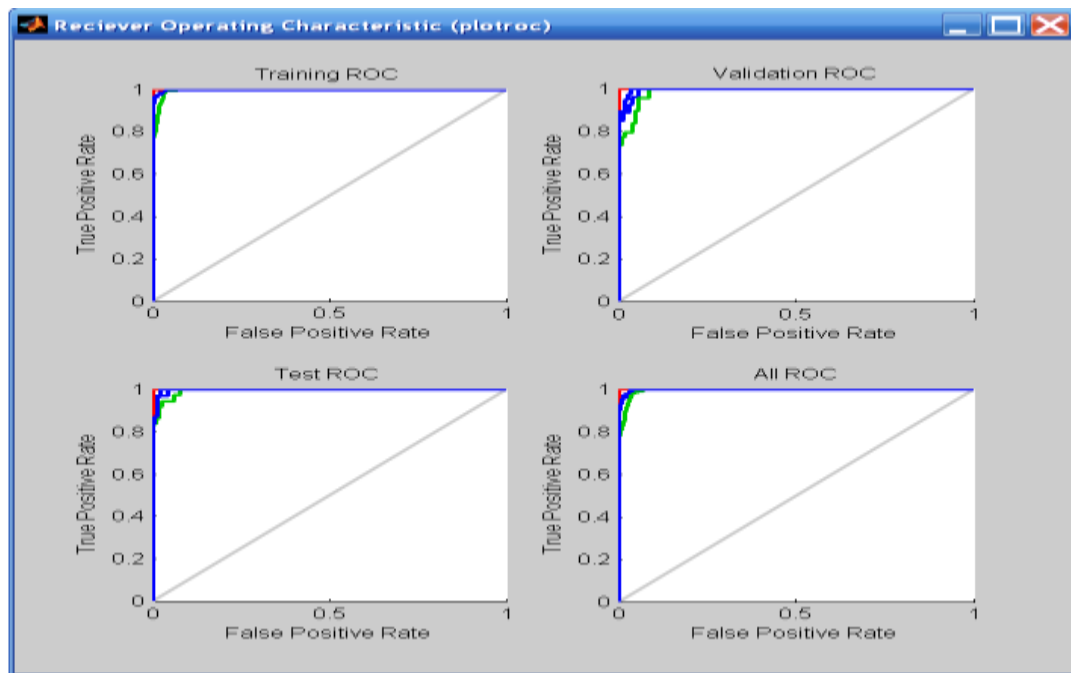


9. Under the **Plots** pane, click **Confusion** in the Neural Network Pattern Recognition Tool.

The next figure shows the confusion matrices for training, testing, and validation, and the three kinds of data combined. The network's outputs are almost perfect, as you can see by the high numbers of correct responses in the green squares and the low numbers of incorrect responses in the red squares. The lower right blue squares illustrate the overall accuracies.

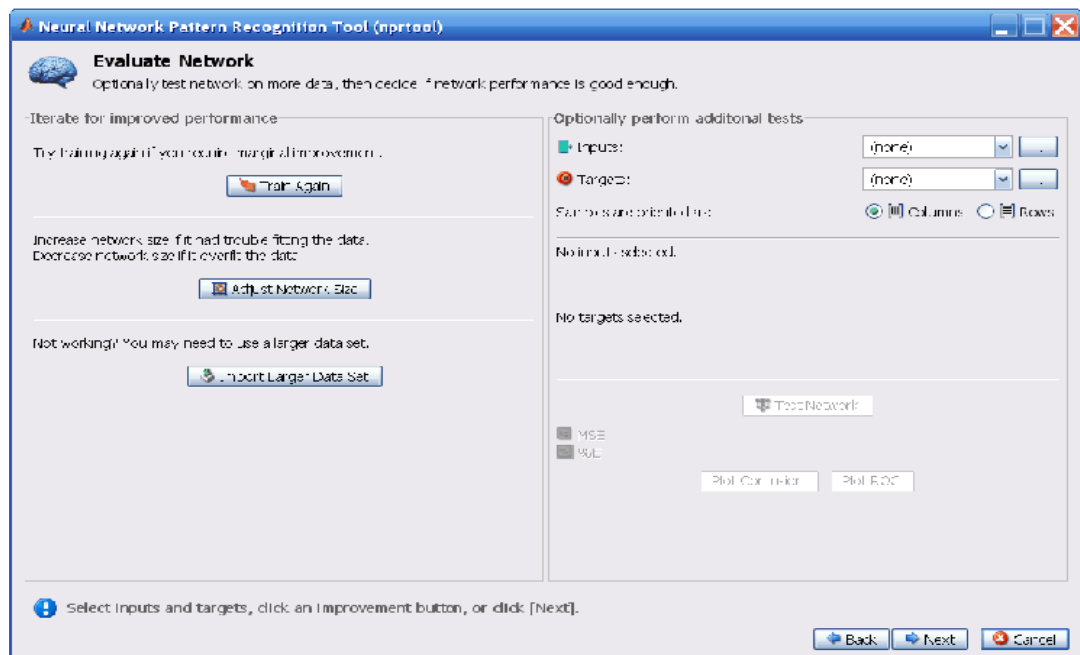


10. Plot the Receiver Operating Characteristic (ROC) curve. Under the **Plots** pane, click **Receiver Operating Characteristic** in the Neural Network Pattern Recognition Tool.

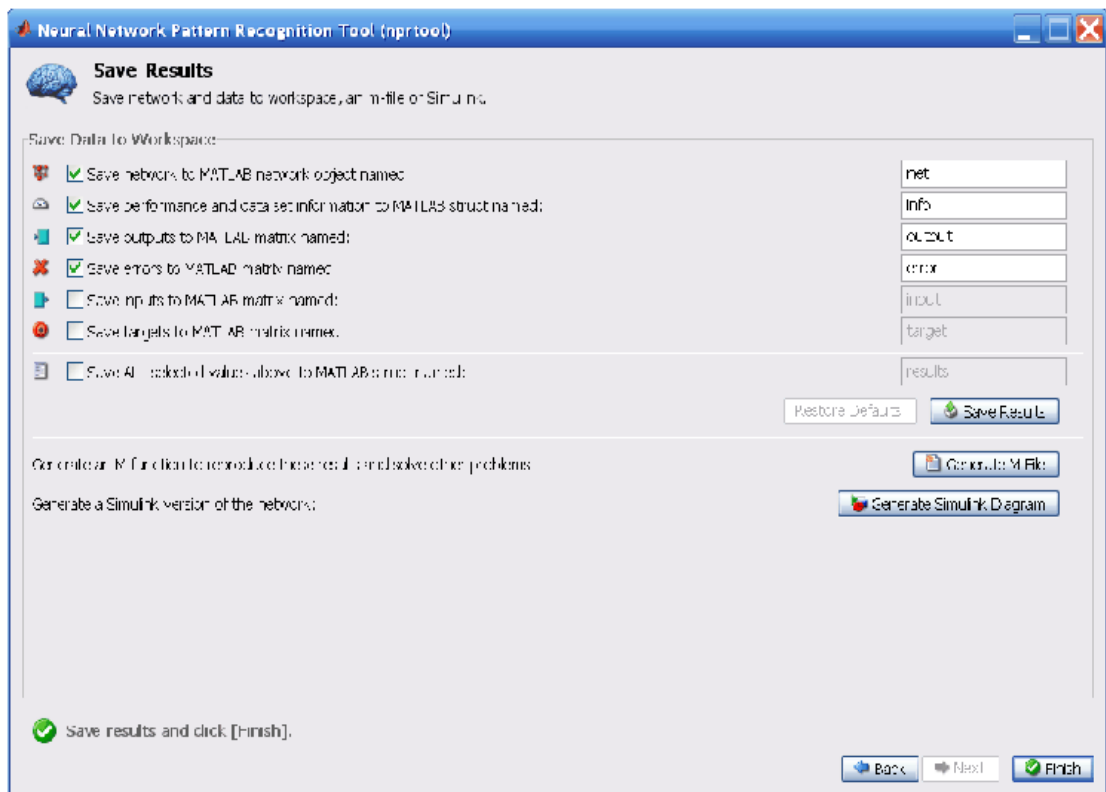


The colored lines in each axis represent the ROC curves for each of the four categories of this simple test problem. The *ROC curve* is a plot of the true positive rate (sensitivity) versus the false positive rate ( $1 - \text{specificity}$ ) as the threshold is varied.

12. In the Neural Network Pattern Recognition Tool, click **Next** to evaluate the network.



13. At this point, you can test the network against new data. If you are dissatisfied with the network's performance on the original or new data, you can train again, increase the number of neurons or perhaps get a larger training data set.
14. When you are satisfied with the network performance, click **Next**.



15. Use the buttons on this screen to save your results.

- You now have the network saved as net1 in the workspace. You can perform additional tests on it or put it to work on new inputs using the sim function.
- If you click **Generate M-File**, the tool creates an M-file, with commands that recreate the steps that you have just performed from the command line. Generating an M-file is a good way to learn how to use the command-line operations of the Neural Network Toolbox™ software.

16. When you have saved your results, click **Finish**.

### Input and Target Data Set:-

To train network we have used a dataset of around 6000 characters of different fonts. Each character was initially in a form of 12x12 image which was then converted into a column vector of size 144. All such vectors were arranged as columns in a single matrix. This matrix was the final input to the network for training purpose. Once the network is trained, the input is also fed to it in the same format. Below is a small part of the network input data.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	1	0	0	0	0	0	0
6	1	0	0	0	1	0	1	1	1	0	0
7	1	1	1	1	1	1	1	1	1	0	1
8	0	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1	0
10	0	0	0	0	0	0	0	0	0	1	0
11	0	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	0
17	1	0	0	0	1	0	0	1	1	0	0
18	1	1	1	1	1	1	1	1	1	1	1
19	1	1	1	1	1	1	1	1	1	1	1
20	0	0	0	0	1	0	1	1	1	0	0
21	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	1	0	0	0	0	0	0
29	1	0	0	0	1	0	1	1	1	0	0
30	1	1	1	1	1	1	1	1	1	1	1
31	1	1	1	1	1	1	1	1	1	0	1
32	1	0	1	0	1	1	1	1	1	0	1
33	0	0	0	0	1	0	1	0	0	0	0
34	0	0	0	0	0	0	0	0	0	1	0
35	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0

The rows extend up to 144 while there are 6188 columns, each for one input.

The targets for training the network were fed in form of 30x6188 matrix. Since the network has to identify 26 english alphabets and four directional arrows thus the target matrix has 30 rows, one row for each input. For every input the value in row corresponding to that character is 1 and is 0 elsewhere. No column has 1 in two places simultaneously.

Below is a small portion of target matrix:-

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0

# **Chapter 7**

# **Software**

# **Design**



## 7.1 Autonomous mode

In autonomous mode of operation the robot asks user to enter a destination to be reached and once it is provided with one it autonomously recognize the path and follows it to reach its destination. It call `autonomous.m` function which takes video objects, port objects, destination, network object as input arguments.

### 7.1.1 Flow of Control Diagram

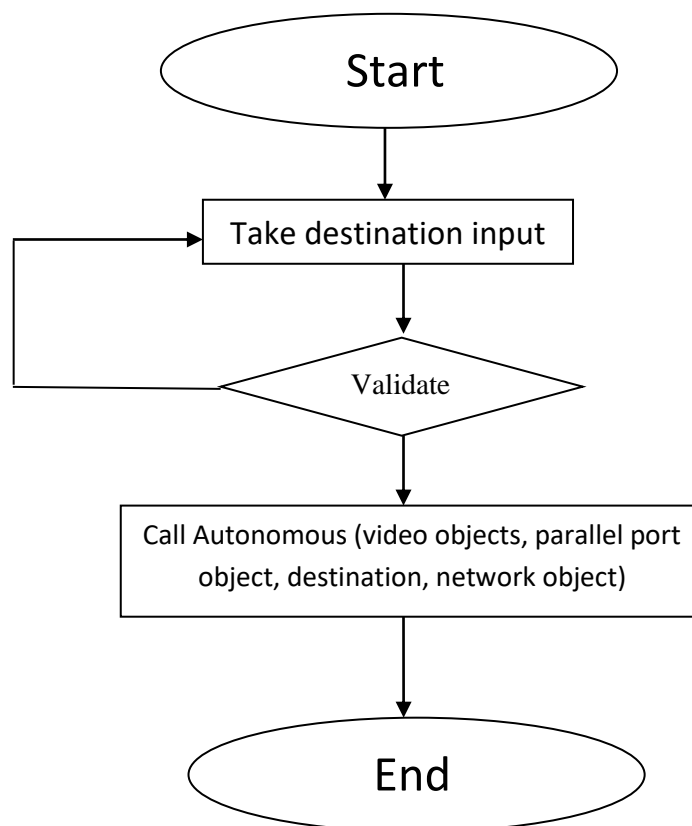


Fig 7.1 Flow of Control Diagram

## 7.1.2 Working and Algorithm

### 7.1.2.1 Autonomous.m

Autonomous.m calls functions for navigation and receives back information about the sign board. This information is then fed into neural network and corresponding output is analyzed. The parameters it receives back from road recognition functions are the image data of different characters, their centroids and their height & width.

This image data is fed to neural network and output is obtained. This is done by the following code snippet:

```
img_data=double(img_data);  
op=sim(net,img_data);  
[max_rec rec_char]=max(op);
```

Here previously image data was logical in format, but then it is type casted into double data type. The output of network is in form of matrix each row depicting one of the English alphabets and each column depicting one of the could be character found in the image. The last line in above code snippet finds max. output in each column and its index.

The function below arranges the characters found according to their centroids.

```
for i=1:s(2)  
    if (centroid(2,i)< avg)  
        if (max_rec(i)>0.9)  
  
            if (k==1)  
                character=char(64+rec_char(i));  
                str1=strcat(str1,character);  
  
            else  
                distbwcent=centroid(1,i)-centroid(1,temp1);  
                sumofwidth=(rect(i,3))/2 + (rect(temp1,3))/2;  
                if (distbwcent>sumofwidth+20)  
                    character=char(64+rec_char(i));  
                    str1=strcat(str1,',',character);  
                else  
                    character=char(64+rec_char(i));  
                    str1=strcat(str1,character);  
                end  
            end  
            temp1=i;  
            k=k+1;  
        end  
    else  
        if (max_rec(i)>0.5)  
            if (j==1)
```

```

        character=char(64+rec_char(i));
        str2=strcat(str2,character);
    else
        distbwcent=centroid(1,i)-centroid(1,temp2);
        sumofwidth=(rect(i,3))/2 + (rect(temp2,3))/2;
        if (distbwcent>sumofwidth+20)
            character=char(64+rec_char(i));
            str2=strcat(str2,',',character);
        else
            character=char(64+rec_char(i));
            str2=strcat(str2,character);
        end
    end
    temp2=i;
    j=j+1;
end
end
end

```

Then in the above found strings of characters destination is searched for and if found, robot is navigated into the direction mentioned in corresponding string.

### 7.1.2.2 Navigation.m

This function manages the whole act of controlling the motion of robot. Once the autonomous.m is called it automatically calls navigation.m . There are two basic functions of navigation.m :-

- To detect the presence of junction
- If junction not found the drive robot along the road

This can be well understood by the flow of control diagram:

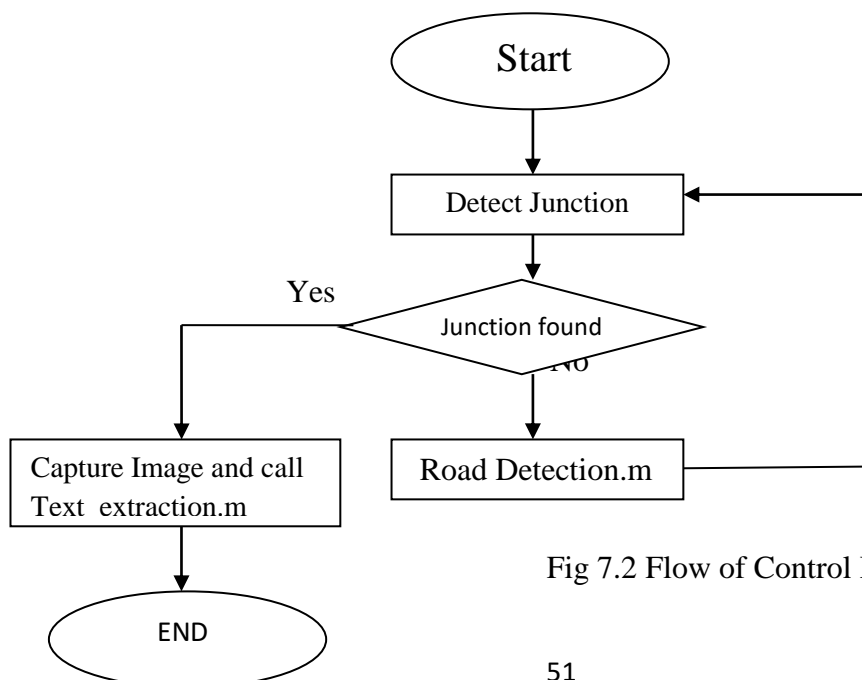


Fig 7.2 Flow of Control Diagram of Navigation

The junction is detected when a red spot is found on the road of a particular size. This operation is performed by the following code snippet:

```
img=getsnapshot(vid1);
img_rgb=ybcr2rgb(img);
r=img_rgb(:,:,1);
g=img_rgb(:,:,2);
b=img_rgb(:,:,3);
r2=((r-g)+(r-b));
r2=((r2==255));
bw=bwconncomp(r2,8);
numPixels = cellfun(@numel,bw.PixelIdxList);
if(max(numPixels)>50000)
    display('Red Blot found')
    pause(2);
    img=getsnapshot(vid2);
    img_rgb=ybcr2rgb(img);
    figure, imshow(img_rgb)
    [img_data centroid rect]=text_extraction1(img_rgb,img);
    stop(vid1);

    return
end
```

Here if the size of red spot is more than 50000 pixel then only it will be classified as a junction.

The road\_navigation function works by finding the centroid of white colour in road image. The position of the centroid decides in which direction the robot has to turn. If the centroid is too much to the left then robot will move in left direction to bring it into centre. . If the centroid is too much to the right then robot will move in right direction. If centroid in close to centre of image then the robot will move straight.

This is done by following code snippet:

```
temp_i=0;
temp_j=0;
count=0;
y=img(:,:,1);
new_img=((y>200));
for i=1:480
    for j=1:640
        if(new_img(i,j)==1)
            temp_i=temp_i+ i;
            temp_j=temp_j+ j;
            count=count+1;
        end
    end
end
cent_i=temp_i/count;
cent_j=temp_j/count;

if(cent_j<288)
    %move left
end
```

```

        bvdata=logical([1 0 1 0 0 0 0 0]);
        putvalue(parport,bvdata);
        pause(t);
        bvdata=logical([0 0 0 0 0 0 0 0]);
        putvalue(parport,bvdata);
elseif((cent_j>=288) & (cent_j<=352))
    %move forward
    bvdata=logical([0 1 1 0 0 0 0 0]);
    putvalue(parport,bvdata);
    pause(t);
    bvdata=logical([0 0 0 0 0 0 0 0]);
    putvalue(parport,bvdata);
elseif(cent_j>352)
    %move right
    bvdata=logical([0 1 0 1 0 0 0 0]);
    putvalue(parport,bvdata);
    pause(t);
    bvdata=logical([0 0 0 0 0 0 0 0]);
    putvalue(parport,bvdata);
end

```

Here *parport* is a parallel port object and data is written to the port using *putvalue* command.

### 7.1.2.3 text\_extraction.m

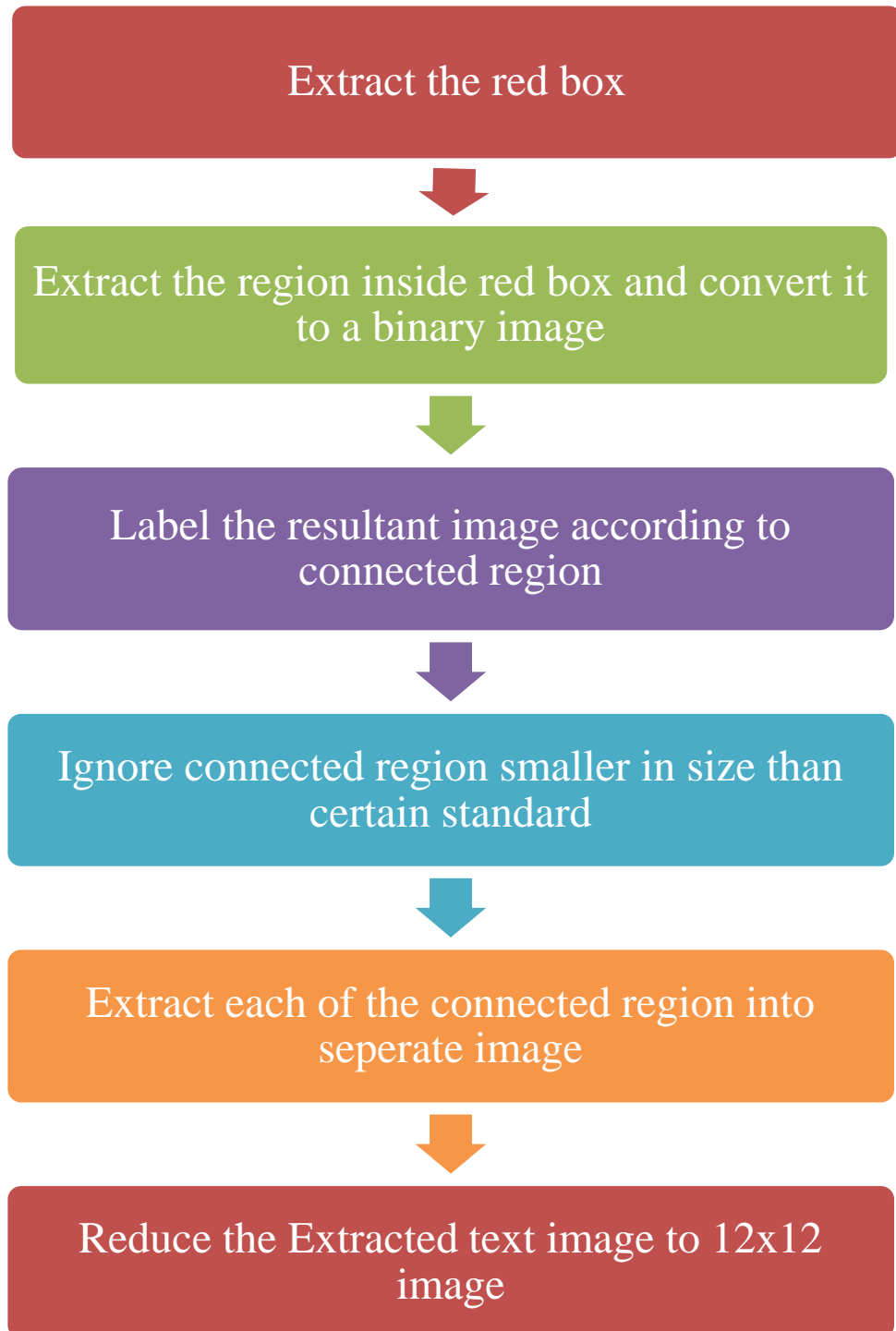


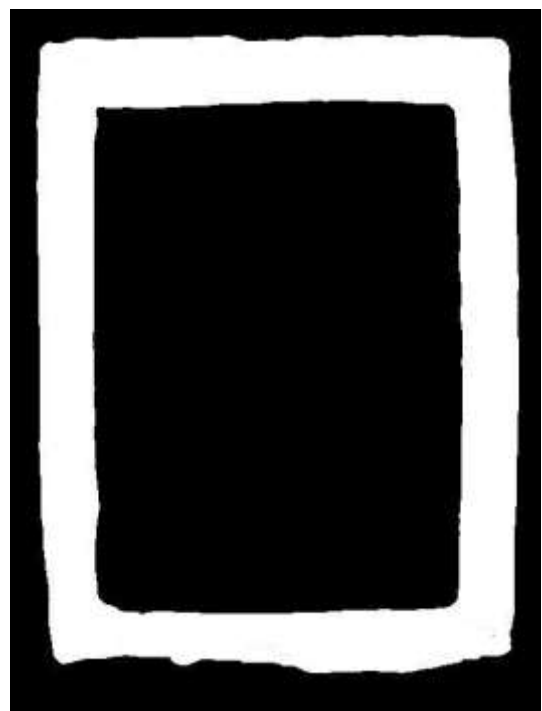
Fig 7.3 Text Extraction Method

### Step 1: Extracting the red region

The red region to be extracted is identified on the basis of luminance, chrominance, and hue. In a YUY2 format image is returned by the webcam. It is similar to ycbcr format. An average value of three components namely, **luminance**(Y), **Chrominance**(Cb), **Hue**(Cr), is found for red region. Then a distance is found for each pixel from this average value. This is calculated as below:-

$$D = \sqrt{(y(i,j) - Y_{av})^2 + (cb(i,j) - Cb_{av})^2 + (cr(i,j) - Cr_{av})^2};$$

Then a limit is put on this distance value. If distance is less than a certain limit, then the point is classified as a red point, else not.



After this point the maximum and minimum indices of white region in the new image is found. These points are extrapolated to make a rectangle and then that rectangle is cropped from the main image. The process of finding the maximum and minimum indices has been explained in following steps.

## Step 2: Extracting the white region

The image acquired from the above step will give a image of red box only.



Fig. 1

To convert the obtained image to the standard RGB format we use the following command.

**Im\_rgb=ycbcr2rgb(Im);** % to convert ycbcr to rgb format

**Im\_rgb=double(Im\_rgb);** % to convert the uint8 image to double so as to ease %  
mathematic operations

Thus the obtained image Im\_rgb will be as shown in fig. 2

Now after this conversion,

The next task will be to extract the white region inside the red box.

For this we find all those points where,

**All three Red, Green and Blue component are greater than 110**

Note: - this value has been determined by observing a test image. This is subject to change according to lighting conditions. Point with these characteristics correspond to white colour

To all such points we initialise a value zero in new image, and to rest of points a value 1. Now in this Binary image we find the minimum and maximum value of x-index & y-index where value is 0. In this way we form a rectangular box around the white region inside the red border. And then we extract this box using the matlab library function *imcrop*.





Fig. 3

Above image is the result of converting the original image to binary image. The code for this is below: -

```
for i=1:s(1)
    for j=1:s(2)
        r=box(i,j,1);
        g=box(i,j,2);
        b=box(i,j,3);
        if r>110 && g>110 && b>110
            new_box(i,j,:)= [0 0 0];
            if i<minx
                minx=i;
            elseif i>maxx
                maxx=i;
            end
            if j<miny
                miny=j;
            elseif j>maxy
                maxy=j;
            end
        end
    end
end
```

```
new_box(i,j,:)= [255 255 255];  
end  
end  
end
```

This also finds the min. and max values of the indices. Now to extract the box according to these determined values of indices we use the function `imcrop` the syntax this is as following:-

```
rect=[miny minx maxy-miny maxx-minx];  
new_img=imcrop(new_box,rect);
```

The resultant image is shown in fig. 4



Fig. 4

### Step 3: Labelling connected components

The image obtained from the above procedure is an binary image which contains the text. Now the next step is to label the connected regions in the image, and also to ignore the connected regions which have a area less than a certain standard in order to distinguish text from the noise or some other sources of error. We will first do this differentiating action and then move forth with the labelling. In order to do this we use a matlab library function *bwconncomp*. The syntax for this is,

```
CC=bwconncomp(BW)
CC = bwconncomp(BW,conn)
```

`CC = bwconncomp(BW)` returns the connected components `CC` found in `BW`. The binary image `BW` can have any dimension. `CC` is a structure with four fields.

**Connectivity** :- Connectivity of the connected components (objects)

**ImageSize** :- Size of `BW`

**NumObjects** :-Number of connected components (objects) in `BW`

**PixelIdxList** :-1-by-NumObjects cell array where the *k*th element in the cell array is a vector containing the linear indices of the pixels in the *k*th object.

The field `PixelIdxList` gives the list of the no. of pixel in each object.

It can be read into a variable by,

```
bw=bwconncomp(new_img,8);

numPixels = cellfun(@numel,bw.PixelIdxList)
```

Now we have to ignore any region of connected pixels which has an area less than 1000(an assumption that every text area is bigger than this).

```
for i=1:bw.NumObjects
```

```
    if numPixels(i)<1000
```

```
        new_img(bw.PixelIdxList{i})=0;
```

```
    end
```

```
end
```

After this, any connected region smaller than 1000 pixels will be ignored.



Fig.5



Fig.6

In Fig.5 the red box show a small dot which can be a source of error. But it can be seen that, it has been removed in Fig. 6 due to the above lines of code.

Now we have a image that possibly contains only text and thus can now be labelled according to the connected regions.

We use *bwlabel* for labelling the image according to connected regions.

Syntax:

```
[L no_of_reg] =bwlabel(img,conn); % conn=4 connectivity or 8 connectivity
```

This returns a label image in which each connected region is assigned a label.

To see this image we use the lib. function *label2rgb*.

```
rgb=label2rgb(x,@jet,'k');
```

#### Step 4: Extracting separate characters.

This is done in the same way we extracted the white region surrounded by the red border. We find the max. & min. values of x & y indices for each connected region in label image and then extract the rectangular region from the image received from previous operations.

The code below performs the above operation: -

```
for i=1:s(1)
    for j=1:s(2)
        for k=1:no_char
            if x(i,j)==k
                if i<minx(k)
                    minx(k)=i;
                elseif i>maxx(k)
                    maxx(k)=i;
                end
                if j<miny(k)
                    miny(k)=j;
                elseif j>maxy(k)
                    maxy(k)=j;
                end
            end
        end
    end
end
end
rect=[miny; minx; maxy-miny; maxx-minx];
rect=transpose(rect);
```

The above *rect* variable contains dimensions for each connected region which is den extracted using *imcrop*.

### Step 5: Reducing the image to smaller size

The character images obtained above are not suitable for recognition purpose since due to their large size, thus we need to reduce it to 12x12 size.

To do this first we divide resize the image to a size of 60x60 using *imresize*. The use block image processing to find average of 5x5 blocks. This image is then reduced to a 12x12 image again using *imresize*. The resultant image is the final character image and is stored in a new image variable, each for one character. The following piece of code performs this task: -

```
for i=1:no_char

    temp=imcrop(new_img,rect(i,:));

    temp=imresize(temp,[60 60]);

    f = inline('uint8(round(mean2(x)*ones(size(x))))');

    temp = blkproc(temp,[5 5],f);

    temp=imresize(temp,[12 12]);

    temp=im2bw(temp,0.15);

    eval(['char' num2str(i) '=temp;']);

    eval(['figure, imshow((char' num2str(i) '))']);

end
```

The resultant images are shown below,



## 7.2 Reading Mode

### 7.2.1 Flow Of Control Diagram Reading Mode

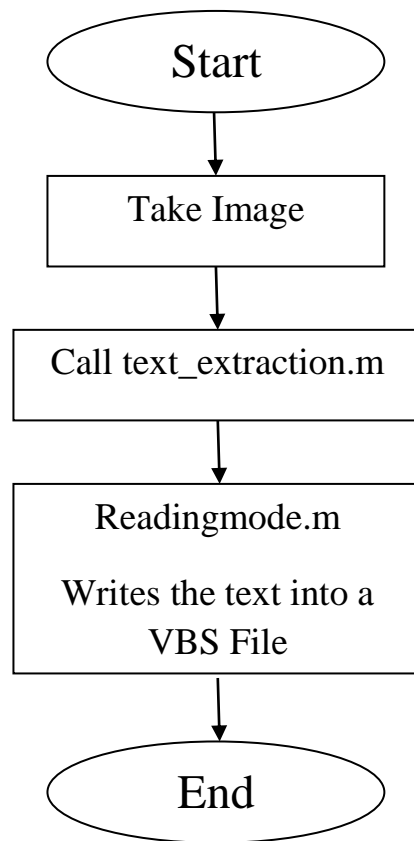


Fig 7.4 Flow of Control Diagram of Reading Mode

## 7.2.2 Working and implementation

### 7.2.2.1 text\_extraction.m

Above MATLAB file contains a function 'text\_extraction1()' which is explained in 7.1.2.2.

### 7.2.2.2 reading.m

```
op=sim(net,img_data);
```

The above code passes the 'image\_data' into the ANN 'net' and the output is stored in the variable 'op'

```
for i=1:s(2)
    if (centroid(2,i)< avg)
        if (max_rec(i)>0.9)

            if (k==1)
                character=char(64+rec_char(i));
                str1=strcat(str1,character);

            else
                distbwcent=centroid(1,i)-centroid(1,temp1);
                sumofwidth=(rect(i,3))/2 + (rect(temp1,3))/2;
                if (distbwcent>sumofwidth+20)
                    character=char(64+rec_char(i));
                    str1=strcat(str1,',',character);
                else
                    character=char(64+rec_char(i));
                    str1=strcat(str1,character);
                end
            end
            temp1=i;
            k=k+1;
        end
    end
```

'op' is then checked for any valid characters and outputs a string 'char' comparing the centroids of the characters are aligned or not.

```
end
fid=fopen('d:\read.vbs','w');
fwrite(fid,'CreateObject("SAPI.SpVoice").Speak"', 'char');
fwrite(fid,str1,'char');
fwrite(fid,' ', 'char');
fwrite(fid,str2,'char');
fwrite(fid,'"', 'char');
fclose(fid);
```



end

The Microsoft Speech API is what is used for the Narrative accessibility in the project feature built into Windows. The above code writes the read character into a VB Script file which is self executable in Windows.

## 7.3 Surveillance Mode

### 7.3.1 Flow of Control Diagram

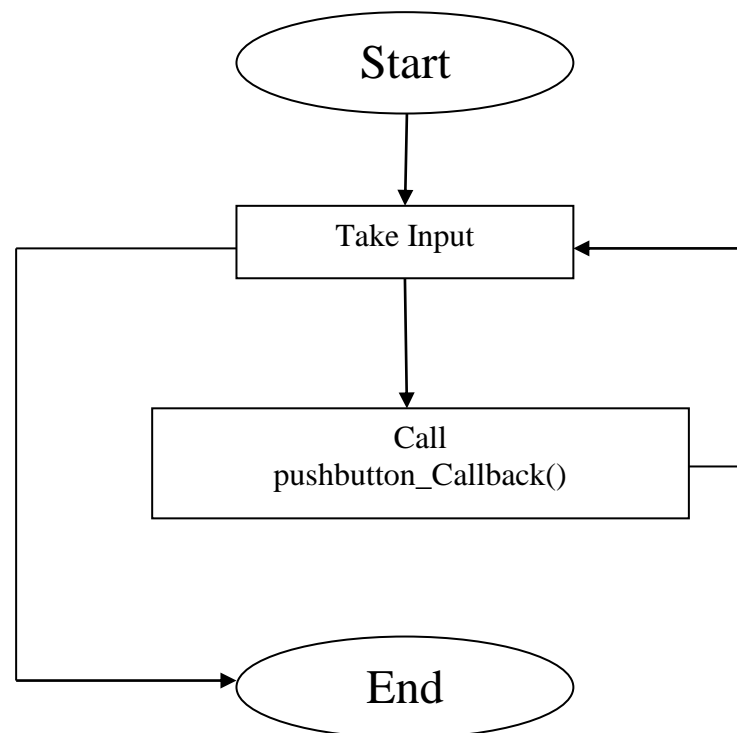
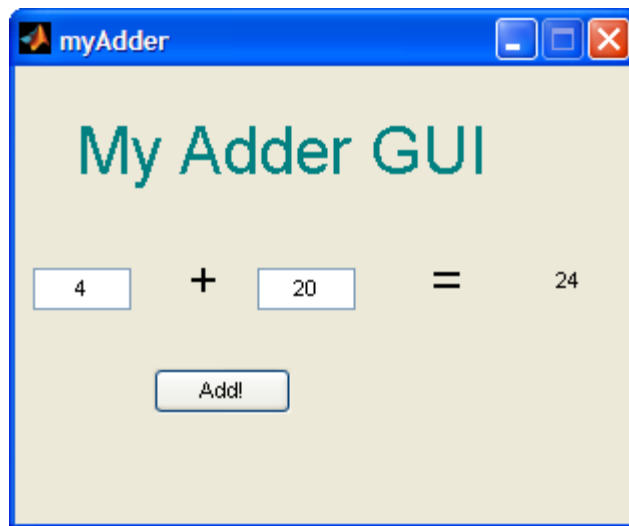


Fig 7.5 Flow of Control Diagram of GUI

### 7.3.2 GUI Tutorial

Why use a GUI in MATLAB? The main reason GUIs are used is because it makes things simple for the end-users of the program. If GUIs were not used, people would have to work from the command line interface, which can be extremely difficult and frustrating. Imagine if you had to input text commands to operate your web browser (yes, your web browser is a GUI too!). It wouldn't be very practical would it? In this tutorial, we will create a simple GUI that will add together two numbers, displaying the answer in a designated text field.



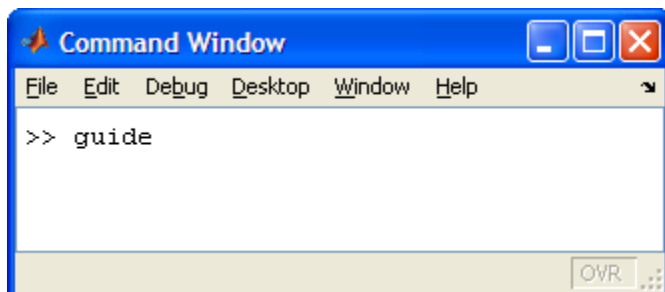
This tutorial is written for those with little or no experience creating a MATLAB GUI (Graphical User Interface). Basic knowledge of MATLAB is not required, but recommended. MATLAB version 2007a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated). Lets get started!

## Contents

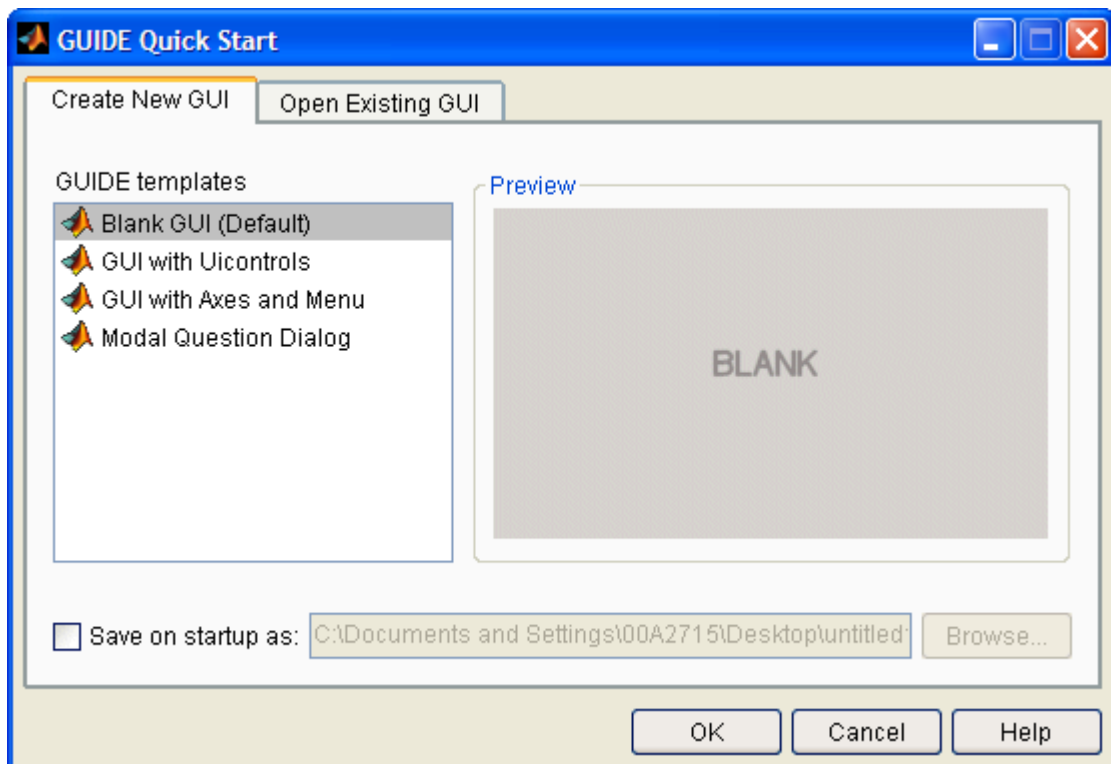
- Initializing GUIDE (GUI Creator)
- Creating the Visual Aspect of the GUI: Part 1
- Creating the Visual Aspect of the GUI: Part 2
- Writing the Code for the GUI Callbacks
- Launching the GUI
- Troubleshooting and Potential Problems
- Related Posts and Other Links

## Initializing GUIDE (GUI Creator)

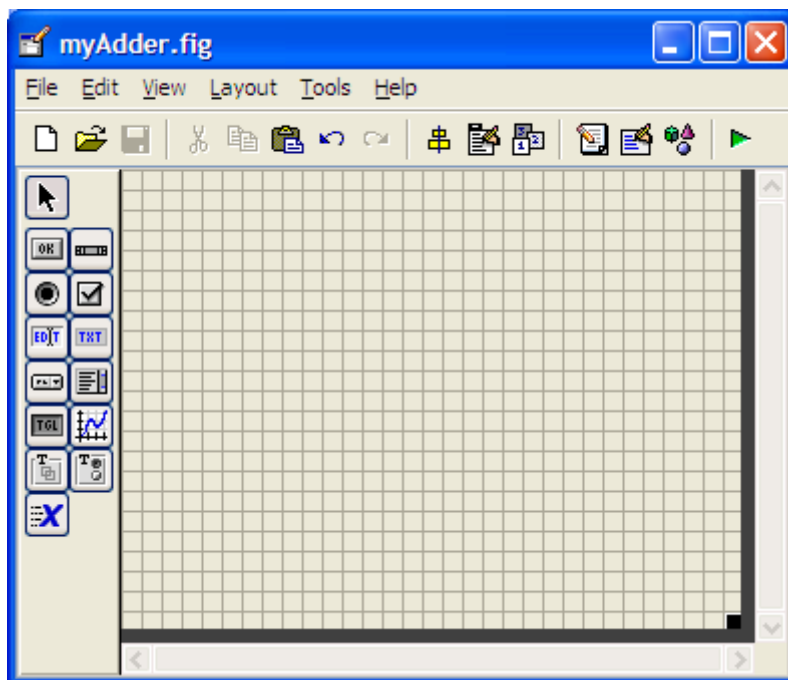
1. First, open up MATLAB. Go to the command window and type in `guide`.



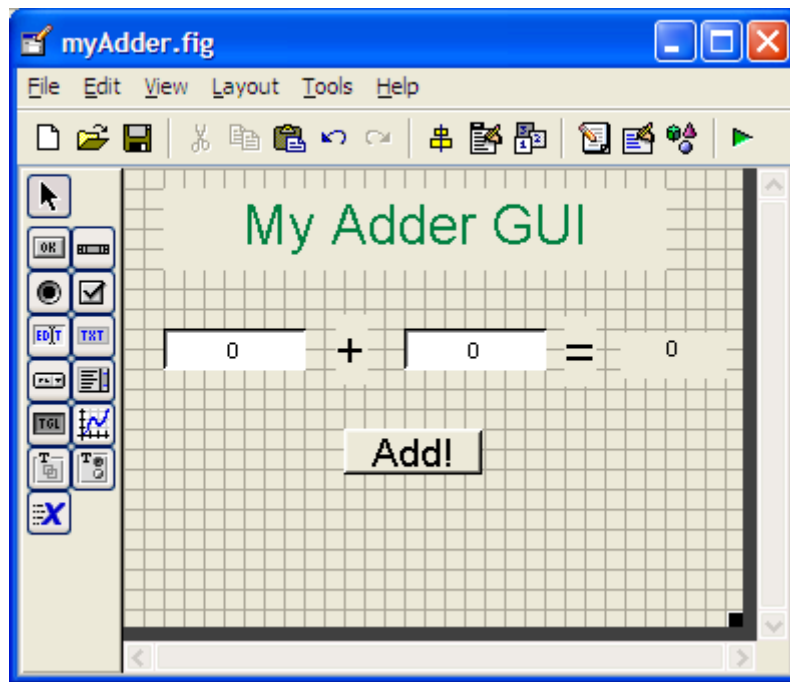
2. You should see the following screen appear. Choose the first option `Blank GUI (Default)`.



3. You should now see the following screen (or something similar depending on what version of MATLAB you are using and what the predesignated settings are):



4. Before adding components blindly, it is good to have a rough idea about how you want the graphical part of the GUI to look like so that it'll be easier to lay it out. Below is a sample of what the finished GUI might look like.



### Creating the Visual Aspect of the GUI: Part 1

1. For the adder GUI, we will need the following components



Two Edit Text components

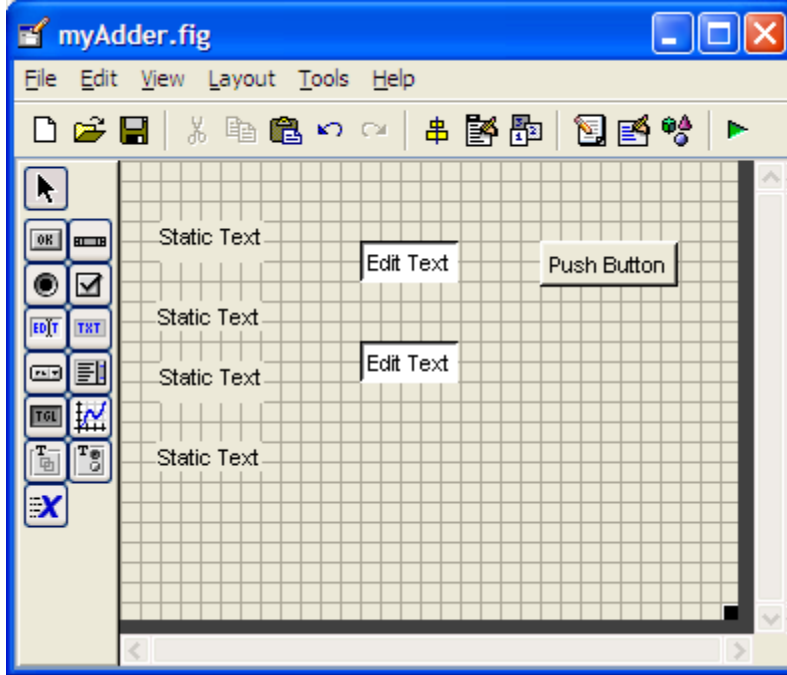


Three Static Text component



One Pushbutton component

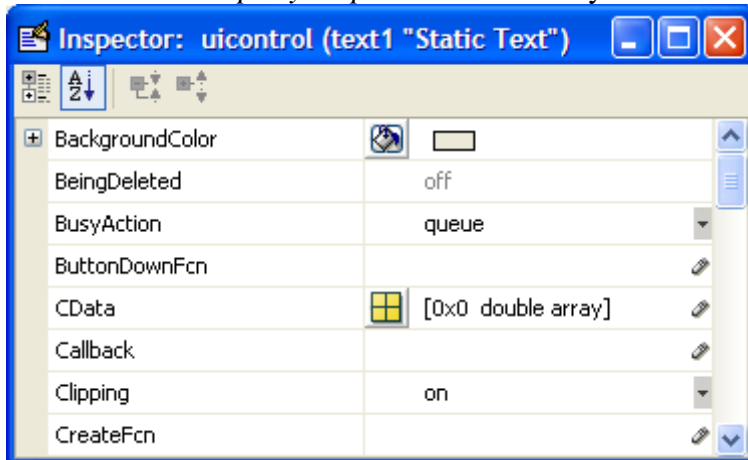
Add in all these components to the GUI by clicking on the icon and placing it onto the grid. At this point, your GUI should look similar to the figure below :



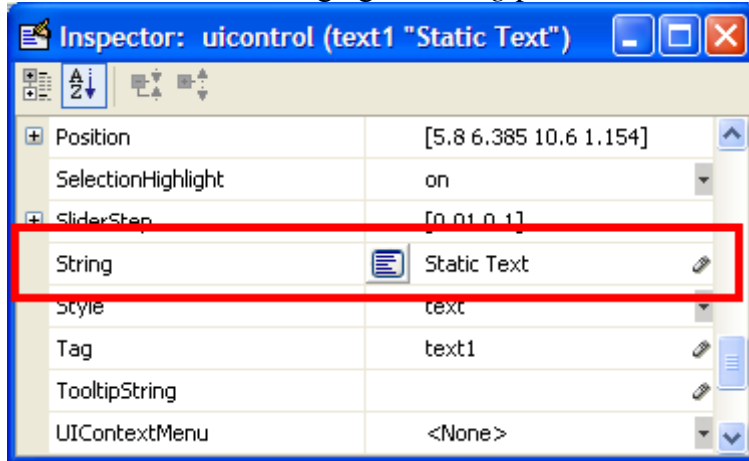
2. Next, its time to edit the properties of these components. Let's start with the static text.

Double click one of the *Static Text* components. You should see the following table appear.

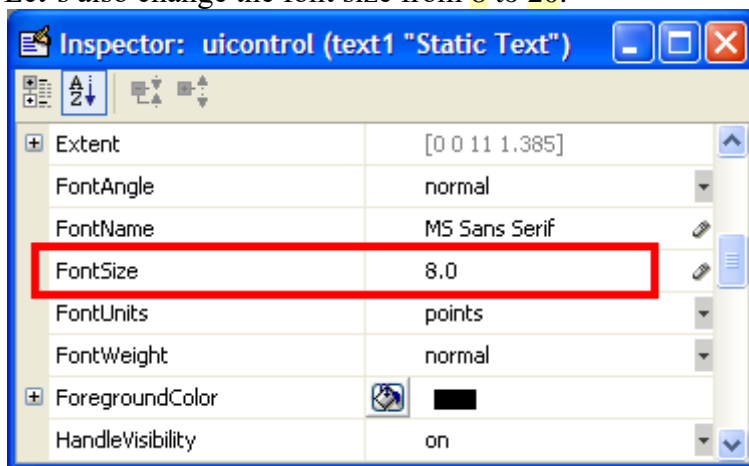
It is called the *Property Inspector* and allows you to modify the properties of a component.



3. We're interested in changing the *String* parameter. Go ahead and edit this text to `+`.



Let's also change the font size from `8` to `20`.

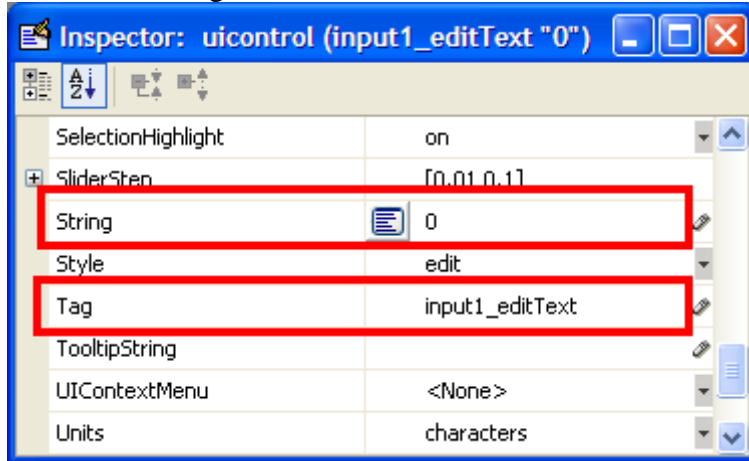


After modifying these properties, the component may not be fully visible on the GUI editor. This can be fixed if you resize the component, i.e. use your mouse cursor and stretch the component to make it larger.

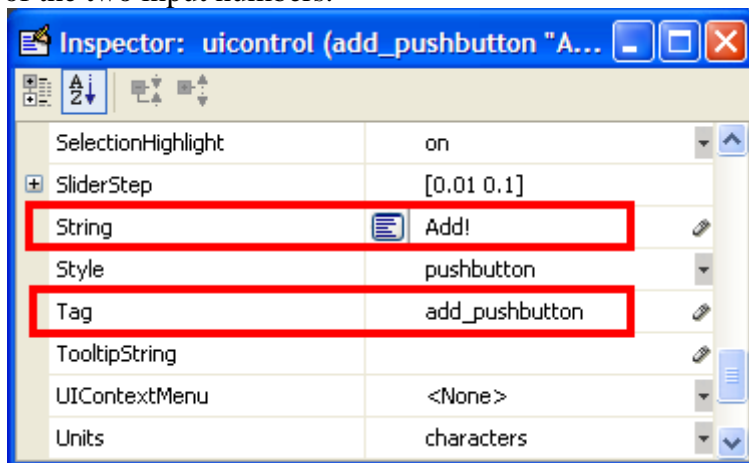
4. Now, do the same for the next *Static Text* component, but instead of changing the *String* parameter to `+`, change it to `=`.
5. For the third *Static Text* component, change the *String* parameter to whatever you want as the title to your GUI. I kept it simple and named it `MyAdderGUI`. You can also experiment around with the different font options as well.
6. For the final *Static Text* component, we want to set the *String* Parameter to `0`. In addition, we want to modify the *Tag* parameter for this component. The *Tag* parameter is basically the variable name of this component. Let's call it `answer_staticText`. This component will be



will be added together.

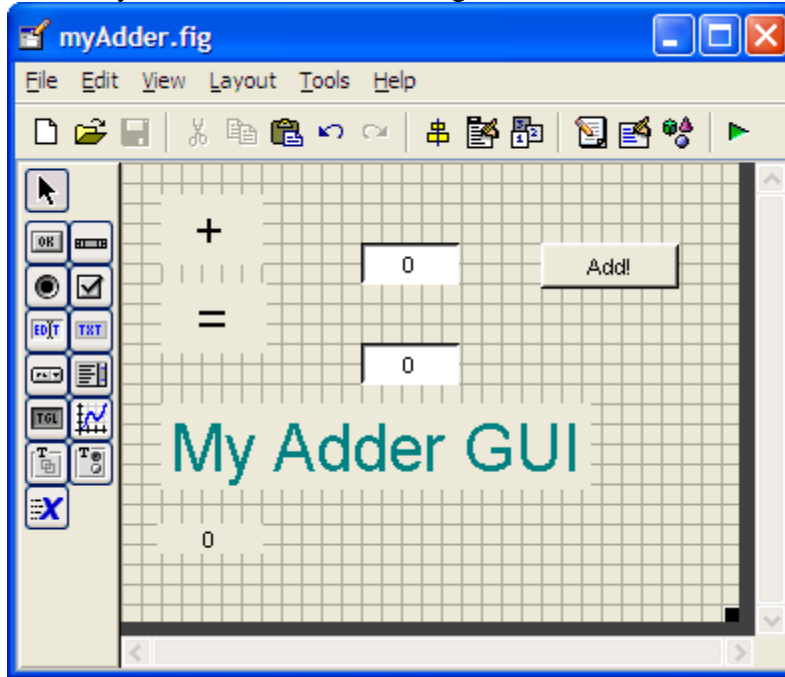


2. For the second *Edit Text* component, set the *String* parameter to **0** BUT set the *Tag* parameter `input2_editText`. This component will store the second of two numbers that will be added together.
3. Finally, we need to modify the *pushbutton* component. Change the *String* parameter to **Add!** and change the *Tag* parameter to `add_pushbutton`. Pushing this button will display the sum of the two input numbers.

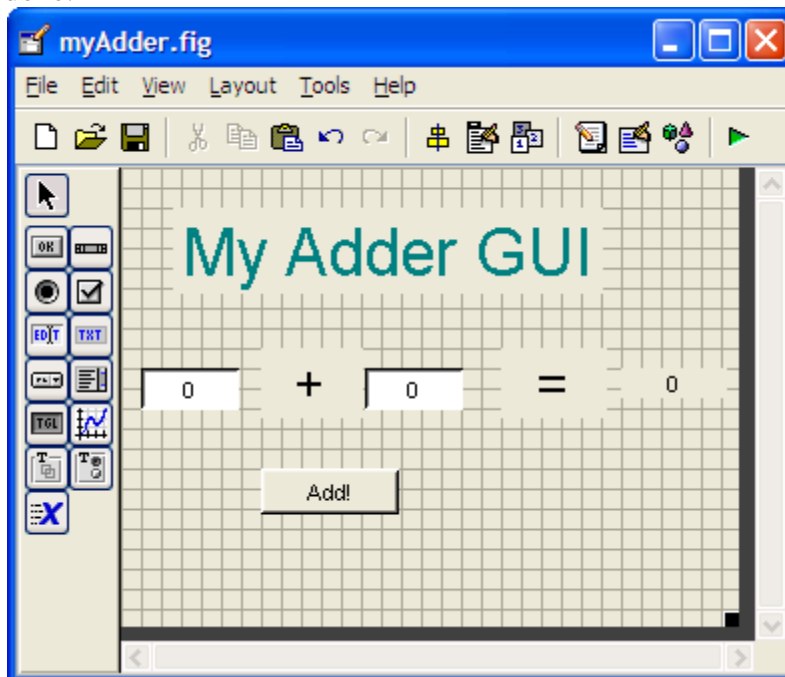




4. So now, you should have something like this:




Rearrange your components accordingly. You should have something like this when you are done:

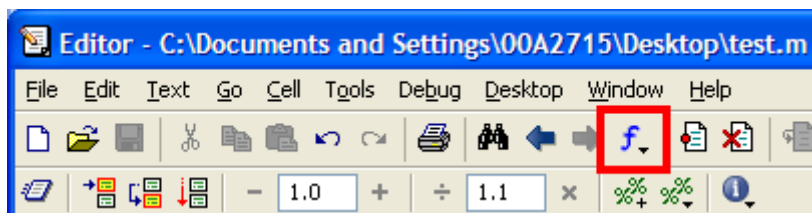


5. Now, save your GUI under any file name you please. I chose to name mine **myAdder**. When you save this file, MATLAB automatically generates two files: *myAdder.fig* and *myAdder.m*. The .fig file contains the graphics of your interface. The .m file contains all the code for the GUI.

## Writing the Code for the GUI Callbacks

MATLAB automatically generates an .m file to go along with the figure that you just put together. The .m file is where we attach the appropriate code to the callback of each component. For the purposes of this tutorial, we are primarily concerned only with the *callback* functions. You don't have to worry about any of the other function types.

1. Open up the .m file that was automatically generated when you saved your GUI. In the MATLAB editor, click on the  icon, which will bring up a list of the functions within the .m file. Select *input1\_editText\_Callback*.



2. The cursor should take you to the following code block:

```
3. function input1_editText_Callback(hObject, eventdata, handles)
4. % hObject    handle to input1_editText (see GCBO)
5. % eventdata  reserved - to be defined in a future version of MATLAB
6. % handles    structure with handles and user data (see GUIDATA)
7.
8. % Hint: get(hObject,'String') returns contents of input1_editText as text
9. %    str2double(get(hObject,'String')) returns contents of
10. %    input1_editText as a double
```

Add the following code to the bottom of that code block:

```
%store the contents of input1_editText as a string. if the string
%is not a number then input will be empty
input = str2num(get(hObject,'String'));

%checks to see if input is empty. if so, default input1_editText to zero
if (isempty(input))
```

```

    set(hObject,'String','0')


end

guidata(hObject, handles);

```

This piece of code simply makes sure that the input is well defined. We don't want the user to put in inputs that aren't numbers! The last line of code tells the gui to update the handles structure after the callback is complete. The handles stores all the relevant data related to the GUI. This topic will be discussed in depth in a different tutorial. For now, you should take it at face value that it's a good idea to end each callback function with `guidata(hObject, handles);` so that the handles are always updated after each callback. This can save you from potential headaches later on.

11. Add the same block of code to *input2\_editText\_Callback*.

12. Now we need to edit the *add\_pushbutton\_Callback*. Click on the  icon and select *add\_pushbutton\_Callback*. The following code block is what you should see in the .m file.

```

13. % --- Executes on button press in add_pushbutton.
14. function add_pushbutton_Callback(hObject, eventdata, handles)
15. % hObject    handle to add_pushbutton (see GCBO)
16. % eventdata  reserved - to be defined in a future version of MATLAB
17. % handles    structure with handles and user data (see GUIDATA)

```

Here is the code that we will add to this callback:

```

a = get(handles.input1_editText,'String');
b = get(handles.input2_editText,'String');

% a and b are variables of Strings type, and need to be converted
% to variables of Number type before they can be added together

total = str2num(a) + str2num(b);

c = num2str(total);

% need to convert the answer back into String type to display it
set(handles.answer_staticText,'String',c);

```

```
guidata(hObject, handles);
```

18. Let's discuss how the code we just added works:

```
19. a = get(handles.input1_editText, 'String');
```

```
20. b = get(handles.input2_editText, 'String');
```

The two lines of code above take the strings within the *Edit Text* components, and stores them into the variables *a* and *b*. Since they are variables of *String* type, and not *Number* type, we cannot simply add them together. Thus, we must convert *a* and *b* to *Number* type before MATLAB can add them together.

21. We can convert variables of *String* type to *Number* type using the MATLAB command `str2num(String argument)`. Similarly, we can do the opposite using `num2str(Number argument)`. The following line of code is used to add the two inputs together.

```
22. total = (str2num(a) + str2num(b));
```

The next line of code converts the *sum* variable to *String* type and stores it into the variable *c*.

```
c = num2str(total);
```

The reason we convert the final answer back into *String* type is because the *Static Text* component does not display variables of *Number* type. If you did not convert it back into a *String* type, the GUI would run into an error when it tries to display the answer.

23. Now we just need to send the sum of the two inputs to the answer box that we created. This is done using the following line of code. This line of code populates the *Static Text* component with the variable *c*.

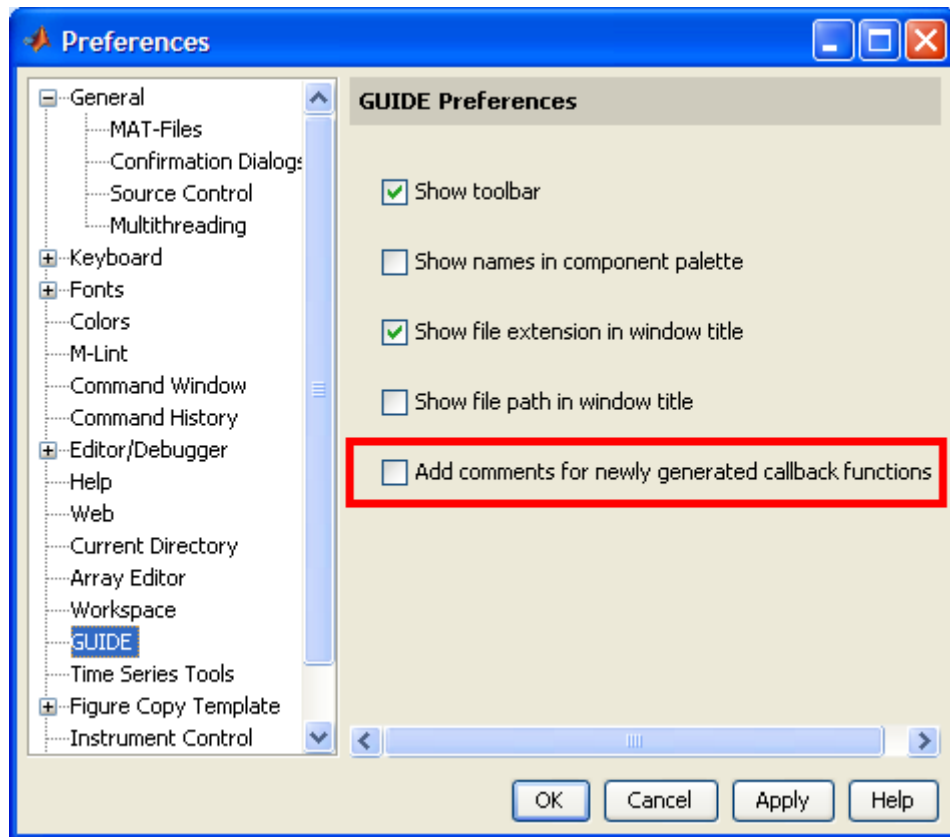
```
24. set(handles.answer_staticText, 'String', c);
```

The last line of code updates the handles structures as was previously mentioned.

```
guidata(hObject, handles);
```


Congratulations, we're finished coding the GUI. Don't forget to save your m-file. It is now time to launch the GUI!

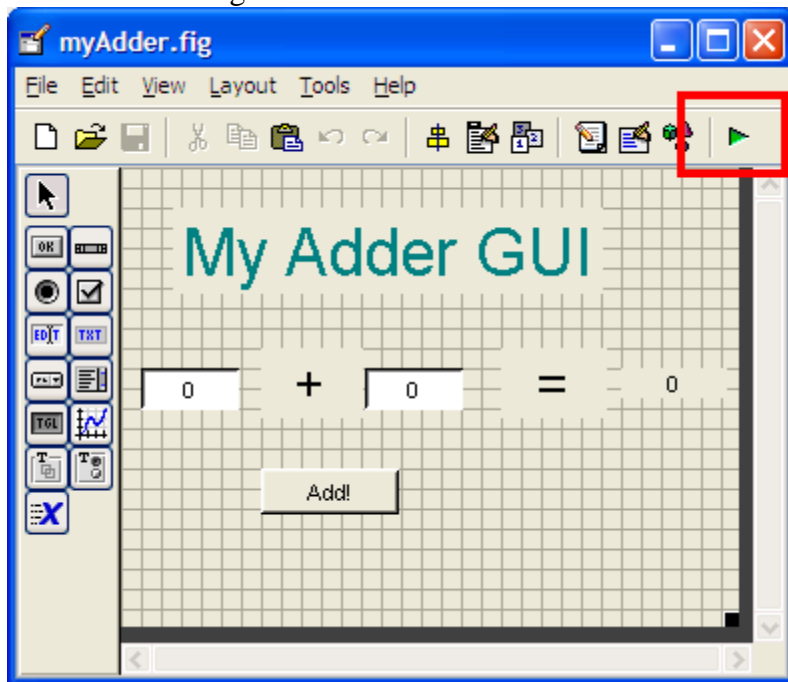
25. If you don't want MATLAB to automatically generate all those comments for each of the callbacks, there is a way to disable this feature. From the GUI editor, go to **File**, then to **Preferences**.



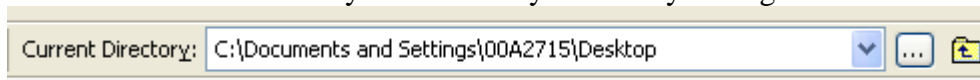
### Launching the GUI

1. There are two ways to launch your GUI.

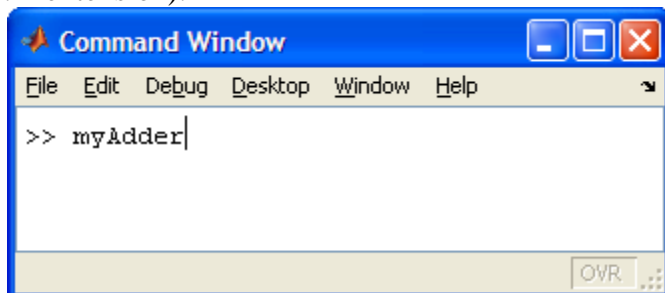
- The first way is through the GUIDE editor. Simply press the  icon on the GUIDE editor as shown in the figure below:



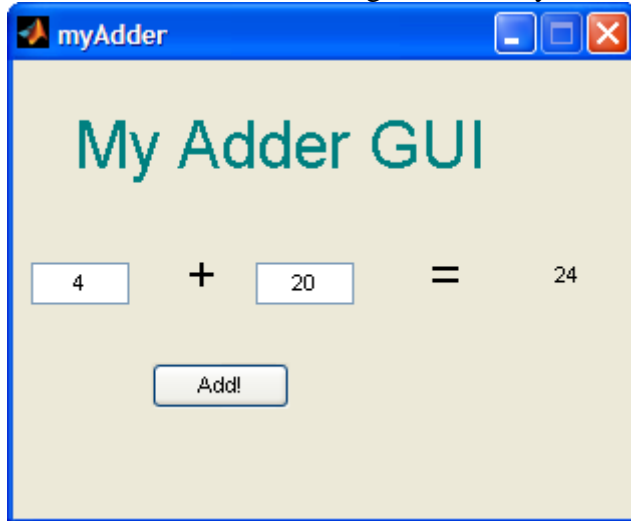
- The second method is to launch the GUI from the MATLAB command prompt. First, set the MATLAB current directory to wherever you saved your .fig and .m file.



Next, type in the name of the GUI at the command prompt (you don't need to type the .fig or .m extension):



2. The GUI should start running immediately:



From this we get a simple idea how to create a GUI in MATLAB, the final GUI of ANS is shown in snapshots.

### **7.3.3 Working and Implementation**

Working principal of surveillance mode is straightforward the robot is controlled remotely in remote desktop configuration in the GUI of ANS the robot is moved using the direction buttons and on the screen several cameras are used to capture the video.

This finds in application in several military because it have a ability to work in LAN and WAN (Internet provided) it can reach inaccessible areas which are dangerous for jawans to guard . It also finds its application in house where a person can control the robot for surveillance even when he is not at home .



# Chapter 8

## Coding

## 8.1 surviellance.m

```
function varargout = surviellance(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @surviellance_OpeningFcn, ...
                  'gui_OutputFcn', @surviellance_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

% --- Executes just before surviellance is made visible.
function surviellance_OpeningFcn(hObject, eventdata, handles, varargin)
set(handles.uipanel1,'Visible','on');
set(handles.uipanel2,'Visible','off');
set(handles.uipanel2,'Visible','off');

vid1=videoinput('winvideo',1,'YUY2_640x480');
vid2=videoinput('winvideo',2,'YUY2_640x480')
handles.vidobj1=vid1;
handles.vidobj2=vid2;
subplot(1,2,1)

vidres=get(handles.vidobj2,'VideoResolution');
nbands=get(handles.vidobj2,'NumberOfBands');
hImage=image(zeros(2*vidres(2),vidres(1),nbands));
preview(handles.vidobj2,hImage);

subplot(1,2,2)

vidres=get(handles.vidobj1,'VideoResolution');
nbands=get(handles.vidobj1,'NumberOfBands');
hImage=image(zeros(2*vidres(2),vidres(1),nbands));
preview(handles.vidobj1,hImage);
```

```

parport=digitalio('parallel','LPT1');
addline(parport,0:7,'out');
putvalue(parport,0) ;
% Choose default command line output for surveillance
handles.portobject=parport;
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = surveillance_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
parport=handles.portobject;
bvdata=logical([0 1 1 0 0 0 0 0]);
putvalue(parport,bvdata);
pause(0.2);
bvdata=logical([0 0 0 0 0 0 0 0]);
putvalue(parport,bvdata);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
parport=handles.portobject;
bvdata=logical([0 1 0 1 0 0 0 0]);
putvalue(parport,bvdata);
pause(0.2);
bvdata=logical([0 0 0 0 0 0 0 0]);
putvalue(parport,bvdata);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
parport=handles.portobject;
bvdata=logical([1 0 0 1 0 0 0 0]);
putvalue(parport,bvdata);
pause(0.2);
bvdata=logical([0 0 0 0 0 0 0 0]);
putvalue(parport,bvdata);

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
parport=handles.portobject;
bvdata=logical([1 0 1 0 0 0 0 0]);
putvalue(parport,bvdata);
pause(0.2);
bvdata=logical([0 0 0 0 0 0 0 0]);
putvalue(parport,bvdata);

```

```

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
set(handles.uipanel1,'Visible','on');
set(handles.uipanel2,'Visible','off');
set(handles.uipanel3,'Visible','off');
guidata(hObject,handles);
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
set(handles.uipanel1,'Visible','off');
set(handles.uipanel2,'Visible','on');
set(handles.uipanel3,'Visible','off');
guidata(hObject,handles);
function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
load net11.mat
destination=get(handles.edit1,'String');
Autonomous(net11,destination,handles);

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
closepreview
vid=handles.vidobj2;
vidres=get(vid,'VideoResolution');
nbands=get(vid,'NumberOfBands');
hImage=image(zeros(vidres(2),vidres(1),nbands));
preview(vid,hImage);

% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
closepreview
vid=handles.vidobj1;
vidres=get(vid,'VideoResolution');
nbands=get(vid,'NumberOfBands');
hImage=image(zeros(vidres(2),vidres(1),nbands));
preview(vid,hImage);

```

```
% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.uipanel1,'Visible','off');
set(handles.uipanel2,'Visible','off');
set(handles.uipanel3,'Visible','on');
```

```
% --- Executes on button press in abort.
function abort_Callback(hObject, eventdata, handles)
par=handles.portobject;
putvalue(par,0) ;
pause(30) ;
```

```
% --- Executes on button press in reading_mode.
function reading_mode_Callback(hObject, eventdata, handles)
vid=handles.vidobj2;
img=getsnapshot(vidobj2);
img_ycb=ycbcr2rgb(img);
[img_data centroid rect]=text_extraction1(img, img_ycb);
reading_mode(img_data,centroid,rect);
```

## 8.2 autonomous.m

```
function [] = Autonomous(net,destination,handles)
vid1=handles.vidobj1;
vid2=handles.vidobj2;
parport=handles.portobject;
% This function is called whenever the bot is functioning in autonomous mode
% This function calls all other function that perform the text extraction
% and recognition and then based on it takes decision
[img_data centroid rect]=navigation(vid1,vid2,parport);
img_data=double(img_data);
op=sim(net,img_data);
[max_rec rec_char]=max(op);
s=size(centroid);
total=sum(centroid(2,:));
avg=total/s(2);
k=1;j=1;
str1=[];str2=[];

for i=1:s(2)
    if (centroid(2,i)< avg)
        if (max_rec(i)>0.9)

            if (k==1)
                character=char(64+rec_char(i));
                str1=strcat(str1,character);

            else
                distbwcent=centroid(1,i)-centroid(1,temp1);
                sumofwidth=(rect(i,3))/2 + (rect(temp1,3))/2;
                if (distbwcent>sumofwidth+20)
                    character=char(64+rec_char(i));
                    str1=strcat(str1,',',character);
                else
                    character=char(64+rec_char(i));
                    str1=strcat(str1,character);
                end
            end

            temp1=i;
            k=k+1;
        end
    else
        if (max_rec(i)>0.5)

            if (j==1)
                character=char(64+rec_char(i));
                str2=strcat(str2,character);

            else
                distbwcent=centroid(1,i)-centroid(1,temp2);
```

```

        sumofwidth=(rect(i,3))/2 + (rect(temp2,3))/2;
        if (distbwcent>sumofwidth+20)
            character=char(64+rec_char(i));
            str2=strcat(str2,',',character);
        else
            character=char(64+rec_char(i));
            str2=strcat(str2,character);
        end

    end
    temp2=i;
    j=j+1;
end
end
end
str1
str2

if (findstr(destination,str1))
    if(~isempty(findstr('left',str1)))
        bvdata=logical([1 0 1 0 0 0 0]);
        putvalue(parport,bvdata);
        pause(0.5);
        bvdata=logical([0 0 0 0 0 0 0]);
        putvalue(parport,bvdata);
    elseif (findstr('right',str1))
        bvdata=logical([0 1 0 1 0 0 0]);
        putvalue(parport,bvdata);
        pause(0.5);
        bvdata=logical([0 0 0 0 0 0 0]);
        putvalue(parport,bvdata);
    end

elseif (findstr(destination,str2))
    if(findstr('left',str1))
        bvdata=logical([1 0 1 0 0 0 0]);
        putvalue(parport,bvdata);
        pause(0.5);
        bvdata=logical([0 0 0 0 0 0 0]);
        putvalue(parport,bvdata);
    elseif (findstr('right',str2))
        bvdata=logical([0 1 0 1 0 0 0]);
        putvalue(parport,bvdata);
        pause(0.5);
        bvdata=logical([0 0 0 0 0 0 0]);
        putvalue(parport,bvdata);
    end
end
else
end
end

```

### 8.3 navigation .m

```
function [img_data centroid rect]=navigation(vid1,vid2,parport)
a=1;
start(vid1);

while(a)
    img=getsnapshot(vid1);
    img_rgb=ycbcr2rgb(img);
    r=img_rgb(:,:,1);
    g=img_rgb(:,:,2);
    b=img_rgb(:,:,3);
    r2=((r-g)+(r-b));
    r2=((r2==255));
    bw=bwconncomp(r2,8);
    numPixels = cellfun(@numel,bw.PixelIdxList);
    if(max(numPixels)>50000)
        display('Red Blot found')
        pause(2);
        img=getsnapshot(vid2);
        img_rgb=ycbcr2rgb(img);
        figure, imshow(img_rgb)
        [img_data centroid rect]=text_extraction1(img_rgb,img);
        stop(vid1);

        return
    else
        road_recognition(img, parport,0.2);
    end
end
stop(vid2);stop(vid1)
end
```



## 8.4 road\_recognition.m

```
function [] = line_tracer(img,parport,t)
temp_i=0;
temp_j=0;
count=0;
y=img(:,1);
new_img=((y>200));
for i=1:480
    for j=1:640
        if(new_img(i,j)==1)
            temp_i=temp_i+ i;
            temp_j=temp_j+ j;
            count=count+1;
        end
    end
end
cent_i=temp_i/count;
cent_j=temp_j/count;

if(cent_j<288)
    %move left
    bvdata=logical([1 0 1 0 0 0 0 0]);
    putvalue(parport,bvdata);
    pause(t);
    bvdata=logical([0 0 0 0 0 0 0 0]);
    putvalue(parport,bvdata);
elseif((cent_j>=288) & (cent_j<=352))
    %move forward
    bvdata=logical([0 1 1 0 0 0 0 0]);
    putvalue(parport,bvdata);
    pause(t);
    bvdata=logical([0 0 0 0 0 0 0 0]);
    putvalue(parport,bvdata);
elseif(cent_j>352)
    %move right
    bvdata=logical([0 1 0 1 0 0 0 0]);
    putvalue(parport,bvdata);
    pause(t);
    bvdata=logical([0 0 0 0 0 0 0 0]);
    putvalue(parport,bvdata);
end
end
```

## 8.5 text\_extraction.m

```
function[img_data centroid rect]= text_extraction1(img,img_ycb)
img2=red_box(img_ycb,50);
img2=uint8(img2);
img2=ycbcr2rgb(img2);
% following part extracts the white region inside the red box

new_img=white_box(img2);

% following part finds the connected regions in the image

s=size(new_img);
bw=bwconncomp(new_img,8);
numPixels = cellfun(@numel,bw.PixelIdxList)
for i=1:bw.NumObjects
    if numPixels(i)<30
        new_img(bw.PixelIdxList{i})=0;
    end
end
figure, imshow(new_img);
bw=bwconncomp(new_img,8);
[x no_char]=bwlabel(new_img,8);
reg_info=regionprops(bw,'basic');
for i=1:no_char
    centroid(:,i)=reg_info(i,1).Centroid;
end
rgb=label2rgb(x,@jet,'k');
figure, imshow(rgb)
hold on
plot(centroid(1,:),centroid(2,:),'or')
hold off
minx(1:no_char)=s(1);
maxx(1:no_char)=1;
miny(1:no_char)=s(2);
maxy(1:no_char)=1;
for i=1:s(1)
    for j=1:s(2)
        for k=1:no_char
            if x(i,j)==k
                if i<minx(k)
                    minx(k)=i;
                elseif i>maxx(k)
                    maxx(k)=i;
                end

                if j<miny(k)
                    miny(k)=j;
                elseif j>maxy(k)
                    maxy(k)=j;
                end
            end
        end
    end
end
```

```

        end
    end
end
end
end
rect=[miny; minx; maxy-miny; maxx-minx];
rect=transpose(rect)
for i=1:no_char
    temp=imcrop(new_img,rect(i,:));
    temp=imresize(temp,[12 12],'bicubic');
    temp2=temp';
    if i==1
        img_data=temp2(:);
    else
        img_data=[img_data temp2(:)];
    end
    eval(['char' num2str(i) '=temp;']);
    eval(['figure, imshow((char' num2str(i) '), xlabel(' num2str(i) ')]);
end

```

## 8.6 red\_box.m

```
function [new_img]= red_box(a,limit);
img=double(a);
s=size(img);
minx=s(1);
maxx=1;
miny=s(2);
maxy=1;
red_box=img;
red_box=zeros(s(1),s(2));
y=img(:,1);
cb=img(:,2);
cr=img(:,3);
for i=1:s(1)
    for j=1:s(2)
        d=sqrt((y(i,j)-81.7946)^2 + (cb(i,j)-102.4029)^2 +(cr(i,j)-214.8273)^2);
        if(d<limit)
            red_box(i,j)=1;
        end
    end
end
figure, imshow(red_box)
B = medfilt2(red_box, [15 15]);
for i=1:s(1)
    for j=1:s(2)
        if B(i,j)==1;
            if i<minx
                minx=i;
            elseif i>maxx
                maxx=i;
            end

            if j<miny
                miny=j;
            elseif j>maxy
                maxy=j;
            end
        end
    end
end
minx=minx+25;
miny=miny+25;
maxx=maxx-25;
maxy=maxy-25;
figure, imshow(B)
new_img=img(minx:maxx,miny:maxy,:);
a=uint8(new_img);
figure, imshow(uint8(new_img))
```

## 8.7 white\_box.m

```
function [new_img]=white_box(img2)
s=size(img2);
minx=s(1);
maxx=1;
miny=s(2);
maxy=1;
new_box=img2;
r=new_box(:,1);
g=new_box(:,2);
b=new_box(:,3);
new_box=~((r>90) & (g>90) & (b>90));
for i=1:s(1)
    for j=1:s(2)
        if new_box(i,j)==0
            if i<minx
                minx=i;
            elseif i>maxx
                maxx=i;
            end

            if j<miny
                miny=j;
            elseif j>maxy
                maxy=j;
            end
        end
    end
end
figure, imshow(new_box);
minx=minx+25;
miny=miny+25;
maxx=maxx-25;
maxy=maxy-25;
rect=[miny minx maxy-miny maxx-minx];
new_img=imcrop(new_box,rect);
```

# Chapter 9

## Testing

## 9.1 Unit Testing of Text Extraction Algorithms

The image below is given input to the text extraction algorithm. The labelled image output and the image data is given below.



Fig 9.1 Text Extraction Testing

Some of the extracted characters are as shown below:-

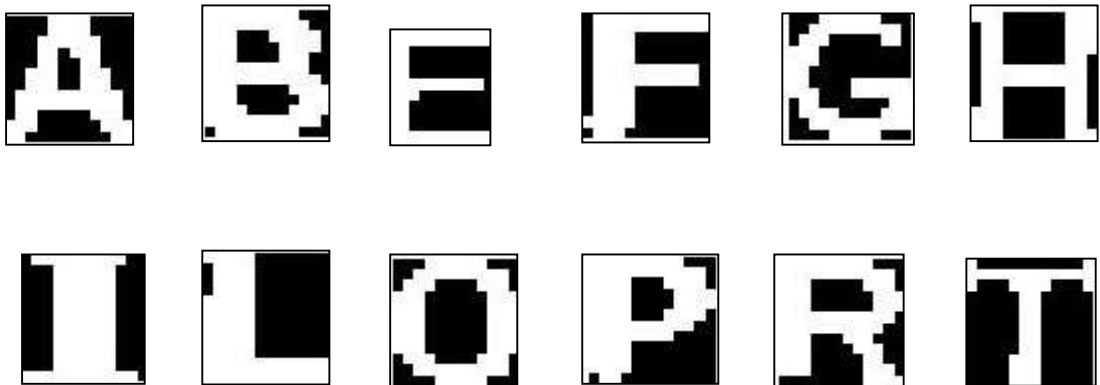


Fig 9.2 Extracted Text 12x12 matrix form

The image data is a matrix of 144 rows and columns equal to no. of characters detected. Each column denotes data corresponding to a character.

A snapshot of the image data is shown below:-

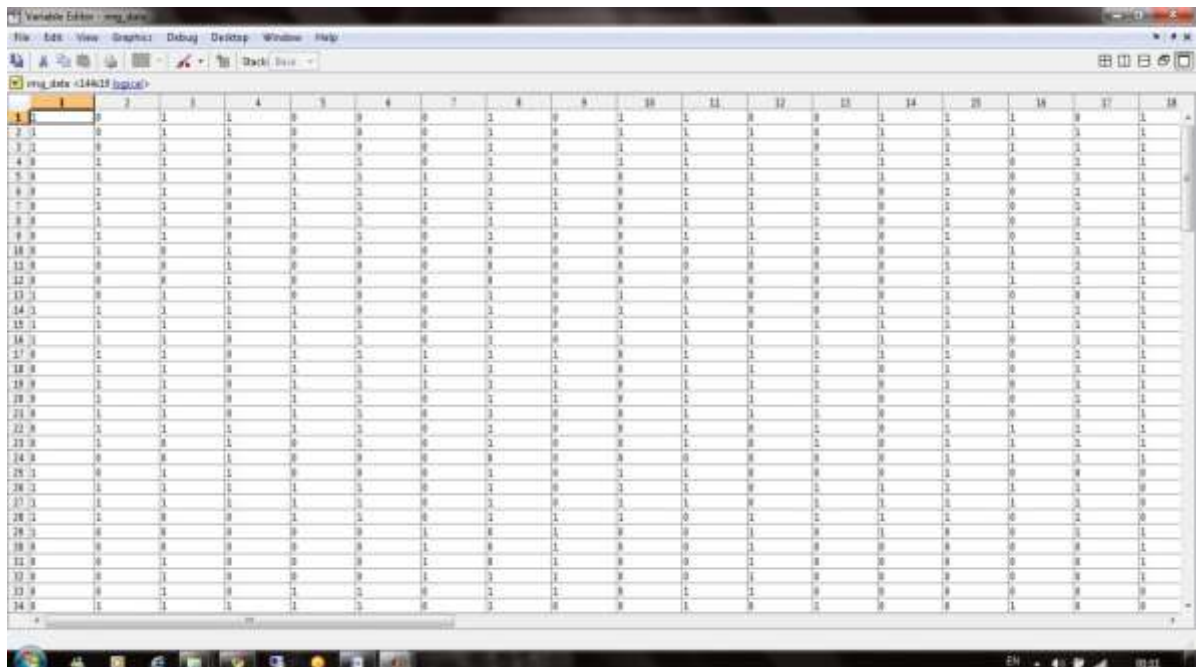


Fig 9.3 Input Matrix of extracted text



## 9.2 Unit Testing of Neural Network

The image data obtained from the above text extraction algorithm is fed to neural network. The output of neural network is form of a matrix of 30 rows and no. of column equal to no. of characters. The value in each row gives a measure of how similar is the character is to a certain character. The red marked cells shows a positive recognition. For example in 2<sup>nd</sup> column 7<sup>th</sup> row has highest value which shows that 2<sup>nd</sup> character was “G”.

	1	2	3	4	5	6	7	8
1	9.9605e-04	1.8506e-08	1.8637e-06	5.9301e-07	3.7398e-10	4.0691e-08	0.9864	5.2322e-05
2	8.8629e-08	7.9787e-06	0.9986	0.0028	7.7892e-05	6.6404e-05	2.2735e-10	8.4646e-06
3	3.1390e-08	0.0020	3.9708e-08	4.7706e-09	7.5267e-04	2.0195e-05	7.3791e-09	2.6340e-07
4	2.1465e-06	4.8700e-06	7.6576e-07	1.2306e-05	4.6243e-04	7.4281e-06	5.5344e-07	4.7918e-05
5	5.4633e-04	1.1116e-06	5.6378e-04	2.2154e-04	3.5535e-05	4.6694e-07	1.4844e-07	1.0297e-09
6	1.2861e-07	6.1362e-06	2.7597e-06	4.9091e-04	1.4318e-07	3.1502e-09	4.2624e-06	2.5232e-06
7	1.4629e-06	0.9980	0.0010	3.3701e-05	1.8391e-06	0.0012	2.2452e-07	1.0038e-08
8	1.6387e-05	2.6322e-09	2.8552e-04	0.9999	1.0501e-06	2.9519e-06	4.2706e-10	2.4008e-09
9	8.3588e-06	1.0971e-05	2.3019e-08	6.9396e-11	2.1699e-06	2.8832e-06	3.8015e-05	2.9954e-08
10	2.1676e-09	3.0289e-06	1.3696e-04	1.0195e-05	1.0931e-05	1.7177e-05	1.3357e-07	1.1119e-05
11	0.3687	3.6833e-06	8.3060e-07	6.8496e-06	2.7202e-08	1.3670e-08	1.9901e-04	7.9203e-10
12	0.1731	5.3306e-07	4.2639e-06	1.0121e-07	7.0115e-07	1.7495e-07	3.4653e-06	5.8316e-08
13	1.2696e-07	6.9189e-07	1.6206e-04	0.0024	4.5385e-05	1.7581e-05	1.7409e-05	3.5350e-05
14	0.7050	4.7273e-06	4.8261e-09	3.0477e-04	2.7476e-06	5.9441e-06	6.2996e-06	2.5383e-08
15	1.6966e-06	5.5430e-05	3.9729e-05	6.5436e-06	0.9957	0.9998	3.5150e-06	2.1977e-06
16	3.3361e-11	4.1504e-09	1.9256e-07	3.4572e-05	9.3395e-06	4.5458e-06	5.6059e-07	0.9990
17	1.1304e-06	2.7562e-04	4.7509e-07	5.6594e-07	2.9837e-05	0.0016	2.2784e-05	1.8343e-05
18	4.2759e-08	1.7757e-08	9.2680e-05	0.0022	2.7984e-05	1.8002e-08	4.1490e-05	9.2560e-05
19	0.0054	2.7268e-05	0.0131	9.0318e-04	1.4713e-07	2.4986e-06	2.3875e-07	9.8219e-09
20	7.6346e-11	1.0504e-06	9.9725e-10	8.1529e-08	2.1907e-05	5.7612e-08	5.3628e-08	9.3379e-06
21	2.0535e-06	9.7949e-04	3.8442e-04	2.9632e-04	0.0011	2.0849e-04	1.1076e-09	3.6497e-08
22	4.1218e-06	1.0884e-07	5.1967e-09	1.2053e-06	6.3883e-07	1.7553e-07	8.5485e-07	2.1549e-05
23	3.5092e-04	2.1700e-06	3.7266e-09	0.0041	7.9707e-05	2.5016e-05	1.5607e-05	4.7091e-08
24	0.0012	1.4021e-09	4.8229e-10	5.2138e-08	6.1318e-09	7.1137e-10	0.0074	6.6086e-07
25	2.6064e-07	2.0233e-07	1.1649e-04	8.2595e-06	1.8596e-08	1.2971e-07	2.0034e-09	4.2036e-08
26	9.8234e-09	6.9714e-08	8.9178e-04	4.6357e-06	1.9572e-05	4.4784e-07	1.8879e-06	2.7020e-05
27	3.0920e-06	9.8771e-07	4.5267e-04	3.8265e-05	4.3348e-05	1.1761e-05	3.3597e-05	3.4495e-05
28	1.1751e-05	4.3958e-06	6.2582e-06	3.0606e-04	1.1488e-05	2.6653e-05	8.5912e-05	1.2033e-04
29	1.1061e-04	2.6832e-05	1.3027e-04	2.4177e-05	2.0710e-05	8.5446e-06	3.2671e-05	2.0650e-06
30	1.5700e-04	5.9346e-05	6.9014e-05	1.2435e-04	8.8462e-05	3.7927e-05	3.1756e-04	1.1334e-05

Fig 9.4 Output Matrix of Network

# **Chapter 10**

## **Screenshots of**

### **A.N.S.**

# 10.1 Autonomous Mode



10.2 Surveillance Mode



## 10.3 Reading Mode



## **Conclusion**

The project is an integration of different areas of engineering such as robotics, computer vision, digital image processing, electronics, neural networks and computer programming. Vision is the most advanced of our senses. So, it is not surprising that images and speech play the most important role in human perception. There are fields such as computer vision, whose ultimate goal is to use computers to emulate human vision, including learning and being able to make inferences and take actions on visual inputs. This area itself is a branch of artificial intelligence whose objective is to emulate human intelligence.

## **Further Enhancements**

- \* Can be used to make autonomous cars .
- \* Can be further be used as a reading device to the visually challenged people.
- \* Robot can be used as a testing model for obstacle avoidance algo ,and collision avoidance algorithm.

## References

### 1) Books

**Digital Image Processing**, by Gonzalez, Woods, and Eddins.

**Digital Image Processing Using MATLAB**, by Gonzalez, Woods, and Eddins.

**Neural Networks - A Comprehensive Foundation**, by Simon Haykin.

**Getting Started with MATLAB 7: A Quick Introduction for Scientists and Engineers**, by Rudra Pratap .

### 2) Web Sites

<http://www.mathworks.com/support/>, for online support.