

AUTONOMOUS NAVIGATION SYSTEM USING NEURAL NETWORK

*Thesis submitted to
Visvesvaraya National Institute of Technology, Nagpur
in fulfillment of requirement for the award of
Degree of*

Bachelor of Technology

By

Aby Abraham	(BT06ECE-013)
Anshul Prakash	(BT06ECE-016)
Rohit Thakur	(BT06ECE-049)
S.Sameer	(BT06ECE-050)
Sachin Kumar Goel	(BT06ECE-051)
Shivdeep Singh	(BT06ECE-055)

Guide

**Prof. A.G. Kothari &
Mr. Parikshit Deshpande**



**Department of Electronics and Computer Science Engineering
Visvesvaraya National Institute of Technology
Nagpur-440010 (India).**

May 2010

Declaration

I, hereby declare that the thesis titled “**Autonomous Navigation System Using Neural Network** ” submitted herein has been carried out by me in the Department of Electronics and Computer Science Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier as a whole or in part for the award of any degree / diploma at this or any other Institution / University.

Aby Abraham (BT06ECE-013)
Anshul Prakash (BT06ECE-16)
Rohit Thakur (BT06ECE-049)
S.Sameer (BT06ECE-050)
Sachin Kumar Goel (BT06ECE-051)
Shivdeep Singh (BT06ECE-055)

Date :

Certificate

The thesis titled “**Autonomous Navigation System Using Neural Network**”
submitted by

Aby Abraham (BT06ECE-013)

Anshul Prakash (BT06ECE-016)

Rohit Thakur (BT06ECE-049)

S.Sameer (BT06ECE-050)

Sachin Kumar Goel (BT06ECE-051)

Shivdeep Singh (BT06ECE-055)

for the award of degree of Bachelor of Technology, has been carried out under my supervision at the Department of Electronics and Computer Science Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is comprehensive, complete and fit for evaluation.

Prof. A.G. Kothari

Assistant Professor,
Department of Electronics
and Computer Science Engineering,
VNIT, Nagpur

Head,
Department of Electronics and Computer Science Engineering
VNIT, Nagpur

Date :

Acknowledgement

Our project, which involved the design and testing of an ‘Autonomous Navigation System’, was an enjoyable as well as a very useful educational experience and we are sure that the skills gained from doing this project would be invaluable for our respective careers.

We would like to thank the Department of Electronics & Computer Science Engineering at the Institute for providing us with this opportunity.

We would also specially like to thank our guide Prof. A.G. Kothari, Dr. A.P. Gokhale, Prof. S.B. Dhok and Mr. Parikshit Deshpande for their continuous support and encouragement. Their guidance has proved invaluable to us.

Lastly, we would like to thank everyone else who has helped in making this project a success.

Mr. Aby Abraham
Mr. Anshul Prakash
Mr. Rohit Thakur
Mr. S.Sameer
Mr. Sachin Kumar Goel
Mr. Shivdeep Singh

1.1 ABSTRACT

Autonomous navigation means that a vehicle is able to plan its path and execute its plan without human intervention. An autonomous robot is one which can not only maintain its own stability but also can plan its movements as it proceeds.

The underlying principle behind the operation is to obtain the image, process it and feed it to a trained neural network, the output of which would then decide the course of action to be taken by the vehicle. Autonomous robots rely on visual perception. Once basic position information is gathered in the form of environmental perception, machine intelligence must be applied to translate some basic motivation (reason for leaving the present position) into a route and plan dynamics of the robot's movement.

Autonomous navigation even though highly useful, has become an extremely challenging engineering problem. This is partly because of the dynamic changing environment in which the system has to operate. Apart from this, the vast array of sensors onboard the system need to coordinate properly to give precise directions to the vehicle. To increase the complexity of an already demanding problem, the whole system needs to operate real-time. Even a slight delay of the order of a millisecond would lead to a vehicle crash or some other equally disastrous effect.

The vast majority of autonomous mobile robots that exist in today's world mostly employ the assistance of a whole pack of sensors which include devices from laser range finders to radar. A camera is only seldom if not used as an input device for autonomous navigation. This thesis presents on the contrary, an autonomous navigation system which relies on only a 2-D image captured from a web camera for its navigation. Traditional computing techniques would definitely not support the problem mentioned above. Taking decisions from information provided by individual pixels of an image using conventional programming techniques would be next to impossible. If attempted, it would require laborious image processing techniques and thousands of lines of code explaining the decisions to be taken with the conditions of each pixel.

Artificial Neural Network forms a very attractive option to solve this problem. The elegance of this approach is that a backbone network of a few layers, each with a specified number of neurons, can be created. [1] The individual pixels of the image can be used as an input to this network. Once the network has been trained with the specified targets, the response time of the network is almost negligible. The network also takes into account the randomness factor and so is more efficient than conventional methods. Apart from this, the size of the code required for the network is also very small. Neural Networks have shown promising results in a variety of areas ranging from pattern recognition to artificial intelligence. The system presented here uses a four layer artificial neural network as its core, the details of which are presented later.

LIST OF FIGURES

1. FIGURE 1(a) :- Overall Block Schematic of the system
2. FIGURE 2(a) 2(b), 2(c), 2(d), 2(e):- Illustration of Backpropagation algorithm
3. FIGURE 2(f) :-Tan-Sigmoid transfer function
4. FIGURE 2(g) :- Linear transfer function
5. FIGURE 2(h) :- Network Architecture
6. FIGURE 2(i) :- MSE vs. No Of Epochs --- Training Curve
7. FIGURE 3(a) :- Block Schematic of Image Pre-processing
8. FIGURE 3(b) :- Original captured image of the road
9. FIGURE 3(c) :- Cropped image
10. FIGURE 3(d) :- Binary Image obtained after color processing
11. FIGURE 3(e) :- Dilated Image
12. FIGURE 3(f) :- Resized image
13. FIGURE 3(g) :- Final image obtained after thinning
14. FIGURE 4(a) :- Block diagram for pointer animation over GUI
15. FIGURE 4(b) :- Steps for image detection over GUI
16. FIGURE 5(a), 5(b), 5(c) ,5(d), 5(e):- GUI snapshots
17. FIGURE 6(a) :- FPGA design flow

LIST OF ACRONYMS

1. **GUI** – Graphical User Interface
2. **MSE**- Mean Square Error
3. **ANS**- Autonomous Navigation System.
4. **HDL**- Hardware Description Language
5. **FPGA**- Field Programmable Gate Array

INDEX

Chapter 1:-	INTRODUCTION
Chapter 2:-	NEURAL NETWORK THEORY AND DESIGN
Chapter 3:-	IMAGE PRE-PROCESSING
Chapter 4:-	PATH PROFILE RECOGNITION
Chapter 5:-	GRAPHICAL USER INTERFACE AND SIMULATION RESULTS
Chapter 6:-	PROBLEMS , MODIFICATIONS AND FUTURE IMPROVEMENTS
Chapter 7:-	ANNEXURE
Chapter 8:-	REFERENCES

1. INTRODUCTION

1.1) HISTORY AND PREVIOUS WORK DONE:

The history of autonomous vehicles starts in 1977 with the **Tsukuba Mechanical Engineering Lab** in **Japan**. On a dedicated, clearly marked course it achieved speeds of up to 30 km/h (20 miles per hour), by tracking white street markers (special hardware was necessary, since commercial computers were much slower than they are today).

In the 1980s a vision-guided **Mercedes-Benz** robot van, designed by **Ernst Dickmanns** and his team at the University of **Munich, Germany**, achieved 100 km/h on streets without traffic. Subsequently, the European Commission began funding the 800 million Euro **EUREKA Prometheus Project** on autonomous vehicles (1987-1995). [2]

Also in the 1980s the **DARPA**-funded Autonomous Land Vehicle (ALV) in the United States achieved the first road-following demonstration that used **laser radar (Environmental Research Institute of Michigan)**, **computer vision (Carnegie Mellon University and SRI)**, and autonomous robotic control (**Carnegie Mellon and Martin Marietta**) to control a driverless vehicle up to 30 km/h. [3]

In 1995, **Dickmanns'** re-engineered autonomous **S-Class** Mercedes-Benz took a 1600 km trip Munich, Bavaria to Copenhagen in Denmark and back, using computer vision and transputers to react in real time. The robot achieved speeds exceeding 175 km/h on the **German Autobahn**, with a mean time between human interventions of 9 km, or 95% autonomous driving. Again it drove in traffic, executing maneuvers to pass other cars. Despite being a research system without emphasis on long distance reliability, it drove up to 158 km without human intervention.

In 1995, the **Carnegie Mellon University** Navlab project achieved 98.2% autonomous driving on a 5000 km (3000-mile) "No hands across America" trip. This car, however, was semi-autonomous by nature: it used neural networks to control the steering wheel, but throttle and brakes were human-controlled. [4]

From 1996-2001, **Alberto Broggi** of the **University of Parma** [5] launched the **ARGO** [6] Project, which worked on enabling a modified **Lancia Thema** to follow the normal (painted) lane marks in an unmodified highway. The culmination of the project was a journey of 2,000 km over six days on the motorways of northern Italy dubbed Mille Miglia in Automatico, with an average speed of 90 km/h. 94% of the time the car was in fully automatic mode, with the longest automatic stretch being 54 km. The vehicle had only two black-and-white low-cost **video cameras** on board, and used **stereoscopic vision** algorithms to understand its environment, as opposed to the "laser, radar - whatever you need" approach taken by other efforts in the field.

In 2008, **General Motors** stated that they will begin testing driverless cars by 2015, and they could be on the road by 2018. [7]

1.2) IDEA AND OBJECTIVE BEHIND THE SYSTEM:

The basic idea is to obtain the images of the road at regular intervals through the camera mounted over the moving vehicle. These images are then subjected to a number of pre-processing steps: cropping, color processing, edge detection, image re-sizing and morphological operations like dilation, thinning etc. to generate a 2-D matrix with reduced and essential features only. By converting this matrix into a 1-D column vector and feeding it to a trained neural network, we can expect the network to produce an output which indicates the correct steering direction.

The objective of this project is to automate an all-terrain vehicle which would then navigate only with the assistance of 2-D images of the path, obtained continuously from a web camera, mounted on the vehicle.

1.3) SCOPE:

The autonomous navigation system can be put to use in diverse real life applications. Some of the possible avenues where it can prove to be very productive to reduce human effort and intervention are detailed below:

1. **Mining applications:** Here the system can be used for exploration in the deep underground mines where humans find it hard to stay and carry out exploration. This particular system can be trained to follow the track laid down and return with vital data.
2. **Assistance to Drivers:** This system can be used to help the new learners. In the initial stages, the new learners can be helped with this trained system and when they are more accustomed to driving, then they can switch to full manual mode.
3. **Railway Navigation:** This system can be very helpful for the functioning of fully autonomous railways. This system can also be equipped with perceiving the railway signals and deciding the whether to stop or proceed further.
4. **Assistance to handicapped persons:** The major application of this system can be to help people with physical disability to ambulate without any external help. With the minimal help of a control panel, which would be controlled by the respective person, such people can experience a more independent life.
5. **Future Combat Systems:** The ANS program will provide navigational, perception, path-planning and vehicle-following algorithms, as well as the requisite on-board sensor package for autonomous mobility. [8]

6.

1.4 BASIC FLOW OF THE SYSTEM

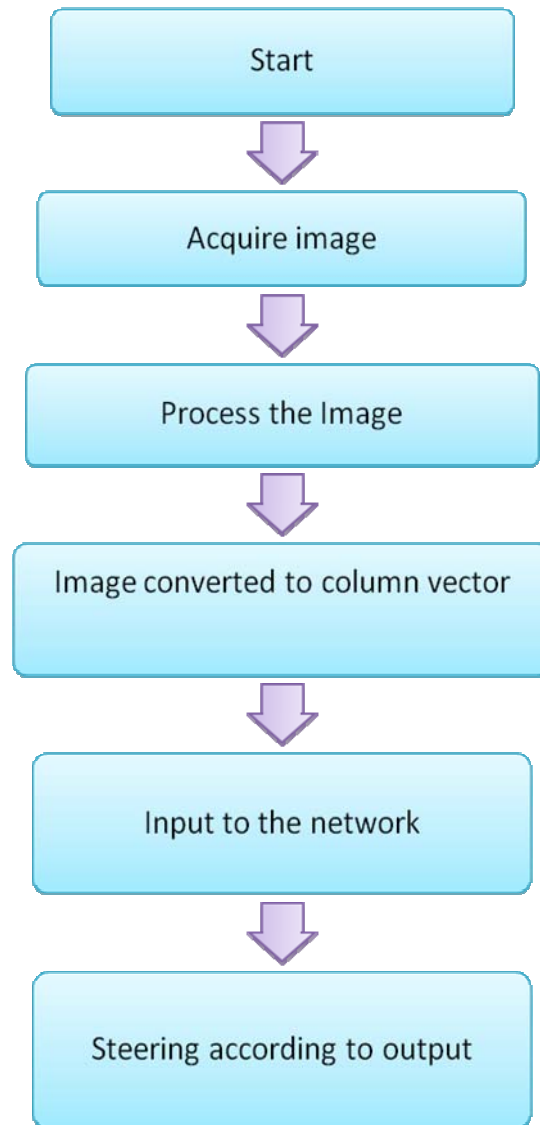


FIGURE 1(a) BLOCK SCHEMATIC

1.5 SPECIFICATIONS OF THE PROJECT:

1. **Neural network** – Multi-layer feed-forward with Backpropagation
2. **Number of layers** =4
3. **Input size** =920 pixels OR 3052 pixels.
4. **Transfer functions employed:** Layer 1: purelin
Layer 2: tansig
Layer 3: tansig
Layer 4: purelin
5. **Output set** = 3 neurons
6. **Training set** = 80 images (920x1) for the model track; 42 images (3052x1) for the actual road used in the case study.
7. **Training function** = trainrp
8. **Learning function**= learngdm
9. **Performance function**= mse

1.6 REQUIREMENTS:

1. **Laptop:** At least 1.5 GHz clock speed, 256 MB free memory, at least 128 MB **RAM**.
2. **Web Camera 2 mega pixel:** This particular web camera must be interfaced with the digital processor which would carry out the required image pre-processing, the required compression and finally feed it to the neural network for detection.
3. **Simulation Software:** Matlab 7.1: to develop the required neural network for detection of the road pattern.

2. NEURAL NETWORK THEORY AND DESIGN

2.1 THEORY:

Artificial Neural networks are physical cellular networks that are able to acquire, store and utilize experimental knowledge related to the network capabilities and performance. They consist of an interconnection of neurons such that the neurons output are connected through weights to all other neurons including themselves. [9]

The basic terminologies associated with a neural network are:-

(1) Layers: The groups of neurons present at each stage while proceeding from the input side to the output form individual “layers”. The layers existing in between the input layer and the output layer are termed as “hidden” layers.

(2) Nodes: These are the summing junctions at which the inputs fed from the preceding layer (multiplied by the corresponding weights) and the biases are combined to generate new outputs to be fed to the forward layers.

(3) Weights: These are multiplying factors by which the inputs get scaled before being summed at the nodes. The weights are adjustable and can be varied at each training stage of the network.

(4) Bias: It is a fixed input which is combined at each node in every layer.

2.2 NEURAL NETWORK ARCHITECTURE:

The different types of neural network architecture possible are :-

(1) Feed Forward network:- It consist of layers of neurons (multi-layer feed forward) fed from the output of the previous layer. Feed forward network can also consist of just a single layer in which they are directly fed the inputs (appropriately weighted).

(2) Feedback network:- A feedback network is obtained from feed-forward network by connecting the neurons output to their input. The essence of closing the feedback loop is to enable control of present output through the previous network output. [10]

2.3 NETWORK LEARNING:

Learning is necessary when the information about input and output is unknown or incomplete so that no design of a network can be performed in advance. “Batch learning” takes place when the network weights are adjusted in a single training step. In this mode of learning, the complete set of input-output training data is needed to determine weights and feedback information produced by the network itself is not involved in developing the neural network. This learning technique is also known as “recording”. In contrast, “incremental learning” involves learning in steps from the environment.

The two basic learning modes are:

- (1) **Supervised learning:** In supervised learning, at each instant of time when the input is applied, the desired response is provided externally. The difference between the actual and desired response serves as an error measure and is used to correct network parameters, since the weights are adjustable, the error can be used to modify weights so that the error decreases. The set of input and output patterns called ‘training set’ is required for this learning mode.
- (2) **Unsupervised learning:** In learning without supervision, the desired response is not known. Thus, explicit information cannot be used to improve network behavior. Since no information is available as to correctness or incorrectness of responses, learning must somehow be accomplished based on observation of responses to inputs that we have marginal or no knowledge about.[11]

Unsupervised learning algorithms use patterns that are typically redundant raw data having no labels regarding their class membership or association. In this mode of learning, the network must discover for itself any possibly existing patterns, regularities, separating properties etc. While discovering these, the network undergoes change of its parameters which is called “self organization”.

For this project we have adopted supervised learning (with batch training) since the learning of different road patterns (not easily discernible even for a human) cannot be accomplished in an unsupervised environment. Also, batch training with a prior set of input and output patterns is necessary before actual automation.

2.4 BACKPROPAGATION ALGORITHM

The following abstract explains the details of the Backpropagation algorithm essential for training the neural network by modifying the weights following each epoch. In the initial step, the output signal of the network (shown below) 'y' is compared with the desired output value 'z' (the target), which is found in the training data set. The difference is called error signal of the output layer neuron.

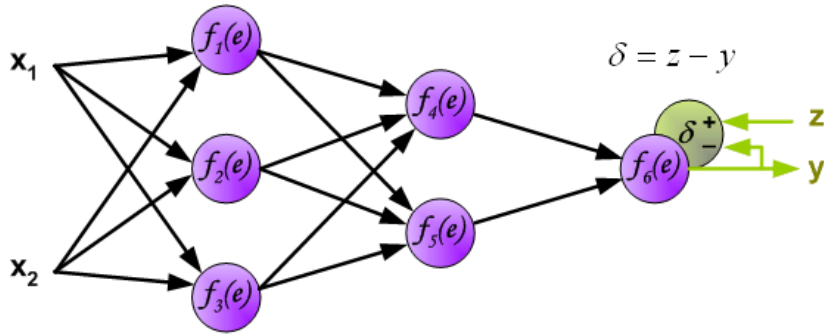


FIGURE 2(a)

It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. The basic idea of Backpropagation is to propagate the error signal (computed in single teaching step) back to all neurons, whose output signals were input for the discussed neuron.

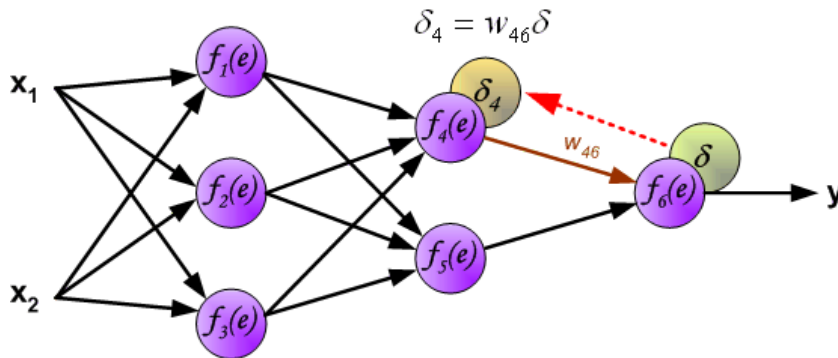


FIGURE 2(b)

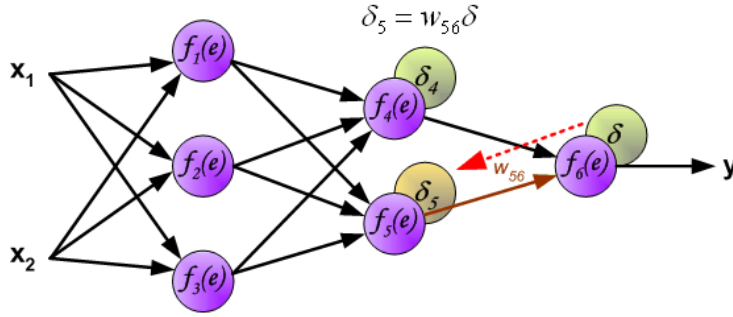


FIGURE 2(c)

The weights' coefficients w_{mn} used to propagate errors back are equal to those used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons, they are added. The illustration is below:

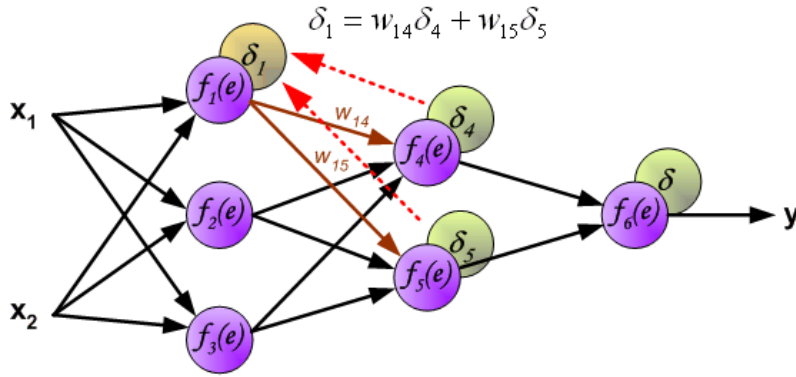


FIGURE 2(d)

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. As shown below, $df(e)/de$ represents derivative of the neuron activation function which is used along with the error information and initial weights in the process of updation of the weights.

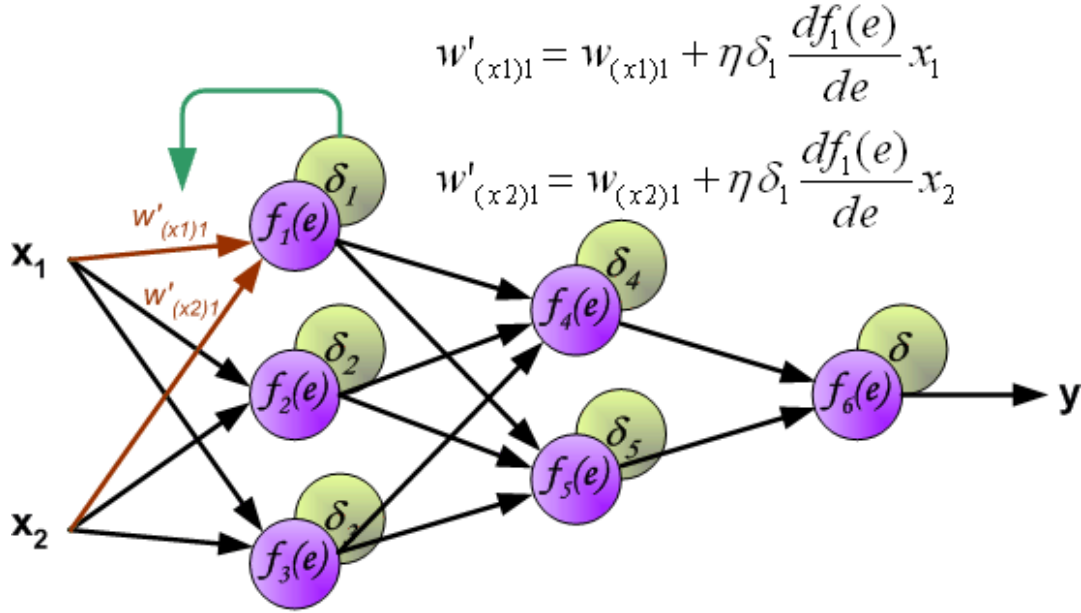


FIGURE 2(e)

This process of modification of the weights is performed for all the layers of the neural network beginning from the initial layer to the final output layer [12]. Backpropagation requires that the neuron activation function be a continuously differentiable function. The basic algorithm adjusts the weights in the steepest descent direction (negative of the gradient)[13]. This is the direction in which the performance function is decreasing most rapidly.

Coefficient η affects the network teaching speed. There are a few techniques to select this parameter. The first method is to start the teaching process with a large value of the parameter. While weights coefficients are being established, the parameter is being decreased gradually. The second, more complicated, method starts teaching with a small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage.

2.5 LEARNING, TRANSFER AND PERFORMANCE FUNCTIONS USED:

The various learning and transfer functions utilized in the neural network design are explained below:

1. TANSIG

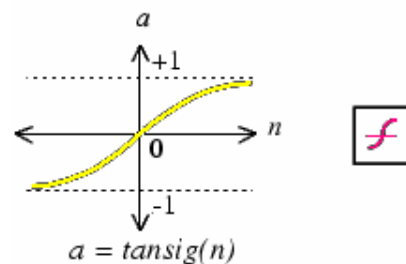
Algorithm

- $a = \text{tansig}(n) = 2/(1+\exp(-2*n))-1$

This is mathematically equivalent to $\tanh(N)$. It differs in that it runs faster than the MATLAB implementation of \tanh , but the results can have very small numerical differences. This function is a good tradeoff for neural networks, where speed is important and the exact shape of the transfer function is not.[14]

Hyperbolic tangent sigmoid transfer function

Graph and Symbol



Tan-Sigmoid Transfer Function

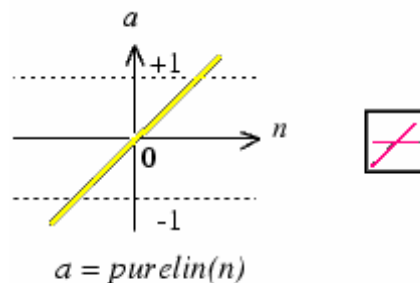
FIGURE 2(f)

2. PURELIN

Algorithm

$$a = \text{purelin}(n) = n$$

Graph and Symbol



Linear Transfer Function

FIGURE 2(g)

3. TRAINRP:-

trainrp can train any network as long as its weight, net input, and transfer functions have derivative functions.

Backpropagation is used to calculate derivatives of performance perf with respect to the weight and bias variables X. Each variable is adjusted according to the following:

- **$dX = \text{deltaX} * \text{sign}(gX);$**

where the elements of deltaX are all initialized to delta0, and gX is the gradient. At each iteration the elements of deltaX are modified. If an element of gX changes sign from one iteration to the next, then the corresponding element of deltaX is decreased by delta_dec. If an element of gX maintains the same sign from one iteration to the next, then the corresponding element of deltaX is increased by delta_inc. [15]

Training stops when any of these conditions occurs:

- The maximum number of epochs (repetitions) is reached.
- The maximum amount of time is exceeded.
- Performance is minimized to the goal.
- The performance gradient falls below min_grad.
- Validation performance has increased more than max_fail time since the last time it decreased (when using validation).

4. LEARNGDM

Purpose

Gradient descent with momentum weight and bias learning function

Algorithm

learnngdm calculates the weight change dW for a given neuron from the neuron's input P and error E, the weight (or bias) W, learning rate LR, and momentum constant MC, according to gradient descent with momentum:

- **$dW = mc * dW_{prev} + (1 - mc) * lr * gW$**

The previous weight change dWprev is stored and read from the learning state LS.

5. MSE

Purpose

Mean squared normalized error performance function

2.6 DETAILS OF THE NETWORK USED:

The architectural description of the neural network employed for the project is given below. The numbers of layers and neurons per layer have been chosen based on the performance of the network. Although increasing the number of neurons and layers provide a modest increase in the system accuracy, it also leads to an undesirable rise in the system latency and training time needed.

Multi-layer feed forward network with Backpropagation has been employed with,

Number of layers: 4 (1 input layer, 2 hidden layers, 1 output layer)

Number of neurons (nodes) per layer:

Layer 1:	35
Layer 2:	25
Layer 3:	25
Layer 4:	03

Transfer functions used for each layer: purelin, tansig, tansig, purelin .

Training function: trainrp .

Learning function: learngdm .

Performance function: mse.

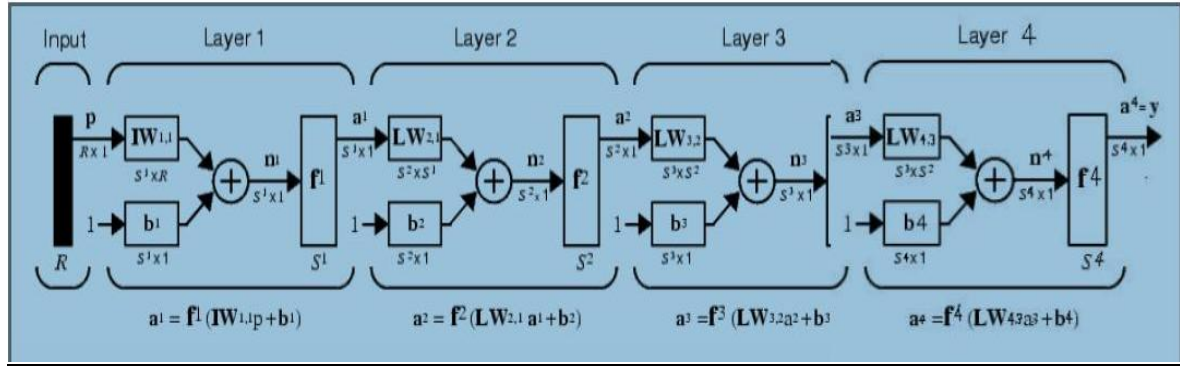


FIGURE 2(h) Network Architecture

2.7 MSE VS NO OF EPOCHS TRAINING CURVE:-

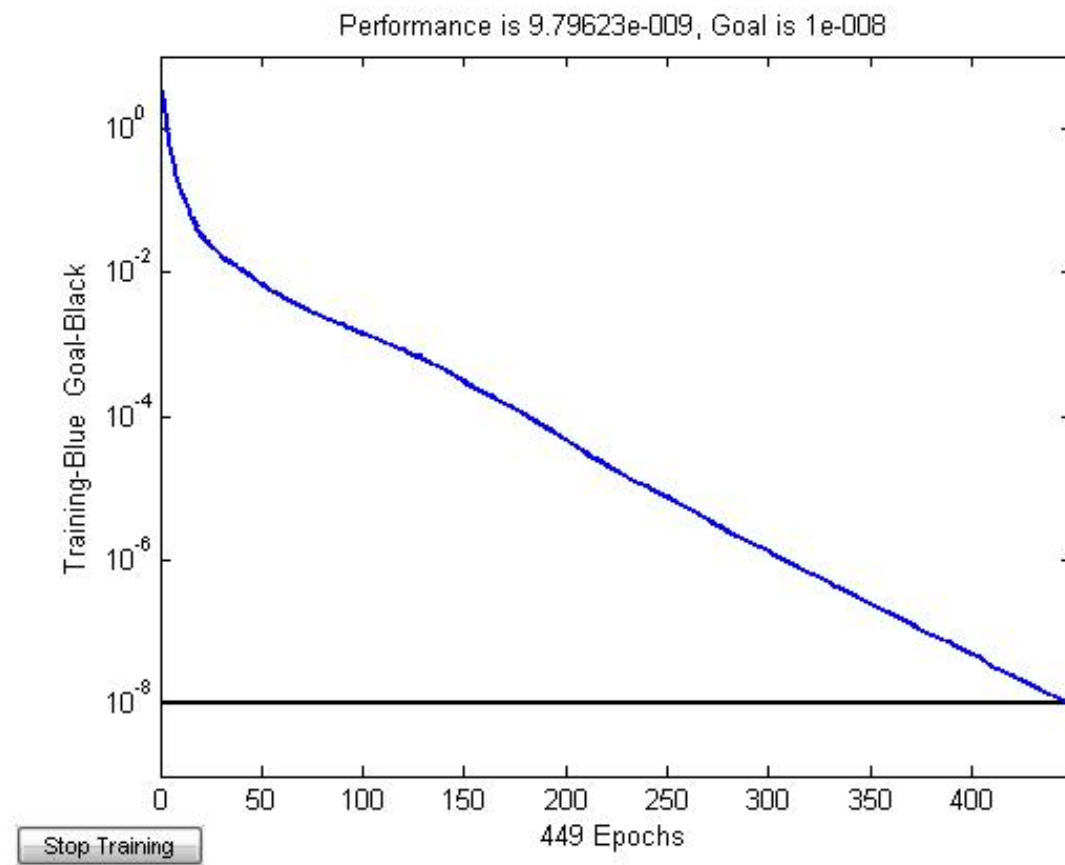


FIGURE 2(i)

2.8 OUTPUTS OF THE NEURAL NETWORK AND THEIR SIGNIFICANCE:-

When trained appropriately the simple connectionist networks can learn to precisely guide a mobile robot in a wide variety of situations. The connectionist model for autonomous road following used in the system is the feed forward multi-layer perceptron. The input layer consists of a single 46x20 or 27x109 unit retina" onto which a pre-processed video image is projected. Each of the 920 or 3052 input units is fully connected to the two unit hidden layer, which is in turn fully connected to the output layer. Each of the 3 output units (neurons) represents a different possible steering direction. The centermost output unit represents the "travel straight ahead" condition, while units to the left and right of center represent sharp left and right turns. The neuron output which is the largest positive value is selected and the corresponding steering action initiated.

3. IMAGE PRE-PROCESSING

3.1 NECESSITY OF FEATURE EXTRACTION AND PRE-PROCESSING:

The first step in the implementation of the project is to capture images of the road with the help of a camera mounted over the moving vehicle. The images taken at periodic intervals are then pre-processed before being fed as input to the neural network.

In **pattern recognition and image processing [16]**, **feature extraction** is a special form of dimensionality reduction. When the input data to an algorithm is too large to be processed and it is suspected to be notoriously redundant (much data, but not much information) then the input data will be transformed into a reduced representation set of features (also named features vector). Transforming the input data into the set of features is called features extraction. If the features extracted are carefully chosen it is expected that the features set will extract the relevant information from the input data in order to perform the desired task using this reduced representation instead of the full size input.

Feature extraction involves simplifying the amount of resources required to describe a large set of data accurately. When performing analysis of complex data one of the major problems stems from the number of variables involved. Analysis with a large number of variables generally requires a large amount of memory and computation power or a classification algorithm which over fits the training sample and generalizes poorly to new samples. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy.

The images obtained include features like trees, buildings etc. which are unimportant from the point of view of actual road detection. Hence, it is necessary to extract only the useful features associated with the road. There are various steps involved in pre-processing of the images like image cropping; color based processing, edge detection, image resizing, filtering and morphological operations like dilation, erosion, thinning etc. Among these we have chosen only specific techniques which were observed to provide better results for our project..

3.2 STEPS INVOLVED IN IMAGE PROCESSING

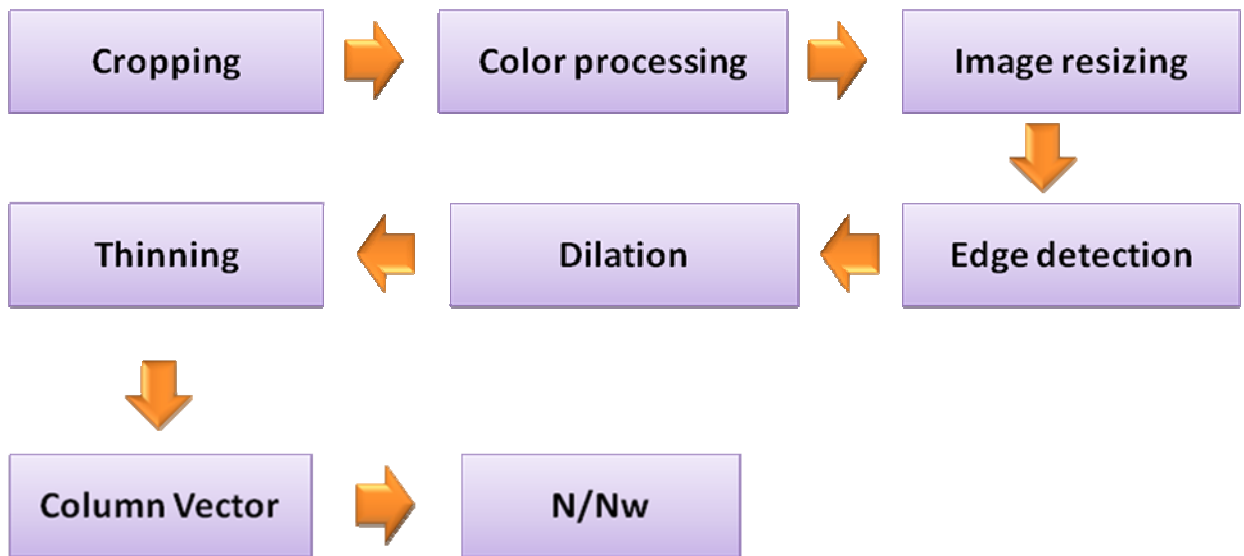


FIGURE 3(a) BLOCK SCHEMATIC

As shown in the block diagram, the image captured through the camera is initially cropped followed by application of color processing techniques. This processed image is then subjected to edge detection and is appropriately resized. The resized image is morphologically processed through repeated dilation and thinning steps before being converted into a column vector to be fed to the neural network. The necessity of each of these steps is explained below.

- (i) **Cropping:** The road or path generally occupies only the middle portion of the total image captured. Cropping of the image from its sides and the top by suitable dimensions not only removes some unwanted external features outside the road path but also enables us to work with the smaller image requiring lesser processing in further steps.
- (ii) **Color processing:** Color processing is employed so as to ensure a good distinction between the actual road and the surrounding features. Using the black color as the indication for the road (i.e. pixels with {R,G,B} values very low) we can easily separate the road portion from the surroundings and then apply edge detection algorithms. This method has been adopted for the model road created. Another modification used in the actual road detection for the real-time video is to only extract the highlighted(white) road boundary(having very high (R,G,B) pixel values). The advantages of color processing over direct edge detection and filtering approaches have been outlined in the next section. [17]

- (iii) **Image resizing:** The process of image resizing is very necessary to reduce the volume of data which is going to be processed by the neural network. It was observed that image resizing performed before edge detection over the binary image of the road path provided better results and also improved the nature of the final dilated and thinned images.
- (iv) **Edge detection:** Edge detection of the road path obtained through color processing is done over the binary image of the path. This enables better detection of the degree of turning involved as compared to the case when the entire binary image available is fed as input to the neural network.[18]
- (v) **Morphological operations :**
 - (a) **Dilation:** The edge detected image is generally observed to produce discontinuities at the road boundaries which may influence the output of the neural network. Dilation ensures that we obtain a continuously linked road edge by filling up the missing gaps between neighboring edge pixels
 - (b) **Thinning:** The process of dilation causes a problem of having extremely thick edges which distort the original shape of the road boundary. So, it is necessary that we ‘thin’ the dilated images so as to obtain a finer boundary which remains completely linked. [19]

This new sequence of operations involving color processing at the start was preferred in place of traditional techniques like edge detection followed by filtering because it helped in removing the unwanted external features directly. This made further processing steps very simple and fast owing to the reduced number of features.

3.3 ILLUSTRATION OF THE IMAGE PRE-PROCESSING STEPS THROUGH AN EXAMPLE:-

The following image sequence show the results obtained at various stages of the image processing algorithm adopted:-



FIGURE 3(b) original image of the road



FIGURE 3(c) cropped image



FIGURE 3(d) binary image obtained after color processing



FIGURE 3(e) dilated image

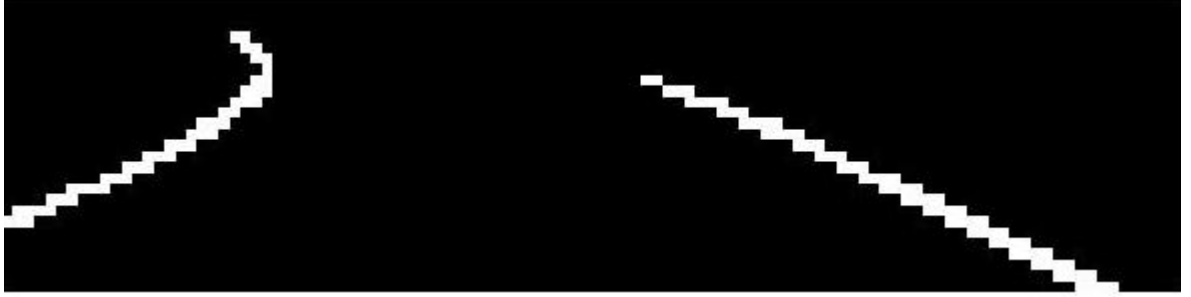


FIGURE 3(f) resized image

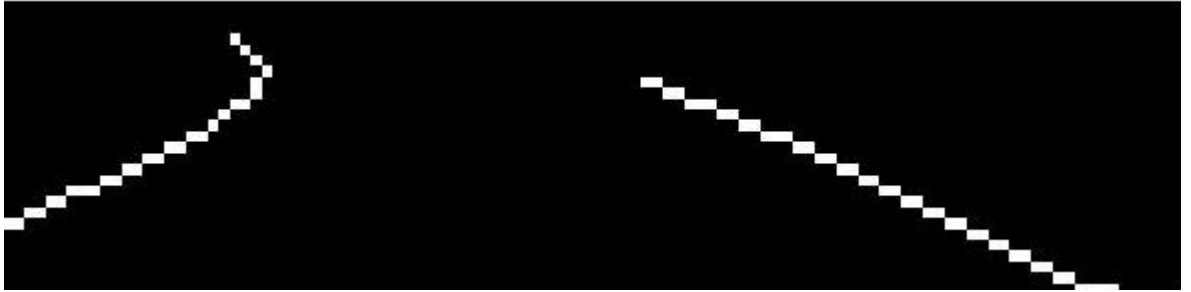


FIGURE 3(g) final image obtained after thinning

The image matrix (2-D) obtained after all these processing steps is then converted into a column vector/matrix (1-D) so that it can be fed as input to the neural network.

4. PATH PROFILE RECOGNITION

This part deals with the navigation from one destination to another and the results being shown over a GUI (graphical user interface).

First the user chooses one particular source and destination from a set of available options. Then he gives the command for navigation following which the images corresponding to the track are loaded and fed to the pre-processor unit one by one in the order of occurrence on the track. After the necessary pre-processing steps the (re-sized) image matrix is converted into a column vector and fed as input to the neural network. The neural network then produces an output indicating as to whether the steering direction is left, right or straight.

The output of neural network is mapped onto the GUI handle, which produces the equivalent motion on the screen. The navigation pointer moves to indicate that the image has been processed and the correct path followed. By default, any turns (right/left) encountered are taken at an angle of 45 degrees. The procedure is followed for the entire set of images corresponding to the path selected and the autonomous navigation from the source to the desired destination is achieved. This process is illustrated in the block diagram below:

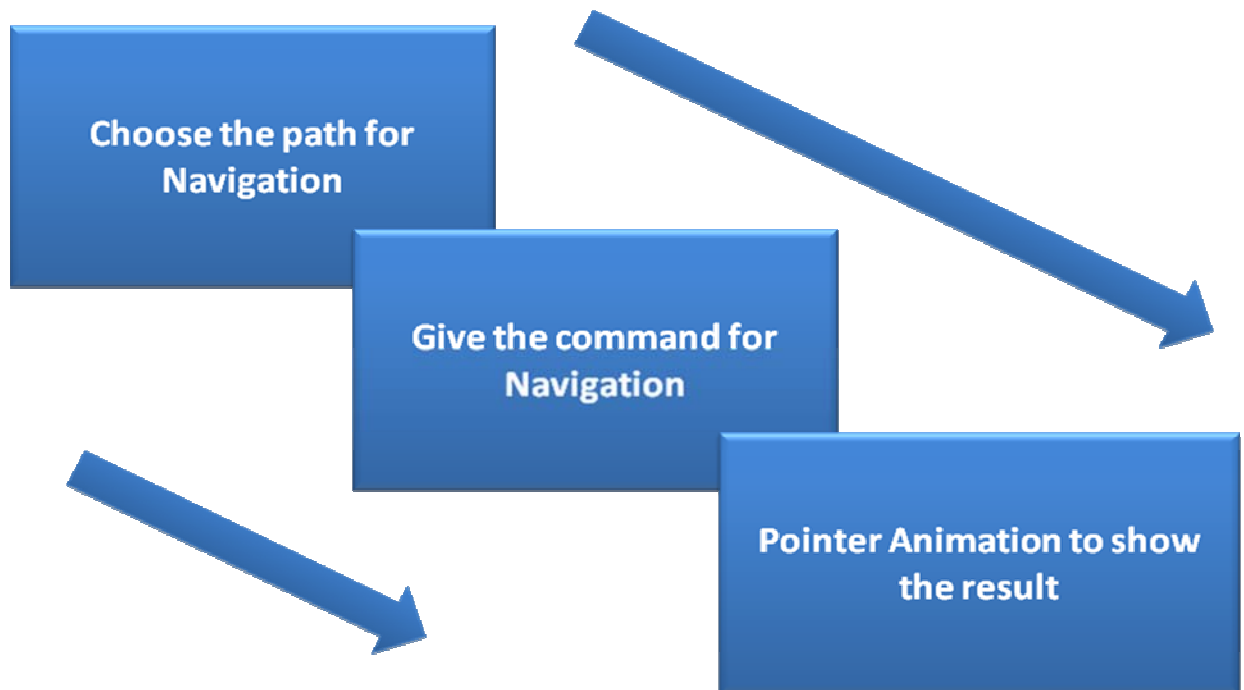


FIGURE 4(a)

The neural network function and calling has been embedded in the form a function call which is invoked with the Graphical User Interface Handle (GUI).

The second part in path profile recognition involves the road detection on an image-to-image basis. Here all the images have been listed in GUI dropdown list box. Whenever an image name is selected, the corresponding pre-processed image vector is loaded and fed as input to the neural network. Then the corresponding steering result is generated by the function with neural network at its core.

The third part of the GUI chooses one of the four paths, each from a source to destination, and displays the images sequentially, displays the result calculated by the GUI and traces the path.

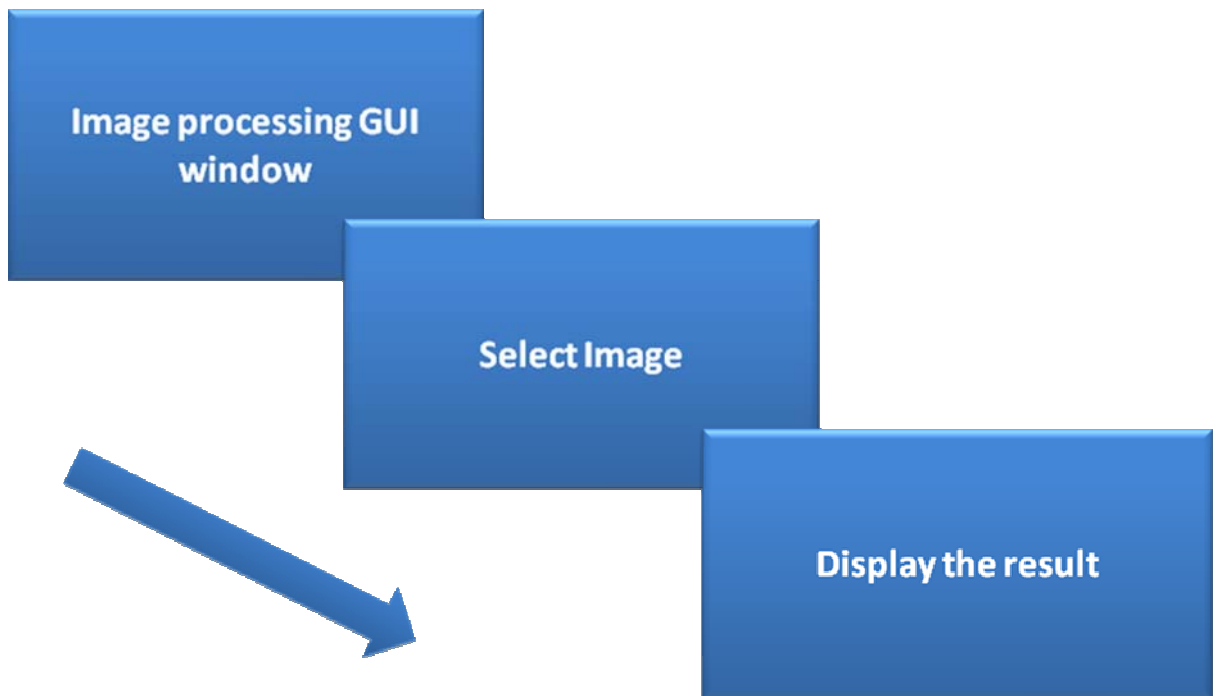


FIGURE 4(b)

5. GRAPHICAL USER INTERFACE AND SIMULATION RESULTS

A graphical user interface (GUI) is a graphical display in one or more windows containing controls, called *components*, that enable a user to perform interactive tasks.[20] The user of the GUI does not have to create a script or type commands at the command line to accomplish the tasks. Unlike coding programs to accomplish tasks, the user of a GUI need not understand the details of how the tasks are performed.

GUI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders—just to name a few. GUIs created using MATLAB tools can also perform any type of computation, read and write data files, communicate with other GUIs, and display data as tables or as plots.

Most GUIs wait for their user to manipulate a control, and then respond to each action in turn. Each control, and the GUI itself, has one or more user-written routines (executable MATLAB code) known as *callbacks*, named for the fact that they “call back” to MATLAB to ask it to do things. The execution of each callback is triggered by a particular user action such as pressing a screen button, clicking a mouse button, selecting a menu item, typing a string or a numeric value, or passing the cursor over a component. The GUI then responds to these *events*. The user as the creator of the GUI, provides callbacks which define what the components do to handle events. This kind of programming is often referred to as *event-driven* programming.

In event-driven programming, callback execution is *asynchronous*, that is, it is triggered by events external to the software. In the case of MATLAB GUIs, most events are user interactions with the GUI, but the GUI can respond to other kinds of events as well, for example, the creation of a file or connecting a device to the computer.

Callbacks can be coded in two distinct ways:

- As MATLAB language functions stored in files
- As strings containing MATLAB expressions or commands (such as 'c = sqrt(a*a + b*b);' or 'print').

Using functions stored in code files as callbacks is preferable to using strings, as functions have access to arguments and are more powerful and flexible. MATLAB scripts (sequences of statements stored in code files that do not define functions) cannot be used as callbacks.

Although a callback can be provided with certain data and made to do anything, the instant of callback execution cannot be controlled. That is, while the GUI is being used, there is no control over the sequence of events that trigger particular callbacks or other callbacks still running at those times. This distinguishes event-driven programming from other types of control flow, for example, processing sequential data files.

WAYS TO BUILD MATLAB GUIS

A MATLAB GUI is a figure window to which user-operated controls are added. The desired size can be selected and the components can be positioned as wanted. Using callbacks the components can be made to perform any desired action when the user clicks or manipulates them with keystrokes.[21]

MATLAB GUIs can be built in two ways:

- **Using GUIDE** (GUI Development Environment), an interactive GUI construction kit.
- Creating code files that generate GUIs as functions or scripts (programmatic GUI construction).

The first approach starts with a figure that can be populated with components from within a graphic layout editor. GUIDE creates an associated code file containing callbacks for the GUI and its components. GUIDE saves both the figure (as a FIG-file) and the code file. Opening either one also opens the other to run the GUI.

In the second, *programmatic*, GUI-building approach, a code file is created that defines all component properties and behaviors. When a user executes the file, it creates a figure, populates it with components, and handles user interactions. The figure is not normally saved between sessions because the code in the file creates a new one each time it runs. As a result, the code files of the two approaches look different.

Programmatic GUI files are generally longer, because they explicitly define every property of the figure and its controls, as well as the callbacks. GUIDE GUIs define most of the properties within the figure itself. They store the definitions in its FIG-file rather than in its code file.

The code file contains callbacks and other functions that initialize the GUI when it opens. MATLAB software also provides functions that simplify the creation of standard dialog boxes, for example to issue warnings or to open and save files. The GUI-building technique you choose depends on the user's experience, preferences and the kind of application needed.

This table outlines some possible GUI building techniques:

TYPE OF GUI	TECHNIQUE
Dialog box	MATLAB software provides a selection of standard dialog boxes that can be created with a single function call.
GUI containing just a few components	It is often simpler to create GUIs that contain only a few components programmatically. Here, each component can be fully defined with a single function call.
Moderately complex GUIs	GUIDE simplifies the creation of moderately complex GUIs

GUI SNAPSHOTS:

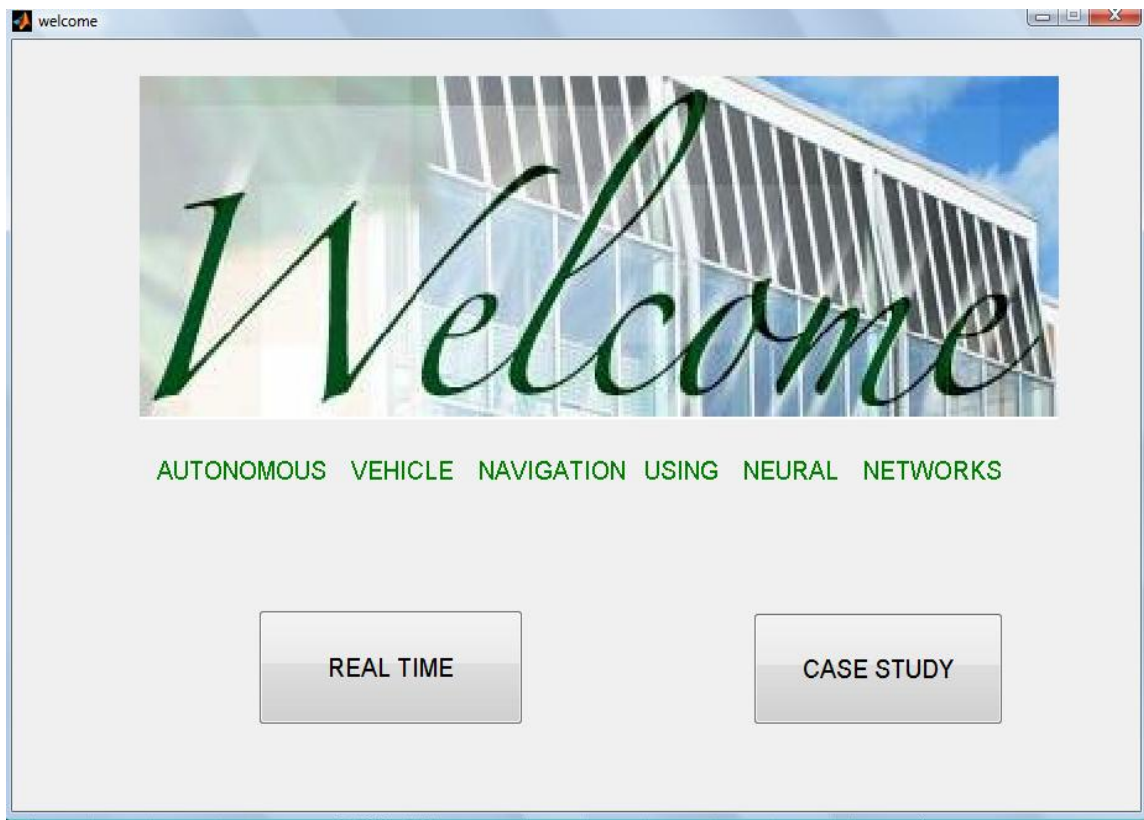


FIGURE 5(a)

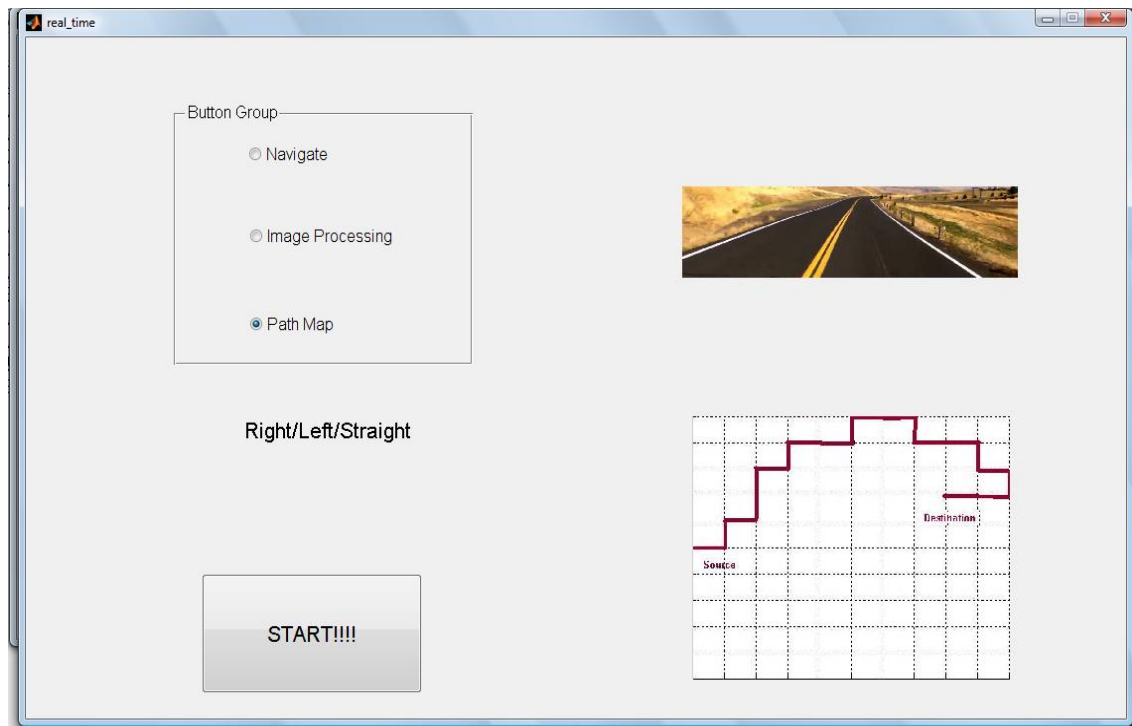


FIGURE 5(b)

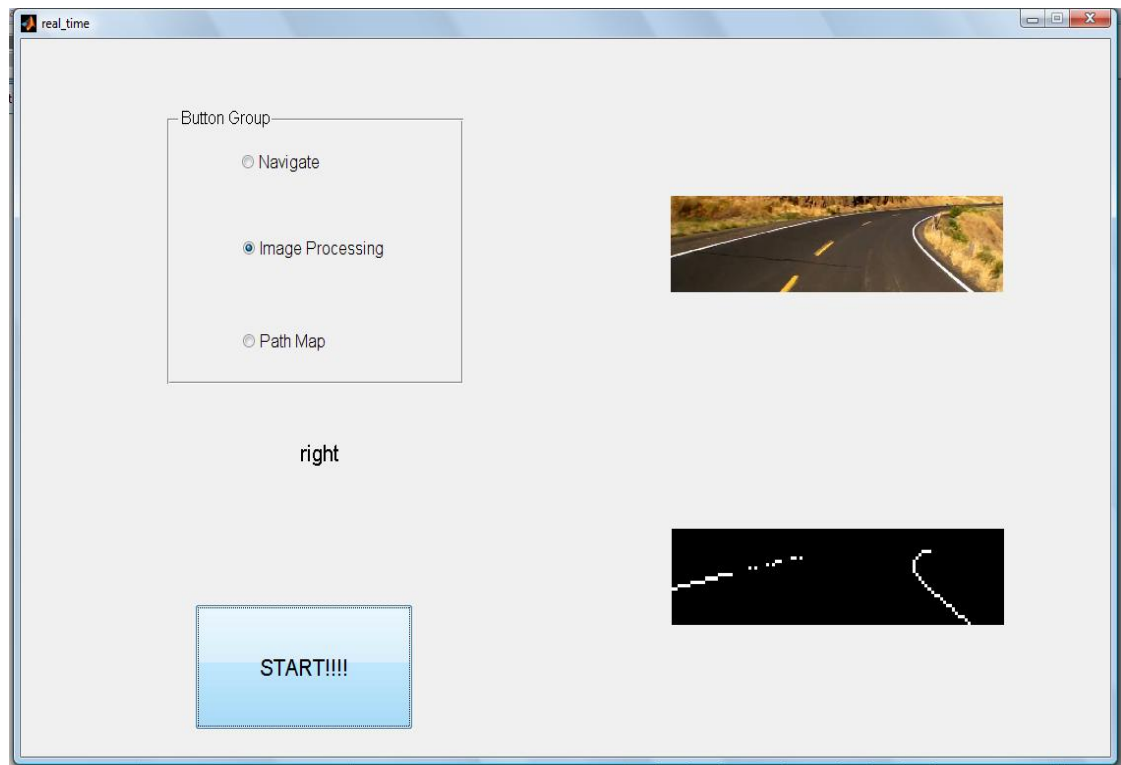


FIGURE 5(c)

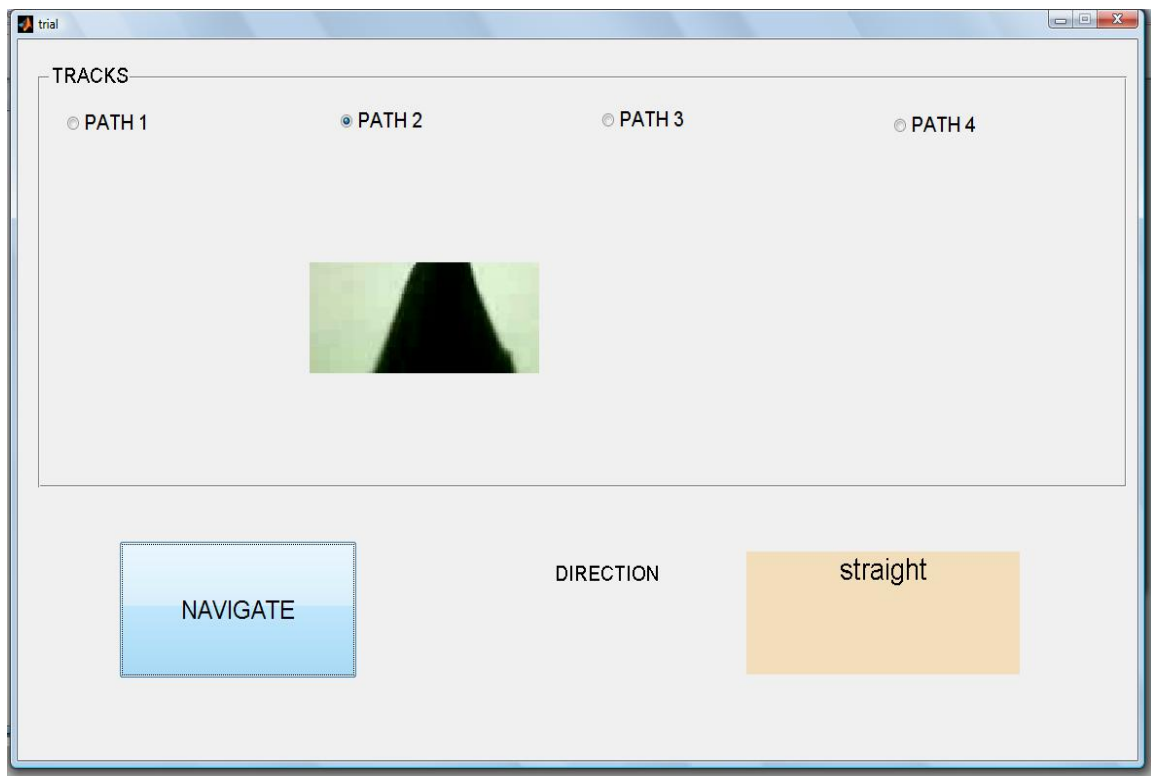


FIGURE 5(d)

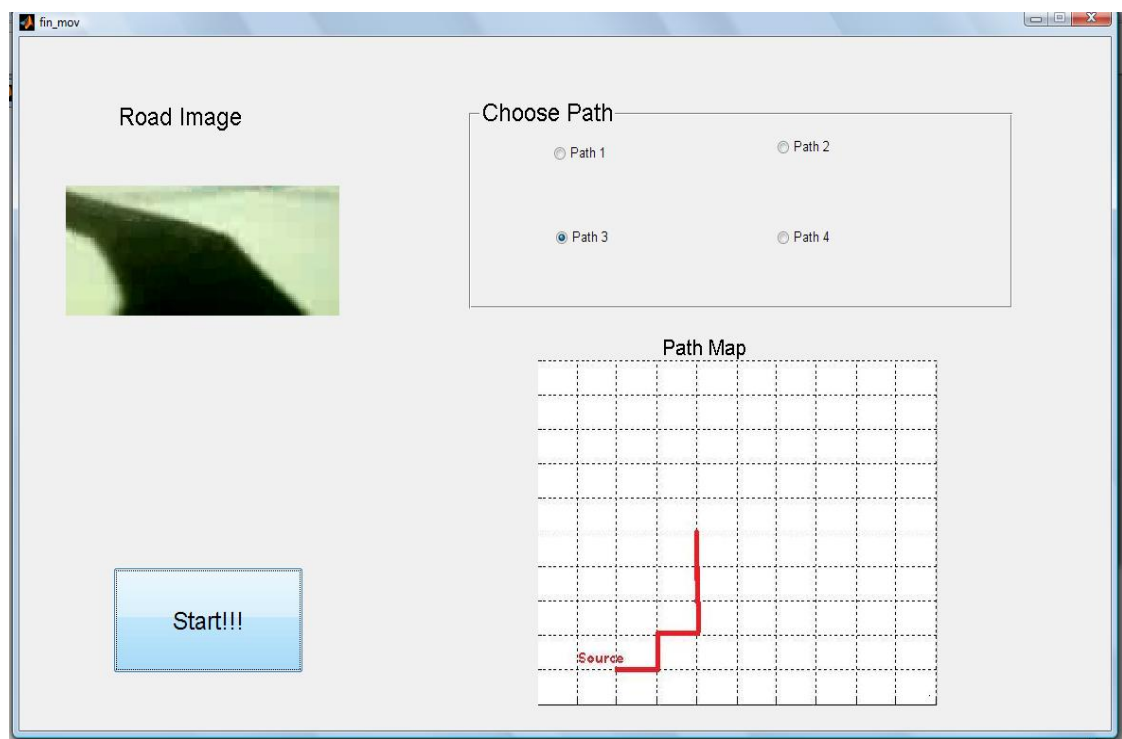


FIGURE 5(e)

6. PROBLEMS, MODIFICATIONS AND FUTURE IMPROVEMENTS

6.1 POTENTIAL PROBLEMS:

There are certain problems associated with direct implementation of the principles of this project under actual road conditions. There are detailed below:-

The algorithms have been designed only for a clear (**vacant**) **road** with no obstacles or moving vehicles, a situation which is not practically obtained. The path detection requires that the images contain some necessary distinctive features about the road like marked boundaries or lines marked along the middle portion of the road etc. Also, for color processing to be possible, we need to ensure that there are very few (major) external features having the same nature or color as the path.

Training of the neural network with images obtained under varying **illumination** conditions is a big challenge especially if color based processing is employed. On-line training or incremental training of the neural network is also not possible.

Since the system must operate under real-time conditions, the maximum possible speed of the vehicle gets restricted by the processing capability of the system. Latency and delay involved in processing of the images may cause undesirable delays in production of the required outputs for steering.

As the vehicle is steered down the center of the road during training or along a corner only, the network will never be presented with situations where it must recover from misalignment errors. When driving for itself, the network may occasionally stray from the trained path, so it must be prepared to recover by steering the vehicle back to the exact path. Another problem is that naively training the network with only the current video image and steering direction may cause it to over learn recent inputs. If the car is driven down a stretch of straight road at the end of training, the network will be presented with a long sequence of similar images. This sustained lack of diversity in the training set will cause the network to "forget" what it had learned about driving on curved roads and instead learn to always steer straight

6.2 MODIFICATIONS AND FUTURE IMPROVEMENTS POSSIBLE:

To overcome the previously mentioned set of deficiencies in the practical implementation of this project, a lot of further work needs to be done in the areas of image processing and neural network training. A number of possible solutions have been suggested below related to the problems encountered. These can be regarded as starting points for further improvement in the system.

Upon encountering obstacles or other vehicles, the neural network can be designed so as to obtain additional outputs which may control (decrease) the vehicle's speed or take a proper turn so that the obstacle can be avoided. [22]

In case there are **non-homogenous conditions** over the road or unwanted external features having the same intensity or color as the road, we can employ **morphological image processing operations** like **region filling**, **cleaning** or **erosion** to develop the isolated road path. [23]

To work with the conditions of **non-uniform illumination**, it is possible to use **image enhancement techniques** like brightness and contrast adjustment. Image subtraction is also a possible solution to this problem.

Faster image processing algorithms apart from image compression may be employed to ensure faster system response. Use of **RBF's (radial basis functions)** in place of traditional multi-layer feed-forward neural networks can ensure network training in a fraction of a second and provide very good results for large sized training sets. [24]

HARDWARE EXTENSION OF THE PROJECT:

For implementation of this project on a moving vehicle there are two techniques possible:

Mounting a camera on the vehicle and then sending the images captured to a laptop for processing using Matlab programs. Then the decision signal generated can be sent to the actuation system of car. Main points to be considered should be:

Transmission can be done wirelessly. Data transfer rate required for it can be given by camera resolution x bytes per pixel x photos per sec.

Spy cameras or security cameras are to be used to reduce time delay between transmission and reception for real time processing.

FPGA's can be used for image processing application's to get high performance at low costs. The SPARTAN 3A DSP platform FPGA is ideal for cost sensitive DSP algorithmic and co-processing applications that require high DSP performance. Using the SIMULINK modeling environment from the MATHWORKS and system generator for DSP from XILINX, complete video hardware accelerators can be quickly constructed without requiring existing knowledge of VHDL design flows.



FIGURE 6(a)

Benefits of using XtremeDSP Video Starter Kit are:

The XtremeDSP™ Video Starter Kit supplies you with everything needed to accelerate video designs using reconfigurable hardware without requiring existing knowledge of VHDL / Verilog design flows.

The Starter Kit comes complete with the Spartan-3A DSP 3400A FPGA development board, an FMC-Video IO daughter card, a Micron VGA CMOS Camera Module and a comprehensive set of Xilinx Tools.

SPARTAN 3A can run application at 40 MHz; hence can be used for our application. One of the main conditions for the successful implementation of this hardware is that the hardware should be able to take decision within a specified time interval.

The maximum possible speed of the vehicle is constrained by the processing delay of the system. Since the images should be captured at a maximum spacing of about 3 meters for obtaining accurate path information and the processing time incurred by the software is about 0.3 seconds, we can roughly sample 3 images per second. Thus, the ideal speed of the moving vehicle which can be realized using this system comes out to be 9 m/sec or 32.4 km/hr.

7. ANNEXURE

1. Newff

Purpose:

Create feed-forward backpropagation network

Syntax:

net = newff(P,T,[S1 S2...S(N-1)],{TF1,TF2...TFNI},BTF,BLF,PF,IPF,OPF,DDF)

Description:

newff(P,T,[S1 S2...S(N-1)],{TF1 TF2...TFNI}, BTF,BLF,PF,IPF,OPF,DDF)

P R x Q1 matrix of Q1 sample R-element input vectors

T SN x Q2 matrix of Q2 sample SN-element target vectors

Si Size of ith layer, for N-1 layers, default = [].
(Output layer size SN is determined from T.)

TFi Transfer function of ith layer. (Default = 'tansig' for hidden layers and 'purelin' for output layer.)

BTF Backpropagation network training function (default = 'trainlm')

BLF Backpropagation weight/bias learning function (default = 'learngdm')

PF Performance function. (Default = 'mse')

IPF Row cell array of input processing functions. (Default = {'fixunknowns','removeconstantrows','mapminmax'})

OPF Row cell array of output processing functions. (Default = {'removeconstantrows','mapminmax'})

DDF Data division function (default = 'dividerand')

This function returns an N-layer feed-forward backpropagation network.

The **transfer functions TFi** can be any differentiable transfer function such as tansig, logsig, or purelin.

The **training function BTF** can be any of the backpropagation training functions such as trainlm, trainbfg, trainrp, traingd, etc.

The **learning function BLF** can be either of the backpropagation learning functions learngd or learngdm.

The **performance function** can be any of the differentiable performance functions such as mse or msereg.

Algorithm:

Feed-forward networks consist of multiple layers using the 'dotprod' weight function, 'netsum' net input function, and the specified transfer function.

The first layer has weights coming from the input. Each subsequent layer has a weight coming from the previous layer. All layers have biases. The last layer is the network output.

Each layer's weights and biases are initialized with 'initnw'.

Adaption is done with 'trains', which updates weights with the specified learning function.

Training is done with the specified training function. Performance is measured according to the specified performance function.

2. Train

Purpose:

Train neural network

Syntax:

[net,tr,Y,E,Pf,Af] = train(net,P,T,Pi,Ai)

Description:

'train' trains a network net according to net.trainFcn and net.trainParam.

train (net,P,T,Pi,Ai) takes

net Network

P Network inputs

T Network targets (default = zeros)

Pi Initial input delay conditions (default = zeros)

Ai Initial layer delay conditions (default = zeros)

and returns as argument

net New network
 Tr Training record (epoch and perf)
 Y Network outputs
 E Network errors
 Pf Final input delay conditions
 Af Final layer delay conditions

Note that T is optional and need only be used for networks that require targets. Pi and Pf are also optional and need only be used for networks that have input or layer delays.

Algorithm:

‘train’ calls the function indicated by **net.trainFcn**, using the training parameter values indicated by **net.trainParam**.

Typically one ‘epoch’ of training is defined as a single presentation of all input vectors to the network. The network is then updated according to the results of all those presentations. Training occurs until a maximum number of epochs occur, the performance goal is met, or any other stopping condition of the function net.trainFcn occurs.

Some training functions depart from this norm by presenting only one input vector (or sequence) each epoch. An input vector (or sequence) is chosen randomly each epoch from concurrent input vectors sequences).

3. Sim

Purpose:

Simulate neural network

Syntax:

- **[Y,Pf,Af,E,perf] = sim(net,P,Pi,Ai,T)**
- **[Y,Pf,Af,E,perf] = sim(net,{Q TS},Pi,Ai,T)**
- **[Y,Pf,Af,E,perf] = sim(net,Q,Pi,Ai,T)**

Description:

‘Sim’ simulates neural networks.

[Y,Pf,Af,E,perf] = sim(net,P,Pi,Ai,T) takes

net Network

P Network inputs

Pi Initial input delay conditions (default
= zeros)

Ai Initial layer delay conditions (default
= zeros)

T Network targets (default = zeros)

and returns

Y Network outputs

Pf Final input delay conditions

Af Final layer delay conditions

E Network errors

perf Network performance

The arguments Pi, Ai, Pf, and Af are optional and need only be used for networks that have input or layer delays.

Algorithm:

sim uses these properties to simulate a network net.

net.numInputs, net.numLayers

net.outputConnect, net.biasConnect

net.inputConnect, net.layerConnect

These properties determine the network's weight and bias values and the number of delays associated with each weight:

net.IW{i,j}

net.LW{i,j}

net.b{i}

net.inputWeights{i,j}.delays

net.layerWeights{i,j}.delays

These function properties indicate how sim applies weight and bias values to inputs to get each layer's output:


```
net.inputWeights{i,j}.weightFcn  
net.layerWeights{i,j}.weightFcn  
net.layers{i}.netInputFcn  
net.layers{i}.transferFcn
```

4. nntool

Purpose:

Open Network/Data Manager

Syntax:

nntool

Description:

‘nntool’ opens the Network/Data Manager window, which allows you to import, create, use, and export neural networks and data.

5. imshow – Display image

Syntax:

imshow(I)

imshow(I,[low high])

imshow(RGB)

imshow(BW)

imshow(X,map)

imshow(filename)

himage = imshow(...)

imshow(..., param1, val1, param2, val2,...)

Description:

imshow(I) displays the grayscale image I.

imshow(I,[low high]) displays the grayscale image I, specifying the display range for I in [low high]. The value low (and any value less than low) displays as black; the value high (and any value greater than high) displays as white. Values in between are displayed as intermediate shades of gray, using the default number of gray levels.

imshow(RGB) displays the truecolor image RGB.

imshow(BW) displays the binary image BW. imshow displays pixels with the value 0 (zero) as black and pixels with the value 1 as white.

imshow(X,map) displays the indexed image X with the colormap map. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0.

imshow(filename) displays the image stored in the graphics file filename. The file must contain an image that can be read by imread or dicomread. imshow calls imread or dicomread to read the image from the file, but does not store the image data in the MATLAB workspace. If the file contains multiple images, imshow displays the first image in the file. The file must be in the current directory or on the MATLAB path.

himage = imshow(...) returns the handle to the image object created by imshow.

Class Support

A true color image can be uint8, uint16, single, or double. An indexed image can be logical, uint8, single, or double. A grayscale image can be logical, uint8, int16, uint16, single, or double. A binary image must be of class logical.

For grayscale images of class single or double, the default display range is [0 1]. If your image's data range is much larger or smaller than the default display range, you might need to experiment with setting the display range to see features in the image that would not be visible using the default display range. For all grayscale images having integer types, the default display range is [intmin(class(I)) intmax(class(I))].

If your image is int8, int16, uint32, int32 or single, the CData in the resulting image object will be double. For all other classes, the CData matches the input image class.

6. bwmorph - Morphological operations on binary images

Syntax:

BW2=bwmorph(BW,operation)

BW2= bwmorph(BW,operation,n)

Description:

BW2 = bwmorph(BW,operation) applies a specific morphological operation to the binary image BW.

BW2 = bwmorph(BW,operation,n) applies the operation n times. n can be Inf, in which case the operation is repeated until the image no longer changes.

operation is a string that can have one of the values listed below.

Operation	Description
'dilate'	Performs dilation using the structuring element ones(3).
'erode'	Performs erosion using the structuring element ones(3).
'fill'	<p>Fills isolated interior pixels (individual 0s that are surrounded by 1s), such as the center pixel in this pattern.</p> <pre> 1 1 1 1 0 1 1 1 1 </pre>
'thin'	<p>With $n = \text{Inf}$, thins objects to lines. It removes pixels so that an object without holes shrinks to a minimally connected stroke, and an object with holes shrinks to a connected ring halfway between each hole and the outer boundary. This option preserves the Euler number. See Algorithm for more detail.</p>

Class Support:

The input image BW can be numeric or logical. It must be 2-D, real and nonsparse. The output image BW2 is of class logical

.

7. edge

-Find edges in grayscale image

Syntax:

BW = edge(I)

BW = edge(I,'log')

BW = edge(I,'log',thresh)

BW = edge(I,'log',thresh,sigma)

[BW,threshold] = edge(I,'log',...)

Description

BW = edge(I) takes a grayscale or a binary image I as its input, and returns a binary image BW of the same size as I, with 1's where the function finds edges in I and 0's elsewhere.

By default, edge uses the Sobel method to detect edges but the following provides a complete list of all the edge-finding methods supported by this function:

- The **Sobel** method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
- The **Prewitt** method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
- The **Roberts** method finds edges using the Roberts approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
- The **Laplacian of Gaussian** method finds edges by looking for zero crossings after filtering I with a Laplacian of Gaussian filter.
- The **Canny** method finds edges by looking for local maxima of the gradient of I. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds, to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

The parameters you can supply differ depending on the method you specify. If you do not specify a method, edge uses the Sobel method.

Laplacian of Gaussian Method

BW = edge(I,'log') specifies the Laplacian of Gaussian method.

BW = edge(I,'log',thresh) specifies the sensitivity threshold for the Laplacian of Gaussian method. edge ignores all edges that are not stronger than thresh. If you do not specify thresh, or if thresh is empty ([]), edge chooses the value automatically. If you specify a threshold of 0, the output image has closed contours, because it includes all the zero crossings in the input image.

BW = edge(I,'log',thresh,sigma) specifies the Laplacian of Gaussian method, using sigma as the standard deviation of the LoG filter. The default sigma is 2; the size of the filter is n-by-n, where $n = \text{ceil}(\text{sigma} * 3) * 2 + 1$.

[BW,thresh] = edge(I,'log',...) returns the threshold value.

Class Support:- I is a nonsparse numeric array. BW is of class logical.

Remarks

For the gradient-magnitude methods (Sobel, Prewitt, Roberts), thresh is used to threshold the calculated gradient magnitude. For the zero-crossing methods, including Lap, thresh is used as a threshold for the zero-crossings; in other words, a large jump across zero is an edge, while a small jump isn't.

The Canny method applies two thresholds to the gradient: a high threshold for low edge sensitivity and a low threshold for high edge sensitivity. `edge` starts with the low sensitivity result and then grows it to include connected edge pixels from the high sensitivity result. This helps fill in gaps in the detected edges.

In all cases, the default threshold is chosen heuristically in a way that depends on the input data. The best way to vary the threshold is to run `edge` once, capturing the calculated threshold as the second output argument. Then, starting from the value calculated by `edge`, adjust the threshold higher (fewer edge pixels) or lower (more edge pixels).

8. `imresize` – Resize image

Syntax

```
B = imresize(A, scale)
B = imresize(A, [mrows ncols])
[Y newmap] = imresize(X, map, scale)
[...] = imresize(..., method)
[...] = imresize(..., parameter, value, ...)
```

Description

`B = imresize(A, scale)` returns image `B` that is `scale` times the size of `A`. The input image `A` can be a grayscale, RGB, or binary image. If `scale` is between 0 and 1.0, `B` is smaller than `A`. If `scale` is greater than 1.0, `B` is larger than `A`.

`B = imresize(A, [mrows ncols])` returns image `B` that has the number of rows and columns specified by `[mrows ncols]`. Either `NUMROWS` or `NUMCOLS` may be `NaN`, in which case `imresize` computes the number of rows or columns automatically to preserve the image aspect ratio.

`[Y newmap] = imresize(X, map, scale)` resizes the indexed image `X`. `scale` can either be a numeric scale factor or a vector that specifies the size of the output image (`[numrows numcols]`). By default, `imresize` returns a new, optimized colormap (`newmap`) with the resized image. To return a colormap that is the same as the original colormap, use the 'Colormap' parameter (see below).

`[...] = imresize(..., method)` resizes the indexed image. `method` can be (1) a text string that specifies a general interpolation method, (2) a text string that specifies an interpolation kernel, or (3) a two-element cell array that specifies an interpolation kernel.

Method Name	Description
'nearest'	Nearest-neighbor interpolation; the output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered.
'bilinear'	Bilinear interpolation; the output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood
'bicubic'	Bicubic interpolation (the default); the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood

Remarks

The function `imresize` changed in version 5.4 (R2007a). Previous versions of the Image Processing Toolbox used a somewhat different algorithm by default. If you need the same results produced by the previous implementation, use the function `imresize_old`.

For bicubic interpolation, the output image may have some values slightly outside the range of pixel values in the input image. This may also occur for user-specified interpolation kernels.

Class Support

The input image can be numeric or logical and it must be nonsparse. The output image is of the same class as the input image. An input image that is an indexed image can be `uint8`, `uint16`, or `double`.

GUI FUNCTIONS

1. `guide` -Open GUI Layout Editor

Syntax:

```
guide
guide('filename.fig')
guide('fullpath')
guide(HandleList)
```

Description:

guide initiates the GUI design environment (GUIDE) tools that allow you to create or edit GUIs interactively.

guide opens the GUIDE Quick Start dialog where you can choose to open a previously created GUI or create a new one using one of the provided templates.

guide('filename.fig') opens the FIG-file named filename.fig for editing if it is on the MATLAB path.

guide('fullpath') opens the FIG-file at fullpath even if it is not on the MATLAB path.

guide(HandleList) opens the content of each of the figures in HandleList in a separate copy of the GUIDE design environment.

What is a Callback?

A *callback* is a function that executes when a specific event occurs on a graphics object. You specify a callback by setting the appropriate property of the object. This section describes the events (specified via properties) for which you can define callbacks.

Graphics Object Callbacks

All graphics objects have three properties for which you can define callback routines:

- ButtonDownFcn — Executes when you click the left mouse button while the cursor is over the object or within a 5-pixel border around the object.
- CreateFcn — Executes during object creation after you set all properties.
- DeleteFcn — Executes just before deleting the object.

User Interface Object Callbacks

User interface objects (e.g., uicontrol and uimenu) have a Callback property through which you define the function to execute when you activate these devices (e.g., click a push button or select a menu).

Figure Callbacks

Figures have additional properties that execute callback routines with the appropriate user action. Only the CloseRequestFcn property has a callback defined by default:

- CloseRequestFcn — Executes when a request is made to close the figure (by a close command, by the window manager menu, or by quitting MATLAB).
- KeyPressFcn — Executes when you press a key while the cursor is within the figure window.
- ResizeFcn — Executes when you resize the figure window.
- WindowButtonDownFcn — Executes when you click a mouse button while the cursor is over the figure background, a disabled uicontrol, or the axes background.
- WindowButtonMotionFcn — Executes when you move the mouse button within the figure window (but not over menus or title bar).
- WindowButtonUpFcn — Executes when you release the mouse button, after having pressed the mouse button within the figure.

2. inspect

Open Property Inspector

Syntax:

```
inspect
inspect(h)
inspect([h1,h2,...])
```

Description:

inspect creates a separate Property Inspector window to enable the display and modification of the properties of any object you select in the figure window or Layout Editor. If no object is selected, the Property Inspector is blank.

inspect(h) creates a Property Inspector window for the object whose handle is h.

inspect([h1,h2,...]) displays properties that objects h1 and h2 have in common, or a blank window if there are no such properties; any number of objects can be inspected and edited in this way (for example, handles returned by the bar command).

The **Property Inspector** has the following behaviors:

- Only one Property Inspector window is active at any given time; when you inspect a new object, its properties replace those of the object last inspected.
- When the Property Inspector is open and plot edit mode is on, clicking any object in the figure window displays the properties of that object (or set of objects) in the Property Inspector.
- When you select and inspect two or more objects of different types, the Property Inspector only shows the properties that all objects have in common.
- To change the value of any property, click on the property name shown at the left side of the window, and then enter the new value in the field at the right.

The Property Inspector provides two different views:

- **List view** — properties are ordered alphabetically (default); this is the only view available for annotation objects.
- **Group view** — properties are grouped under classified headings (Handle Graphics objects only)

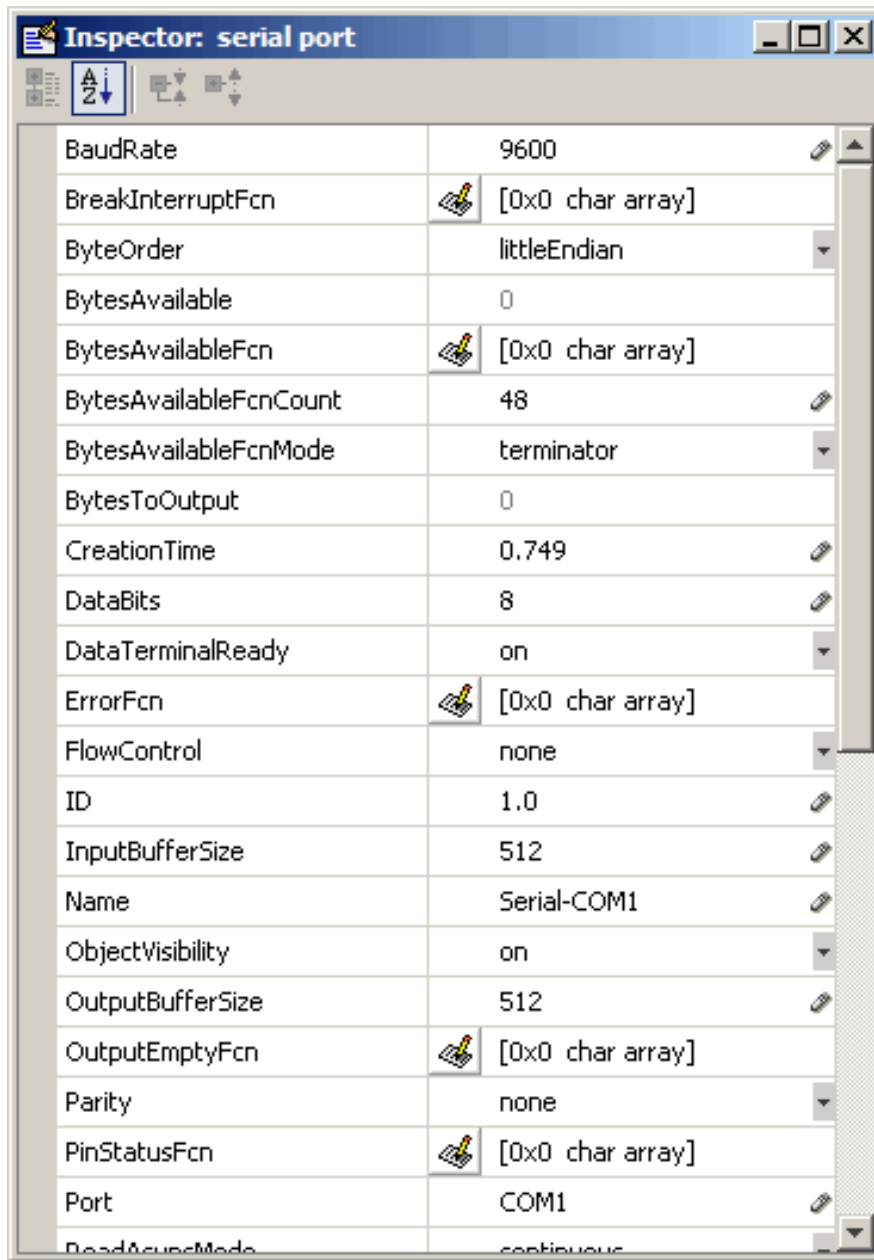



FIGURE 7(a)

Viewing the Object Hierarchy

The Object Browser displays a hierarchical list of the objects in the figure, including both components and menus. As you lay out your GUI, check the object hierarchy periodically, especially if your GUI contains menus, panes, or button groups. Open it from **View > Object Browser** or by click the Object Browser icon  on the GUIDE toolbar.

The following illustration shows a figure object and its child objects. It also shows the child objects of a uipanel.

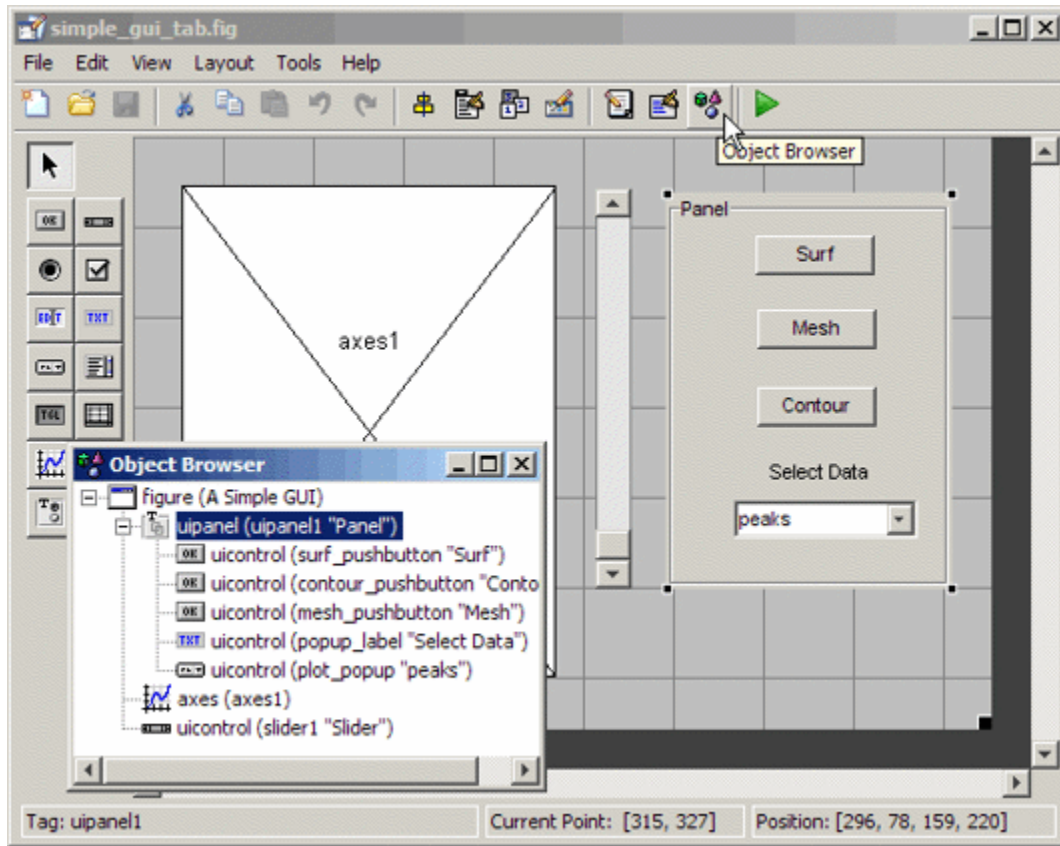


FIGURE 7(b)

To determine a component's place in the hierarchy, select it in the Layout Editor. It is automatically selected in the Object Browser. Similarly, if you select an object in the Object Browser, it is automatically selected in the Layout Editor.

ADDITIONAL RESULTS OF IMAGE PROCESSING:



Original image of road (straight)



Cropped image



Image obtained after color processing



Dilated image



Resized image



Thinned image



Original image of road (right)



Cropped image

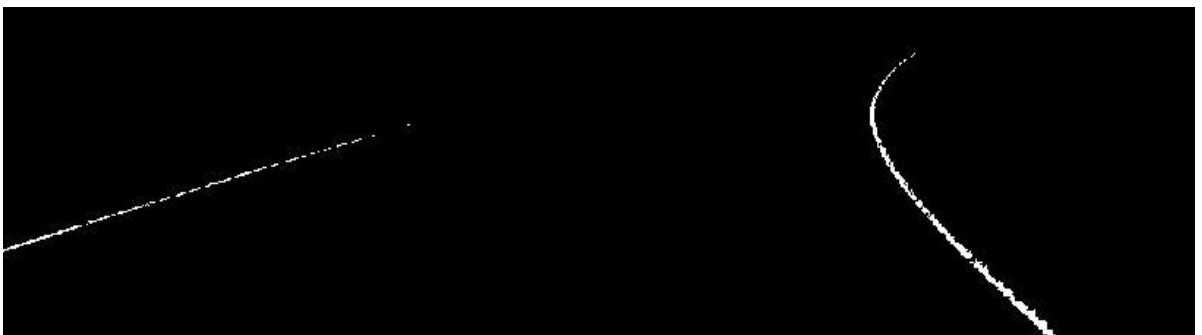
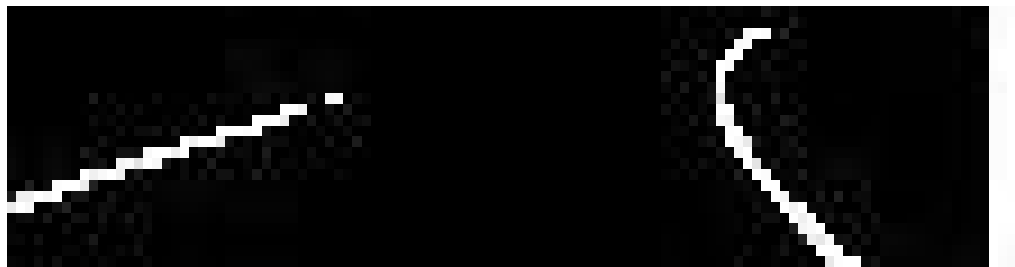


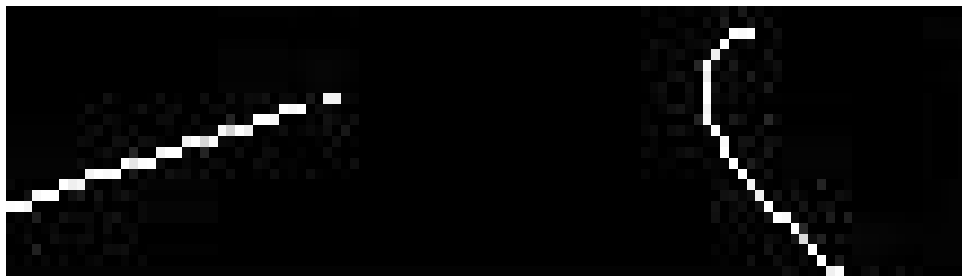
Image after color processing



Dilated image



Resized image



Thinned image

8. REFERENCES

1. Dean A. Pomerleau, ALVINN: an autonomous land vehicle in a neural network, Advances in neural information processing systems 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1989
2. <http://www.unibw.de>
Dickmanns, E. D., and Zapp, A. 1987. Autonomous high speed road vehicle guidance by computer vision. In Proc. 10th World Congress Automatic Control, Vol. 4, Munich, West Germany.
3. <http://www.umich.edu>
4. <http://www.cmu.edu> :- Pomerleau, D. A. 1990. Neural network based autonomous navigation. In Vision and Navigation: The CMU Navlab, Charles Thorpe, ed., pp. 83-92. Kluwer Academic Publishers, Boston, MA.
5. <http://www.unipr.it/>
6. <http://www.argo.ce.unipr.it/ARGO/english/>
7. <http://www.gm.com/>
8. http://www.ri.cmu.edu/research_project_detail.html
9. Artificial Neural Network: A Tutorial By Anil K. Jain, Michigan State University, K.M. Mohiuddin IBM Almaden Research Centre
10. Haykin, S., Neural Networks: A Comprehensive Foundation, Prentice Hall, Upper Saddle River, NJ, 1999.
11. Zurada, J., Introduction to Artificial Neural Systems, West Publishing Co, St. Paul, MN, 1992.
12. Efficient Training of Artificial Neural Networks for Autonomous Navigation By Dean A. Pomerleau
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213
13. http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html
Mariusz Bernacki
Przemysław Włodarczyk (2005)
14. Vogl, T.P., J.K. Mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon, "Accelerating the convergence of the backpropagation method," Biological Cybernetics, Vol. 59, 1988, 257-263

15. Riedmiller, Proceedings of the IEEE International Conference on Neural Networks (ICNN), San Francisco, 1993, pp. 586-591
16. Rafael C. Gonzalez, Richard E. Woodes, Digital Image Processing, Beijing: Publishing House of Electronics Industry, 2nd ed, 2003.
17. Fundamentals of digital image processing
Anil K. Jain
18. Canny, John, "A Computational Approach to Edge Detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8
19. Pratt, William K., Digital Image Processing, John Wiley & Sons, Inc., 1991.
20. Matlab basics
http://www.mathworks.com/access/helpdesk/help/techdoc/creating_guis/bqz79mu.html
21. MATLAB® Advanced GUI Development By Scott T. Smith
22. Correlation based optical ranging and proximity detector
United States Patent 6307622
23. Digital Image Processing Using MATLAB Gonzalez, Woods, and Eddins
24. Introduction to Radial Basis Function Networks By Adrian G. Bors Department Of Computer Science , University Of York York YO10 YDD U.K.