# A low cost vision-guided car for autonomous racing car competitions

Samuel N. Cubero, Donal Tjoe
Department of Mechanical Engineering
Curtin University of Technology
Perth, Western Australia

**Abstract**

*This paper presents a brief "case study" report on the design and performance of a low-cost self-driving, self-learning vision-guided robot car (VIC) that can drive in a left or right side lane and even identify and overtake obstacles or vehicles in front of it. In the future, it is hoped that more of these types of low-cost robots will be built and used for fully autonomous robotic racing car competitions.*

*Topics that are covered herein include a brief description of CMU's NAVlab self-driving car, which inspired this work, low-cost mechanical and electronic hardware, image analysis, object detection, ANN (Artificial Neural Network) training, the control software, test results and future work to improve robot performance and capabilities.*

## 1 Introduction

This paper discusses in detail a "case study" report on the design and development of a "Vision-guided Intelligent Car" (called VIC), designed and built by Donal Tjoe and Sam Cubero at Curtin University of Technology, Australia. The Vision-guided Intelligent Car (VIC) project started in early 2004. The predecessor of the VIC, called the "CARbot" [1], built in 2003, relied on a single 1D line-scan camera and it was found that it could not collect enough information to execute reliable overtaking manoeuvres. The goal was to build a miniature version of the NAVlab vision-guided automobile and implement automatic steering and speed control for a radio-controlled toy car so that it can serve as a basic platform for image analysis and vision guidance experiments in the future.

## 2 CMU "NAVLab" self-driving car

"Autonomous Land Vehicle in Neural Network" (ALVINN), Figure 1, was a project developed at Carnegie Mellon University (CMU), headed by Dean Pomerleau [2] in 1989. The test vehicle was called "NAVLab".

NAVLab took advantage of "Artificial Neural Network" (ANN) learning ability to establish an intelligent navigator for the vehicle. NAVLab was equipped with a vision system that was aimed at the road ahead of the vehicle. The artificial intelligence software learned the driving methodology by watching all the actions of a human driver in response to video images seen during normal driving conditions. As the vehicle was driven on the road under human guidance, the computer system recorded the video images and the human steering movements during a recording phase. The recorded data was then used for training, ie. adjusting the weights of the neural network, so that the ANN could learn how to drive.

**Figure 1. NAVLab vision-guided car [2]**

During on-road tests, ALVINN showed satisfactory performance and was reported to have been able to successfully drive a car across the USA "hands free" for 98% of the entire journey. It managed to navigate autonomously and safely in different types of environments including single lane dirt roads, suburban neighbourhood streets and on two-lane highways at speeds up to 80 km/hour. However, despite the robustness of the system, there were a few drawbacks to its design. Firstly, the network training procedure was performed "on-the-fly", which means that the training was done at the same time as the recording, ie. the network was set to learn instantaneously as the human trainer drives the vehicle. Thus, the network could not learn how to perform a recovery manoeuvre unless faced with a real disaster, eg.

Another motorist performs an illegal move which could lead to a collision or a car in front suddenly stops quickly.

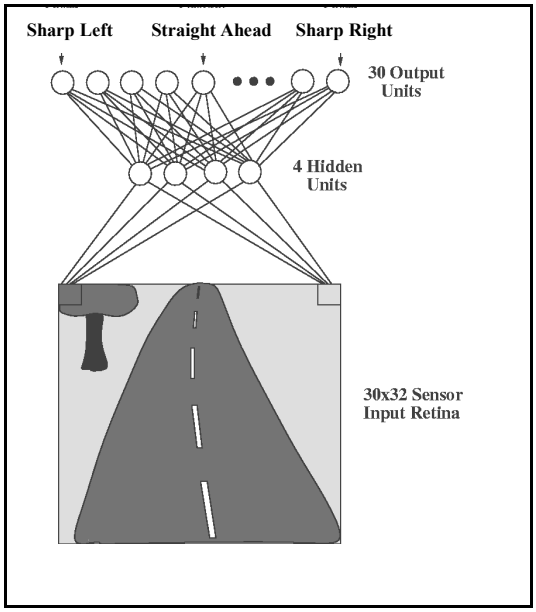


**Figure 2. Neural Network [2 Pomerleau]**

Effective emergency manoeuvres cannot be controlled by the driver or trained "on the fly", and hence, corrective action must be preprogrammed in order to implement "defensive driving" techniques, like swerving to avoid obstacles, other vehicles or pedestrians. The second drawback of the system comes from the ineffective image processing applied to the acquired images. Since the project was done in the late 1980's, complex image processing was not feasible because the available hardware simply was not fast enough to process the vast amount of video data in real-time and respond at high speeds. Another common problem of forward facing video cameras is saturation of the CCD sensors caused by the reflection of sunlight from the road, especially when the vehicle is driving towards the sun during early morning or late afternoon hours. Sensor saturation means that the data values for most pixels are at their maximum levels due to excessive, uncontrollable light energy from the sun, hence, this renders CCD sensor data useless for image analysis and guidance purposes because there is not enough contrast between the brightest and darkest pixels in the image. This last drawback is a common problem for most forward facing vision systems because it is impossible to control the position of the sun relative to the direction of the road in order to avoid direct or intense reflected sunlight reaching the CCD sensor. Several automobile manufacturers (including BMW and Mercedes Benz), have developed prototype cars that can automatically steer and drive themselves using forward facing video cameras, however, such vehicles were probably not put into mass production due to the high risk of sensor saturation and unavoidable system failure. Unfortunately, even today, the risks of failure and costs of implementing such systems in existing cars are very high. (eg. Assistware$^{TM}$ in the USA marketed their automatic steering system for about US$30,000 per vehicle).

## 3 Artificial Neural Networks (ANN)

Humans possess the ability to learn from experience. This ability allows people to make accurate predictions and judgements while dealing with new situations by associating the situation faced with previous experiences. This explains how humans can increase their performance in sports after going through several training sessions, and how humans can perform tasks such as recognising a person from a photograph, or by only hearing the person's voice. It is not easy to appreciate that a simple task that many of us take for granted, like catching a ball, is actually comprised of several complex procedures. To catch a ball from the air, the brain has to make judgements based on the 3D image captured from the stereo eyes and relate this to the position of the hands in 3D space so that the hand can move towards the expected future 3D position of ball in order to grab it at the expected time of impact. Subsequently, the brain has to send signals to the motoric nerves to actuate the hands accordingly. Such complex procedures are almost impossible to describe using pure logic and mathematical formulae, but surprisingly, a trained human brain can execute such tasks relatively quickly and efficiently, and with a high degree of accuracy because of the way that human brains are constructed. A biological brain consists of billions of information-processing units called neurons. These neurons are interconnected by junctions called synapses. A neuron receives information from other neurons or senses through synapses. The information is then processed to generate an output, which is later sent to other neurons or response units, also through a synapse. The strength of each synapse connection is constantly adjusted in the learning process. This is how memories are stored in a human brain. Memories in the brain are chemical residues created and built up by frequent repetition of connective actions between brain cells, although, the entire process of how brain cells behave and operate is still largely a mystery to scientists even today.

Artificial Neural Network (ANN) technology is a branch of artificial intelligence that models the construction of a biological brain with the aim of attaining human-like abilities in performing cognitive tasks. Kohonen (1988) [3] defined ANN as massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organisations, which are

intended to interact with the objects of the real world in the same way as biological nervous systems.

ANN is prominent in performing tasks such as pattern recognition and object classification. A neural network can be trained using a set of sample input patterns and expected output patterns. When faced with other input patterns, the network can predict the output patterns based on its training.

Neural networks are widely used in technologies such as character recognition, pattern matching and speech recognition. The main advantages of ANN include the capacity to be trained to classify poorly structured inputs, robustness against noise in data (fault tolerance), flexibility (the ability to generalise) and the ability to "learn from experience or training". However, ANN is not without its disadvantages, as listed below. Neural networks have poor knowledge representation and explaining capability, thus making it difficult to learn or follow its logic of making or predicting future decisions.

## 3.1 Neural Network Architecture

An artificial neural network consists of a number of very simple and highly interconnected processors, also called neurons, which are analogous to the biological neurons in the brain. The neurons are the main component of an ANN. An individual neuron can collect a number of inputs from the system inputs or other neurons and then generate a single output, all through links that are analogous to synapses in biological brains. The output signal from the neuron, in turn, splits into a number of branches that transmit the same signal.

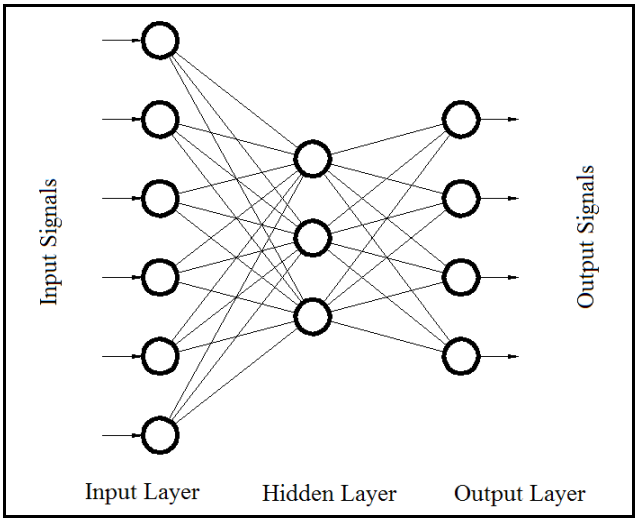


**Figure 3. Typical Neural Network Architecture**

Each of the links that connect the neurons has a numerical weight associated with it. Weights are the basic means of long-term memory in neural networks. They express the importance of each neuron input, and similar to the biological brain, a neural network learns through repeated adjustments of these weights.

A typical ANN is made up of a hierarchy of layers (Figure 3) and the neurons in the networks are arranged along these layers. The neurons connected to the external environment form input and output layers. The weights are modified to bring the network input/output behaviour into line with that of the environment.

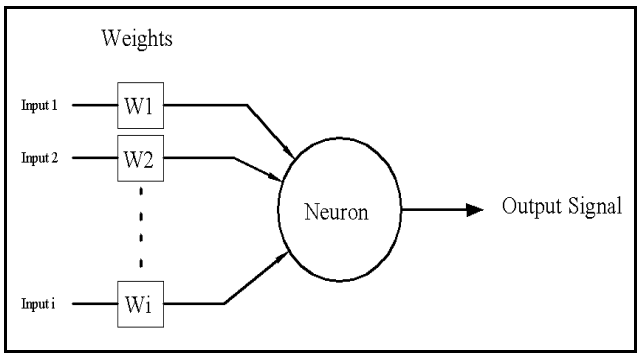


**Figure 4. Neuron Connections**

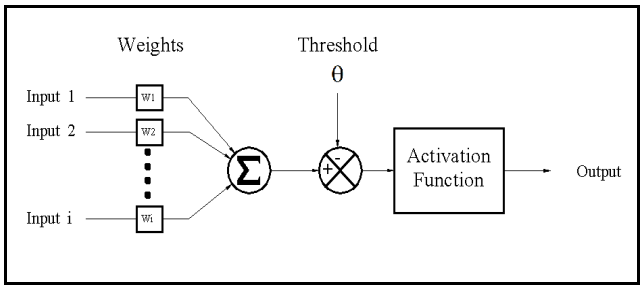


**Figure 5. Internal Neuron Computation Process**

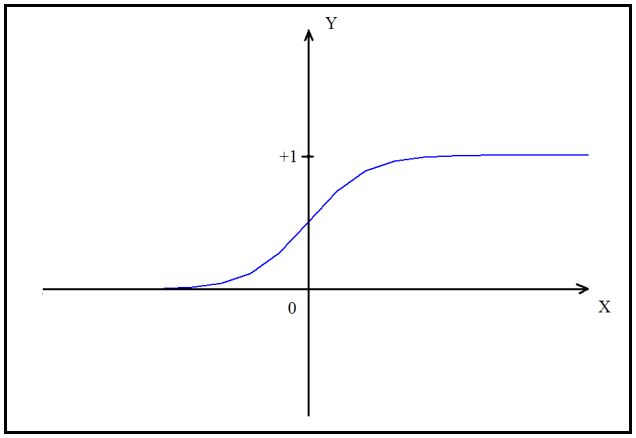


**Figure 6. Sigmoid activation function**

There are four types of activation functions commonly used in the neural networks field, namely the step, sign, linear and sigmoid activation functions. The form of activation currently utilized in this project is the sigmoid function. Hence, it is appropriate to further discuss this

method, while the various other methods of activation will not be described in this "case study" report.

The equation of a sigmoid activation function is expressed as follows:

$$Y^{sigmoid} = \frac{1}{1+e^{-X}} \quad (1)$$

$$X = \sum_{i=1}^{n} xiwi - \theta \quad (2)$$

Where *xi* and *wi* is the value of input *i* and weight *i* respectively, $\theta$ is the local threshold and *n* is the number of inputs fed to the neuron. The sigmoid function is shown in Figure 6.

## 3.2 Back-propagation Algorithm

"Back-propagation" is a technique used for training feed-forward neural networks (networks that have no feedback or those with no connections that loop). During the training, the network is presented with a set of sample input signals along with the expected output signals and through a number of iterations or epochs, the output of the network is forced to converge to the expected output signals by adjusting the weights of neuron connections. The term is an abbreviation for "backwards propagation of errors". As the algorithm's name implies, error signals are propagated backwards for weight modification purposes. For a three layer feed-forward neural network, Negnevitsky [4] derived the back-propagation learning algorithm into the following steps:

*Step 1: Initialisation*

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range, eg. (-0.5, 0.5)

*Step 2: Activation*

Activate the back-propagation neural network by applying inputs $x_1(p)$, $x_2(p)$,… $x_n(p)$ and desired outputs $y_{d,1}(p)$ $y_{d,2}(p)$,…..$y_{d,n}(p)$.

(a) Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j\right] \quad (3)$$

where n is the number of inputs of neuron j in the hidden layer, and sigmoid is the sigmoid activation function.

(b) Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \times w_{jk}(p) - \theta_k\right] \quad (4)$$

where m is the number of inputs of neuron k in the output layer.

*Step 3: Weight training*

Update the weights in the back-propagation network propagating backward the errors associated with output neurons. Steps (a) & (b):

(a) Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p)\,[1-y_k(p)]\,e_k(p) \quad (5)$$

where

$$e_k(p) = y_{d,k}(p) - y_k(p) \quad (6)$$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha\ y_j(p)\ \delta_k(p) \quad (7)$$

where α is the *learning rate parameter.*

Update the weights at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \quad (8)$$

(Step 3 continued on next page…)

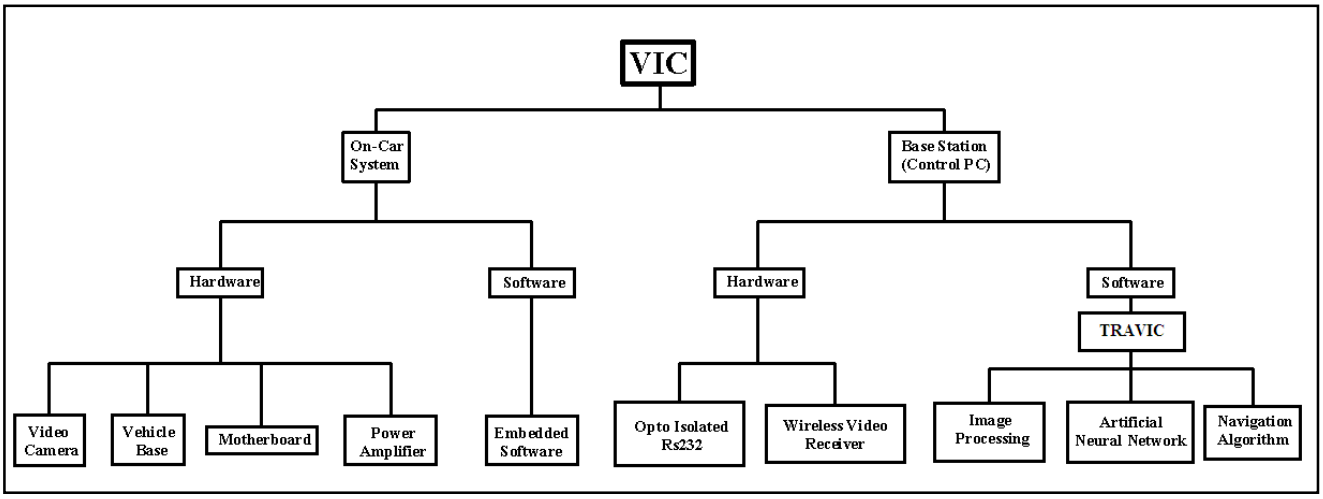


**Figure 7. System Hierarchical Chart for the VIC (Vision-guided Intelligent Car) robot**

(Step 3) (b) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) \times w_{jk}(p) \quad (9)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha \; x_i(p) \; \delta_j(p) \quad (10)$$

Update the weights at the output neurons:

$$W_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p) \quad (11)$$

*Step 4: Iteration*

Increase iteration p by one, go back to Step 2 and repeat the process until the selected error criterion is satisfied.

The error criterion in this design is based on the sum of the squared error of the output layer neurons. The sum of the squared error acts as an indicator of the network's performance. When the value of the sum of squared errors in an entire pass through all training sets, or epoch, is sufficiently small, the network is considered to have converged to the desired pattern and the training can be stopped.

The artificial neural network used in this design was coded in software using Microsoft® Visual Basic™ 6.0 Professional for the Windows® 9x/ME/2000/XP 32-bit operating system (OS), which is capable of accessing video streams from many sources. There were numerous free downloadable ANN packages available on the internet and using them may have simplified this particular part of the design. However, since the neural network was only one part of the control software and it had to work together with the other software control components, such as the image processing and the navigation algorithm, it was easier to build a neural network specially designed to match VIC's software requirements rather than adjust 3rd party software which could introduce many developmental limitations. The implementation of the ANN in the control software will be discussed later in this paper.

## 4 Hardware for the VIC robot

Figure 8 illustrates the flow of data communications between the vehicle and the base station (PC, or personal computer). As the vehicle moves around the track, video streams are sent from the on-car camera to the base station. The base station would then create a navigation decision to keep the vehicle on the track (even staying within an intended lane) based on the received road images. This navigation decisions were sent as command lines to a microcontroller unit (MCU) on the vehicle through a serial communication line. The microcontroller controlled the speed of the drive motor and the steering angle of the vehicle based on this command line. The detailed design description of the entire system is clarified in following sections. The base of the VIC vehicle is the chassis of a Tamiya™ m-02 (1:10 scale) radio controlled car [5] with rear drive and independent front steering.

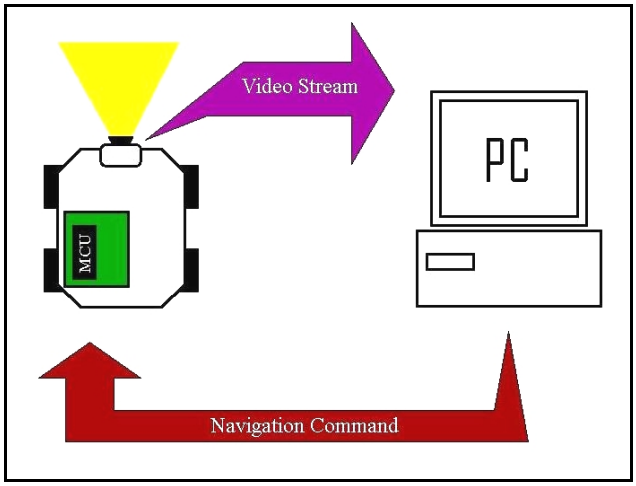


**Figure 8. Communication between VIC and PC**

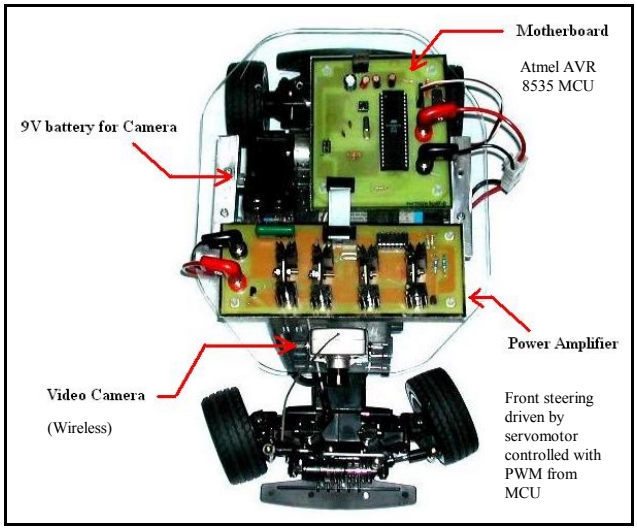


**Figure 9. Top view of the VIC robot**

The position for the Futaba S3003 servomotor (for steering the front wheels) was set by feeding it a PWM (Pulse Width Modulated) signal with a pulse width ranging between 1ms to 2ms, at a frequency of about 50 to 60Hz (or a period between 15-20ms). The relationship between the width of the pulses and angular position had first to be observed experimentally to achieve accurate control because the 1ms to 2ms pulse width dictates the stopping position for the servomotor output shaft.

Normally, sending a 1ms wide pulse would position the steering servomotor shaft to $0^o$, and sending a 2ms wide pulse would turn the shaft to the maximum position, $180^o$. Therefore, to make the servo turn to the neutral position ($90^o$), the pulse had to be 1.5ms wide. The rear wheels were driven by a single H-bridge controlled DC motor (with gearbox) which could be controlled to move the VIC robot forwards and backwards at variable speeds. The H-bridge drive circuit is shown in Figure 10, showing the input signal "a" (PWM signal with a duty value ranging between 0 and 100%). Both drive and steering motors were completely controlled by the software in the MCU. The vehicle was powered by two separate batteries. The first battery was a 7.2V 1500mAH battery pack, normally used for radio control cars. With such sizeable capacity, the battery was able to power the entire vehicle (except the video camera); this included the rear motor, the servomotor and the microcontroller. The video camera was powered using a standard 9Vdc battery.

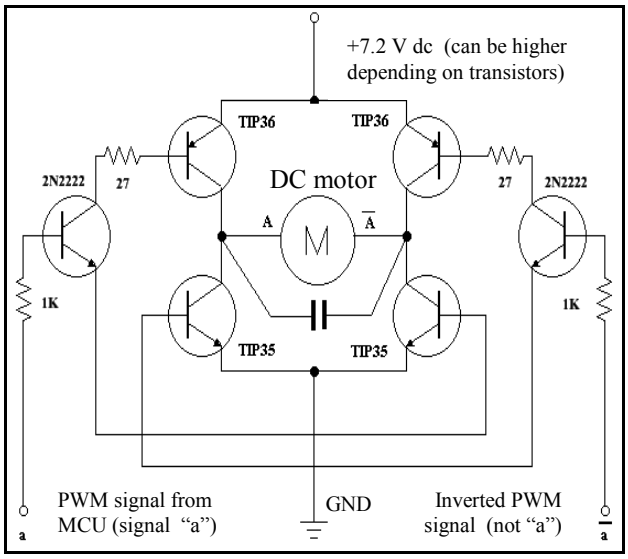


**Figure 10. H-bridge circuit to drive rear wheels**

The camera used for the vision system was a wireless CMOS camera manufactured by Shenzhen Lianyida Science Company [6]. The camera was equipped with a transmitter device and the video stream was sent to the receiver through a 2.4GHz RF transmission. The camera can transmit video images up to a distance of 50m.

The camera provided a two dimensional image with a resolution of 384x289 pixels (PAL-B format). The graphic input capture card in the PC converted this data into 24-bit RGB format. The wireless video receiver was made by the same company that manufactured the wireless camera. The LYD-RC100 is powered by 9V DC source. It received data with the same frequency of 2.4GHz. The video receiver outputted analogue video steams in PAL-B format to the base station PC video capture card. The graphic input capture card used was a TV tuner card, namely a PlayTV ProUltra™, manufactured by PixelView™. The capture card was capable of receiving a composite video stream from colour CCD camera, recording the video input and displaying it on the monitor. The video stream could be accessed from the Microsoft Windows™ clipboard in form of RGB arrays (24-bit colour, 8-bit Red, 8-bit Green, 8-bit Blue) and analysed in Visual Basic™ 6.0 for Microsoft Windows™ 98SE/XP.

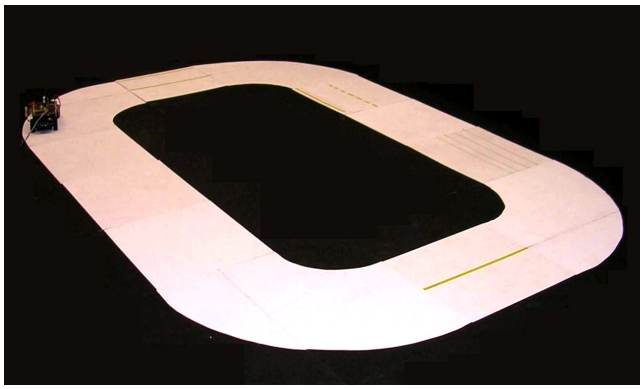

**Figure 11. White race-track for testing VIC robot**

The VIC robot was tested on a white track painted over black panels. The test track, shown in Figure 11, was made 55cm wide to provide enough room for the vehicle to overtake obstacles and other vehicles.

## 5 Embedded control software for the MCU

The embedded software was written in C, compiled using CodeVision™, targetting the Atmel™ AVR AT90S8535 microcontroller [7] (MCU). The software was designed essentially to receive instructions from the base station PC and generate the appropriate PWM signals for both the steering and driving motors. The instructions sent by the base station were in the form of an 8-bit command byte which was divided into two 4-bit command codes or nybbles by the MCU. The first nybble (bits 7, 6, 5 and 4) carried the desired speed level and the second nybble (ie. bits 3-0) carried the desired steering direction.

Figure 12 shows the "control byte" sent from the PC to the MCU via serial communications (RS232 COM1 port to UART) to control drive and steering motors, allowing a range of up to 16 control values for each motor.

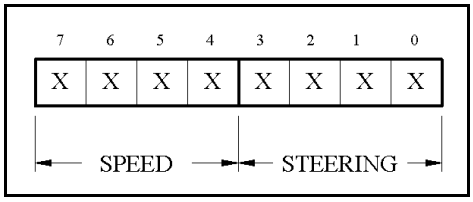


**Figure 12. Control byte sent from PC to MCU**

The "control byte" allows the VIC robot to have up to 16 steering angles and 16 speed levels. However, in the final design, in order to simplify the neural network architecture, the steering direction was limited to seven. The 16 speed levels were divided to eight forward speeds and eight backward speeds, where zero corresponded to the maximum forward speed and fifteen corresponded to the maximum reverse speed. Levels 7 and 8 were in the middle of the range, thus, the DC average voltages of the PWM signals at these particular levels were very close to zero and resulting torques generated by the driving motor were not sufficient to move the vehicle. Consequently, these two levels were used for braking purposes.

The seven steering codes carried by the command line were represented by an integer ranging from zero to six. The binary code 3 corresponded to the straight steering direction. Binary codes lower than 3 were for left-turning with 0 as the sharpest left-turn steering angle. Binary codes higher than 3 corresponded to right-turn steering directions with 6 giving the sharpest right-turning steering angle. The command codes for controlling all steering and driving movements are shown in Figure 13.

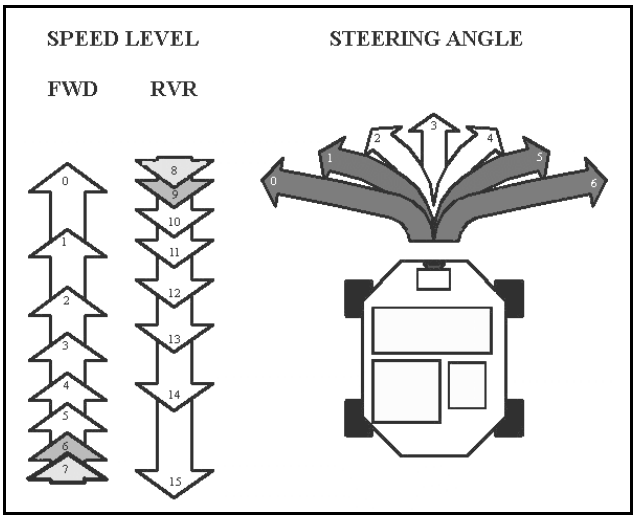


**Figure 13. Command codes for the VIC robot**

The Visual Basic "MS-Comm" ActiveX control component was used to send single-byte commands from the RS232 COM1 PC serial port to a MAX232 chip and the RxD (receive) pin of the MCU's UART hardware.

## 6 Image analysis for VIC navigation

The control software was named TRAVIC (Track Routing Algorithm for VIC). It includes components for image processing, artificial neural network control and a navigation algorithm. The relationship between these three components in TRAVIC is illustrated in Figure 14.

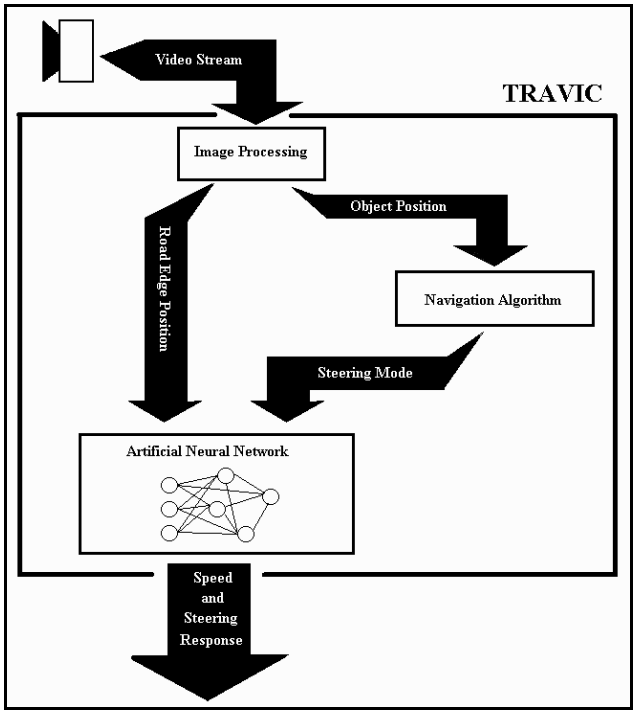


**Figure 14. Components of TRAVIC controller**

The TRAVIC software has only one input from the vehicle vision system. The image processing interprets the visual input, acquires the position of the road edge and detects the presence of object in front of the vehicle. If an object (in this case another vehicle) is present, the image processing software obtains the position of the object.

The navigation algorithm determines the steering mode based on conditions in front of the vehicle. The available modes are "normal", "overtaking" and "transition" mode. In normal mode, the vehicle cruises in the left hand side of the road at its normal speed. It is assumed that the object or vehicle in front is only visible in the left lane of the track, hence, when overtaking, VIC must change its orientation by performing a translation from the left lane to the right lane of the track. VIC maintains its position on the right hand side of the road for a specific time before changing back to left hand side. The transitional mode is very short in duration and is activated when the vehicle completes the overtaking manoeuvre. As its name suggested, it is a transitional phase from overtaking to normal mode. During this mode, the vehicle switches back to the left hand lane and the speed is set at a lower level. The purpose of this is to bring back the vehicle to its normal speed quickly prior to lane changing.

The acquired road edge position and the steering mode are used by the neural network to predict a desirable steering response. The neural network needs to be trained twice so that the vehicle can drive on both sides of the track. Thus, there are two sets of weights that must be trained, one set for each "lane" of the dual-lane track.

When the vehicle is in normal mode, the Left Hand Side (LHS) weights are set to "active" so that the vehicle can maintain its position in the left side or left lane of the road. If the overtaking mode is activated, the network must switch from the LHS weights to the Right Hand Side (RHS) weights. Once the overtaking is completed and the mode is set to transition mode, the neural network switches back to the LHS weights and it remains active until the next overtaking procedure occurs. The road edge positions are basically the inputs for the neural network. The inputs and the active set of weights generate the steering direction to maintain the vehicle's position in its corresponding lane.

The forward speed level of the car is determined based on the steering direction generated by the neural network and the active steering mode. For each mode, the straight steering direction corresponds to the highest speed level. The sharper the turning direction, the lower the speed level. The speed levels for each steering direction and modes are given in Table 1 (Compare Figure 13). The values in Table 1 were obtained from several experiments and were chosen to best suit each mode of driving.

**Table 1. Speed Level for different modes of VIC**

| STEERING DIRECTION | SPEED LEVEL | | |
|---|---|---|---|
| | Normal Mode | Overtaking Mode | Transition Mode |
| 0 | 5 | 6 | 6 |
| 1 | 5 | 5 | 6 |
| 2 | 5 | 5 | 6 |
| 3 | 4 | 3 | 6 |
| 4 | 5 | 5 | 6 |
| 5 | 5 | 5 | 6 |
| 6 | 5 | 6 | 6 |

## 6.1 Road Edge Detection

The main goal of the road edge detection was to provide the neural network with a set of data that indicated the current shape of the track ahead in a simple manner rather than feeding all the RGB data of the entire screen to the network. The edge detection only analysed a part of the screen that displays the track. The area of interest was divided into 8 rows where in each row the software searched for both edges of the road by looking for the points where the colour turns from dark to bright and vice versa. Basically, the outcomes of the edge detection were a set of sixteen coordinates which were divided into two groups of eight. One group contained the left road edge coordinates and the other group contained the right road edge coordinates.

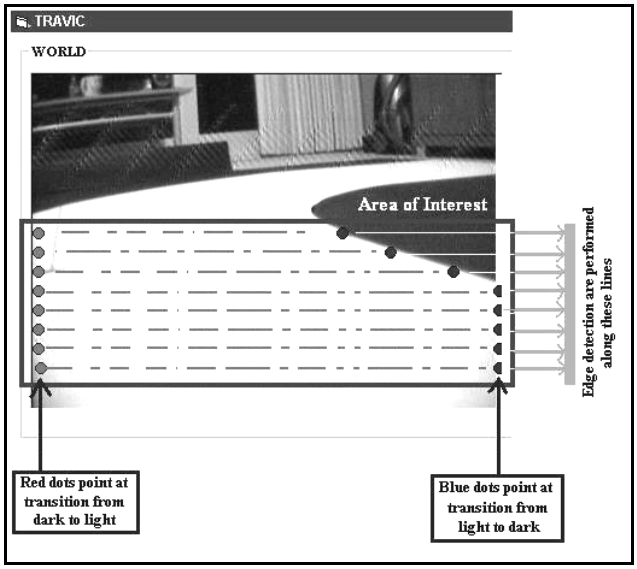


**Figure 15. Edge detection in the area of interest**

Figure 15 shows a snapshot of the vision system screen displaying the road with edge detection applied to the image. The eight left-side dots indicate the pixel positions where the software detected transitions from dark to bright, and the eight right-side (darker) dots indicate the pixel positions where the software detected transitions from bright to dark. Due to the camera's limited field of view, the vision system can only show one edge of the road. In Figure 15, part of the right-side edge is detected. A simple "bisection" algorithm is used to scan along each of the eight scanned "lines" to find the transition positions between "dark to bright" pixels (left side dots) and "bright to dark" pixels (right side dots).

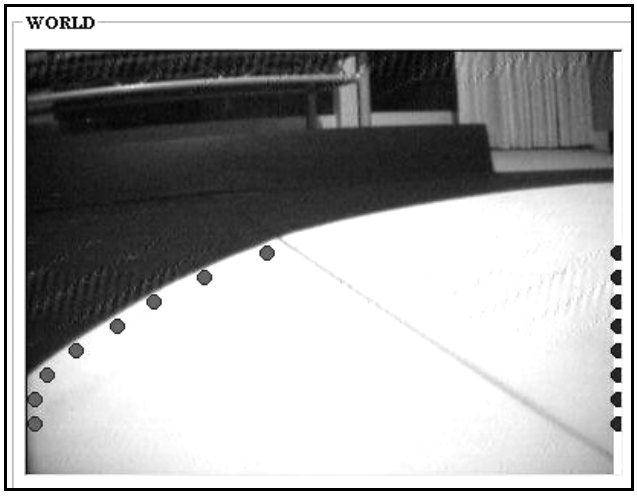


**Figure 16. Edge detection for left side of track**

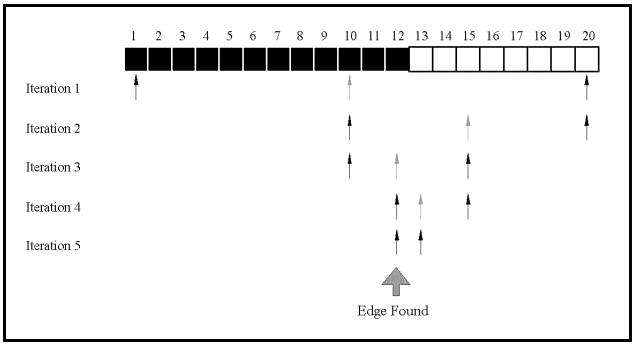


**Figure 17. Bisection method for edge detection**

The bisection method allows an edge to be detected very quickly and efficiently. To show the effectiveness of binary search, consider a line of 20 black and white pixels shown in Figure 17. For each iteration, the light coloured central arrow shows the "bisection" point or middle value between the two outer dark arrows. The transition from dark to bright occurs at pixel 12 and 13 because the pixels are side-by-side and the difference between their brightness values exceeds a set threshold value. If the edge detection is done by applying linear search, (ie. checking the difference in brightness between every pair of neighbouring pixels to see if it exceeds a threshold value), it would take 12 iterations or loops until the target is achieved (up to 19 for a worst case edge). A binary search needs only 5 iterations for this simple example.

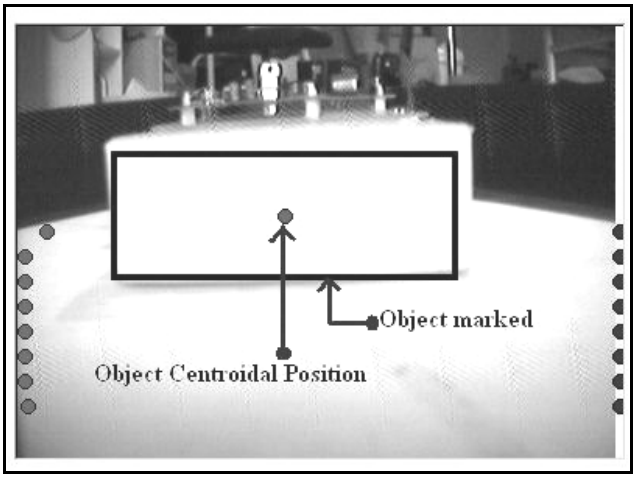


**Figure 18. Object recognition**

## 6.2 Object Recognition

The purpose of object recognition for the VIC robot is to detect the presence of a vehicle or object in front of the car. When another vehicle is found present in front of VIC, the object recognition algorithm draws a rectangular box around the image of that vehicle, measures the area of the box and locates the centre of the area. (Figure 18) This information is used later in the navigation algorithm for driving around or overtaking the object so that a collision is avoided and the VIC robot can continue driving around the vehicle or obstacle.

Figure 18 shows a snapshot of the TRAVIC software detecting another vehicle, identified by the black box. The software also calculated the position of the object's centre of area, marked with a centroidal black dot. The back part of the vehicle in front was covered by a yellow card box so that it would stand out from the black and white environment to make it easy to recognize.

The object detection was carried out by performing a scan over a specific area of interest searching for the colour yellow (shown shaded in Figure 19, ie. pixels with strong red and green colour data but with weak blue). The area of interest was divided into 16 rows and 36 columns of cells. Each of these cells included 25 pixels (5x5 rows and columns). When the majority of these pixels in one cell were filled with the colour yellow then the cell was said to be "positive". Figure 19 illustrates the colour-scanning scheme over the area of interest. It shows how the positive or yellow cells are marked with "1" (those with a majority of yellow pixels) and the rest are marked with "0" (those with a minority of yellow pixels). A rectangular box is drawn surrounding the positive "1" cells and the centre of this box is located mathematically.

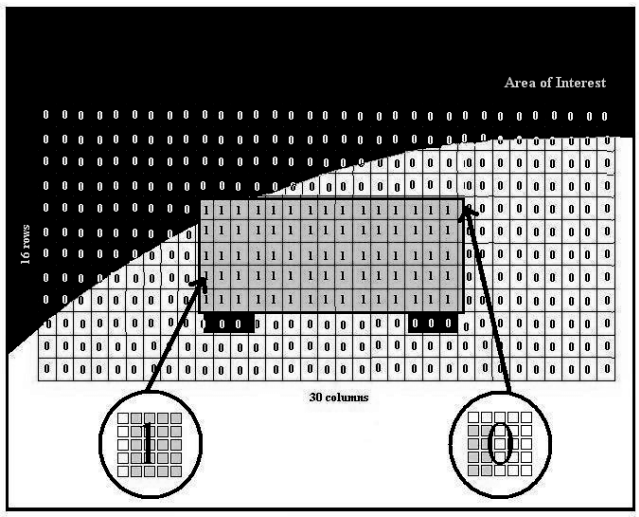


**Figure 19. Object scanning**

## 7 Artificial Neural Network implementation

The neural network in this design gave the TRAVIC control software the ability of learning so that it could grasp the concept of navigation and automatically drive and steer the VIC robot based on "experience" learned from training runs. The ANN is shown in Figure 20.

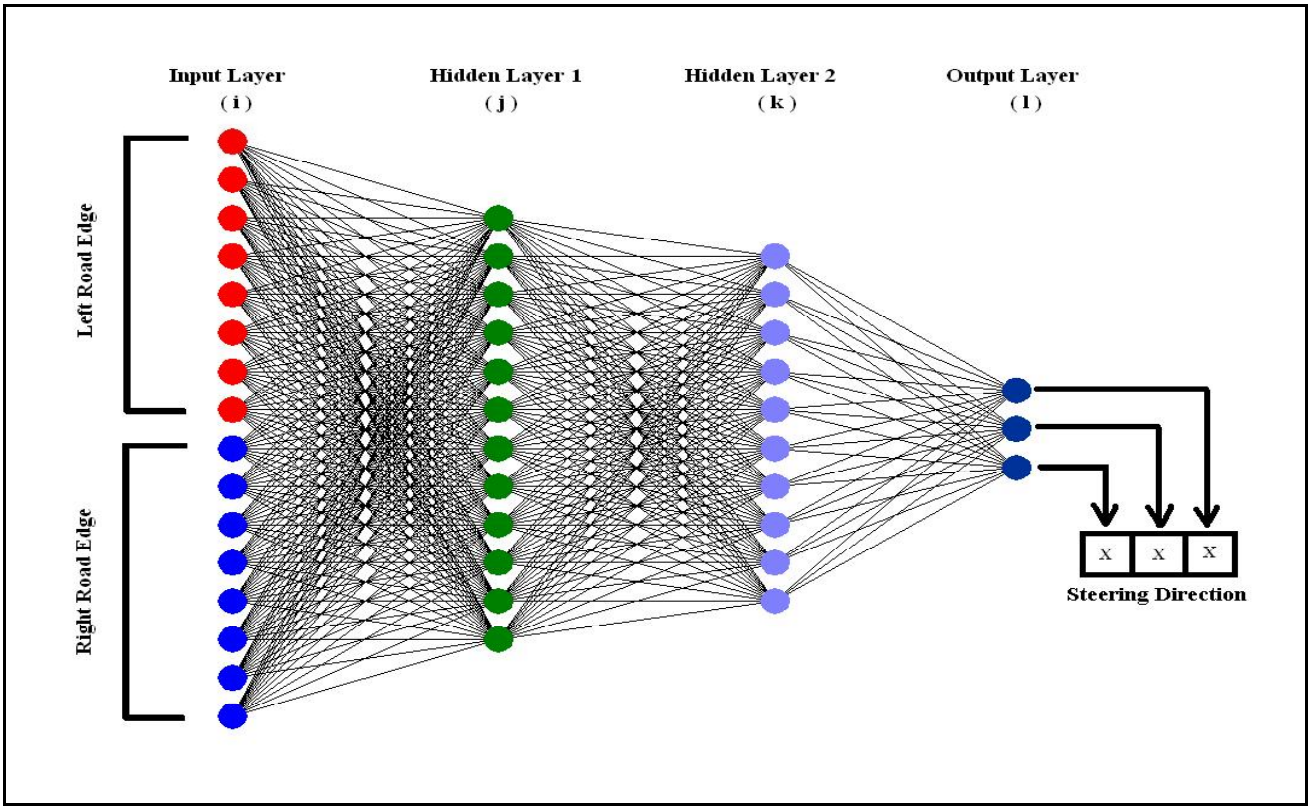


**Figure 20. Artificial Neural Network for the TRAVIC controller software**

## 7.1 Weight training for automatic steering

The network was trained with a set of recorded sample data so that it could deduce or predict the most suitable steering response by observing the left and right road edge positions. The neural network implemented in this design was a four layer feed forward network, trained with a back propagation algorithm.

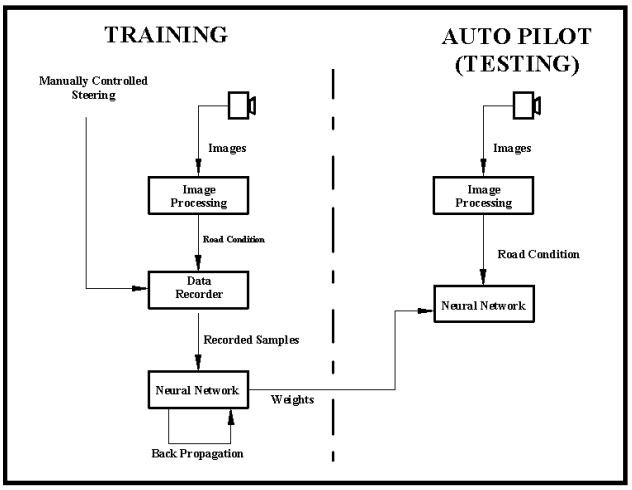


**Figure 21. Training the ANN for self-steering**

The input layer consists of sixteen input units and the output layer consists of three output units. The inputs to the neurons in the input layer came from the output of the road edge detection. The first eight units were sourced by the coordinates of the left edges and the other eight units were sourced by the coordinates of the right edges. Between the input and output layers, there are two hidden layers. The first hidden layer consists of 12 neurons and the second hidden layer consists of 10 neurons. The outputs of the three output neurons forms a binary number that represents a 3-bit steering direction (value 0 to 6).

Figure 21 shows the training method used in the design to teach the vehicle how to maintain its position on the track. One of the advantages of having separate steering and driving modules was that during the training, the driving motor could be kept off, which made the process easier. In the training stage, the vehicle was pushed around the track while the steering was controlled manually. The road edge coordinates and the manually selected steering angle were recorded into a file. The recorded samples were used later in the ANN weight training using back propagation. The samples provided the neural network with a set of possible inputs and the expected steering output. Therefore, the sample road edge data and steering data was used by the back propagation learning algorithm to force the neural network to adapt to

the trainer's steering methodology. The trained weights represented the system's memory for a particular mode of driving. This set of weights was used in the testing of autonomous control for VIC, where the network deduced the steering response from the processed data coming from live video rather than the recorded training video.

In the development, the teaching stage was divided into two stages: left lane and right lane training. The recorded samples from both training sessions were saved into two separate files. The neural network was trained twice. The first training session was for generating a set of weights that would keep the vehicle centred in the left lane of the track and the second one was for producing a set of weights that would keep the vehicle centred on the right lane of the track. The control software could change the set of weights used in the neural network whenever a lane transition or mode change was necessary.

## 7.2 Software implementation

Three separate programs were written in Visual Basic for the neural network. The first program was for recording the data sample, the second program was for training the neural network to adapt to the recorded samples, and the third one was for testing the trained network. Figure 22 shows the "Graphical User Interface" (GUI) for the data recording software.

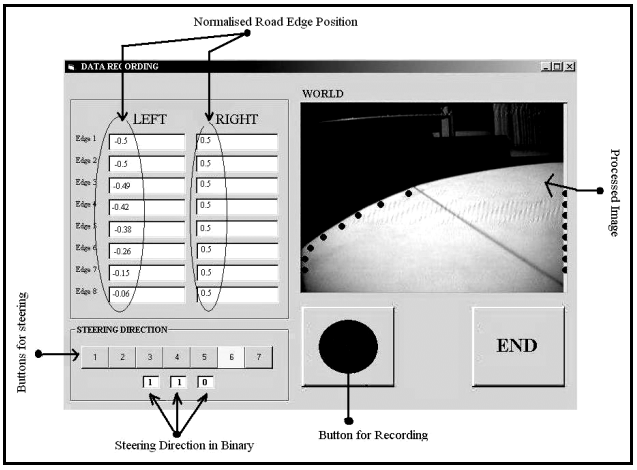


**Figure 22. Data recording software GUI**

In the recording stage, the steering direction was controlled manually, the drive motor was turned off and the vehicle was pushed slowly around the track. The GUI of the recording software is shown in Figure 21 and shows the post-processed video stream where the road edges are marked with red (left) and blue (right) edge dots. The normalised values of the dot coordinates are displayed to the left of the video window. The software also allows the user to manually control the steering direction while the car is being pushed around the track.

The record button is for saving the current values of the normalised road edge positions and the selected steering direction into a text file. The training samples were saved in the form of a two dimensional array of numbers and the dimension of the array was 19 x *n*, where *n* here represents the number of samples recorded during the training (typically 80-125) while 19 is allocated for 16 road edge positions and the 3-bit binary number representing the steering direction. The training samples described the way the trainer responded to different road conditions and this file was used for training the network so that it could imitate the trainer's responses to visual images of the track.

The training program adjusted the weight of each neuron connection in the network so that it could mimic the behaviour represented by the recorded samples. The software was designed for training a four layer neural network with up to 150 records of samples using the back propagation learning algorithm.

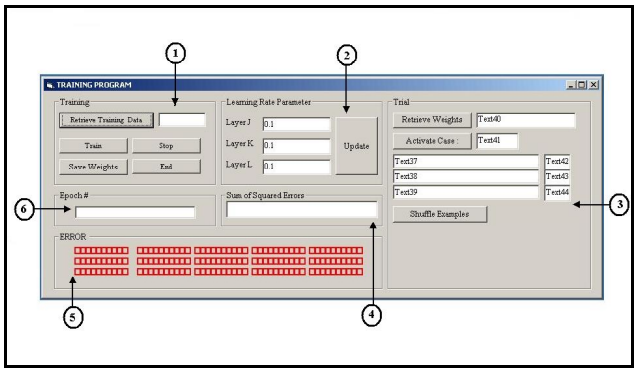


**Figure 23. Data recording software GUI**

Figure 23 shows the GUI of the training program. The user interface is comprised of 6 frames (marked with numbers in the figure). The buttons in the first frame are for retrieving recorded samples, starting and stopping the training session and saving the adjusted weights. The buttons in the Frame 2 allow the user to change the learning rate parameter of each layer during the training. In Frame 3, the user can test the performance of the neural network against a specific sample. The actual output recorded in the sample and the output generated by the neural network are displayed next to each other so the user can compare the estimated results to the desired output. Frame 4 displays the sum of squared errors that represents the network's overall aptitude. Frame 5 displays a table of blank red boxes. Each of these boxes represents a sample. The box is filled with the colour red when the network had managed to memorise the corresponding case, ie. the output of the network was close to the recorded output.

Hence, during the training the user can see how many cases had been memorised by the network. Frame 6 displays the number of iterations or epochs the network had so far completed.

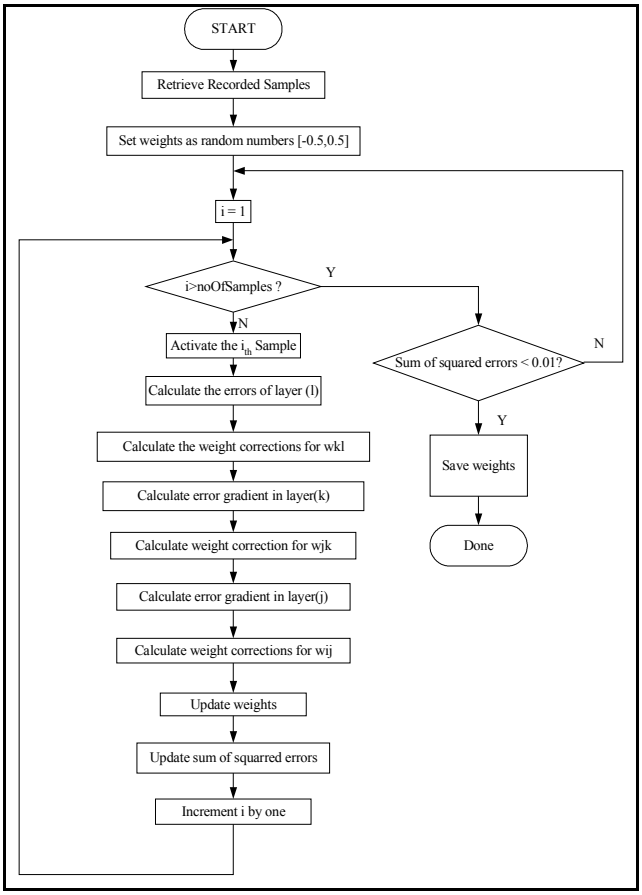


Figure 24. ANN weight-training algorithm

The flowchart representation of the training algorithm is shown in Figure 24. Initially, the training samples were retrieved from the text files and saved into an array. The weights of the network were set to random values within the range of [-0.5, 0.5]. One by one, the inputs from the samples were executed throughout the network. The outputs were then compared to the recorded outputs and the errors were calculated. As mentioned earlier in section 3, the errors are calculated and propagated backwards from the output layer to the input layers. During the propagation of the errors in each layer, the weight corrections were calculated based on the errors or error gradients of the corresponding layer. After all the weight corrections had been calculated, the weights were updated and the network repeated to execute the next sample with the adjusted weights. The errors of the output layer (l) of each sample were accumulated for calculating the sum of squared errors. When the network has gone through all the samples, the software is said to have finished one iteration or epoch.

The sum of squared errors indicates the network's level of understanding of the training samples. At the end of each epoch, this parameter is compared to a specific error criterion, usually a very small number. An error criterion of 0.01 was found to be adequate. The network must repeat the entire epoch until the sum of squared errors becomes less than this specified tolerance or error value. When this occurs, the network is said to have converged or adapted to the behaviour recorded in the training samples. The fully tuned weights are saved to another text file and the neural network is then ready to be used in the complete software (TRAVIC) for automatic steering.

The testing program was specifically written for examining the performance of the neural network which employed the trained weights. The user could load the tuned weights to the network and then observe the competency of the control software in automatically steering the vehicle around the track. Therefore, the user would be able to tell whether the neural network was ready to be implemented in the navigation system or still required more training.

The navigation algorithm is responsible for the manoeuvering decisions. Each decision is made based on the output from the object recognition and the road edge detection ANN weight training. Figure 25 illustrates the navigation algorithm in flowchart form.

Initially, the navigation was set to the default mode, which was the normal forward driving mode (vehicle cruising in the left lane at normal speed). The first two procedures to be performed are the road edge detection and the object recognition. The next thing to be done if the vehicle is still in normal mode is to check whether an object is present or not. If no object is detected, the vehicle should continue to cruise in normal mode. Once another vehicle or obstacle is detected, the distance between the vehicle and the front vehicle is checked. This is done by looking at the y coordinate of the object's centre of area on the screen. Once the value reaches what corresponds to a safe overtaking distance, the overtaking mode is adopted and an "overtaking timer interrupt procedure" (or interrupt routine) is activated. The neural network decides the steering direction based on the road edge detection results and the current steering mode. The speed is determined from the steering direction and the navigation mode as shown in Table 1.

The flowchart in Figure 25 describes the foreground process. In the background, two timer interrupt routines are used to organise the navigation mode of the vehicle. Visual Basic™ has a real-time interrupt facility that is able to generate a periodical interrupt request and the time interval between each request can be set from 1 ms up to 1 second. The two timer interrupt routines used were named

"Overtake Timer" and "Trans Timer" and each had a time variable ("overtake time" and "trans {transition} time") to set the duration of each routine. Time variables were set manually, but driving mode can be changed automatically based on changes in the image or available track space.

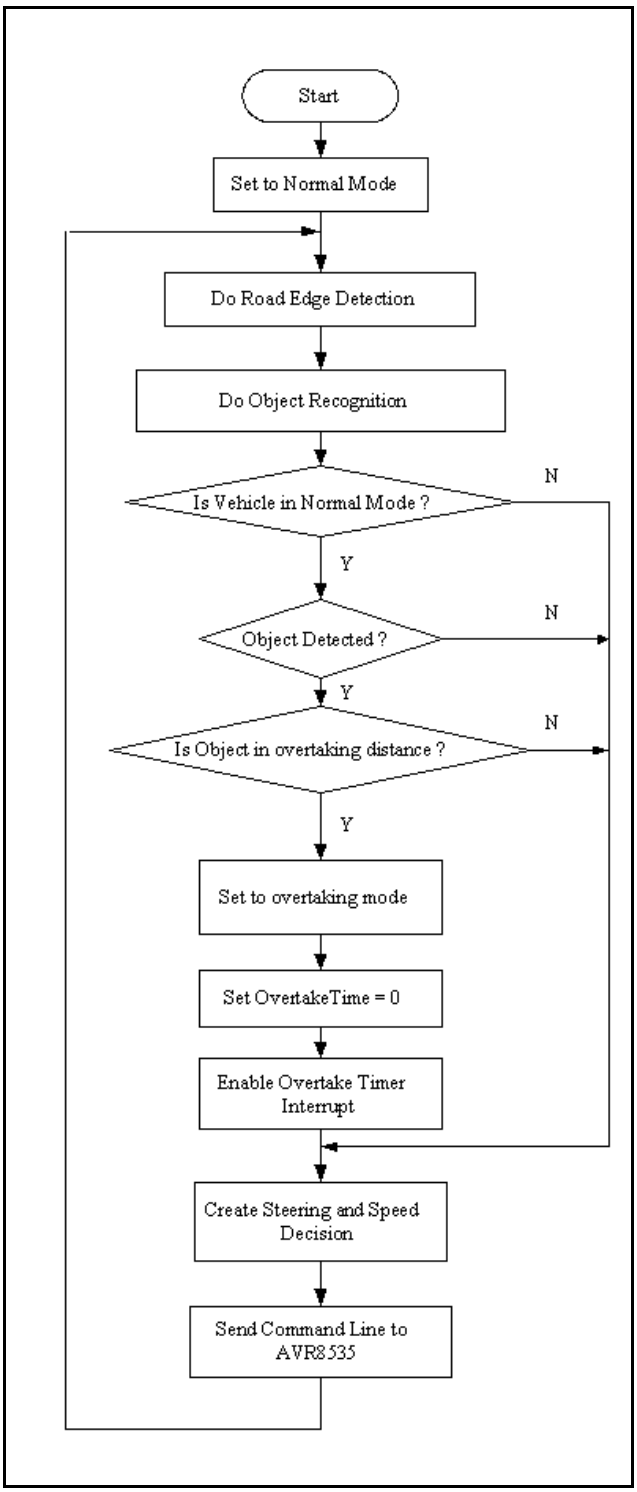


Figure 25. TRAVIC navigation algorithm

Both timers are set to generate an interrupt request every second when activated. As shown in Figure 25, the overtaking mode and the overtake timer interrupt are activated when the front vehicle is close enough to overtake. After a few seconds in the "overtaking mode", the navigation mode switches to "transitional mode" and the transitional timer interrupt routine is activated for a few seconds, during which the driving speed is lowered to adjust from the overtaking speed back to the normal speed in "normal mode". After this time elapses, TRAVIC is automatically set back to "normal mode" driving mode.

These changes in driving modes could have been based entirely on vision information, such as road edge obstacle position or vision road edges, however, this would require some manual control programming whereby TRAVIC would have to pass control from the ANN-directed navigation algorithm in Figure 25 to different procedural routines which handle the "overtaking" and "transitional" driving modes without using rigid or manually set time limits for each of these two modes. ie. mode changing is done automatically and each mode is given as much time as it needs to execute, based on relative velocity between VIC and the object, object proximity and available visible lateral width of the track ahead. After these modes are completed, control can then be passed back to the ANN to control the "normal driving" mode. Unfortunately, this highly flexible automatic method of mode changing was not implemented due to time constraints, however, it will be implemented in future to make the VIC more adaptable to its environment.

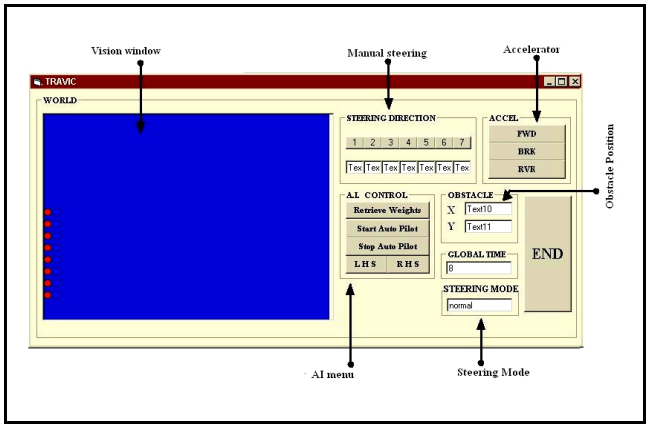


Figure 26. Control software GUI

Figure 26 shows the GUI of TRAVIC. The interface was designed so that the user can view the video image and choose between using autopilot navigation or manual control of the vehicle's movements. The user interface also displays the "centre of area" coordinates for an object in front, when detected, and the steering mode of the vehicle (ie. normal, overtake and transition modes).

## 8 Results

The "Vision-guided Intelligent Car" (VIC) robot described in this paper was able to successfully perform automatic steering and speed control and able to learn how to drive consistently in the left hand (outer) or right hand (inner) lane of the closed loop test track, like NAVLab on a small scale. When presented with an obstacle (such as a yellow box), VIC was able to identify it and perform obstacle avoidance or an "overtaking" procedure, in order to drive around it automatically.

## 9 Future work

The VIC robot, at present, is only suitable for driving on tracks, like that shown in Figure 11, and its TRAVIC control software is unable to navigate on outdoor, off-road terrain or on track surfaces that lack clearly defined road edges. It is hoped that the VIC robot will be programmed for driving in these types of unstructured environments in the future. It could be used for applications such as search and rescue or automatic mine detection and clearing.

As mentioned in Section 7.2, the control software can be improved to implement automatic mode changing without having to use inflexible, preprogrammed timers which do not take into consideration the relative speed between the obstacle ahead and the VIC robot.

A minor problem that was encountered in this project was unreliable wireless serial communications between the PC and the MCU (AVR microcontroller), hence a wire tether was used. Long range and high speed TCP/IP communications transmitted using "WiFi" hardware (based on the IEEE 802.11 "b" or "g" standard) will also be attempted to give a communications range of up to 400 metres from the remote human operator at the base station.

Additional cameras could be used around the vehicle (eg. Pointing left, right and behind) to acquire more vision information for the control software to analyse. Extra analysis would greatly improve the ability of the control software to avoid collisions and perhaps even implement "defensive driving" manoeuvres in the event that other cars on the road perform dangerous movements which could lead to a crash. These types of control systems may even assist drivers of motor vehicles, helping them to escape from imminent danger or to avoid collisions.

## 10 Conclusion

An operational vision guided robot car that can "learn how to drive" was described in detail in this "case study" report. It is hoped that this work will inspire others to develop similar types of intelligent vehicles for practical applications, entertainment, education and even future competitions between mechatronic engineering students.

## References


1. S N Cubero, J Layanto, M Goode: Autonomous Racing Car Competition for Mechatronics Engineering Education, Proc 10th MMVIP Perth, Dec 2003, pp 9-16. Research Studies Press ISBN: 0-86380-290-7

2. D A Pomerleau: ALVINN: an autonomous land vehicle in a neural network. Technical Report CMU-CS-89-107, Computer Science Dept. Carnegie Mellon University, Pittsburgh PA, 1989.

3. T Kohonen : An Introduction to Neural Computing, Neural Networks, vol. 1, no. 1, pp 3#16, 1988.

4. M Negnevitsky: Artificial Intelligence: A Guide to Intelligent Systems. 1st edition. Essex, Pearson Education Ltd, 2002.

5. Tamiya America, Inc. 2004: Tamiya R/C Car Chassis Lineup : M-Chassis, F1, OFF-Road Car, [Online], URL www.tamiya.com/english/rc/beginner/chassis2.htm

6. Shenzhen Lianyida Science Co., Ltd. 2004: Wireless Transmitter and Receiver,[Online], URL: http:// lianyd.en.alibaba.com/product/50021575/50117576/Wireless_Transmitter _And_Receiver/Wireless_Transmitter_And_Receiver.html

7. Atmel Co. 2004: AT90S8535 Datasheet. http://www. atmel.com/dyn/resources/prod_documents/DOC1041.PDF