Four Point Calibration and a Comparison of Optical Modeling and Neural Networks for Robot Guidance

A thesis submitted to the
Division of Graduate Studies and Research
Of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of Mechanical, Industrial and Nuclear Engineering
of the College of Engineering

1999

by

Sameer S. Parasnis

BS, (Mechanical Engineering), Govt. College of Engineering, Pune, India, 1997.

Committee Chair: Dr. Ernest L. Hall

Abstract

A truly autonomous robot must be able to sense its environment and react appropriately. This issue attains greater importance in a variable environment. The University of Cincinnati Robot team is actively involved in building a small, unmanned,autonomously guided vehicle for the International Ground Robotics Contest organized by Association for Unmanned Vehicle Systems International (AUVSI) each year. The unmanned vehicle is supposed to follow an obstacle course bounded by two white/yellow lines, which are four inches thick and 10 feet apart. The navigation system for one of University of Cincinnati's designs,Bearcat II used 2 CCD cameras and an image tracking device for the front end processing of the image captured by the cameras. The three dimensional world co-ordinates were reduced to two dimensional image coordinates as a result of the transformations taking place from the ground plane to the image plane. A novel four-point calibration system was designed to transform the image co-ordinates back to world co-ordinates for navigation purposes using fuzzy logic. This transformation is the main focus of this study. The calibration algorithm and the vision control algorithm were coded in C++. Using these algorithms, the robot built with this design was able to track and follow the lines successfully as proven at the 7th International Ground Robotics Competition held on June 7, 1999, at Oakland University, Michigan .This research goes one step further in exploring other alternatives for modeling vision systems for the mobile robot. Recent techniques involve the use of artificial neural networks to process sensor data for mobile robot guidance. A comparison of a fuzzy logic controller for an Autonomous guided vehicle and neural network perception is also described in this work. The fuzzy logic controller uses both vision and obstacle information and provides

the steering and speed controls to the robot. A feed-forward neural network is described to guide the robot using vision and range data. A comprehensive comparison of these two systems, with certain criteria such as input-output relationship,preprocessing required,complexity of algorithms, is made. The Matlab simulation shows very promising results. More onsite research is required in order to implement neural networks on the exiting design. The significance of this work is that it provides comparative tradeoffs on two important robot guidance methods.

Table of Contents

List of Figures/Illustrations

Chapter 1

INTRODUCTION

1.1 Motivation

Mobile robots have come of age. Autonomous road vehicles, guided by computer vision systems, are a topic of research in numerous places in the world. For effective functioning the robot must be capable of purposeful movement in the environment .A truly autonomous robot must sense its environment and react appropriately. These issues attain greater importance in an outdoor, variable environment.

The rise in popularity of the single chip microcomputer and the drastic reductions in size and cost of integrated circuits in recent years have opened up huge new opportunities for intelligent systems and robotics. Building an unmanned ground robotic vehicle is a challenging task. A robot can be thought of as an intelligent connection of perception to action. Implementing this is a formidable task and might take on a wide variety of disciplines, ranging from mechanical logic to microprocessor control to networks of neuron-like gates. Mobile robots pose a unique challenge to artificial intelligence researchers. They are inherently autonomous and they force us to deal with key issues such as uncertainty, reliability and real time response. They also require an integration of mechanical strength, reliable control systems, and sensors for vision and obstacle avoidance. Navigation and mapping are crucial to all robotic systems and are an integral part of autonomous mobile robots. [1,2,3]

While it is possible for a robot to be mobile and not do mapping and navigation, sophisticated tasks require that a mobile robot build maps and use them to move around. Levitt and Lawton [4] (1990) pose three basic questions that define mobile robot mapping and navigation:

- Where am I?
- How do I get to other places from here?
- Where are the other places relative to me?

Humans receive a large amount of their information through the human vision system, which enables them to adapt quickly to changes in their environment. The Center for Robotics Research at the University of Cincinnati has been involved in a worldwide competition to build a small-unmanned autonomous ground vehicle that can navigate around an outdoor obstacle course. Vision-based mobile robot guidance has proven difficult for classical machine vision methods because of the diversity and real time constraints inherent in the task. Vision for motion control must always be real time vision. In this context of unmanned guided vehicles, the vision system must enable the robot to perceive changes in its environment while they are occurring and soon enough to react to these changes and make decisions accordingly. For mobile systems, a speed of reaction similar to human beings is desirable. For this, a robot vision system should not introduce a delay of more than 100 ms in reporting an event in the environment or in providing data for some visible motion. [5]

There are several factors which affect the functioning of the outdoor autonomous systems. [6]

Some of them are:

- Variations of road type
- Appearance variations due to lighting and weather conditions
- Real time processing constraints
- High level reasoning constraints

A general autonomous vehicle should be capable of driving on a variety of road surfaces like grass, concrete, sand, boards etc. The vehicle should function equally well inside, i.e., on a plane surface as well as outside on a varying terrain.

The second factor making autonomous driving difficult is the variation in appearance that results from environmental factors. Lighting changes and deep shadows make it difficult for perception systems to pick up important and desired features during daytime driving.

The threshold of the perception system has to be adjusted in such a way that the desired features are identified correctly. Any change to the light affects the threshold and performance of the system. Also to be considered is the fact that missing or obscured lane markers make driving difficult for an autonomous system even under favorable lighting conditions.

Adequate computer hardware is a key to practical robot vision. General-purpose computers are definitely not adequate. Multi-processor systems containing a small number of processing elements, each of them based on a standard microprocessor of moderate performance have been demonstrated to outperform much more expensive computer systems in robot vision applications. Flexibility is a very important factor, including the flexibility of random access pixel data by the processing elements, and

flexibility in dynamically concentrating the computing power of the system on those parts of an image containing the most relevant information at any moment. The system should also be flexible in restructuring under software control to match the inherent structure of the vision task. [7,8]

There is always a limited amount of time for processing sensor information. The speed of the front end processing system should be such that the vehicle reacts very quickly to the changes in the environment. For example at 5 miles per hour a vehicle is traveling nearly 7.5 feet per second. A lot can happen in 7.5 feet, like losing track of the lane or straying a significant distance from the lane or colliding with an obstacle if the system does not react accurately or act quickly enough. [7]

1.2 Objective

The objective of this study was to design the vision system for an autonomous guided vehicle (Bearcat II) built for the Association for Unmanned Vehicle Systems (AUVS) 1999 International Ground Robotics Competition. This robot has several unique features such as the Zero Turning Radius (ZTR) feature. The competition was held on June 7 1999. This thesis describes the design, development and exploratory research on the vision system for the autonomous guided vehicle Bearcat II .The three dimensional (3-D) vision system makes use of 2 CCD cameras and an image-tracking device for the front end processing of the image captured by the camera. The camera reduces the three dimensional world co-ordinate system into two dimensional image co-ordinate system. After getting the information regarding image co-ordinates, at any time, the challenge is to extract three-dimensional information from them. A mathematical as well as geometrical transformation occurs via the camera parameters in transforming a 3-D

coordinate system to a 2-D system. If these mathematical and geometrical relations are known, a 3-D coordinate point on a line can be autonomously determined from its corresponding 2-D image point. To establish these mathematical and geometrical relationships, the camera has to be calibrated. The calibration of the camera of the vision system is the main task of this project. This is because if the vision system is well calibrated, accurate measurements of the coordinates of the points on the line with respect to the robot can be made. From these measurements, the orientation of the line with respect to the robot can be computed. With these computations, the next task is to guide the robot.

The motion control of the AGV designed has the capability of turning about the center of its drive axis, which is called the zero turning radius feature. It is gaining popularity and expanding commercially in the U.S. mowing market. This feature provides exceptional maneuverability and can make sharp turns possible with relatively greater ease than those without the ZTR feature. Rotating one wheel forward and the other wheel backward generally accomplishes the ZTR function. However in our design we instead vary the speeds of the left and right drive wheels while negotiating a curve. This enables the AGV to make a curved turning path parallel to the track lines.

Specific goals of the thesis:

1) Design of a novel 4-point calibration method for the vision system for the 7th AUVS competition.
2) Use of fuzzy logic and neural networks for the vision control system for the unmanned guided vehicle.

3) Comparison of the optical modeling and neural networks' approach for the guidance methods for the vehicle.

1.3 System Design

The robot base is constructed from an 80/20 aluminum Industrial Erector Set. The AGV is driven and steered by two independent 24 volt, 12 amp motors. These motors drive the left and right drive wheels respectively through two independent gearboxes, which increase the motor torque by about forty times. The power to each individual motor is delivered from a amplifier that amplifies the signal from the Galil DMC motion controller. To complete the control loops a position encoder is attached on the shaft of each of the drive motors. The encoder position signal is numerically differentiated to provide velocity feedback signal. There is a castor wheel in the rear of the vehicle, which is free to swing when the vehicle has to negotiate a turn.

Recent techniques involve the use of artificial neural networks to process sensor data for mobile robot guidance. A comparison of a fuzzy logic control for an AGV and a neural network perception is also described in this work. The fuzzy logic controller uses both vision and obstacle information and provides the steering and speed controls to the robot. A feed-forward neural network is described to guide the robot using vision and range data. The significance of this work is that it provides comparative tradeoffs on two important robot guidance methods.

1. 4 Outline of the thesis

In Chapter 2, a brief survey of the background and previous research is given. The main focus is on the vision system of the vehicle and the significance of the calibration is elaborated in Chapter 3. The 4-point calibration method is explained in detail and its

relative advantages over the previous method are given. Chapter 4 explains the mechanism of the fuzzy logic controller. The recent use of neural networks for mobile robot guidance and their use in this project are explained in Chapter 5. Chapter 6 gives an comparison between the two methods. The details and an account of the 7th Annual IGRC competition are given in Chapter 7. Conclusions and recommendations for future work are described in Chapter 8.

Chapter 2

Background and Previous Research

Calibration of a camera means determining the geometric properties of the imaging process i.e. the transformation that maps a 3-D point, expressed with respect to a reference frame onto its 2-D image whose co-ordinates are expressed in pixel units. This problem has been a major issue in photogrammetry and computer vision for years. The main reason for such interest is that the knowledge of the imaging parameters allows one to relate the image measurements to the spatial structure of the observed scene. [11]

The fundamental theorem of robot vision says that manipulation of a point in space $\mathbf{x_1}$ by either a robot manipulator that moves it to another point $\mathbf{x_2}$ or through a camera system that images the point onto a camera sensor at $\mathbf{x_2}$, is described by the a matrix transformation, which is of the form

$$\mathbf{x_2 = Tx_1}$$

The transformation matrix **T** describes the first-order effects of translation, rotation, scaling, and projective and perspective projections. Camera calibration is a complex problem because of the following problems [13]:

(1) Calibration of internal parameters of a camera, the so-called intrinsic parameters, including the optical and mechanical (geometrical) properties of the camera, such as focal length, lens distortion parameters, the intersection point of the optical axis with the image plane etc. Sometimes the manufacturers supply these parameters but they are usually not accurate enough for computations. Some of them such as focal length

vary with adjustments, while some of them such as the lens center are calibrated once and for all depending upon the optical stability of the camera.

(2) Estimation of the location of the camera (system) relative to the 3-D world reference system, including rotation and translation between these two systems is required. These are called extrinsic parameters. These parameters are not directly related to the camera itself, but the set up of a camera, which means they have to be calibrated at each set up.

Robert[13] in his paper "Camera Calibration without Feature Extraction" has presented an approach to this problem using a calibration pattern. The approach is different from the classical calibration techniques, which involve extraction of image features and computation of camera coefficients. A classical iterative technique is used to search for the camera parameters that best project 3-D points of a calibration pattern.

Li and Lavest[14] have thrown light on some aspects of zoom lens camera calibration. A lot of care has to be taken in the electronic stability of the camera and frame grabber, and the way calibration points are measured and detected in images. In this paper they have addressed some practical aspects of camera calibration, in particular, of a zoom lens system. Through a systematic approach they describe all the keys points that have to be checked in order to obtain accurate calibration results.

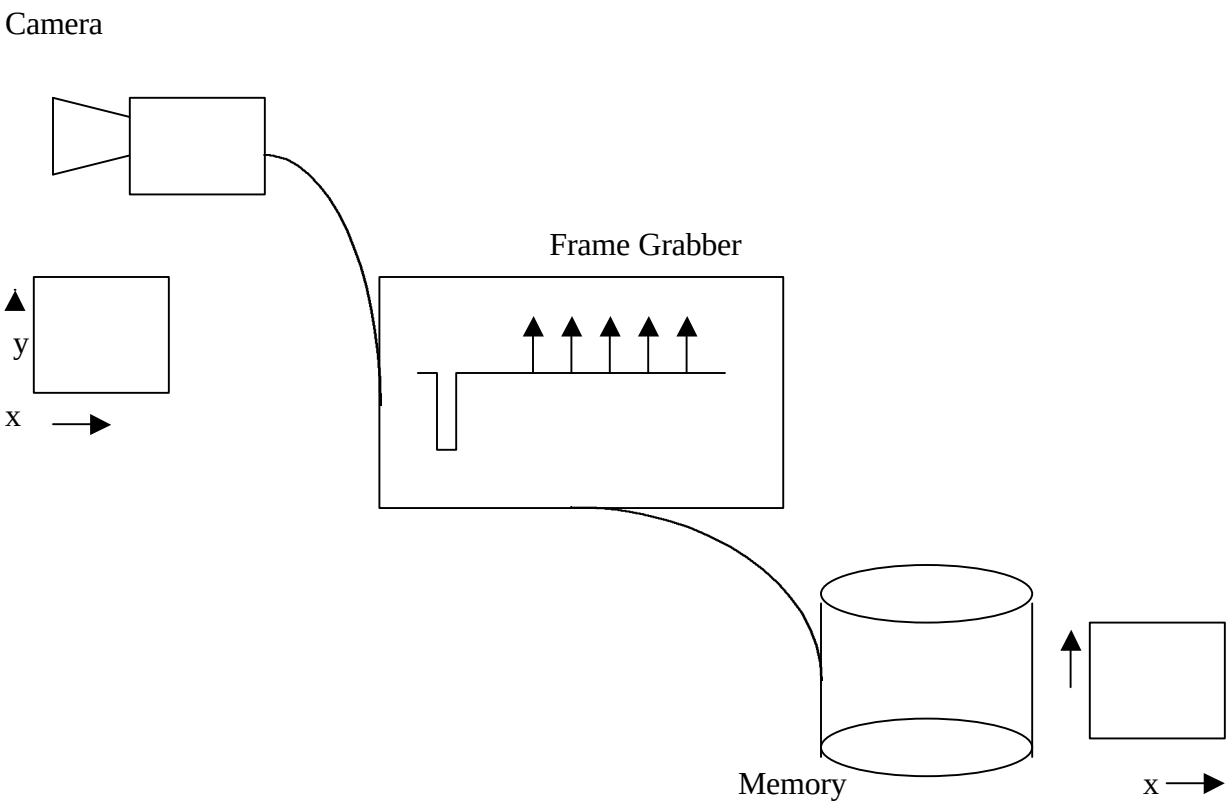


Figure 1. Image Grabbing Scheme.[13]

Weng,Cohen, and Herniou [14,15] use Weng's camera model based on the pinhole model of perspective projection considering radial and tangential distortions.

A lot of caution is required during calibration. Hong, et al [16] list two points that should be considered in camera calibration: 1. reducing the location error of image features as far as possible, by exploiting image processing technique, 2. compensating system error by the optimal pattern of approximating residual error of image points, namely the posterior compensation of the system error. Based on these two points, the calibration process discussed in [16] are of three parts: (1) The direct transformation error approximation camera calibration algorithm; (2) the subpixel image feature location

algorithm combined with the 3D control point field delicate design and fabrication; (3) The precisely movable stage, which provides the reliable means of accuracy checking.

Tsai [17] presented an algorithm that decomposes a solution for 12 transformational parameters (nine for rotation and three for translation) into multiple stages by introducing a radial alignment constraint. The radial alignment constraint assumes that the lens distortion occurs only in the radial direction from the optical axis Z of the camera. Using this constraint, six parameters are computed first, and the constraint of the rigid body transformation is used to compute five other parameters. The remaining parameters are computed by radial lens distortion parameter and estimating it by a nonlinear optimization procedure.

Faugeras [18] et. al in "Analysis of a Sequence of Stereo Scenes Containing Multiple Moving Objects Using Rigidity Constraints "describe a method for computing the movement of objects as well as that of a mobile robot from a sequence of stereo frames. Stereo frames are obtained at different instants by a stereo rig, when the mobile robot navigates in an unknown environment possibly containing some moving rigid objects.

Zhang and Faugeras[19] present a method for estimating 3D displacements from two stereo frames. It is based upon the hypothesize-and-verify paradigm, which is used to match 3D line segments between the two frames. In order to reduce the complexity of the algorithm, objects are assumed to be rigid. In the experimental sections, the algorithm is shown to work on indoor and natural scenes.

"A 3D World Model Builder with a Mobile Robot "[20]- An article written by the same authors describes a system to incrementally build a world model with a mobile

robot in an unknown environment. The model is segment-based. A trinocular stereo system is used to build a local map about the environment. A global map is obtained by integrating a sequence of stereo frames taken when the robot navigates in the environment.

Luong, et al [21,22]in their paper " Motion of an Uncalibrated Stereo Rig: Self-Calibration and Metric Reconstruction " address the problem of self-calibration and metric reconstruction (up to a scale) from one unknown motion of an Uncalibrated stereo rig, assuming the coordinates of the principal point of each camera are known.

They also present a novel technique for calibrating a binocular stereo rig by using the information from both scenes and classical calibration objects. The calibration provided by the classical methods is only valid for the space near the position of the calibration object. Their technique takes the advantage of the rigidity of the geometry between two cameras. The idea is to first estimate precisely the epipolar geometry, which is valid for a wide range in space from all available matches.

During the execution of a task the vision-system is subject to external influences such as vibrations, thermal expansion etc. which affect and possibly render invalid the initial calibration. Moreover, it is possible that the parameters of the vision-system such as the zoom or the focus are altered intentionally in order to perform specific vision-tasks.

“Self-Maintaining Camera Calibration Over Time “ by Schenk et al [23] describes a technique for automatically maintaining calibration of stereovision systems over time without using again any particular calibration apparatus.

Worrall, Sullivan and Baker [24] in the paper “A simple, intuitive camera calibration tool for natural images” present an interactive tool for calibrating a camera, suitable for use in outdoor scenes. The motivation for the tool was the need to obtain an approximate calibration for images taken with no explicit calibration data. The method decomposes the calibration parameters into intuitively simple components, and relies on the operator interactively adjusting the parameter settings to achieve a visually acceptable agreement between a rectilinear calibration model and his own perception of the scene.

Most of the previous research deals with the theoretical aspects of zoom lens camera calibration. The intrinsic parameters are obtained by building a pinhole camera model. The research here deals with designing a calibration algorithm keeping in mind the significant practical aspects.

Chapter 3

The calibration system

3.1 Significance

The objective of the vision system is to make the robot follow a line using a camera. In order to obtain accurate information about the position of the line with respect to the centroid of the robot, the distance and the angle of the line with respect to the centroid of the robot has to be known. The camera system reduces the 3-D information about the obstacle course into 2-D image co-ordinates. In order to obtain a relationship between the two co-ordinate systems, the camera needs to be calibrated.

Camera calibration is a process to determine the relationship between a given 3-D coordinate system (world coordinates) and the 2-D image plane a camera perceives (image coordinates). More specifically, it is to determine the camera and lens model parameters that govern the mathematical or geometrical transformation from world coordinates to image coordinates based on the known 3-D control field and its image. The CCD camera maps the line from the 3-D coordinate system to the 2-D image system. Since the process is autonomous, the relationship between the 2-D system and the 3-D system has to be accurately determined so that the robot can be appropriately controlled to follow the line. The objective of this section is to explain the entire calibration process and its significance in this project.

Camera calibration is considered very important in many computer vision problems. Camera calibration in the context of three-dimensional machine vision is the process of determining the internal camera geometric and optical characteristics (intrinsic parameters) and/or the 3-D position and orientation of the camera frame relative to a certain world coordinate system (extrinsic parameters). Camera projection is often modeled with a simple pinhole camera model. In reality, the camera is a much more complicated device, and if it is used as a measurement instrument, a proper calibration procedure should be performed.

In order to follow a track, which is separated by two lines, which are 10 ft apart, 2 CCD cameras and an image-tracking device (ISCAN) are used. The ISCAN image tracking system finds the centroid of the darkest or the brightest region in an image and returns the co-ordinates of these points. These are the image co-ordinates. These co-ordinates are two-dimensional while the real world co-ordinates are three-dimensional. An algorithm is developed to establish a mathematical and geometrical relationship between the physical three-dimensional (3-D) and its corresponding digitized two-dimensional (2-D) co-ordinates. In an autonomous situation the challenge is to determine 3-D co-ordinates given the image co-ordinates. This is established by what is popularly known as “Calibration” of the camera. The objective is to find any corresponding ground co-ordinate given an image co-ordinate. What makes this the most important and crucial task is that the process of following the line is autonomous and dynamic and hence the relationship between these co-ordinates should be accurately determined. This, in turn, determines how closely the robot follows the line and hence the success of the robot.

The objective was to design a calibration method, which was not only accurate but also is easy and less time consuming. Some calibration methods are very accurate but are extremely time consuming. To understand the calibration process of the cameras for the mobile robot, it is important to understand the robot geometry as well as the geometrical as well as mathematical transformations that take place when the 3-D co-ordinates are reduced to 2-D co-ordinates.

The first and most important thing is to set up the co-ordinate system for the robot. A point in global coordinates is related to the corresponding point in image coordinates using exactly the same transformations that are encountered in the physical system. The global coordinates are first translated to the center of the image plane, then rotated counter clockwise about the x-axis by the tilt angle, θ, then rotated counter clockwise about the z-axis by the pan angle, ϕ, then mapped through a perspective transformation with lens center located at $y=y_0$, and finally projected onto the x-z plane. These parameters are difficult to measure practically,an approach where all the parameters are embedded in the camera coefficients is required. This approach is described in the next few paragraphs and the camera coefficients are calculated using the calibration procedure. [11]

3.2 Direct Coefficients Computation Approach

The vision system was modeled by the following equations.

$$X_{PI} = A_{11}\, x_g + A_{12} y_g + A_{13} z_g + A_{14} \quad (2)$$

$$Y_{PI} = A_{21} x_g + A_{22} y_g + A_{23} z_g + A_{24} \quad (3)$$

where A_{nm} are coefficients, X_{PI} and Y_{PI} are x and y image coordinates, and x_g, y_g, and z_g are the ground coordinates. In transforming the ground coordinate points to the image coordinate points the following transformation operations occur on the points: scaling, translation, rotation, perspective, and projective. Solving for the transformation parameters to obtain the image and ground coordinate relationship is a difficult task. Fortunately, in the model equations given above, the transformation parameters are imbedded into the coefficients.

3.3 The Four point calibration:

Brief History:

During the initial research for the calibration many alternatives were explored. A mathematical model was built using the homogeneous matrix transformation. The homogeneous matrix transformation approach maps the physical coordinates to the image coordinates using matrix transformation. A point in global coordinates is related to the corresponding point in image coordinates using exactly the same transformations that are encountered in the physical system. This approach is theoretically perfect, but has practical difficulties. In this approach, system parameters such as the focal length, the pan angle and the tilt angle are used.

In the linear model, there are two equations and eight unknowns. Each point corresponds to two unknowns. So in order to get eight unknowns from two equations, we need four points. This is the basic principle of the four-point calibration.

To compute the coefficients, a calibration device was constructed to obtain four data points. With the four points, a matrix equation was yielded as shown below :

$$X_{PI} = CA_{1k} \quad (4)$$

$$Y_{PI} = CA_{2k} \quad (5)$$

where

$$C = \begin{bmatrix} xg_1 & yg_1 & zg_1 & 1 \\ xg_2 & yg_2 & zg_3 & 1 \\ xg_3 & yg_3 & zg_3 & 1 \\ xg_4 & yg_4 & zg_4 & 1 \end{bmatrix}; XPI = \begin{bmatrix} X_{PI1} \\ X_{PI2} \\ X_{PI3} \\ X_{PI4} \end{bmatrix}; YPI = \begin{bmatrix} Y_{PI1} \\ Y_{PI2} \\ Y_{PI3} \\ Y_{PI4} \end{bmatrix}; A_{1k} = \begin{bmatrix} A_{11} \\ A_{12} \\ A_{13} \\ A_{14} \end{bmatrix}; A_{2k} = \begin{bmatrix} A_{21} \\ A_{22} \\ A_{23} \\ A_{24} \end{bmatrix}$$

and k = 1,…4.

Equations (4) and (5) consist of 8 linearly independent equations and four unknowns, the least-square regression method is applied to yield a minimum mean-square error solution for the coefficients. Below are the equations for the solution:

$$A_{1K} = (C^T C)^{-1} C^T X_{PI} \quad (6)$$

$$A_{2K} = (C^T C)^{-1} C^T Y_{PI} \quad (7)$$

Given an image coordinate xPI and yPI, and z ground coordinate (the z coordinate of the points with respect to the centroid of the robot is maintained constant because of the ground constraint) the corresponding x_g and y_g ground coordinates are computed as indicated by the following matrix equations.

$$\begin{pmatrix} x_g \\ y_g \end{pmatrix} = Q^{-1}\mathrm{B} \qquad (8)$$

where

$$\mathrm{Q} = \begin{bmatrix} \mathrm{A}_{11} & \mathrm{A}_{12} \\ \mathrm{A}_{21} & \mathrm{A}_{22} \end{bmatrix}; \mathrm{B} = \begin{bmatrix} \mathrm{X}_{\mathrm{PI1}} - \mathrm{A}_{14} - (\mathrm{A}_{13}\mathrm{z}_{\mathrm{g}}) \\ \mathrm{Y}_{\mathrm{PI1}} - \mathrm{A}_{24} - (\mathrm{A}_{23}\mathrm{z}_{\mathrm{g}}) \end{bmatrix}$$

The experimental procedure and use of the newly written C++ program is described in the further subsections. The C++ program is listed in appendix A.

3.4 Experimental Methods

A calibration device was constructed to permit measurement of corresponding three dimensional object points and image points. The determination of the camera focal lengths and the orientation of the projection system (system identification) with respect to the global coordinates system can be obtained by several methods as described in the introduction. In this study, a calibration device was constructed to accomplish system identification.

3.5 Calibration Device

The calibration device is shown in Figure 3.1. It consists of 2 metallic plates which are 10'' x 6'' x 0.5''. The plates are very precisely machined so as to be used on an optical bench. The two plates are separated by four steel shafts, which are 1'' in diameter and 5'' in length. The shafts are surface ground and are securely fixed to the two plates. The metallic plates are coated with a black anodized material to ensure no change in dimension and minimum reflectance. The upper plate has a grid of holes separated by 1''. The main advantage of using a metallic device as against a wooden device, which was previously used, is the dimensional accuracy.

Four precision 1'' diameter aluminum spheres are used as the test points. Two of the spheres are placed on the same plane (the surface) of the upper metallic plate while the other two spheres are supported by the screws from underneath which are pinned to the upper metallic plate upside down. They are raised in height by using a bushing, which is about 0.5625 '' in length. This ensures that all the spheres do not lie on the same plane. Starting from the surface of the base, the balls are given alphabetic labels in a clockwise direction. Each of the corners of the base is numerically labeled, also in a clockwise direction, with the corner #1 .

Accurate measurements of the exact coordinates of all the four points with respect to the reference point is an essential factor in the calibration process. To attain the needed high accuracy, a coordinate measuring machine, whose accuracy is ten thousandth of an inch. 0.0001'', was utilized to measure the centroid of the four balls with respect to the tip of corner of the lower metallic plate. To obtain the actual physical measurements of each of the four calibration points with respect to a reference point, in

this case the centroid of the robot, the X and Y distances between the tip of corner of the lower base and the centroid of the robot is carefully measured.

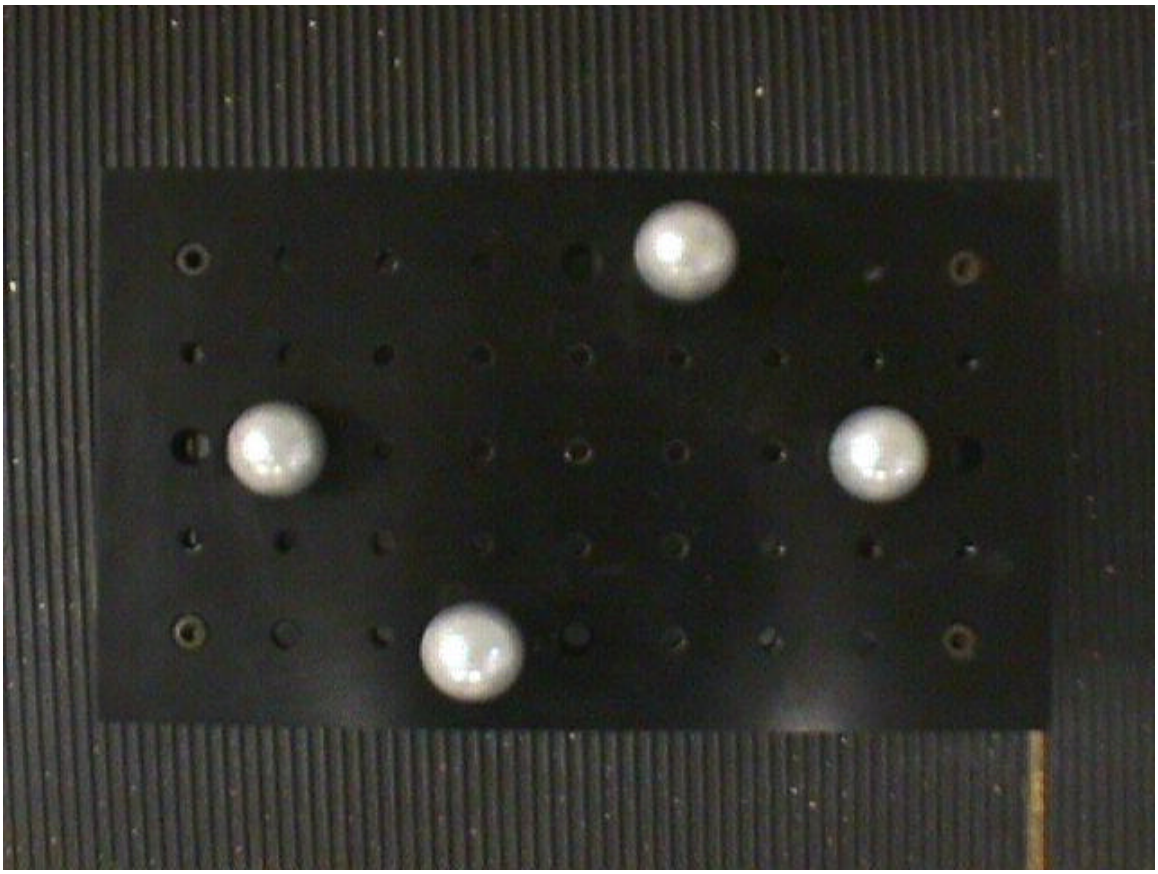


Figure 3.1 Top view of the calibration device

Each point of the calibration system now has an accurate physical coordinate reference to the centroid of the robot and their corresponding image coordinates are obtained via Iscan, the image-processing tool. From the physical and image coordinates, the camera parameters (coefficients) are computed.

The order of points is clockwise for both the left and right camera (starting from the point nearest to the camera). Both the cameras are calibrated independently of each other, but by using the same principle and method.

We know that the 2 white lines are 10 ft apart and 4 inches wide. So the robot center is 5 feet from the line. The most important part of the entire calibration process is to get an optimum vision window. The position of the camera should be such that it gives a full view of the line and the surrounding area. The pan angle and the tilt angle of the cameras are to be adjusted in such a way that the line is approximately in the center of the image screen. The main reason for doing this is, at that position, the line is at an angle of zero degrees with respect to the robot. The vision algorithm is designed in such a way that the robot should always be parallel to the lines and in the center of the track.

Once the line is set parallel to the robot, the next task is to set the window limits on either side of the line. At any given time, only one camera is functional. When the robot is run in its auto-mode, two I-Scan windows are formed at the top and bottom of the image screen and they flicker back and forth on the line as shown in Figure 3.2. The window size can be adjusted through the C++ program. The X and Y image co-ordinates of these windows are sent to the program and using the calibration coefficients, their real X and Y co-ordinates are calculated. These values are fed to the vision control program to get the actual distance and the angle of the line from the robot, as shown in Figure 3.3.

3.6 Procedure for Vision Calibration: (with the new four-point calibration)

1. Take the vehicle to the test track and place the white lines parallel to the robot at a distance of 3 feet from the sides.
2. Make all measurements cautiously.
3. Check to see if the vehicle is in the center of the test track.
4. Adjust the camera so that the white line is approximately vertical in the center of the monitor.
5. Place a T-scale in the front of the drive wheels to have an X-axis across the track with reference to the machine.
6. Place the long scale with a rotatable arm along with the length of the machine and perpendicular to the T-scale. This would be Y-axis going down the track .
7. Since the robot is rectangular in shape, mark the centers on the robot on the X- axis and Y-axis. This can be marked permanently so that it can be useful in further calibration. In this way, we know that the centroid of the robot is the geometrical center and we don't have to explicitly know the centroid. All the measurements can be taken from the marked X and Y centers.
8. Place the 4-point calibration device on the white line such that the all the 4 points are visible on the monitor .
9. Get the Xo and Yo distances. Xo = distance between the X center and the bottom tip of the calibration device measured along the X-axis. Similarly Yo = distance between the Y center and the bottom tip of the calibration device along Y-axis.
10. The Zo distance is known because the distances of the spheres on the calibration device are exactly known.

11. Adjust the size of the window to approx. 4 X 4 and have threshold set to bright. Make sure the ISCAN is in the manual mode.
12. Focus on the 1st sphere. Run the "g-cart" program from the computer and choose the Visdemo option. Get the X and Y image co-ordinates of the sphere. Note it down in the table below. Repeat this procedure for all the 4 points. Measure thrice and take the average to avoid any clerical mistakes. Make a table as shown below and enter the co-ordinates.

Sample table for calibration:

Sphere	X value	Y value	Z value	X_{PI}	Y_{PI}
1					
2					
3					
4					

Table 3.1 Sample table for noting down the image co-ordinates during calibration

13. The real world co-ordinates of the spheres (Xg, Yg, Zg) and their image co-ordinates are then entered in the C++ program "matrix.c" shown in Appendix A. The output is the camera coefficients. These can be now inserted into the vision control program. In the C++ program, the matrices are expressed as two-dimensional arrays and they are manually filled as the co-ordinates are taken. The comments provided in the program are self explanatory.

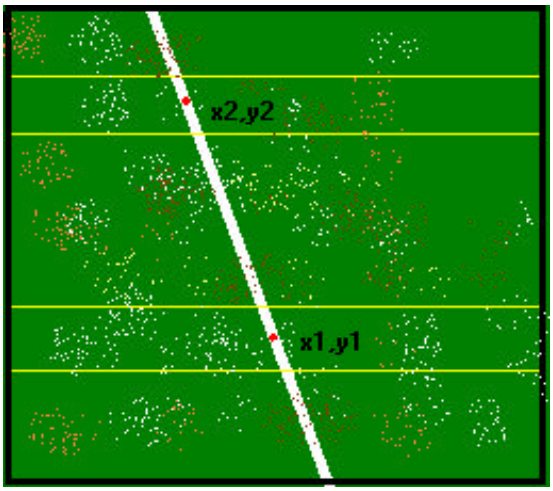


Figure 3.2 Image coordinates as seen on the monitor

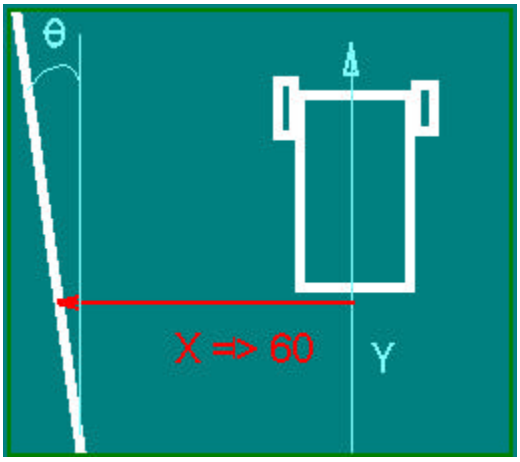


Figure 3.3 The real world coordinate system.

3.7 Experimental Points:

Points	Physical Co-ordinates			Image Co-ordinates	
	x_g	y_g	z_g	X_{PI}	Y_{PI}
1	58	12.125	11.5625	215	149
2	60	11.125	12.5625	203	163
3	62	15.125	11.5625	190	125
4	60	17.125	12.5625	214	113

Table 3.2 Points for the left camera

Camera Coefficients for the left camera							
A11	A12	A13	A14	A21	A22	A23	A24
-7.625	1.833	-5.0833	576.244	0.25	-8.333	-5.1667	175.802

Table 3.3 Camera coefficients

Points	Physical Co-ordinates			Image Co-ordinates	
	x_g	y_g	z_g	X_{PI}	Y_{PI}
1	58	26.375	11.5625	326	138
2	60	28.375	12.5625	330	124
3	62	24.375	11.5625	359	156
4	60	22.375	12.5625	342	174

Table 3.4 Points for the right camera.

Camera Coefficients for the right camera							
A11	A12	A13	A14	A21	A22	A23	A24
-7.25	-2.00	6.5	33.4063	0.3333	-8.333	-2.000	315.333

Table 3.5 Camera coefficients.

Points	Original Physical Coordinates			Image Coordinates		Computed Physical Coordinates	
	x_g	y_g	z_g	X_{PI}	Y_{PI}	x_c	y_c
1	58.00	26.375	-11.5625	326	138	58.00	26.375
2	60.00	28.375	-12.5625	330	124	60.00	28.375
3	62.00	24.375	-11.5625	359	156	62.00	24.375
4	60.00	22.375	-12.5625	342	174	60.00	22.375
5	59.00	25.375	-12.5625	332	149	59.446	25.3527
6	61.00	25.375	-12.5625	334	145	60.4128	25.87
7	59.50	28.375	-12.5625	324	124	59.1632	28.3415
8	62.00	21.375	-12.5625	356	182	61.68	21.48

Table 3.6 Comparison of the original physical co-ordinates and the computed co-ordinates.

Correlation plots for the original and the computed x and y coordinates are shown in Figures (3.4) and (3.5). The linearity of the plots means that the difference between the original coordinates and the computed ones is very small. It is interesting to note that with the 4-point calibration we have achieved the same accuracy in a very simple and quick procedure. This is the main highlight of the 4-point calibration. This procedure has been designed in such a way that it takes minimum time for the initial setup.

Also computed to ascertain or test the discrepancies between the two sets of coordinates is the mean square error. For each of the correlation plots, the mean square error was 0.094944 for the x-axis and 0.179469 for the y-axis. With a mean square error of within two tenth of an inch, the calibration process is considered as accurate and reliable enough to compute the physical coordinate of a real life point on a ground. Having dealt with the computation of the x and y coordinate of a physical point with respect to the centroid of the robot, the next item considered in this study is how to spot more than one point, two points to be precise, on a line and establish some geometrical relationship between the points and relate it to the centroid of the robot.

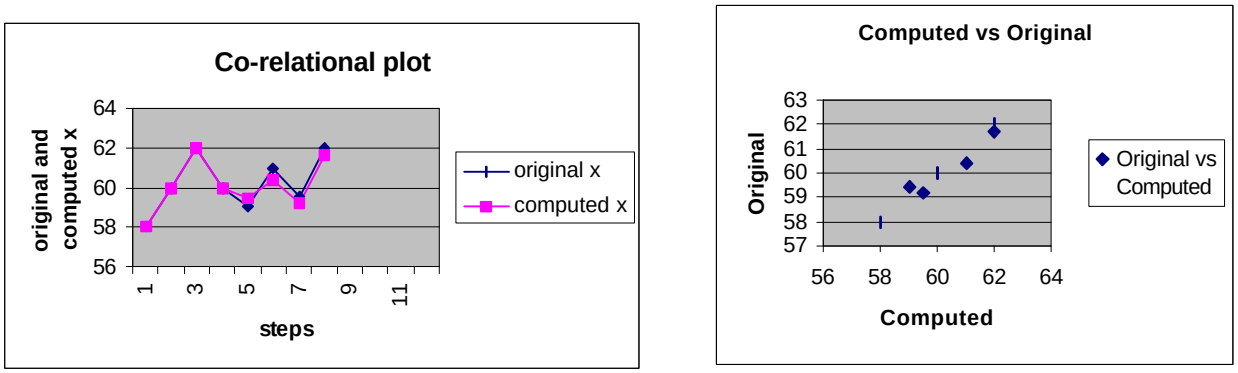


Fig 3.4 Plots for X axis

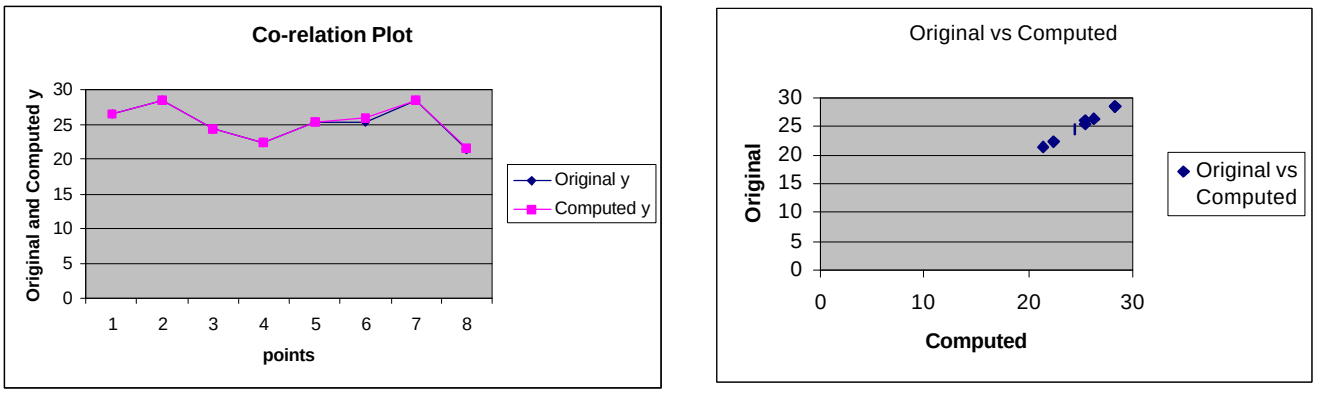


Fig 3.5 Plots for Y axis

Chapter 4

The Use of Fuzzy Logic

4.1 Basics

The specific challenge in the design of an intelligent controller is to know what information is needed, how to measure it and how to use it so as to satisfy the requirements of the system. This involves a mathematical description of the relation among inputs to the process, its state variables and its output. In other words, we build a mathematical model of the system. The modeling of a mobile robot is a very complex task, which has multiple inputs and multiple outputs, and incorrect modeling can lead to unstable systems and unsuitable performance. The fuzzy logic control (FLC) uses the qualitative aspects of the human decision process to construct the control algorithm. [26]

Logic deals with propositions, which may or may not be true. A proposition can be true on one occasion, and false on another. If a proposition is true, it has a truth-value of 1 and if it is false it has a value of 0. These are the only possible truth-values using Aristotelian logic. Propositions can be combined to generate other propositions, by means of logical operators. The truth-value of a proposition thus obtained can be related to the truth-values of propositions that were combined to generate it. Fuzzy Logic is a way of interfacing inherently analog processes that move through a continuous range to a digital computer that likes to see things as well-defined numeric values. Conventional Boolean logic contains only binary truth-values (0 and 1) where there is a sharp distinction between true and false. Fuzzy logic allows these states to change gradually from one state

to the next. In other words, there are values which range between 0 and 1.These are called the membership functions.

Fuzzy logic deals with propositions that can be true to a certain degree – somewhere from 0 to 1. It can be best understood in the context of set membership. The inclusion of degree of membership in the set makes it convenient for developers to come up with a set theory based on fuzzy logic. Fuzzy sets are sets in which members are represented as ordered pairs that include information on degree of membership.

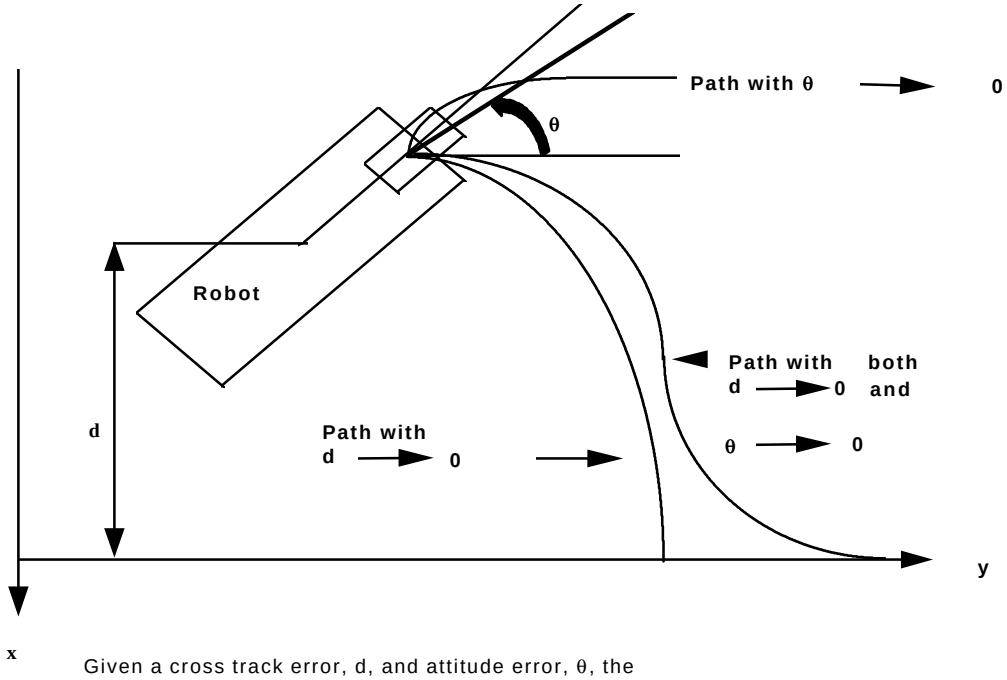
4.2 Applications of fuzzy logic:

Applications of fuzzy sets and fuzzy logic are found in many fields, such as artificial intelligence, engineering, computer science, operations research, robotics and pattern recognition. Fuzziness should also be included in neural networks.[28]

The development of techniques for autonomous robot navigation constitutes one of the major trends in the current research in mobile robotics. By autonomous, we mean the ability of the robot to move on its own without human assistance in environments that have not been engineered specifically for them. The design requirements were to build an autonomous robot that would follow a given track separated by two lines and, in turn, avoid the obstacles. This called for a design of separate subsystems at lower levels which would satisfy specific design objectives and integration of these systems at a higher level to enable the robot to function properly.

The subsystems that are designed include a speed control system, a steering system, a vision system and an obstacle avoidance system. In our discussion here, we primarily focus on the vision system, which consists of 2 CCD cameras and an image-tracking device from ISCAN Inc. The information from the vision system is used as input

to a closed loop fuzzy logic controller to control the steering and speed of the robot. Fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and a output stage. The input stage maps sensor or other inputs (in our case the distance and the angle of the line) to the appropriate membership functions and truth-values. The processing stage consists of the fuzzy inference engine, which invokes the rules and fires the appropriate response, and then combines the results of the rules. The output stage converts (defuzzifies) the combined result back into a crisp value that is given to the control computer that takes necessary actions.



Given a cross track error, d, and attitude error, θ, the control problem is to determine how to make both d(t) and θ(t) approach 0.

Figure 4.1 Distance error and angle error of the position of the robot w.r.t the line. [13]

The ISCAN tracking device returns the X and Y co-ordinates of 2 points on the line (Z is constant) and the corresponding ground co-ordinates are calculated. From the computed X and Y co-ordinates of the points, the angle of the line with respect to the centroid of the robot is computed from a simple trigonometric relationship. The distance of the line from the centroid of the robot is also obtained. Both these parameters need to be controlled. If only the angle is controlled, the robot might draw closer to the line and eventually go out of bounds. The same is true for minimizing only the distance error. By knowing the angle error (θ_{error}) and the distance error (d_{error}), we have to find the steering angle to guide the robot in the proper direction.

4.3 The Fuzzy Logic Controller

The fuzzy control for the autonomous robot was developed based on empirical methods, counting on human experience, chiefly by trial and error. The general process was as follows:

A) Fuzzy Set Definition:

As defined above θ_{error} and d_{error} form the input variables for the controller. Fuzzy subsets are formed on these variables and membership functions are defined. To model the problem in a simple manner, triangular membership functions have been used.

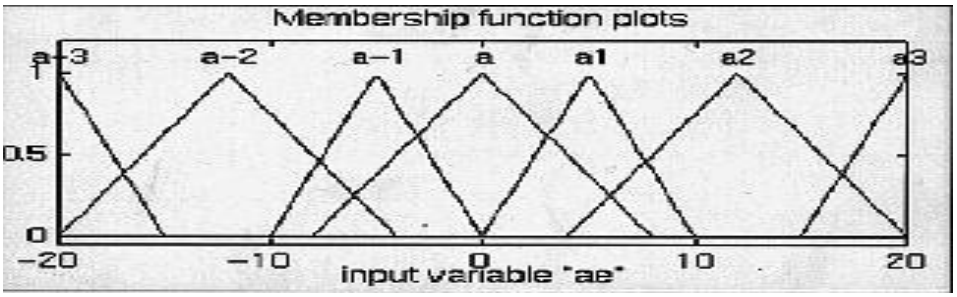


Fig 4.2.Input fuzzy sets (angle error) [26]

The inputs are then expressed in words describing the degree of membership of the variable.

Fuzzy Set	Description
a-3	Extreme Left
a-2	Left
a-1	Soft Left
a	Zero
a1	Soft Right
a2	Right
a3	Extreme Right

Table 4.1.Lingustic variables for the steering angle.

The output variable i.e. steering angle of the vehicle is similarly fuzzified.

B) Rule Base:

Since navigation as done by humans is an intuitive process, the rule base for this system was derived empirically from extensive testing. Moreover, it was felt that describing navigation in words was a “natural” and “simple” process for the human expert. This description is translated into linguistic rules. Thus a rule base was created. The experimental process yielded a set of 32 rules for guiding the robot along the track and avoiding the obstacles [26].

C) Fuzzy Operators:

Fuzzy Operators are employed to unify the multiple input sets into a single value. The two inputs θ_{error} and d_{error} are combined with the AND operator (product). The resulting value is then applied to the output membership function.

D) Defuzzification:

Centroidal Defuzzification is carried out to obtain the desired steering angle of the robot.

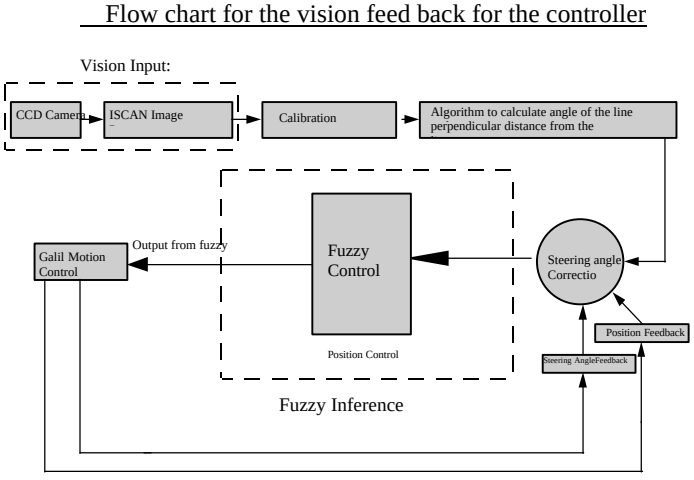


Figure 5.3 Flow chart for the vision feedback for the controller

4.4 Obstacle Avoidance

The obstacle avoidance system consists of a rotating ultrasonic transducer. An ultrasonic ranging system from Polaroid is used for calibration of the ultrasonic transducers.

The operational principle of the system is that a pulse of electronically generated sound is transmitted towards the target and the resulting echo is detected. The time between the pulse sent and received back is measured and, knowing the speed of sound in air, the system can convert the elapsed time into a distance.

A fuzzy logic controller to perform the obstacle avoidance was developed as a separate module. The θ_{error} and d_{error} , the distance of the obstacle from the robot, along with the position of the obstacle relative to the robot (LEFT,RIGHT) were the input variables. The desired steering angle is the output. A rule base similar to the one used to avoid obstacles was developed in a manner similar to the one described above. A Pseudo code was developed for obstacle avoidance in the control algorithm, which is then programmed in C and interfaced with the other modules. Testing of this dual level fuzzy system was done. First a theoretical simulation was run using the MATLAB fuzzy logic toolbox . The input used for running these simulations were from theoretical cases. The inputs were θ_{error} and d_{error} . The outputs were $\theta_{steering}$ and the speed of the robot.

Chapter 5

The Neural Networks Approach

5.1 Basics

The human brain uses a web of interconnected processing elements called neurons to process information. Each neuron is autonomous and independent. Many real life problems like face recognition, prediction, etc have two characteristics, which make them different from ordinary problems. First, the solutions are non-algorithmic and the data provided may be noisy and incomplete.

The vast processing power inherent in biological neural structures has inspired the study of the structure for hints on organizing man-made computers. Inspired by the biological nervous system, artificial neural network technology is being used to solve a wide variety of complex scientific, engineering and business problems. Neural networks are ideally suited for such problems because like their biological counterparts, an artificial neural network can learn, and therefore can be trained to find solutions, recognize patterns, classify data, and forecast future events. [27]

In contrast to classical approaches in fields such as statistics and control theory, neural nets require no explicit model or limiting assumptions of normality or linearity. Neural networks are a uniquely powerful tool in applications where formal analysis would be extremely difficult or impossible, such as pattern recognition and nonlinear system identification and control.

A neural network is a group of processing elements divided into computing layers. One subgroup makes independent computations and sends the results to the second subgroup. A subgroup of processing elements is called a layer in the network. Each subgroup makes its independent computations and passes on the output to the next layer. Each processing element makes its computation based on a weighted sum of inputs. The first layer is called the input layer and the last layer is the output layer. The layers between the first and the last layers are called the hidden layers. The processing elements are seen as units that are similar to the neurons in a human brain, and hence are referred to as neurons. A non-linear function followed by a threshold function is used to determine the output of the neurons in each layer. Synapses between neurons are referred to as connections. Processing in neural networks is done in parallel, rather than sequentially, as in a digital computer. Digital computers require accurate information and algorithms. These observations show how neural networks are different from expert system, which require precise formulations of rules for firing based on the information.[28]

Output of a Neuron:

The activation or raw output of a neuron in a neural network is a weighted sum of its inputs, but a threshold function is also used to determine the final value, or the output. When the output is 1, the neuron is said to fire and when it is 0, the neuron is said not to be fired. When a threshold function is used, different results of activations, all in the same interval of values, can cause the same final value.

Weights:

The weights used on the connections between different layers have much significance in the working of the neural network and the characterization of a network. The following steps are possible in a neural network:

- Start with one set of weights and the run the network.
- Modify some or all the weights and run the network again with new sets of weights
- Repeat this process until some predetermined goal is met.

Training:

The reason to alter the weights is that the output(s) is not what is expected. There has to be some rule and some criterion to specify when the process of successive modification of weights ceases. This process of changing or refining the weights is called learning. A network in which learning is employed is said to be subjected to training. Training is an external process or regimen. Learning is the desired result that takes place internally.

Feedback:

If you wish to train a network so that it learns some predetermined patterns, it would be important to have information fed back from the output neurons to neurons in the layer before that, to enable further processing and adjustments of the weights. The feedback can be from the output layer to the input layer or sometimes from the output layer to the hidden layers and then back to the input layer. The error in the output is fed back and modified appropriately according to some paradigm. The process of feedback continues till the training procedure is continued.

Noise:

A data set used to train a neural network may have inherent noise in it, or an image may have random speckles in it. The response of the neural network to noise is an important factor in determining its suitability to a given application. Sometimes it is even advisable to introduce noise intentionally in training to find out if the network is indeed learning in the presence of noise.

Thus we can say that,

"*A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use*."[27]

It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.

An Artificial Neural Network is a network of many simple processors ("units"). The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training. The weighted sum of inputs to unit j is given by

$$net_j = \sum_i w_{ij} i_i.$$

w_{ij} is the weight connecting the i^{th} input to the j^{th} neuron……(1)

The design of a neural network begins with network architecture. The task of navigation and obstacle avoidance together is treated as a function approximation problem. Hence, a feed-forward architecture is used to model the problem. The weighted sum of all the inputs is fed to each layer of the network in succession. The output of one layer becomes the input of the next until the next layer is reached.

Another important design aspect is the choice of the training algorithm.

5.2

Back-Propagated Delta Rule Networks (BP)

The backpropagation network is a very popular model in neural networks. It does not have feedback connections, but errors are back propagated during training. Least mean squared error is used. The errors in the output determine measures of hidden layer output errors, which are used as a basis for adjustment of connection weights between the input and hidden layers. Adjusting the two sets of weights between the pairs of layers and recalculating the outputs is an iterative process that is carried on until the errors fall below a tolerance level.

Backpropagation is a development from the simple Delta rule in which extra hidden layers (layers additional to the input and output layers, not connected externally) are added. The network topology is constrained to be feedforward: i.e. loop-free. Generally connections are allowed from the input layer to the first (and possibly only) hidden layer; and so on to the output layer. The hidden layer learns to recode (or to provide a representation for) the inputs. More than one hidden layer can be used. The architecture is more powerful than single-layer networks; it can be shown that any mapping can be learned, given one hidden layer (of units). [27]

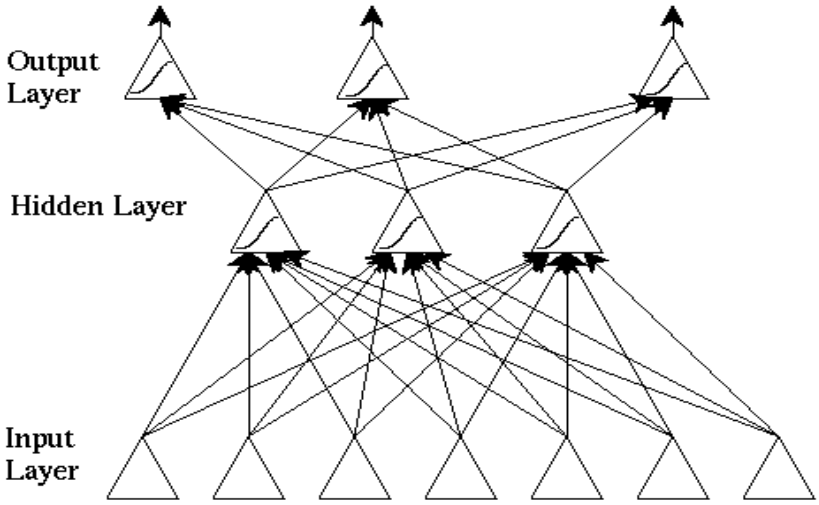


Figure 5.1 Feed forward network architecture. [29]

5.3 Training BP Networks

The weight change rule is an application of the steepest descent search algorithm. Weights are changed by an amount proportional to the error at that unit times the output of the unit feeding into the weight.

Running the network consists of:

Forward pass: The outputs are calculated and the error at the output units calculated as the difference between the computed output and the desired output.

Backward pass: The output unit error is used to alter weights on the output units. Then the error at the hidden nodes is calculated (by back-propagating the error at the output units through the weights), and the weights on the hidden nodes altered using these values.

For each data pair to be learned, a forward pass and backward pass is performed. This is repeated for a fixed number of iterations or until a predetermined acceptable error is reached.

Analog processing units with sigmoidal activation function have been used.

As a function:

$$f(net)=1/(1 + \exp(-k*net)) \quad \text{.....(2)}$$

Figure 5.2 shows the output for k=0.5, 1, and 10, as the activation varies from -10 to 10.

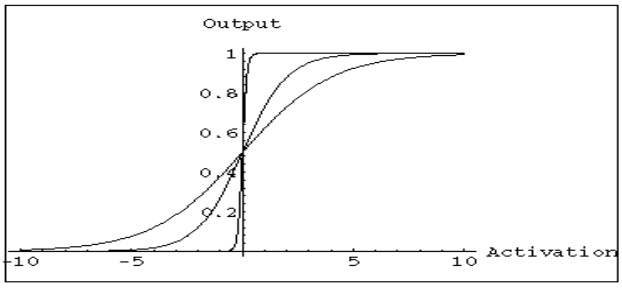


Fig.5.2 Activation Function (sigmoid) [27]

5.4 Input Representation

The most important factor determining the performance of a neural network on a given task is the input representation. For autonomous navigation and obstacle avoidance of a mobile robot, as described earlier, we use one CCD camera to capture the images of the obstacle course. The input images are analog, i.e. continuous color, images. Before providing these images to the network, they are converted into binary form using a

threshold function. The justification of this step lies in the fact that outdoor lighting conditions may vary. Also the obstacles and the guidelines (the 2 lines which are 10 ft. apart) may be white or yellow. Converting to binary form simplifies the task of the network at a very little computation cost. This is followed by resizing the image to a frame of 30 x 30 pixels. The track conditions are such that most of the information contained in the image is redundant. Also resizing, which can be easily achieved by a standard algorithm, reduces the size of the network drastically.

The binary values of the pixel i.e. 0 or 1 are used as input to the network. The 30 x 30-image matrix is vectorized to a 900 element vector when it is fed to the network. The preprocessing performed on the image seeks to reduce the complexity of the problem by standardizing the “features” needed for navigation and avoiding the obstacles. The task of identifying these features is left to the network.

5.5 Output Representation

The next crucial decision in determining the networks’ mechanism is the output representation. The one out of N- coding technique was used to represent the desired steering response to a given condition. The steering angle is continuous and bounded in an interval. This interval is discretized to a finite number (5 in this case) of responses.

Steering Angle in degrees	Output Vector
-10	[1 0 0 0 0]
-20	[0 1 0 0 0]
0	[0 0 1 0 0]
10	[0 0 0 1 0]
20	[0 0 0 0 1]

Table 5.1.Output Representation.

The parameters of the network, called synapses or weights, have to be fixed before the network is put to use. Training performs this task by iterative updating of randomly initialized weights. The back propagation algorithm was used for this purpose. Since this problem does not require knowledge of previous states, recurrent models were not considered. Also, since training is off-line, the speed of convergence is of less consequence. Another advantage of training off-line is that structured noise can be added to increase the network's ability to handle a variety of situations.

Matlab Neural Network Toolbox

The Neural Network Toolbox is a comprehensive environment for neural network research, design, and simulation within MATLAB Neural networks offer engineers and scientists a powerful way to explore, classify, and identify patterns in data. Because they require intensive matrix computations, MATLAB provides a natural framework for rapidly implementing neural networks and for studying their behavior and application.

The Neural Network Toolbox provides comprehensive support for the design, implementation, and simulation of many proven network paradigms and is completely extensible and customizable. Its consistent methodology and modular organization facilitate research, provide a flexible framework for experimentation, and simplify customization.

For this simulation, the backpropogation program (demobp1.m) was used. The main advantage of using Matlab in this simulation that it is very powerful and also gives the user a neat user interface.

Chapter 6
Comparison of the Rule Based Model vs. the Connectionist Model

Before delving into a comparison of the results from applying the different paradigms, it is interesting to note the basic differences between them.

Fuzzy information processing technology links numerical data treatment based on mathematics with human experience and knowledge expressed linguistically. Neural Network technology shows good performance in memorizing numerical input output data out of past experience but has the drawback of hiding knowledge in a black box after acquiring it through learning.

The Rule based algorithm relies on specifying the input variables explicitly; in other words, using exclusively the features identified as being influential in determining the output drives the output. It is critical to identify all the important features and only these features. Missing a critical feature will lead to unpredictable results, while having extraneous features increases the size of the rule base, making the algorithm inefficient. Therefore, it is said that a fuzzy system is modeled by drawing inferences from the real system.

The Neural Network is trained using real instances of the system. This is called learning from examples. The purpose of training is to fix the weights of the network in such a way that the influential features are highlighted. This method ensures that all the critical features are identified. However, the accuracy of the network depends on the training data provided to it. It is quite possible that ill-chosen examples bias the network

output. Also the network may pick up features present in examples by coincidence, since it has no way of distinguishing relevant features from irrelevant ones.

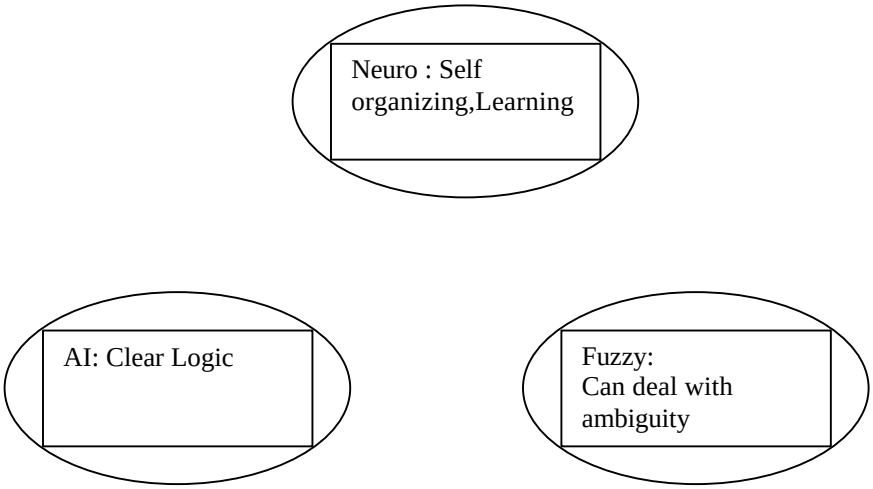


Figure 6.1 A comparison of AI, Fuzzy and Neural Networks [28]

In addition to the explicit inputs, it is necessary to come up with a set of rules to be able to model a fuzzy system. Fuzzy systems are amenable for use only in situations where the inputs and outputs have explicit relations. Once these relations are identified, coding a fuzzy system is very straightforward. The parameters and fuzzy sets are defined according to experience, making the system transparent, hence, easy to maintain and upgrade. A typical program consists of a set of IF-THEN statements. More often than not some form of preprocessing is required for numerical input and output representation. In its simplest form, this could be scaling. As discussed earlier, a neural network relies on training to relate inputs and outputs. Although observing the weights gives us an idea about the input –output relations, the governing rules cannot be explicitly stated. Thus training a neural network is a black box.

The mechanism of a fuzzy logic program involves running through a series of conditional (if-then) statements. In the worst case, all the rules have to be checked, which can lead to high computational time. A neural network involves simple (multiplication) weighted summations equivalent to matrix multiplication. A feed-forward neural network has a fixed finite number of such computations on every operation.

As the number of input variables or resolution into linguistic variables of a fuzzy system increases, the rule base increases in size geometrically. A feed forward neural network with a single hidden layer can model highly non-linear systems. This is due to the massively parallel nature of the network. It must be mentioned, however, that no deterministic methods exist to determine the optimum number of hidden layer of neurons required to model a given system. Also, any change in size of the network means that the network has to be trained again.

Any errors during the formulation of the rule base or while coding severely affect the performance of the system, as there is a one to one correspondence between an output and a rule. Fuzzy systems are also sensitive to errors in data.

Neural Networks are relatively more robust and tolerant of noisy data. Since each neuron contributes to every neuron in the next layer, failure of just a few does not greatly affect the network performance. The subtle relationship between the input and the output makes the network robust and capable of generalization. The following table provides a summary of the comparison between the two algorithms.

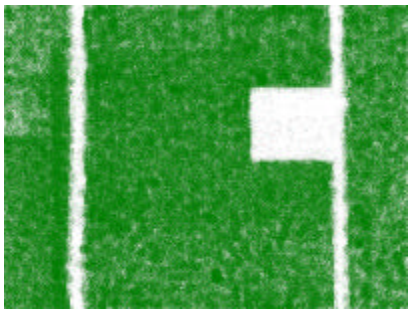
Criteria for Comparison	Fuzzy Controller	Neural Networks
Principle	Rules for navigation of the robot in the obstacle course inferred from human drivers actions.	Human drivers' actions mimicked for "learning" by the network from images.
Inputs and Preprocessing Required	Calibration of camera required to obtain the d_{error} and θ_{error} –the input variables	Raw image data converted to binary form and resized to 30 x30 frame
Complexity of Algorithm	In the worst case 32 conditionals (If-Then statements) to be evaluated.	For a network of size 'n' the same number of multiplication and addition operations are required.
Size of the Model	Increases geometrically with increase in resolution or number of input variables.	Increases linearly with increase in number if processing units.
Robustness of the Algorithm	Sensitive to noise in data and coding errors.	Massive parallel structure gives robustness and ability to generalize.
Coding and Maintenance	Transparent code helps in maintenance.	Involves matrix operations for efficient implementation.
Hardware Implementation	Relatively difficult.	Easily hard wired.
Tuning	No analytical technique exists, based purely on trial and error.	Tuning can be achieved by suitable training algorithms
Stability	Difficult to develop a model that can prove the stability of the controller.	A feed forward network with one hidden layer can provably model highly non-linear systems.

Table 6.1.Comparison between the two techniques

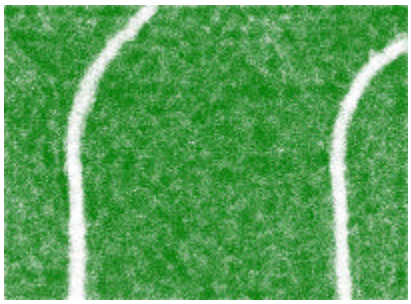
6.2 Results: A) *Fuzzy Logic Controller:* Table 3.

Remarks:	Distance Error:(de)	Angle Error: (ae)	Steering angle: (st)
Case 1:Ideal	0	0	0
Case 2:	2	0	-5.2
Case 3:Extreme	3	0	-10
Case 4:Extreme	-3	0	10
Case 5:Extreme	0	20	19.6
Case 6:Extreme	2.5	20	-19.6
Case 7:Extreme	-3	20	19.6
Case 8:	-1.5	-1	10
Case 9:	1.5	10	-13.6
Case 10:	-2	13	10.8

B) *Neural Networks:* The following test images were presented to the network. The output steering direction is shown below.



Test Image 1: Small Left Turn 10 deg



Test Image 2: Small Right Turn 10 deg

Test Image 3: Sharp Left Turn 20 deg

Chapter 7

The IGRC Contest

7.1 Introduction

This chapter reviews the Annual International Ground Robotics Competition. Starting in 1991, the Oakland University has been organizing an Annual International Ground Robotics Competition. In this competition, unmanned, automated guided robotic vehicles have to travel around an obstacle course. The robots have to stay within the track, and avoid the obstacles laid out on the course. This is a classic example of where the navigation and response of the various control systems play a very important role.

The design challenge set for the competitors by the organizers has been to make the course challenging and realistic. Until 1996, no team had been successful in their attempts to complete the course, without knocking over any of the obstacles, or straying off the track. In 1997, however, a few teams did complete the course successfully, and hence the deciding factor was the time and who would be able to complete the course in the least time. In 1998, most of the teams had a faster and smaller robot with better maneuverability than the previous year but no one completed the course because it was shaped like the boundary of the islands of Japan. The AGV's had to be faster to finish the track in a lesser time. Bearcat II was 3 months old when it was entered in the 1998 Contest and not all of the subsystems were fully developed. However, in 1999, the team

was all geared up with a very good chance of winning the contest and there in lies the success story of Bearcat II.

7.2 Rules of the Competition

1. The vehicles are not allowed to stray off the course. The vehicles are required to stay inside the edges of the track, and going off the track results in loss and elimination.
2. Each vehicle is allowed three runs, and the best run of the vehicle is taken into account for the final result.
3. The vehicles are not allowed to collide with obstacles. Touching the obstacle results in a loss of points, and moving or knocking the obstacles, results in termination of the run.
4. The vehicles are not allowed to travel at a speed greater than 5 mph, and should have facilities for emergency stopping.

The vehicles are subjected to a safety inspection called the qualification before they are allowed to compete in the actual competition. This inspection ensures that all the vehicles comply with the requirements. During the runs, if the judges feel the vehicle may be safety hazard or if it violates the safety requirements, they can disqualify the vehicle. Emergency stops on the vehicle and the remote stops are tested as a measure of safety and as a qualification for the competition. The maximum speed is also tested on a special track.

7.3 The 1999 Contest :

There were four different competitions involved: vehicle performance competition, vehicle design competition, road debris bonus event and follow the leader bonus event.

a) Vehicle performance competition

- OBJECTIVE:

1. Robotic vehicle to autonomously navigate around an outdoor obstacle course under a prescribed time while avoiding the obstacles on the track.

2. Judges will rank the entries, which complete the course based on the shortest adjusted time. In the event that a vehicle does not finish the course, the judges will rank the entry based on the longest adjusted distance traveled.

- BEARCAT II performance

On the first day, the BEARCAT II had the remote E-stop burn out. The next day, a new remote E-stop unit was bought and connected in series with the motor power control circuit, which is controlled by a solenoid. This effort helped the BEARCAT II to pass the qualification test. During the formal contest, the computer motion system closely followed the commands from the computer and the sonar rotation system. The vision system worked perfectly and it was visible that the robot was responding to the turns. The calibration was accurate and the robot always tried to remain at the center of the track. It also avoided three obstacles on its way and had no problems on different ground surfaces. The BEARCAT II made a distance of 153.3 feet during its second effort, which is the fourth best among all the teams. The detail results are given in the Appendix. Figure 7.1 shows the track that the vehicle traveled during the contest. The robot stopped because it ran into a small white obstacle which was placed very close to the boundary line.

Figure 7.1 Bearcat II in the 1999 AUVSI Contest held on June 7, 1999.

b) Road Debris Bonus Event

In this competition, the robot vehicle will encounter irregular size and shape obstacles that could be seen during regular highway or roadway driving. This can include, construction signs, barrels and cones, hubcaps, tire treads, tailpipes, barricades, parked cars and pedestrians.

Each autonomous vehicle system (AVS) entering the Bonus Competition Event (BCE) was given 2 attempts at negotiating a 150'-250' stretch of road.

Again, the Bearcat II moved on the complicated track for 65 feet before it ran off the track. This was the 3rd best performance in the Road Debris Event.The vision system worked great as the robot tracked the lines. Both the wheel motion and the sonar rotation acted accurately to support the robot moving in the right direction.

c) Follow the header:

This event required that the BEARCAT II robot autonomously follow a side lawn mower that was driven by one of the judges. An omnidirectional motion system was used to determine the angle of the target and the rotating sonar to determine the distance. The BEARCAT II placed 4th in the competition.

d) Design competition:

The design competition is held before the performance events and judged on the quality of a written design report, an oral presentation and a vehicle visual inspection by the judge.

The BEARCAT II was ranked 4th in this competition for its clear organization and outstanding salesmanship.

Chapter 8

Conclusions and Future Work:

A novel four-point calibration method has been designed and implemented for the vision system of Bearcat II. It has been successfully tested and implemented on Bearcat II in the international competition. The calibration program is written in C++ (as is all the control software for BearcatII). The new calibration technique saves time and is simpler to understand. The C++ software is robust and provides a neat interface, also saving time. MathCAD was used just to counter-check the answers. C++ provides accurate answers and saves considerable time. The objective of this research as stated in section 1.2 was achieved.

As discussed earlier, the tuning of the fuzzy logic controller is still an open problem. No analytical techniques are known to exist for the tuning of the rule-based system. Hence it is difficult and time consuming to develop a set of guidelines for the rule-based system. On the other hand, the neural network behaves like a black box and require training. The relationship between inputs and outputs of the network is subtle and cannot be extracted explicitly. [29]

Incorporating speed control in the fuzzy system entails adding another layer of rules above the existing rule-base [30]. The information required for speed control is the desired steering angle and the position of the obstacle with respect to the robot. Such a two-layered system could increase the computational cost . The raw image data presented to the neural network does not contain information necessary to achieve speed control.

Implementing speed control requires development of a different input and output representation.

When an output of the neural network active is with values limited in the [0,1] area, the output can be regarded identical with membership values added to each factor of a fuzzy set. So it is relatively easy to combine both processes or replace one of them with the other.

Advantages of neural networks over fuzzy logic:

Some of the questions like knowledge acquisition, construction of fuzzy sets, creating rules for inference remain unanswered in fuzzy logic. So many researchers have tried to fuse fuzzy logic with neural networks to utilize the performance of linguistic expression interface for generating membership functions and inference rules. Also, most of the current fuzzy inference often results in an explosion of ambiguity. (ambiguity increases with each new inference step).

Advantages of fuzzy logic over neural networks:

In neural networks, there is a problem of knowledge spread over and hidden in the neural network cannot be explicitly known. It is a **black box.** The fuzzy knowledge representation is useful to interpret learning result of neural networks logic and to speed up the learning process of neural networks logic with the knowledge obtained in fuzzy inference rules as initial knowledge.[29]

The general and problem specific limitations of the two systems are being addressed by a new field of research called fuzzy-neural systems or neuro-fuzzy systems. Tuning of fuzzy systems can be achieved by neural networks. In a different model, fuzzy neurons, with membership functions as activation functions are used to model fuzzy-logic

controller with learning ability [30]. The number of input neurons is equal to the number of rules. Other hybrid models that utilize the similarities of fuzzy and neural systems to develop more versatile solutions provide directions for future work.

9. References:

1. Ernest L. Hall, **Robotics: A User-Friendly Introduction**, Holt, Rinehart, and Winston, New York, NY, 1985, pp. 23.

2. Z. L. Cao, S. J. Oh, and E. L. Hall, Dynamic omnidirectional vision for mobile robots," **Journal of Robotic Systems**, 3(1), pp. 5-17, 1986.

3. Berthold Klaus Paul Horn , **Robot vision** , MIT Press , New York , McGraw-Hill, pp. 1-4,1986.

4. Ichiro Masaki, **Vision-based vehicle guidance**, New York, Springer-Verlag, 1992.

5.David Kortenkamp, R. Peter Bonasso, and Robin Murphy, **Artificial intelligence and mobile robots: case studies of successful robot systems** / AAAI Press, Cambridge, Mass. MIT Press, pp. 16-20,1998.

6. Martial H. Hebert, Charles Thorpe, Anthony Stentz, **Intelligent unmanned ground vehicles: autonomous navigation research at Carnegie Mellon,** Kluwer Academic Publishers, Boston, pp. 1-4,1997.

7. Dean A. Pomerleau, "**Neural network perception for mobile robot guidance"** Kluwer Academic Publishers, Boston, pp. 1-4,1993.

8. A. Meystel, **Autonomous mobile robots: vehicles with cognitive control** / Singapore, Teaneck, NJ, World Scientific, pp. 2-6, 1991.

9. Joseph L. Jones, Bruce A. Seiger, Anita M. Flynn, "**Mobile robots: inspiration to implementation",** A.K. Peters, Massachusetts, pp. 1-3, 1999.

10.Yiannis Aloimonos, **Visual navigation: from biological systems to unmanned ground vehicles** , Mahwah, NJ : Lawrence Erlbaum Associates,pp. 317-320, 1997.

11. C. E. Thorpe, editor. *Vision and navigation : the Carnegie Mellon Navlab*. The Kluwer international series in engineering and computer science ; Robotics. Boston : Kluwer Academic Publishers,pp 4, 1990.

12. Hanqi Zhuang, Zvi S. Roth, **Camera-aided robot calibration**, Boca Raton, Fla., CRC Press, pp. 1-6,1996.

13. Tayib Samu, Nikhil Kelkar, David Perdue, Mike Ruthemeyer, Bradley Matthews and Ernest Hall, "Line following using a two camera guidance system for a mobile robot," *SPIE Conference 2904-39*, Boston, MA, November 1996.

14. M. X. Li and J.M. Lavest, ``Some Aspects of Zoom-Lens Camera Calibration", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, pp. 1105-1110, Nov., 1996.

15. S. Chen and J. Weng, ``Calibration for peripheral attenuation in intensity images," in Proc. *First IEEE International Conference on Image Processing*, Austin, Texas, pp. 992-996, Nov. 13-16, 1994.

16. J. Weng, P. Cohen and M. Herniou, ``Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10. Oct. 1992, pp. 965-980.

17. F. Hong and Y.Baozong “An Accurate and Practical Camera Calibration System for 3D Computer Vision” **Chinese Journal of Electronics.** Vol. 1, No. 1, pp. 63-71, June 1991.

18. R. Y.Tsai, “A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-The-Shelf Cameras and Lenses.” *IEEE Transaction on Robotics and Automation*, vol. 8, no. 2, pp. 129-139.

19. Z. Zhang and O.D. Faugeras. “Determining Motion from 3D Line Segments: A Comparative Study”, *International Journal of Image and Vision Computing*, Vol. 9, No. 1, pp. 10-19, February 1991**.**

20. Z. Zhang, O.D. Faugeras and N. Ayache. R. Kasturi and R.C. Jain , **“Analysis of a Sequence of Stereo Scenes Containing Multiple Moving Objects Using Rigidity Constraints**” *Computer Vision: Principles,* IEEE computer society press, 1991.

21. Z. Zhang and O.D. Faugeras. “A 3D World Model Builder with a Mobile Robot *International Journal of Robotics Research*,” Vol. 11, No. 4, pp. 269-285, 1992.

22. Z. Zhang, Q.-T. Luong, and O. Faugeras,” Motion of an Uncalibrated Stereo Rig: Self-Calibration and Metric Reconstruction “. *IEEE Trans. Robotics and Automation* Vol.12, No.1, pp. 103-113, February 1996.

23. Z. Zhang, R. Deriche, O. Faugeras, Q.T. Luong. “A Robust Technique for Matching Two Uncalibrated Images Through the Recovery of the Unknown Epipolar Geometry *Artificial Intelligence Journal*, Vol.78, pp. 87-119, October 1995.

24. A D Worrall, G D Sullivan and K D Baker “A simple, intuitive camera calibration tool for natural images” Department of Computer ScienceThe University of Reading, Berkshire, RG6 2AY, UK.

25. Yongduek Seo and Ki-Sang Hong, “Auto-Calibration of Rotating and Zooming Camera”, IAPR Workshop on Machine Vision Applications, Chiba, Japan, 1998

26. Nikhil Kelkar and E.L. Hall, “Fuzzy Logic Control of an AGV,” *Proc. of Intelligent Robots and Computer Vision XVI*, Oct. 15-17, 1997, Pittsburgh, PA.

27. “Simon Haykin,” **Neural networks a comprehensive foundation**”, Maxwell Macmillan International, New York, pp. 10 1994.

28. Dean A Pomerleau, “**Neural Network Perception for Mobile Robot Guidance**”, Kluwer Academic Publishers, Boston, MA, 1993.

29. Toru Yamaguchi, Kenji Goto and Tomohiro Takagi, “**Applied Research in Fuzzy Technology**”, Kluwer Academic Publishers, Boston, MA, pp.270, 1994.

30. Robert J. Schalkoff, “**Artificial Neural Networks**”, McGraw Hill Companies Inc., New York, pp.361-370, 1997.

Appendix A

Code for the Calibration:

The cpp file :

```
#include <stdio.h>
#include <malloc.h>
#include <iostream.h>

typedef struct matrix
{
  int rows;
  int cols;
  double *block;
} *matrixptr;

/* Function prototypes */
void mprint(matrixptr);
void smmult(matrixptr,double);
int madd(matrixptr m1,matrixptr m2,matrixptr dm);
int mmult(matrixptr m1,matrixptr m2,matrixptr dm);
int mcopy(matrixptr sm,matrixptr dm);
int mtrans(matrixptr sm,matrixptr dm);
double det(matrixptr m);
double determin2(matrixptr m);
int minv(matrixptr sm,matrixptr dm);
int minverse2(matrixptr sm,matrixptr dm);
int nsolve(int rows,double *data);
int mid(matrixptr);
void mzero(matrixptr);


/* function definitions */
void mprint(matrixptr m)
{
  int i,j;
  int cols;
  cols=m->cols;
  for (i=0;i<m->rows;i++)
    {
    printf("\n");
    for (j=0;j<cols;j++)
      {
      printf("%8.4lf",*(m->block+i*cols+j));
      }
    }
}
```

```
/* add two matrices giving a third matrix */
int madd(matrixptr m1,matrixptr m2,matrixptr dm)
{
  int i,j;
  if (m1->rows!=m2->rows||m1->cols!=m2->cols)
          {
                  //fprintf(stderr,"\nmadd error - matrices different sizes");
                   cout<<endl<< "madd error - matrices different sizes";
                  return(1);
           }
  if (m1->rows!=dm->rows||m1->cols!=dm->cols)
          {
                  // fprintf(stderr,"\nmadd error - matrices different sizes");
                   cout<<endl<< "madd error - matrices different sizes";
                  return(1);
          }
  for (i=0;i<m1->rows;i++)
    for (j=0;j<m1->cols;j++)
      *(dm->block+i*m1->cols+j)=*(m1->block+i*m1->cols+j)+\
      *(m2->block+i*m1->cols+j);
  return(0);
}

int mcopy(matrixptr sm,matrixptr dm)
//matrixptr sm,dm;
{
  int i,j;
  if (dm->rows!=sm->rows||dm->cols!=sm->cols)
          {
                  //fprintf(stderr,"\nmcopy error - matrices different sizes");
                   cout<<endl<< "mcopy error - matrices different sizes";
                  return(1);
          }
  for (i=0;i<dm->rows;i++)
    for (j=0;j<dm->cols;j++)
      *(dm->block+i*dm->cols+j)=*(sm->block+i*dm->cols+j);
  return(0);
}

/* multiply matrix by scalar double */
void smmult(matrixptr m,double s)
//matrixptr m;
//double s;
{
  int i,j;
  for (i=0;i<m->rows;i++)
```

```
   for (j=0;j<m->cols;j++)
     *(m->block+i*m->cols+j)=*(m->block+i*m->cols+j)*s;
}

int mmult(matrixptr m1,matrixptr m2,matrixptr dm)
//matrixptr m1,m2,dm;
{
  int i,j,k;
  double cellval;
  if (m1->cols!=m2->rows)
          {
                    // fprintf(stderr,"\nmmult error - matrix 1 cols must = matrix 2 rows");
                   cout<<endl<< "mmult error - matrix 1 cols must = matrix 2 rows";
                    return(1);
           }
  if (m2->cols!=dm->cols)
          {
                    // fprintf(stderr,"\nmmult error - dest matrix cols must = matrix 2 cols");
                     cout<<endl<< "mmult error - dest matrix cols must = matrix 2 cols";
                   return(1);
          }
  if (m1->rows!=dm->rows)
          {
                     //fprintf(stderr,"\nmmult error - dest matrix rows must = matrix 1 rows");
                    cout<<endl<< "mmult error - dest matrix rows must = matrix 1 rows";
                    return(1);
    }
  for (i=0;i<m1->rows;i++)
    for (j=0;j<m2->cols;j++)
      {
      cellval=0.0;
      for (k=0;k<m1->cols;k++)
          {
       cellval+=*(m1->block+i*(m1->cols)+k) * (*(m2->block+k*(m2->cols)+j));
           }
      *(dm->block+i*dm->cols+j)=cellval;
      }
  return(0);
}

int mtrans(matrixptr sm,matrixptr dm)
//matrixptr sm,dm;
{
  int i,j;
  if (dm->rows!=sm->cols)
          {
```

```
        // fprintf(stderr,"\nmtrans error - dest matrix rows must = source matrix cols");
        cout<<endl<< "mtrans error - dest matrix rows must = source matrix cols";
          return(1);
        }
  if (sm->rows!=dm->cols)
        {
        //fprintf(stderr,"\nmtrans error - source matrix rows must = dest matrix cols");
            cout<<endl<< "mtrans error - source matrix rows must = dest matrix cols";
        return(1);
        }
  for (i=0;i<sm->rows;i++)
    for (j=0;j<sm->cols;j++)
        {
        *(dm->block+j*dm->cols+i)=*(sm->block+i*sm->cols+j);
        }
  return(0);
}

double det(matrixptr m)
//matrixptr m;
{
  double p1,p2,p3,d;
  int i,j,k;
  if (m->cols!=m->rows)
    {
  //  fprintf(stderr,"\ndet error - matrix must be square");
            cout<<endl<< "det error - matrix must be square";
    return(0.0);
    }
  d=0;
  for (i=0;i<m->cols;i++)
          {
                  p1=p2=1.0;
                   p3=*(m->block+i);
                  k=i;
                  for (j=1;j<m->cols;j++)
                          {
                                  k=(k+1)%m->cols;
                                  p1*= *(m->block+j*m->cols+k);
                                  p2*= *(m->block+(m->cols-j)*m->cols+k);
                          }
                  p3*=(p1-p2);
                 d+=p3;
    }
  return(d);
}
```

```
int minv(matrixptr sm,matrixptr dm)
//matrixptr sm,dm;
{
  int i,j,k,l;
        int nrow,ncol;
  double *d;
  if (sm->rows!=dm->rows||sm->cols!=dm->cols)
                {
                        //printf(stderr,"\nminv error - matrices must be same size");
                          cout << endl<< "minv error - matrices must be same size";
                        return(1);
                }
  if (det(sm)==0.0)
                {
                        // fprintf(stderr,"\nminv error - matrix is singular");
                         cout << endl<< "minv error - matrix is singular";
                        return 1;
                 }
  d=(double *)(malloc(sizeof(double)*sm->rows*(sm->cols+1)));
  if (d==(double *)NULL)
                 {
                        //fprintf(stderr,"\nminv error - insufficient memory");
                         cout << endl<< "minv error - insufficient memory";
                        return(1);
    }
  for (i=0;i<sm->rows;i++)
                {
                        nrow=i-1;
                        ncol=i-1;
        for (j=0;j<sm->rows;j++)
                {
                         nrow=(nrow+1)%sm->rows;
                         if (j==0)
                                 *(d+j*(sm->cols+1)+sm->cols)=1;
                        else
                                *(d+j*(sm->cols+1)+sm->cols)=0;
                        for (k=0;k<sm->cols;k++)
                                {
                                ncol=(ncol+1)%sm->cols;
                                *(d+j*(sm->cols+1)+k)=*(sm->block+nrow*sm-
>cols+ncol);
                                }
     }
    if (nsolve(sm->rows,d))
     {
```

```
                //        fprintf(stderr,"\nminv error - cannot use nsolve on row %u",i);
                cout << endl<< "minv error - cannot use nsolve on row,row ="<<i;
    free((char *)d);
    return 1;
    }
   else
                        {
                        nrow=i-1;
                        for (j=0;j<sm->rows;j++)
                                {
                                nrow=(nrow+1)%sm->rows;
                                *(dm->block+nrow*sm->cols+i)=*(d+j*(sm->cols+1)+sm-
>cols);
                                }
    }
   }
  free((char *)d);
  return 0;
}

int nsolve(int rows,double* data)
//int rows;
//double *data;
{
  int i,j,k;
  int cols;
  double dtemp;
  cols=rows+1;
  for (i=0;i<rows;i++)
                {
                        for (j=i;j<rows&&*(data+j*cols+j)==0.0;j++);
                        if (*(data+j*cols+j)==0.0)
                         {
                                //fprintf(stderr,"\nnsolve error - singular matrix");
                                cout << endl<< "nsolve error - singular matrix";
                                 return 1;
                          }
   if (j!=i)
    {
    for (k=0;k<cols;k++)
      {
      dtemp=*(data+i*cols+k);
      *(data+i*cols+k)=*(data+j*cols+k);
      *(data+j*cols+k)=dtemp;
      }
    }
```

```
                for (j=cols-1;j>=0;j--)
    {
                        *(data+i*cols+j) /= *(data+i*cols+i);
    }
   for (j=i+1;j<rows;j++)
    {
                        for (k=cols-1;k>=i;k--)
     *(data+j*cols+k)-=*(data+j*cols+i) * *(data+i*cols+k);
    }
                }
        for (i=rows-2;i>=0;i--)
                {
                for (j=cols-2;j>i;j--)
                        {
                        *(data+i*cols+cols-1)-= \
                        *(data+i*cols+j) * *(data+j*cols+cols-1);
                        *(data+i*cols+j)=0;
                        }
                }
 return 0;
}

int mid(matrixptr m)
//matrixptr m;
{
 int i,j;
 if (m->rows!=m->cols)
                {
                        //fprintf(stderr,"\nmid error - matrix must be square");
                        cout<< endl<< "mid error - matrix must be square";
                         return 1;
                }
 for (i=0;i<m->rows;i++)
                {
                        for (j=0;j<m->cols;j++)
                        *(m->block+i*m->cols+j)=0.0;
                        *(m->block+i*m->cols+i)=1.0;
                }
 return 0;
}

void mzero(matrixptr m)
//matrixptr m;
{
 int i,j;
 for (i=0;i<m->rows;i++)
```

```
    for (j=0;j<m->cols;j++)
      *(m->block+i*m->cols+j)=0.0;

}

double determin2(matrixptr m)
//matrixptr m;
{
  double p1,p2,p3,d;
  int i,j,k;
  if (m->cols!=m->rows)
                    {
                              //  fprintf(stderr,"\ndet error - matrix must be square");
                                cout<<endl<< "det error - matrix must be square";
                               return(0.0);
                    }
  d=0;
  for (i=0;i<m->cols;i++)
                    {
                               p1=p2=1.0;
                              p3=*(m->block+i);
                              k=i;
                              for (j=1;j<m->cols;j++)
                                        {
                                                  k=(k+1)%m->cols;
                                                  p1*= *(m->block+j*m->cols+k);
                                                   //  p2*= *(m->block+(m->cols-j)*m->cols+k);
                               }
    p3*=(p1);

          d-=p3;
    }
  return(d);
}

int minverse2(matrixptr sm,matrixptr dm)
//matrixptr sm,dm;
{
  int i,j,k,l;
          int nrow,ncol;
  double *d;
  if (sm->rows!=dm->rows||sm->cols!=dm->cols)
                    {
                              //printf(stderr,"\nminv error - matrices must be same size");
                               cout << endl<< "minv error - matrices must be same size";
                               return(1);
```

```
                }
  if (determin2(sm)==0.0)
                 {
                        // fprintf(stderr,"\nminv error - matrix is singular");
                        cout << endl<< "minv error - matrix is singular";
                        return 1;
                 }
  d=(double *)(malloc(sizeof(double)*sm->rows*(sm->cols+1)));
  if (d==(double *)NULL)
                {
                        //fprintf(stderr,"\nminv error - insufficient memory");
                         cout << endl<< "minv error - insufficient memory";
                        return(1);
   }
  for (i=0;i<sm->rows;i++)
                {
                nrow=i-1;
                ncol=i-1;
                        for (j=0;j<sm->rows;j++)
                                 {
                                        nrow=(nrow+1)%sm->rows;
                                       if (j==0)
                                                *(d+j*(sm->cols+1)+sm->cols)=1;
                                        else
                                                *(d+j*(sm->cols+1)+sm->cols)=0;
                        for (k=0;k<sm->cols;k++)
                                {
                                ncol=(ncol+1)%sm->cols;
                                *(d+j*(sm->cols+1)+k)=*(sm->block+nrow*sm-
>cols+ncol);
                                }
     }
    if (nsolve(sm->rows,d))
     {
                //        fprintf(stderr,"\nminv error - cannot use nsolve on row %u",i);
                cout << endl<< "minv error - cannot use nsolve on row,row ="<<i;
     free((char *)d);
     return 1;
     }
    else
                        {
                        nrow=i-1;
                        for (j=0;j<sm->rows;j++)
                                {
                                nrow=(nrow+1)%sm->rows;
```

```
                                *(dm->block+nrow*sm->cols+i)=*(d+j*(sm->cols+1)+sm->cols);
                                }
     }
   }
  free((char *)d);
  return 0;
}
```

The implementation file :

```
//#include"matrix.h"
#include"c:\sameer\matrix.c"
#include<stdio.h>
#include <iostream.h>
#include <conio.h>

void main()
{
        //int m,n,t,a,b,c;
        double vision_matrix[4][4];
        double inverse_matrix [4][4];
        double transpose_matrix[4][4];
        double x_image_matrix[4][1];
        double y_image_matrix[4][1];
        double mult1_matrix[4][4];
        double mult2_matrix[4][4];
        double coeff1k_matrix[4][1];
        double mult4_matrix[4][4];
        double coeff2k_matrix[4][1];
        double q_coeff_matrix[2][2];
        double b_matrix[2][1];
        double world_matrix [2][1];
        double q_inverse_matrix[2][2];

/*      for(m=0; m<4;m++)
        {
                for(n=0;n<4;n++)
                {
                   printf("Enter the value:");
                        scanf ("%d", &vision_matrix[m][n]);
                        printf("%d",vision_matrix[m][n]);
                        printf("\n");
```

```
                }
        }
*/
/*
        double vision_matrix2[4][4]=
{
  6,1,6,6,
  1,6,6,0,
  0,3,2,1,
  8,6,1,9
};*/
//These are values for calibration done on June 03 1999
        vision_matrix[0][0]=60.25;
        vision_matrix[0][1]=17.125;
        vision_matrix[0][2]=-11.5625;
        vision_matrix[0][3]=1;
        vision_matrix[1][0]=62.25;
        vision_matrix[1][1]=15.125;
        vision_matrix[1][2]=-12.5625;
        vision_matrix[1][3]=1;
        vision_matrix[2][0]=64.25;
        vision_matrix[2][1]=19.125;
        vision_matrix[2][2]=-11.5625;
        vision_matrix[2][3]=1;
        vision_matrix[3][0]=62.25;
        vision_matrix[3][1]=21.125;
        vision_matrix[3][2]=-12.5625;
        vision_matrix[3][3]=1;

        //The x and y image co-ordinate matrices are as follows

        x_image_matrix[0][0]= 326;
        x_image_matrix[1][0]= 330;
        x_image_matrix[2][0]= 359;
        x_image_matrix[3][0]= 342;

        y_image_matrix[0][0]= 138;
        y_image_matrix[1][0]= 124;
        y_image_matrix[2][0]= 156;
        y_image_matrix[3][0]= 174;

/*
        identity_matrix[0][0]=0;
        identity_matrix[0][1]=0;
        identity_matrix[0][2]=0;
        identity_matrix[0][3]=0;
```

```
        identity_matrix[1][0]=0;
        identity_matrix[1][1]=0;
        identity_matrix[1][2]=0;
        identity_matrix[1][3]=0;
        identity_matrix[2][0]=0;
        identity_matrix[2][1]=0;
        identity_matrix[2][2]=0;
        identity_matrix[2][3]=0;
        identity_matrix[3][0]=0;
        identity_matrix[3][1]=0;
        identity_matrix[3][2]=0;
        identity_matrix[3][3]=0;
*/

        //The definitions of the matrices

  //struct matrix test;
        struct matrix main= {4,4,&vision_matrix[0][0]};
        //test.rows =4;
        //test.cols=4;


        struct matrix inverse = {4,4,&inverse_matrix[0][0]};

        struct matrix transpose = {4,4,&transpose_matrix[0][0]};

        struct matrix x_image_coord = {4,1,&x_image_matrix[0][0]};

        struct matrix y_image_coord = {4,1,&y_image_matrix[0][0]};

        struct matrix mult1 = {4,4,&mult1_matrix[0][0]};

        struct matrix mult2 = {4,4,&mult2_matrix[0][0]};

        struct matrix coeff1k = {4,1,&coeff1k_matrix[0][0]};

        struct matrix mult4 = {4,4,&mult4_matrix[0][0]};

        struct matrix coeff2k = {4,1,&coeff2k_matrix[0][0]};



        //The definition of matrix pointers
```

```
        matrixptr mainp=&main;

        matrixptr transposep=&transpose;

        matrixptr inversep =&inverse;

        matrixptr x_image_coordp=&x_image_coord;

        matrixptr y_image_coordp=&y_image_coord;

        matrixptr mult1p=&mult1;

        matrixptr mult2p=&mult2;

        matrixptr coeff1kp=&coeff1k;

        matrixptr mult4p=&mult4;

        matrixptr coeff2kp=&coeff2k;

        printf ("Behold the calibration program in C++");
        getch();
        printf("\n");
        printf("The world co-ordinate matrix is:");

        mprint(mainp);
        printf ("\n");
        getch();
        printf ("The X image-coordinate matrix is:");

        mprint(x_image_coordp);

        printf ("\n");
        getch();
        printf ("The Y image-coordinate matrix is:");
        mprint(y_image_coordp);
        printf ("\n");
        getch();

    mtrans (mainp,transposep);

        mmult (transposep,mainp,mult1p);

        minv (mult1p,inversep);

        mmult (inversep,transposep,mult2p);
```

```
        mmult (mult2p,x_image_coordp,coeff1kp);    //a1k parameters

        mmult (inversep,transposep,mult4p);

        mmult (mult4p,y_image_coordp,coeff2kp);   //a2k parameters

        printf("\n");

        printf("The coefficients a1k are: \n");

   mprint(coeff1kp);

        printf("\n");
        getch();
        printf ("The coefficients a2k are: \n");


        mprint(coeff2kp);

        printf("\n");
        getch();

        //m = minv(main1,inverse1);



   //mprint(inverse1);


        /* Now the calibration values are computed, we need to use these coefficients to
get the
        3D world co-ordinates given the image co-ordinates
        */


        //construct a matrix which will give you the values of xg and yg



  double zg = -12.75;



        q_coeff_matrix[0][0] = coeff1k_matrix[0][0];
        q_coeff_matrix[0][1] = coeff1k_matrix[1][0];
```

```
        q_coeff_matrix[1][0] = coeff2k_matrix[0][0];
        q_coeff_matrix[1][1] = coeff2k_matrix[1][0];


  double test_x_image;    //crd1->pos_x

  double test_y_image;     //crd2->pos_y

  test_x_image = 226.00;

  test_y_image = 77.00;

  b_matrix[0][0] = test_x_image-coeff1k_matrix[3][0] - (coeff1k_matrix[2][0]*zg);

 b_matrix[1][0]=test_y_image - coeff2k_matrix[3][0] - (coeff2k_matrix[2][0]*zg);


  struct matrix q_coeff = {2,2,&q_coeff_matrix[0][0]};


  struct matrix q_inverse ={2,2,&q_inverse_matrix[0][0]};


 struct matrix convert = {2,1,&b_matrix[0][0]};


struct matrix world = {2,1,&world_matrix[0][0]};


  matrixptr q_coeffp = &q_coeff;

  matrixptr convertp = &convert;

  matrixptr worldp = &world;


        matrixptr q_inversep = &q_inverse;
        printf ("The coeffmatrix a11,a12,a21,a22 is:");
        mprint(q_coeffp);
        printf ("\n");
        getch();
        printf ("The conversion matrix B is:");
```

```
        mprint (convertp);

        minverse2 (q_coeffp,q_inversep);
        printf ("\n");
        getch();
        printf("The inverse is:");

        mprint(q_inversep);

        mmult (q_inversep,convertp,worldp);

    printf("\n");
        getch();
        printf(" The 3d world co-ordinates are:\n");


        mprint(worldp);
        getch();
        clrscr();




   // printf ("The coefficient matrix is %f",q_coeff_matrix[0][1]);

/*

        //This is a test for getting the determinant and the inverse
        double test_matrix[2][2];

        double inv[2][2];

    test_matrix[0][0]= 12.4;
        test_matrix[0][1]= -6;
        test_matrix[1][0]= 44;
        test_matrix[1][1]= 50;



        struct matrix test = {2,2,&test_matrix[0][0]};

        struct matrix invtest ={2,2,&inv[0][0]};


        matrixptr testp =&test;
```

```
        matrixptr invtestp = &invtest;
        double d;
        d=determin2(testp);
        printf("the determinant is %f",d);
        minverse2(testp,invtestp);
        printf("\n");
        mprint(invtestp);
*/
}
```

Appendix B Neural Networks:

```
%DEMOBP1 Training a nonlinear neuron.


clf;
figure(gcf)
colordef(gcf,'none')
setfsize(500,200);
echo on
clc

%    INITFF   - Initializes a feed-forware network.
%    TRAINBP  - Trains a feed-forward network with backpropagation.
%    SIMUFF    - Simulates a feed-forward network.

%    PATTERN ASSOCIATION WITH A LINEAR NEURON:

%    Using the above functions a neuron is trained
%    to respond to specific inputs with target outputs.

pause % Strike any key to continue...
clc
%    DEFINING A VECTOR ASSOCATION PROBLEM
%    ====================================

%    P defines two 1-element input vectors (column vectors):

load v.m
for I =1:900
v(I,1) = v(I,1) + 0.1;
end
P=v;

%    T defines the associated 1-element targets (column vectors):

T = [0 0 1 0 0; 0 1 0 0 0;0 0 0 0 1;0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 1 0 0;1 0 0 0 0; 0 0 1
0 0 ; 0 0 0 1 0]';

pause % Strike any key to see the error surface...
clc
%    PLOTTING THE ERROR SURFACE AND CONTOUR
%    ======================================

%    ERRSURF calculates errors for a neuron with a range of
%    possible weight and bias values.  PLOTES plots this error
%    surface with a contour plot underneath.
```

```
wv = -4:0.4:4;
bv = -4:0.4:4;
es = errsurf(P,T,wv,bv,'logsig');
plotes(wv,bv,es,[60 30]);

%    The best weight and bias values are those that result in the
%    lowest point on the error surface.

pause % Strike any key to design the network...
clc
%    DESIGN THE NETWORK
%    ==================

%    INITFF is used to initialize the weights and biases for
%    the LOGSIG neuron.
%

[w,b] = initff(P,T,'logsig')
echo off
k = pickic(1);

if k == 2
  w = -2.1617;
  b = -1.7862;
elseif k == 3
  subplot(1,2,2);
  h = centext('* CLICK ON ME *');
  set(h,'color',[0 0 0]);
  [w,b] = ginput(1);
  delete(h)
end

echo on
clc
%    TRAINING THE NETWORK
%    ====================

%    TBP1 uses backpropagation to train 1-layer networks.

df = 5;    % Frequency of progress displays (in epochs).
me = 100;  % Maximum number of epochs to train.
eg = 0.01; % Sum-squared error goal.
lr = 2;    % Learning rate.

%    Training begins...please wait...
```

```
[w,b,ep,tr] = tbp1(w,b,'logsig',P,T,[df me eg lr],wv,bv,es,[60 30]);

%    ...and finishes.

%    TRAINBP has returned new weight and bias values, the number
%    of epochs trained EP, and a record of training errors TR.

pause % Strike any key to see a plot of errors...
clc
%    PLOTTING THE ERROR CURVE
%    ========================

%    Here the errors are plotted with respect to training epochs:

ploterr(tr,eg);

pause % Strike any key to use the classifier...
clc
%    USING THE PATTERN ASSOCIATOR
%    ============================

%    We can now test the associator with one of the origonal
%    inputs, -3, and see if it returns the target, 0.4.

p = -1.2;
a = simuff(p,w,b,'logsig')
%    Training to a lower error goal would reduce this error.

%    Use SIMUP to check the neuron for an input of 2.0.
%    The target response is 0.8.

echo off
disp('End of DEMOBP1')
```

Reading in the test data:

Reading the test data into the matrix:

```
A1 = zeros (768,1024,3);[A1] = imread (‘track1.jpg’,’jpg’);
A2 = zeros (768,1024,3);[A2] = imread (‘track2.jpg’,’jpg’);
A3 = zeros (768,1024,3);[A3] = imread (‘track3.jpg’,’jpg’);
A4 = zeros (768,1024,3);[A4] = imread (‘track4.jpg’,’jpg’);
A5 = zeros (768,1024,3);[A5] = imread (‘track5.jpg’,’jpg’);
A6 = zeros (768,1024,3);[A6] = imread (‘track6.jpg’,’jpg’);
```

Resizing & Thresholding the TEST images:

```
B1 = im2bw (imresize (A1,[30,30],’bilinear’,0)0.5);
B2 = im2bw (imresize (A2,[30,30],’bilinear’,0)0.5);
B3 = im2bw (imresize (A3,[30,30],’bilinear’,0)0.5);
B4 = im2bw (imresize (A4,[30,30],’bilinear’,0)0.5);
B5 = im2bw (imresize (A5,[30,30],’bilinear’,0)0.5);
B6 = im2bw (imresize (A6,[30,30],’bilinear’,0)0.5);
```

Creating an array of TEST images:

```
B= [B1, B2, B3, B4, B5.B6];
```

Vectorization of TEST matrix:

```
Vt = zeros (900,6);
for I=1:6
for j= 1:30
D = zeros (30,1);D(j) =1;
C= Zeros (900,30);C(j-1)*30 +1: (j-1)*30 +30,1:30) = eye(30);
Vt(1:900,i)= Vt(1:900,i)+ C* (double (B(imslice[30,30,10],I)))*D);
end
end
```

TESTING THE NETWORK FOR GENERALIZATION :

Example 1:

This example shows the output to be a small left turn:

```
>>p =vt(:,3);
>>a= simuff(p,w,b,'logsig')

a=
   0.0000
   0.3321
   0.0000
   0.0003
   0.0000
```

Example 2:

This example shows the output to be a small right turn:

```
>>p =vt(:,2);
>>a= simuff(p,w,b,'logsig')

a=
   0.0110
   0.0000
   0.0000
   0.7126
   0.1929
```

Example 3:

This example shows the output to be a small right turn:

```
>>p =vt(:,4);
>>a= simuff(p,w,b,'logsig')

a=
   0.0001
   0.0000
   0.0001
   1.0000
   0.7625
```

Example 4:

This example shows the output to be a small right turn:

```
>>p =vt(:,1);
>>a= simuff(p,w,b,'logsig')

a=
   0.0005
   0.0000
   0.0200
   0.9998
   0.0003
```

Example 5:

This example shows the output to be straight for a small obstacle

```
>>p =vt(:,5);
>>a= simuff(p,w,b,'logsig')

a=
   0.0000
   0.0302
   0.8386
   0.0000
   0.0004
```

1999 International Ground Robotics Competition

Team	Heat 1 Adj Distance	Heat 2 Adj Distance	Heat 3 Adj Distance	Best Distance	Ran
University of Colorado at Denver – CUGAR	440	502.75	440	502.75	1
Hosei University	-5.5	384.5	56	384.5	2
Michigan Technological University - Veronica	-6	193		193	3
University of Cincinnati – Bearcat II	-4		153.3	153.3	4
University of Colorado Boulder –Robotic Autonomous Transport	6	115	-1	115	5
Wayne State University – Wayne Rover	11	-0.5	20.5	20.5	6
Virginia Tech University – Artemis	2	6.5	6	6.5	7
University of Alberta – Polar Bear	-1.5			-1.5	8
Virginia Tech University – Ivan				Demo Only	
Oakland University – Coyote					
University of Detroit Mercy – WHY2K					
University of Tulsa – Hurricane Ariene					

Appendix C

Design Competition Prize Money Breakdown

$1000 Virginia Tech – Artemis
$600 University of Alberta – Polar Bear
$400 University of Detroit Mercy – WHY2K

Follow-the-Leader

$1000 1st Place Virginia Tech – Artemis
(no $) 2nd Place Hosei University – Nectar
(no $) 3rd Place University of Colorado at Denver – CUGAR

Road Debris

$1000 1st Place Hosei University – Nectar
$600 2nd Place University of Colorado at Denver – CUGAR
$400 3rd Place University of Cincinnati – BEARCAT II

First to Run

University of Colorado at Boulder – RAT

TOP SCORERS IN DESIGN COMPETITION

WRITTEN REPORT

Conduct of Design Process	Va. Tech-Artemis
Completeness	Cincinnati
Quality of Writing	Alberta
Innovation	Va. Tech.-GNAT
Electronics, Software, Integration	Alberta
Safety, Reliability, Durability	U.of D.-Mercy
Overall-	Alberta

ORAL PRESENTATION

Organization	Colo. Denver
Graphic Aids	Va. Tech.-Artemis*
	Va. Tech.-GNAT*
	Va. Tech.-IVAN*
Articulation	Va. Tech.-Artemis*, Alberta*,
	Colo. Denver*
Response to Questions	Va. Tech.-Artemis*, Alberta*,

	Colo. Boulder*
Salesmanship	Cincinnati
Overall-	Va. Tech.-Artemis

EXAMINATION OF VEHICLE

Packaging Neatness	Va. Tech.-Artemis
Serviceability	Va. Tech.-Artemis*, Tulsa*, Colo. Boulder*
Ruggedness	Alberta*, Mich. Tech.*
Safety	Va. Tech.-Artemis
Original Content	Va. Tech.-GNAT
Style	Va. Tech.-Artemis*, Alberta*
Overall-	Va. Tech.-Artemis*, U.of D.-Mercy*

Grand Winner	Va. Tech.-Artemis

* Tie Scores