A Motion Control Algorithm for Steering an AGV in an Outdoor Environment

A thesis submitted to the Division of Graduate Studies and Advanced 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 2000

by

Ramesh Thyagarajan
B.S (Production Engineering), Osmania University, Hyderabad India

Thesis Advisor and Committee Chair: Dr. Ernest L. Hall
Abstract

The design of a mobile robot is a challenging task. A truly autonomous robot must be able to sense its environment and react appropriately. This issue becomes even more important if the environment is varying. Generally motion-planning algorithms for a mobile robot are developed based on the assumption that there are no changes in the surrounding environment. One of the goals in robotics is to endow robots with the ability to move and operate autonomously in an environment with unknown, perhaps moving obstacles. Robot navigation is described as the guiding of a mobile robot to a desired destination or along a desired path in an environment characterized by a terrain and a set of distinct objects, such as obstacles and landmarks. The robot must successfully navigate around obstacles, reach its goal and do so efficiently. Not only must a robot avoid colliding with an obstacle such as a rock, it must also avoid falling into a pit or ravine and avoid travel on terrain that would cause it to tip over. The Center for Robotics Research at the University of Cincinnati has built a small, unmanned, autonomous guided vehicle (AGV), named Bearcat II for the International Ground Robotics Competition being conducted by the Association for Unmanned Vehicle Systems (AUVS) each year. The objective is to follow an obstacle course 10 feet wide bounded by white/yellow/dashed lines. For the process of navigating safely through the obstacle-ridden path, the robot has a Vision System consisting of two CCD cameras mounted on its top, and an ISCAN image-tracking device. The three dimensional world co-ordinates are reduced to two-dimensional image co-ordinates by the transformations...
taking place from the ground plane to the image plane. For this purpose, the cameras
need to be accurately calibrated. In addition to this, the robot is provided with a rotating
ultrasonic transducer to detect the presence of obstacles. The main focus of this thesis is
to derive the logic that will enable data from these two systems to be fused together to
ensure that the robot successfully avoids obstacles while staying in its path at the same
time. This logic was tested and successfully proved at the 7\textsuperscript{th} International Ground
Robotics Competition held on June 7, 1999 at Oakland, Michigan.
Thus in short, using the integrated vision system, the vehicle senses its location and
orientation. A rotating ultrasonic sensor is used to map the location and size of possible
obstacles. With these inputs, the logic controls the speed and the steering decisions of
the robot. This logic helps us in obtaining the optimum speeds of the left and right drive
wheels that will help in robot navigation.
Acknowledgements

The work has been a continual learning experience and would not have been possible without the base provided by the various team members who worked and are currently working on Bearcat II as well as Bearcat I.

My deepest gratitude also go to my Thesis Advisor Dr. Ernie Hall for his valuable guidance, support and assistance in helping me conceive the idea of a motion control algorithm into its final form and inspiring me to finish this work.

I also express my thanks for the members of my thesis committee for their suggestions in this research work. Their advice was useful in giving the report its final shape.

My special thanks go to Karthik and Sameer, who have given me constant encouragement all along and who were always there to help me in learning about the functioning of the robot.

My heartfelt gratitude also go to my parents and family, who have always been a source of constant support and encouragement in my pursuit for achieving excellence in all my career goals.
# Table of Contents

Chapter 1  
Introduction  

1.1  Motivation  9  
1.2  Robot Vision  11  
1.3  Robot Obstacle Avoidance  15  
   1.3.1 Representation of Robot in Grid Environment  16  
   1.3.2 Determination of Direction of Obstacle  19  
1.4  Aim of the Exercise  20  
1.5  Design of the Robot  23  

Chapter 2  
Background and Previous Research  

2.1  Line Following Vision System  26  
2.2  Obstacle Avoidance System and Path Planning  28  

Chapter 3  
Vision Calibration  

3.1  Insight  33  
3.2  The Calibration Device and Procedure  35  
3.3  Experimental Results  
   3.3.1 Results of the calibration (right camera)  41  
   3.3.2 Results of the calibration (left camera)  45  

Chapter 4  
Obstacle Avoidance  

4.1  Sonar System  50  
4.2  Challenge  51  
4.3  Range Detection  52  

Chapter 5  
Motion and Control Logic  

5.1  Steering Mechanism  55  
5.2  Actual Logic  59  
5.3  Rotating Ultrasonic Transducer  63  
5.4  Extension of the Logic  64  

Chapter 6  
The IGRC Competition  

6.1  Review  72
6.2 Rules of the Contest 73
6.3 The 1999 Contest 74
6.4 The 2000 Contest: A preview 77

Chapter 7 Conclusions and Future Work 79

References 81

Appendix A (This appendix contains the codes relevant to the logic)

Appendix B (This appendix contains the results of the 1999 IGRC competition)
# List of figures

## Chapter 1

1.1  Representation of Environment in Grid Method  
1.2  Control Equipment Schematic of Bearcat II  
1.3  Various Sub-Systems of Bearcat II  

## Chapter 3

3.1  Representation of the Contest Track  
3.2  Calibration Device  
3.3  Environment (World) Co-ordinate Representation  
3.4  Image Co-ordinate Representation  
3.5  Maxim Video Switch Circuit Diagram  
3.6  Comparison Plots (X-axis) for right camera  
3.7  Comparison Plots (Y-axis) for right camera  
3.8  Correlation Plot (X-axis) for right camera  
3.9  Correlation Plot (Y-axis) for right camera  
3.10  Comparison Plot (X-axis) for left camera  
3.11  Comparison Plot (Y-axis) for left camera  
3.12  Correlation Plot (X-axis) for left camera  
3.13  Correlation Plot (Y-axis) for left camera  

## Chapter 4

4.1  Obstacle Avoidance System (Rotating Sonar)  
4.2  Control Schematic of Sonar Connections  

## Chapter 5

5.1  Steering Mechanism  
5.2  Alignment of Wheels in Motion  
5.3  SIMULINK Model  
5.4  Rotating Ultrasonic Sensor  
5.5  Simple Representation of Rotating Sonar  
5.6  Two Stationary Sonars  
5.7  Robot Navigation
Chapter 6

6.1 Bearcat II in IGRC Contest
Chapter 1

Introduction

1.1 Motivation

Manufacturing companies are taking advantage of automation technologies for achieving rapid increases in productivity. One of the key areas that is being exploited is robotics. The number of industrial robots in the United States was around 70,000 in the early 1990’s and is expected to reach around 90,000 by the turn of the century. But the number is very much less compared to Japan, which has almost 400,000 robots in use at present. The U.S industrial market was predominantly using robots manufactured by Cincinnati Milacron and Unimation in the early 1970’s. Presently, there are about 200 robot manufacturers in the United States. The automotive industry accounts for an estimated 70% of the industrial robots in use, followed by heavy machinery and aerospace industries, each representing approximately 10% of robot sales [1]. Because of the speed with which the robotics industry is developing, new technologies developed specifically for robots is expected to appear, particularly focussing on user-friendly languages, higher-level generative languages, standard protocol for interfacing and communications devices, easier off-line programming capabilities, and hardware and software modularity. Modular construction is the ultimate requirement of every user since it makes the robot easy to maintain. There are currently two basic types of robots: those that are mathematically based (which necessarily require digital computers as their base) and those that are not. The robots that were manufactured by some American
companies like Cincinnati Machines, Unimation and U.S Robotics offered the so-called “smart” technology. By smart technology, we mean the ability of the robot to plan its actions. But as the technology developed, many other small manufacturers started using the technology. Another way to classify industrial robots is based on their weight-carrying capability: light, medium or heavy. The light load robots are often known as assembly robots and are powered by electric motor drives or pneumatic drives. Medium load robots are either electrically or hydraulically driven while the heavy robots are predominantly hydraulically driven. In the future, robot control is expected to use a combination of sensory inputs and mathematically based controls. These techniques will enable a robot to identify objects and then carry out an instruction regarding what to do with such objects from large databases, such as those that created from CAD systems.

A justification for using robots is the potential they offer for enhancing productivity and decreasing cost. Another contributing factor is the capability they offer to increase quality and to do it consistently. But the most vital factor is the makeup of the robot itself. As a computer based product, it depends on a database, and is therefore capable of generating management reports, diagnostics, and, ultimately, linking it into other forms of computerized automation. If robotics is beginning to bloom, the vision is still in its seedling stage.

The design of a mobile robot is a challenging task. Generally motion-planning algorithms for a mobile robot are developed based on the assumption that there are no changes in the surrounding environment. One of the goals in robotics is to endow robots
with the ability to move and operate autonomously in an environment with unknown, perhaps moving obstacles.

1.2 Robot Vision

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 [2] (1990) pose three basic questions that define mobile robot mapping and navigation: They are as follows:

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

Humans acquire a lot of information through their vision system, which enables them to sense their environment and react accordingly. An intelligent machine such as an autonomous robot that must sense changes in its environment and act accordingly, must also be equipped with such a vision system that will help in collecting visual information and using this information to adapt to changes in its environment. When a robot tries to move, it must first make sure that it is free to move: that is, there is no obstacle in its path. This is the most important consideration when designing the vision system for a robot [3]. 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. For autonomous guided vehicles, by real time vision, we mean that the vision system must
allow the vehicle to anticipate any changes in its outdoor environment as and when they occur and make decisions accordingly. For a mobile robot, the vision system should not introduce a delay of more than 100ms in reporting an event in the environment or in providing data for visible motion [4]. A robot vision system uses a sensor sensitive to light, temperature, radiation and the like, that transmits a signal from the environment to a measuring or control device. The robot vision system actually has an input device such as one or more solid-state camera sensors. The input video signal is digitized via a data acquisition and conversion system and is transferred to a computer as an image. A computer treats the visual image as an array of pixels. A typical scene may consist of a number of pixels varying in number from 128 by 128 = 16,384 to about 1000 by 1000 = 1 million. Thus the time taken to analyze a single picture varies from a few seconds to several minutes depending on the volume of data. In order to speed up the response time, several methods have been suggested. One of the methods is to illuminate the scene so that the objects appear as black and white silhouettes. Another is to ensure that no two objects of interest overlap. But even under such artificial circumstances, robot vision is a very complex problem and subjected to many difficulties [5].

Vision systems in industry are generally used for the following functions [5]:

- Identification
- Classification
- Sorting and Measuring
- Inspection
- Verification
- Quality Control
Forging such as for Turbine Blades

The six basic areas of a robot vision system for industrial machine vision are the following [5]:

- Lighting and Optics
- Video Imaging
- Video Image Digitizer
- Window Comparator
- Machine Control Logic
- Operator Interfacing

There are several factors affecting the functioning of outdoor mobile robots. Some of them include [6]:

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

A mobile robot should possess the ability to navigate or guide itself on all terrains like concrete, grass, sand etc. with equal ease. The vehicle should maintain the same functional efficiency both indoors and outdoors.

Another important factor for the navigation of autonomous systems is the variation of environment resulting from the illumination. The vision may system may not be able to maintain the same level of performance during daytime and periods of darkness.
In order for the desired features to be identified correctly, the threshold of the perception system needs to be adjusted properly. The threshold is susceptible to changes in the lighting. Also, missing or obscured line markers make driving an autonomous system difficult even with sufficient lighting.

An additional important feature to be considered is the processing ability of the supervisory control computer. Adequate computer hardware is a key to practical robot vision. It has been found that multiprocessor systems containing a small number of processing element works better than expensive computer systems in robot vision applications. Flexibility is an important factor. The system must also be flexible to restructuring under software control to match the inherent structure of the vision task [7,8].

During motion of the mobile robot, the amount of time for processing the information received from the sensors is very limited. Also the amount of data received and waiting to be processed is enormous. Therefore the speed of the front-end processing system should be such that the vehicle reacts quickly to changes in environment. If the information is not processed fast and quick decisions are not made, then there is every chance that the robot may stray out of its path or collide with an obstacle.

Hall discusses the fundamental theorem of robot vision [9]. The manipulation of a point in space ‘\(x_1\)’ by either a robot manipulator that moves it to another point ‘\(x_2\)’ or through a camera system that images the point onto a camera sensor at ‘\(x_2\)’, is described by a matrix transformation of the form:
\[ x_2 = T x_1 \]  

(3.0)

The transformation matrix ‘T’ can describe the first-order effects of translation, rotation, scaling, parallel and perspective projections. The robot vision theorem suggests that sensing of a point or a collection of points on an object has some relation. In order to exploit this relation to the fullest, the camera calibration has to be done as accurately as possible [10].

1.3 Robot Obstacle Avoidance

In order to carry out intelligent tasks, an autonomous mobile robot must be capable of perception, decision-making, planning, navigation and control. For the successful application of a mobile robot to real time situations, the most important consideration lies in real time obstacle avoidance and navigation. In order for this to be done effectively, the problem lies in designing a system that is capable of processing large amounts of data very quickly to enable the robot to act according to the required situation. Various methods have been proposed to answer the problem of obstacle avoidance: some of the most common and popular being the Potential Field Methods (PFM’s) and the Grid Methods. Other methods include the Vector Field Histogram (VFH) Method developed by Borenstein in which the obstacle is represented in the form of grids. This method permits the detection of unknown obstacles and avoids collisions while simultaneously steering the robot towards the target. This method was developed from the certainty grids method previously developed by Moravec and Elfes.
and uses a 2-dimensional Cartesian histogram grid as a world model. It has been noticed
that the mobile robots navigating obstacle courses using the VFH methods perform very
well. It has been noted that the VFH system needs a fine-tuned threshold only for most
challenging applications (e.g., travel at high speed and in densely cluttered
environments); under less demanding conditions, the system performs well even with an
imprecisely set threshold. But there are some inherent problems in grid methods. The
major problem is the fact that the information needed for environment-representation
will continue to increase in order to satisfy the high resolution of environmental
representation when the robot is in motion [11].

For any robot, the characteristics of the sensors strongly determine the
representation of the obstacles. The use of sonars using the conventional time-of-flight
(CTOF) approach has been widespread for the navigation of mobile robots, due to their
low purchase cost, easy availability, and the ability to provide information about
volumes of space due to its beam width. But they face the problems of poor angular
resolution, limited range resolution, specular reflections, and frequent misreadings due
to external ultrasound sources or cross talk.

1.3.1 The Representation of the Environment in the Grid Method

In the grid method, a two-dimensional Cartesian histogram grid represents the
obstacles. In such a representation, the confidence of existence of obstacles in the event
that there is an obstacle at that location is expressed in terms of a certainty value (CV).
Each cell in the histogram grid holds a certainty value. A probability profile is projected onto a cell in the histogram grid by a range reading. For an ultrasonic sensor, this cell lies on the acoustic axis of the sensor and corresponds to the (measured) distance from the robot to the obstacle. Continuous and rapid sampling of each sensor helps in obtaining a probabilistic distribution of the presence of obstacles. Thus, the same cell (which maps the obstacle) and its neighboring cells are updated. Thus high CV’s are obtained in cells close to the actual location of the obstacle. This is referred to as a “histogrammic probability distribution”. The performance of the control algorithm is directly affected by the cell size. The ability to find a free path in a cluttered environment is best if the cell size is small. This is because a smaller cell size results in a higher resolution representation of environment and hence better detection of obstacle presence. But it suffers from several drawbacks: the weak capability of resistance to disturbances, the large amount of information representing the environment and the long and computationally intensive procedure of making a decision based on the results. The size of a cell in the histogram grid depends on the properties of the sensors. Better sensors mean lower cell size. The cell size can be smaller if the sensors are quick and accurate at sampling the environment, i.e. their response time is less. Else, the size of the cell will be large. In this approach, a detailed description of the robot’s environment is contained in an environment co-ordinate system. The two-dimensional Cartesian grid is continuously updated in real time with range data sampled by the onboard range sensors [11]. The histogram grid is independent of the momentary location of the mobile robot. Figure 1.1 below gives a representation of the mapping of the measured points corresponding to obstacles onto the environment co-ordinate system.
Using the equations (1.1) and (1.2) below, we can obtain the points on the environment co-ordinate system for the measured points corresponding to the obstacles [11].

\[ x_c = x_r + d_0 \cos(\alpha_r + \alpha_i) \]  \hspace{2cm} (1.1)
\[ y_c = y_r + d_0 \sin(\alpha_r + \alpha_i) \]  \hspace{2cm} (1.2)

Fig 1.1: Representation of Environment in Grid Method [11]
By using equations (1.3) and (1.4), the co-ordinate \( \left( x'_c, y'_c \right) \) can be mapped into a cell \((i, j)\) at that location in the environment co-ordinate system [11].

\[
x_c = \text{INT} \left( \frac{x'_c}{w} \right) \ast w + \text{INT} \left( \frac{w}{2} \right) \quad (1.3)
\]

\[
y_c = \text{INT} \left( \frac{y'_c}{w} \right) \ast w + \text{INT} \left( \frac{w}{2} \right) \quad (1.4)
\]

where \((x_c, y_c)\) = Co-ordinate of the cell \((i, j)\) in the environment co-ordinate system.

\( w \) = Width of the cell \((i, j)\)

### 1.3.2 Determination of the direction of obstacle

In the grid method, there is a notional window that moves along with the robot and this window tracks a sector of radius ‘R’ in the environment co-ordinate system. This region is called as the ‘movable window’ and the cell that momentarily belongs to this window is called as the window cell. A procedure similar to this is being implemented for obstacle avoidance in the Bearcat II robot built at the Center for Robotics Research in the University of Cincinnati (details later). Only the window cells have an immediate influence on robot control. The center of this movable window corresponds to the geometric center of the robot in the case of a symmetrically shaped mobile robot. In the case of rectangular mobile robots, the point from which the
distance to the robot’s front edge equals its distance to each side of the robot defines the center of the movable window. A window co-ordinate system is set up at the movable window with its center defined at the center of the window. The direction of the x-axis is defined as the direction from the window center to the front of the robot, looking straight ahead. To facilitate easy and accurate detection of obstacles, the movable window is divided into ‘n’ angular sectors of width $\theta$. The value of ‘$\theta$’ may be chosen arbitrarily, but the value of $n=180/\theta$ should be an integer. It has been determined that ‘$H_k$’ represents the obstacle density in each sector ‘$k$’ in the direction of ‘$k$’ [11].

1.4 Aim of the Exercise

The objective of this study is to describe a motion control algorithm for the steering of an autonomous guided vehicle (Bearcat II), developed for the International Ground Robotics Competition conducted by the Association for Unmanned Vehicle Systems (AUVS) that was held in June 1999. The purpose is to describe a high-level path planning logic, which processes the data from the vision systems and an ultrasonic obstacle avoidance system and steers an autonomous mobile robot between obstacles. The test bed is the autonomous robot built at University of Cincinnati, and this logic was tested and debugged on this machine. Attempts have already been made to incorporate fuzzy systems on a similar robot, and this paper extends them to take advantage of the robot’s extra sensor capability. The Center for Robotics Research is involved in the process of building an autonomous robot that can navigate itself in an
outdoor obstacle course. Solid and dashed lines ten feet apart, and 4 inches wide bound the obstacle course that the robot is supposed to follow. The course can assume different shapes such as the shape of the islands of Japan. The robot has several unique and outstanding features. It is strictly built off-the-shelf components and is a fully developed technological kit almost ready for immediate use. The Control Equipment Schematic for Bearcat II is shown in the Figure 1.2.

![Control Equipment Schematic of Bearcat II](image)

**Fig 1.2: Control Equipment Schematic of Bearcat II**

The important components of the mobile robot include the following:
- **Vision Guidance System**, which includes two JVC CCD cameras [12] mounted on top of the robot and a vision tracking I-Scan equipment [13].

- **Steering Control System**, which includes two 40:1 gear boxes, two pairs of flexible couplings and a single outdoor castor wheel at the rear for easy maneuverability.

- **Obstacle Avoidance System**, which consists of a rotating sonar (making mirror stops on each side) mounted on the front of the robot.

- **Galil Drive System**, to drive the motors, run the robot and help in the functioning of the rotating sonar [14].

- **Safety and Emergency Stop System**, consisting of the remote E-Stop.

- **Other Components**, which include the speed control system, power unit and a supervisor controlled personal computer. This computer is the heart of the mobile robot and has the logic stored in it.
1.5 Design of the Robot

The robot structure is built of commercially available 80/20 Industrial Erector Set. The drive system for the robot includes two independently driven 24 volt, 12 amp motors. These motors are connected to the drive wheels through two gearboxes having a gear ratio of 40:1. This is meant to increase the speed of the robot by about 40 times compared to the speed of the motor. The motors derive power from an amplifier, which amplifies the signal from a Galil DMC motion controller. In order to complete the control loop, there is an encoder attached to the shafts of each drive motor. The encoder signal is numerically differentiated to provide a velocity feedback signal. There is a castor wheel at the rear of the robot, which is free to swing when the robot negotiates a turn.

The other notable features of this robot include the “Modular Design”, i.e., the design of each part has been done in such a way that it can be modified at any stage of the life of the robot without much problem. Another feature is the ability of the robot to move with equal ease in both the directions. But without doubt, the most important feature is the ability of the robot to turn about its radius. This ability is called the Zero Turning Radius (ZTR) and is the most important asset of design. This feature provides exceptional maneuverability as it enables the robot to make sufficiently sharp turns with great ease. Rotating one wheel forward and other wheel backward generally accomplishes the ZTR function. In order to negotiate a curve, we vary the speeds of the left and right drive wheels. This enables the robot to make a curved turning path parallel
to the track lines. Other notable features include the presence of the rotating ultrasonic transducer (to help in accurate obstacle detection) and the relatively low cost of manufacturing.

This thesis describes an algorithm for steering the robot in the obstacle-ridden path. In order for this to happen, the information from the vision guidance system as well as the obstacle avoidance system needs to be taken into consideration simultaneously. Using the integrated vision system, the vehicle senses its location and orientation. A rotating ultrasonic sensor is used to map the location and size of possible obstacles. With these inputs the fuzzy logic controls the speed and the steering decisions of the robot.

As previously mentioned, the vision system of the robot consists of two CCD cameras and an image-tracking device for the front end processing of the image captured by the camera. Using the CCD camera as the medium, the three dimensional world co-ordinate system is reduced to a two-dimensional image co-ordinate system. Information about the 2-dimensional image co-ordinates is easily obtained from the image system. After getting the information regarding the image co-ordinates, at any time, the challenge is to extract the three dimensional information from them. Mathematical as well as geometrical transformations occur via the camera parameters in transforming the 3-dimensional co-ordinate system to a 2-dimensional co-ordinate system. Knowledge of these mathematical and geometric relations will help us in determining a 3-dimensional co-ordinate on the line from its corresponding 2-dimensional image point. To establish these mathematical and geometric relationships, the camera calibration has to be done. If the camera is accurately calibrated, accurate
measurement of the co-ordinates of the point on the line with respect to the robot can be done.

The specific goals of this thesis include:

- Give an overview of the 4-point calibration method for calibrating the cameras of Bearcat II (Vision System).
- Give an introduction to the obstacle avoidance system using the rotating ultrasonic transducer.
- Finally, propose an algorithm integrating the output from these two systems to safely navigate a robot through an obstacle-ridden path.
- Experimentally measure the effectiveness of the algorithm.
Chapter 2

Background and Previous Research

In order that the algorithm work perfectly, the vision systems and the obstacle avoidance system should work faultlessly. For accurate functioning of the vision system, the camera calibration should be done as accurately as possible. Another aspect is the effective functioning of the rotating sonar (obstacle avoidance system). This section lists some background in these areas.

2.1 Line Following Vision System

Calibration of the camera, as mentioned before, is needed for obtaining the 2-dimensional image co-ordinates of a 3-dimensional point in the environment. It means determining the geometric properties of the imaging process, i.e., the transformation that maps a 3-dimensional point, expressed with respect to a reference frame onto its 2-dimensional image whose co-ordinates are expressed in pixel units. The knowledge of the imaging parameters allows us to relate the image measurements to the spatial structure of the observed scene [16]. It is known that camera calibration is a complex problem because of the following reasons:

1. All the intrinsic and extrinsic parameters should be computed from the two-dimensional projections of a limited number of feature points. The intrinsic
parameters include the internal parameters of the camera including the optical and mechanical (geometrical) properties, such as focal length, lens distortion parameters, the intersection point of the optical axis with the image plane etc. These parameters come adjusted when the camera is purchased, but for further accuracy, this exercise needs to be done all over again depending on the required performance. The estimation of the location of the system relative to the 3-dimensional world reference system, including rotation and translation between these two systems (the so-called extrinsic parameters) is required.

2. The parameters are inter-related. A small change in one of the parameters will cause the values of others to change. Hence this is basically a trial-and-error method, which needs to be done iteratively.

3. The formulation is non-linear due to the perspectivity of the pinhole camera model.

According to Hong, et al [17], two points should be considered during camera calibration. They are: (1) reducing the location error of image features as far as possible, by exploiting image processing technique, and (2) compensating system error by the optimal pattern of approximating residual error of image points, namely the posterior compensation of the system error.

It is to be kept in mind that the camera system is subjected to external influences such as vibrations, thermal expansion etc. which necessitate the calibration process to be performed repetitively. Moreover it is possible that the parameters of the vision system such as the focal lens, zoom etc. be altered intentionally in order to perform specific tasks.
2.2 *Obstacle Avoidance System and Path Planning*

Generally motion-planning algorithms for a mobile robot are developed based on the assumption that there are no changes in the surrounding environment. One of the goals in robotics is to endow robots with the ability to move and operate autonomously in an environment with unknown, perhaps moving obstacles. Robot navigation is described as the guiding of a mobile robot to a desired destination or along a desired path in an environment characterized by a terrain and a set of distinct objects, such as obstacles and landmarks. The robot must successfully navigate around obstacles, reach its goal and do so efficiently. Not only must a robot avoid colliding with an obstacle such as a rock, it must also avoid falling into a pit or ravine and avoid travel on terrain that would cause it to tip over.

The purpose of a path planner is to compute a path, i.e., a continuous sequence of configurations. The primary concern of path planning is to compute collision-free paths. Another key issue is the uncertainty problem. It is related to various sources of uncertainty affecting the actual robot (control and sensing errors, inaccurate models of the environment, etc.). Approaches to path planning for mobile robots can be broadly classified into two categories - those that use exact representations of the world and those that use discretized representations. The main advantage of discretization is that adjusting the cell size can control the computational complexity of path planning. In contrast, the computational complexity of exact methods is a function of the number of obstacles and/or the number of obstacle facets, which we cannot normally control [18].
Some approaches have been proposed to steer the robot in order to stay clear of some close obstacles, but they fail to plan maneuvers if they are necessary. In order to solve this problem, Laumond has proposed a three-stage algorithm. This method relies on seeking a collision free path for a holonomic robot and rebuilding it in order to take into account the nonholonomic (a wheel is a nonholonomic system; it can only move in a direction perpendicular to its axle) constraints. In this way a feasible path is created. During the last stage, the path is optimized. Using the same idea, the most widespread procedure is collision detection. In order to simplify it, usually the configuration space approach is used, which allows us to take into account the robot size [19].

Horacio et al. [20] (1998) describe the development and implementation of an automatic controller for path planning and navigation of an autonomous mobile robot using simulated annealing and fuzzy logic. This paper demonstrates the simulated annealing algorithm being used to obtain a collision-free optimal trajectory among fixed polygonal obstacles. The trajectories were represented by B-spline curves, the working space was represented by C-space and simulated annealing was used to obtain the optimal path. The trajectory tracking was performed with a fuzzy logic algorithm. The objectives of the control algorithm were to track the planned trajectory and to avoid collision with moving obstacles. The results of simulation and implementation of the logic are given.

A paper by Vanualailai et al., [21] describes a solution to the path-planning problem (also known as the findpath problem) by using the second or direct method of Liapunov (using the Liapunov function). This method is used to construct control functions for the collision avoidance between two point masses, which are required to
move to designated areas or targets located in the horizontal plane. It also presents two
new results: the possibility of analyzing in a more effective manner, the dynamics of
more than two point masses and a discussion about other important collision avoidance
issues like improving collision avoidance between objects, having the optimum time to
reach a target safely etc.

A paper presented by Lin et al., [22] describes the fuzzy logic approach to the
problem of obstacle avoidance. The paper presents some difficult issues concerning
AGV navigation, like *sensor modeling* (which relates to finding the minimum number
of sensors and their optimal arrangement on the AGV, so that views of all angles can be
seen by the AGV) and *trap recovering* (in which fuzzy logic and crisp reasoning are
combined to guide an AGV to get out of a trap). The paper presents some simulation
results to show the feasibility of the proposed approach. They propose a fuzzy
navigation algorithm consisting of four fuzzy control modules:

- Static obstacle avoidance and target directing module.
- Trap detection module
- Trap recovery module
- Moving obstacle avoidance module

In their paper presented in 1997, Springer et al., [23] examine simple strategies
for the problems of collision-avoidance in the robot soccer competition. They discuss
the strategies for guidance of the robot to capture the ball and for obstacle and
constraint avoidance. These strategies are described to have been developed in such a
way that it is possible for computationally efficient code (based on potential field
techniques) to be written for their implementation on the micro-robot. They assume that
the obstacles exert a repulsive force on the robot while its target exerts an attractive force pulling the robot towards its destination. The magnitude of the forces is taken to be proportional to the distance of the robot from the object. The path followed by the robot is along the direction, which corresponds to the vector sum of the attractive and repulsive forces. They describe three obstacle avoidance methodologies:

- **Global obstacle avoidance**: using on the overhead vision system to provide positional data on every obstacle in the robot’s world.

- **Local obstacle avoidance**: using infrared sensors to detect obstacles in the local environment.

- **Regional obstacle avoidance**: using a combination of the overhead vision system and dead reckoning navigation to determine the robot’s proximity to the limits of the operational region.

A paper by Jorg et al., [24] describes the experimental results of a new approach, which allows one to operate a set of sonar sensors simultaneously without the conventional problems associated with the normal CTOF sonar sensing. This paper mentions that using CTOF sonar sensing, two or more consecutive objects cannot be distinguished if they are so closely spaced that their individual echoes overlap. The limited range of resolution depends on the width ‘T’ of the emitted burst. The echoes overlap if the relative distance ‘Δd’ of the targets is smaller than cT/2 (where c = Speed of sound). In order to overcome this ‘T’ must be made small. To eliminate interference, ‘Δd’ between two targets must be greater than cT/2. Assuming the speed of sound to be 33cm/ms, both echoes do not overlap if Δd > 16.5cm for T = 1ms. But this approach has its own inherent problem. Reducing the duration ‘T’ of the burst improves the sensor’s
range resolution but reduces the sensor’s maximum range resulting from the lower echo energy level. This approach eliminates the frequent misreadings caused by cross talk or external ultrasound sources, and both, range resolution and lateral resolution of the sonar sensor are increased. This is made possible by carefully designing the emitted bursts, i.e., by using appropriate pseudo-random sequences together with a matched filter receiver used for radar applications. The paper presents some results from physical experiments.
Chapter 3

Vision Calibration

3.1 Insight

This chapter gives a brief overview about the calibration device, the four-point calibration method. As mentioned previously, in order for the robot to avoid obstacles and move efficiently, it must be ensured that the vision system must function effectively.

The main purpose of the vision system is to ensure that the robot tracks the line properly using the camera and remain in its path. In order to obtain accurate information about the position of the line with respect to the robot, the distance and orientation of the line with respect to the robot are to be known. Once this is known, camera calibration ensures that the 3-D world co-ordinates are transformed into 2-D image co-ordinates, which appear on the monitor. The CCD camera maps the line from a 3-D world co-ordinate system into a 2-D image co-ordinate system. In the context of 3-D machine vision, camera calibration 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 co-ordinate system (extrinsic parameters) [6].
The robot is designed in such a way that it is able to navigate itself between two lines, which are 10 feet apart. The thickness of the lines is about 4 inches. The appearance of the lines may vary from being solid to dashed to invisible for a certain region. When such a situation occurs, switching of the cameras is achieved through a maxim video switch. This ensures that the other camera now tracks the line and helps maintain the robot in its path. Measurements are referenced to the robot centroid as a global co-ordinate system. For navigation, the cameras must be located with respect to this centroid. A model of the track is given in the Figure 3.1 below:

![Figure 3.1: A representation of the contest track][10]

The functioning of the vision system is as follows: The vision system uses 2 CCD cameras and an image-tracking device (ISCAN). The ISCAN image tracking system finds the centroid of the brightest or the darkest region in an image and returns the co-
ordinates of these points, which are the image co-ordinates. These co-ordinates are the two-dimensional representation of the three-dimensional world co-ordinates. An algorithm needs to be developed to establish a relationship between the two sets of co-ordinates. The real challenge in calibration is to find the 3-dimensional co-ordinates from knowledge of the two-dimensional image co-ordinates. What makes the task further complicated is that the dynamic nature of the environment.

In order to set up the co-ordinate system for the robot, the global co-ordinates are first translated to the center of the image plane, then rotated counter clockwise about the x-axis by the tilt angle ‘\( \theta \)’, then rotated counter clockwise about the z-axis by the pan angle ‘\( \phi \)’ and then mapped through a perspective transformation with lens center located at \( y = y_0 \), and then finally projected on to the x-z plane. These parameters are difficult to measure practically [6].

### 3.2 The calibration device and procedure

The calibration device is shown in Figure 3.2. It consists of two precisely machined metallic plates (capable of being placed on an optical bench) which are 10” x 6” x 0.5”. There is a steel shaft at each corner, which is 1” in diameter and 5” in length. The shafts are surface ground and securely fixed to the two plates. The metallic plates are coated with a black anodized material to ensure constant dimensions and minimum reflectance. The upper plate has a grid of holes 1” apart. Using metallic device gives better accuracy. There are 4 aluminum spheres on the top plate, each 1” in diameter.
Two of these spheres are placed on the same plane (the upper surface) while screws fixed to the upper plate (upside down) support two other spheres. They are raised by 0.5625” by means of metallic bushings. This ensures that all the spheres do not lie on the same plane. In order for calibration to be accurate, accurate measurements of the exact co-ordinates of all the four points with respect to the reference point is essential.

![Fig 3.2: The Calibration Device (Top View) [6]](image)

To obtain the desired accuracy, we utilize a Co-ordinate Measuring Machine (CMM), whose accuracy is ten thousandth of an inch, to measure the centroid of the four balls with respect to the tip of the corner of the lower metallic plate. In order to obtain 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 the corner of the lower base and the
centroid of the robot is carefully measured. The order of points is clockwise for both the left and the right camera (starting from the sphere nearest to the camera). Both the cameras are calibrated independently. The most important part of the vision calibration is that we need to get a good “Vision Window”. The tilt and pan angles of the cameras are to be adjusted in such a way that we get a good view of the line and sufficient area on both sides. Also for best results, it is to be ensured that the line is approximately at the center of the vision window. The main reason for doing this is that at this position, the line is parallel with respect to the robot. The vision algorithm has been designed in such a way that the robot always maintains the ideal position with respect to the lines, i.e., the robot always aligns itself parallel to the lines and at the center of the two lines. The next task is to set the window limits on either side of the line. When the robot is run in its auto mode, at any time there are 2 I-Scan windows are formed at the top and bottom of the image screen, which flicker back and forth. The size of these windows can be adjusted in the program software. The X and Y image co-ordinates of these windows are sent to the program and using the calibration co-efficients, their real X and Y co-ordinates are calculated. These values are fed to the vision control program to get the actual distance and angle of the line from the robot [6]. A view of the world co-ordinates and the corresponding image co-ordinates is shown in Figure 3.3 and Figure 3.4 respectively. It is to be noted that when the calibration is being done, the robot must be positioned in such a way that its center aligns with the center of the track. If no object is found, the ISCAN generates a loss of track (LOT) signal.
This information is updated every 16msec. However the program must wait for 10ms after moving the window to get the new data. This results in a 52msec update time for the vision system.

Fig 3.3: Actual environment of robot motion (3-D co-ordinates) [25]

Fig 3.4: 2-D image co-ordinates corresponding to the 3-D world co-ordinates [25]
During the calibration process, it is necessary to measure $X_0$, $Y_0$ and $Z_0$ where:

$X_0$ = Distance between the X-axis center and the bottom tip of the calibration device, which is placed on the white line measured along the X-axis (across the track).

$Y_0$ = Distance between the Y-axis center and the bottom tip of the calibration device, which is placed on the white line measured along the Y-axis (parallel to the track downward).

$Z_0$ = Distance of the top of the spheres on the calibration device from the base of the calibration device (or from the floor).

Also the size of the window must be approximately 4’ x 4’, and the threshold must be set to “bright”, indicating that we are interested in setting the threshold for the bright points on the vision window.

The cross hairs must be positioned on the first sphere and the program “g-cart” from the computer must be executed. The option “Test the Vision System” is selected from the menu and the ‘X’ and ‘Y’ co-ordinates for the sphere are noted down. The same procedure is repeated for all the four spheres. It is better to take the average of two or three readings in order to overcome observation errors. A sample table for calibration of the cameras is shown below: [6]

<table>
<thead>
<tr>
<th>Sphere</th>
<th>X value</th>
<th>Y value</th>
<th>Z value</th>
<th>$X_i$</th>
<th>$Y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Here $X_i$ and $Y_i$ are the physical co-ordinates corresponding to the measurements. When the real world co-ordinates of the spheres ($X_g, Y_g, Z_g$) and their image co-ordinates are entered in vision program, we get the output, which are the values for the camera calibration co-efficients [6]. It is to be noted that if one of the cameras loses track, then switching of cameras occur. This helps the robot maintain its path even if one of the cameras loses track. The switching of cameras is achieved by using a maxim video switch. The circuit diagram for the video switch is shown in Figure 3.5.

![Circuit Diagram of the Video Switch using for switching cameras](image)

Fig 3.5: Circuit Diagram of the Video Switch using for switching cameras [26]
3.3 Experimental Results

3.3.1 Results of the calibration (right camera)

This is the data set for the calibration of the right camera done on April 28, 2000. It lists the physical (actual ground) co-ordinates measured from the X, Y, Z centers and the measured image co-ordinates. Inserting the image co-ordinates into the vision program, we compute the ground co-ordinates and compare it with the actual ground co-ordinates. Perfect agreement (or near perfect agreement) between these values is the aim of this exercise.

\[ X_0 = \text{Distance of the X-center from the edge of the calibration device} = 57 \text{ inches} \]
\[ Y_0 = \text{Distance of the Y-center from the edge of the calibration device} = 27.0625 \text{ inches} \]
\[ Z_0 = \text{Distance of the top of the calibration device from the Z-center} = -13.5625 \text{ inches} \]

<table>
<thead>
<tr>
<th>Spheres</th>
<th>Actual Ground co-ordinates</th>
<th>Image co-ordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( X_g )</td>
<td>( Y_g )</td>
</tr>
<tr>
<td>1</td>
<td>58.00</td>
<td>33.0625</td>
</tr>
<tr>
<td>2</td>
<td>60.00</td>
<td>29.0625</td>
</tr>
<tr>
<td>3</td>
<td>62.00</td>
<td>31.0625</td>
</tr>
<tr>
<td>4</td>
<td>60.00</td>
<td>35.0625</td>
</tr>
</tbody>
</table>
Here $X_g$, $Y_g$, $Z_g$ are calculated by adding the distance of the edge of the calibration device from the $X$, $Y$, $Z$ centers to the distance of the respective spheres from the edge of the calibration device along the $X$, $Y$, $Z$ axes.

Once these values are obtained, we can then use the program to compute the camera coefficients. This is represented in the form of a $2 \times 4$ matrix $A_{ij}$. Using this matrix, we can then compute the ground co-ordinates from the image co-ordinates.

<table>
<thead>
<tr>
<th>Camera Coefficients for the right camera, $A_{ij}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-10.1667$</td>
</tr>
<tr>
<td>$0.9167$</td>
</tr>
</tbody>
</table>

$i = 1, 2$  \hspace{1cm}  $j = 1, 2, 3, 4$

The following table shows the ground co-ordinates computed by using the image co-ordinates in the vision algorithm. These values are then compared with the actual ground co-ordinates to obtain an idea of the accuracy of the calibration.

<table>
<thead>
<tr>
<th>Test Points</th>
<th>Actual Ground co-ordinates</th>
<th>Image co-ordinates</th>
<th>Computed co-ordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X_g$</td>
<td>$Y_g$</td>
<td>$Z_g$</td>
</tr>
<tr>
<td>1</td>
<td>58.00</td>
<td>29.0625</td>
<td>-13.5625</td>
</tr>
<tr>
<td>2</td>
<td>59.00</td>
<td>31.0625</td>
<td>-13.5625</td>
</tr>
<tr>
<td>3</td>
<td>60.00</td>
<td>32.0625</td>
<td>-13.5625</td>
</tr>
<tr>
<td>4</td>
<td>61.00</td>
<td>30.0625</td>
<td>-13.5625</td>
</tr>
<tr>
<td>5</td>
<td>62.00</td>
<td>33.0625</td>
<td>-13.5625</td>
</tr>
</tbody>
</table>
Comparison of X values (right camera)

X coordinate (inches)

Test Points

Comparison of Y values (right camera)

Y coordinate (inches)

Test Points

Fig 3.6: Comparison of X-values for the right camera

Fig 3.7: Comparison of Y-values for the right camera
Plots showing the comparison between the original ground co-ordinates (actual values) and the computed ground co-ordinates (as obtained from the results of the calibration) for the X-axis and the Y-axis (for 4 test points) are shown in figures 3.6 and 3.7. The plots show that there is a perfect correlation between these co-ordinates.

Another set of plots can be drawn to find the correlation between the actual and computed values for the X and Y positions for the right camera. The plots are shown in figures 3.8 and 3.9 respectively. Once again from these plots, it is seen that the slope of the curve between the actual and computed values is a straight line with slope of about 45 degrees. Hence there is perfect correlation between the values.

Fig 3.8: Correlation plot for right camera (X axis)
3.3.2 Results of the calibration (left camera)

This is the data set for the calibration of the left camera done on April 28, 2000.

\[ X_0 = \text{Distance of the X-center from the edge of the calibration device} = 57 \text{ inches} \]

\[ Y_0 = \text{Distance of the Y-center from the edge of the calibration device} = 27.75 \text{ inches} \]

\[ Z_0 = \text{Distance of the top of the calibration device from the Z-center} = -13.5625 \text{ inches} \]
The following table gives a set of experimental test points. A comparison between the actual ground co-ordinates and the computed ground co-ordinates is shown.

<table>
<thead>
<tr>
<th>Spheres</th>
<th>Actual Ground co-ordinates</th>
<th>Image co-ordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X_g$</td>
<td>$Y_g$</td>
</tr>
<tr>
<td>1</td>
<td>58.00</td>
<td>31.75</td>
</tr>
<tr>
<td>2</td>
<td>60.00</td>
<td>29.75</td>
</tr>
<tr>
<td>3</td>
<td>62.00</td>
<td>33.75</td>
</tr>
<tr>
<td>4</td>
<td>60.00</td>
<td>35.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Camera Coefficients for the left camera, $A_{ij}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 1, 2$</td>
</tr>
<tr>
<td>$j = 1, 2, 3, 4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Points</th>
<th>Actual Ground co-ordinates</th>
<th>Image co-ordinates</th>
<th>Computed co-ordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$X_g$</td>
<td>$Y_g$</td>
<td>$Z_g$</td>
</tr>
<tr>
<td>1</td>
<td>58.00</td>
<td>29.75</td>
<td>-13.5625</td>
</tr>
<tr>
<td>2</td>
<td>59.00</td>
<td>34.75</td>
<td>-13.5625</td>
</tr>
<tr>
<td>3</td>
<td>60.00</td>
<td>33.75</td>
<td>-13.5625</td>
</tr>
<tr>
<td>4</td>
<td>61.00</td>
<td>34.75</td>
<td>-13.5625</td>
</tr>
<tr>
<td>5</td>
<td>62.00</td>
<td>31.75</td>
<td>-13.5625</td>
</tr>
</tbody>
</table>
Plots showing the comparison between the original ground co-ordinates (actual values) and the computed ground co-ordinates (as obtained from the results of the calibration) for the X-axis and the Y-axis (for 4 test points) for the left camera are shown in figures 3.10 and 3.11. The plots show that there is a near-perfect correlation between these co-ordinates.

Another set of plots can be drawn to find the correlation between the actual and computed values for the X and Y positions for the right camera. The plots are shown in figures 3.8 and 3.9 respectively. Once again from these plots, it is seen that the slope of the curve between the actual and computed values is a straight line with slope of about 45 degrees. Hence there is perfect correlation between the values.
Comparison of Y values (left camera)

Fig 3.11: Comparison of Y-values for the left camera

Actual vs Computed Values

Fig 3.12: Correlation plot for left camera (X axis)
Fig 3.13: Correlation plot for left camera (Y axis)
Chapter 4

Obstacle Avoidance

4.1 Sonar System

In this chapter, we talk about the obstacle avoidance system set-up and working on the Bearcat II. For accurate path navigation, in addition to the proper functioning of the vision system, the obstacle avoidance system must also function to perfection.

Obstacle avoidance system consists of a single rotating transducer. Polaroid ultrasonic ranging system is used for the purpose of calibrating the transducer. An Intel 80C196 microprocessor and a circuit are used to process the distance calculations. The distance value is returned through a RS232 port to the control computer. A pulse of electronically generated sound is transmitted toward the target and the resulting echo is detected. The system converts the elapsed time into a distance value. The digital electronics generate the ultrasonic frequency. All the digital functions are generated by the Intel microprocessor. Operating parameters such as transmit frequency, pulse width, blanking time and the amplifier gain are controlled by software supplied by Polaroid.

The drive system for the transducer consists of a DC motor and its control circuitry. With this arrangement the transducer is made to sweep and angle depending on the horizon (range between which we need detection). The loop is closed by an encoder feedback from an encoder. The drive hardware comprises of two
interconnected modules, the Galil ICB930 and the 4-axis ICM 1100. The ICM 1100 communicates with the main motion control board the DMC 1030 through an RS232 interface. The required sweep is achieved by programming the Galil. Adjusting the Polaroid system parameters and synchronizing them with the motion of the motor maintain distance values at known angles with respect to the centroid of the robot [27].

4.2 Challenge

A view of the robot with the position of the rotating sonar is shown in Figure 4.1. The path of robot travel is marked by solid white lines (which can be dashed at certain regions). Sometimes the lines may get obscure or change color from white to green. The road direction may change but the width of the road is always approximately 10 feet. The obstacles are positioned on the track in such a way that they block the path of the robot. It is expected that there is always enough clearance that the robot can pass these obstacles without touching them or knocking them down. These obstacles may vary from 5 gallon buckets; construction barrels (aligned at an angle to the track) or it can be a sand trap and also a ramp (inclined at about 15 degrees). In short, the robot must be able to follow the changing track while avoiding obstacles at the same time. In order for this to happen, the robot must be designed in such a way that it can sense and deal with the issue of uncertainty and incomplete information about the environment in a reasonably short duration of time.
4.3 Range Detection

This section discusses the basic nature of relationships between the robot and the obstacle. Before the system makes any decision, it is important that we know the distance, width and shape of the obstacle. Depending upon these factors, the robot has

Fig 4.1: Robot with the rotating sonar on the track [28]
to make a decision as to whether it will go straight, turn left or turn right. Also it has to
decide upon the amount of turn depending on the nearness of the target to it. The
optimal angle of sweep per reading should be obtained in such a way that it does not
slow down the overall system performance. From the Figure 4.1, we can obtain the
value of the distance of the obstacle (L) from the robot center (O).

Another important thing to be known is the width of the obstacle. Assuming that
‘θ_F’ is the angle of the first sonar contact with the obstacle, ‘θ_L’ is the angle of the last
sonar contact with the obstacle, ‘θ_{F-1}’ is the angle just before the first contact with the
obstacle, ‘θ_{L+1}’ is the angle just after the last contact with the obstacle, we can get the
value for the width of the obstacle by the difference in the width of the two angles. The
sonar uses the “Time of Flight” approach to detect these angles.

Once the values of ‘θ_F’ and ‘θ_L’, we can estimate the direction of the obstacle
with respect to the robot. Three possibilities arise:

- θ_L < 90° and θ_F < 90°; this is an indication that the obstacle is to the right.
- θ_L > 90° and θ_F > 90°; this implies that the obstacle is to the left.
- θ_L < 90° and θ_F > 90°; this implies that the obstacle is straight ahead.

Depending on the cases above, the robot makes an effort to avoid the obstacle, while
simultaneously making sure that it stays inside the track. This is an overview of the
range detection method. For a detailed and complete discussion regarding range
detection, it would be worth while to refer to the paper by Chiang et al. [28]. Fig 4.2
below shows the schematic of the sonar connections from the Galil DMC controller.
Fig 4.2: Control Schematic of the Sonar Connections
Chapter 5

Motion Control and Logic

5.1 Steering Mechanism

The motion control of the AGV designed, has the capability of Zero Turning Radius (ZTR), which is gaining popularity and expanding commercially in the U.S. mowing market. ZTR is the ability to make a turn about the center of the axis of the drive wheels. This unique design offers exceptional maneuverability such as sharp turns or turning in place. Figure 5.1 below shows the steering mechanism of the mobile robot.

Fig 5.1: Steering Mechanism of Bearcat II
Rotating one wheel forward and the other wheel backward accomplishes the ZTR function. However, in our design we can also 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. One important factor to note is that the wheels do not steer. They negotiate a turn by only changing their individual speeds.

The AGV is driven and steered by two independent 48 volt, 15 amp motors. These motors drive the left and right drive wheels, respectively, through two independent gearboxes, which increase the motor torque by forty times. The power to each individual motor is delivered from an AMC DC 48A amplifier, which amplifies the signal from the Galil DMC motion controller. To complete the control loops; a position encoder is mounted on each of the drive motors.

![Fig 5.2: Position of the drive wheels and castor wheel](image_url)

Fig 5.2: Position of the drive wheels and castor wheel [25]
There is an outdoor castor wheel in the back of the vehicle, which is free to swing when the vehicle has to negotiate a turn. Figure 5.2 gives an idea of the way in which the wheels are arranged. The position of the instant center varies according to the speed and the direction of motion of the two wheels. The figure shows the position when the wheels are moving in the same direction with different speeds, i.e., \( V_R > V_L > 0 \). The robot makes a turn about its “Instant Center”, the position of which varies according to the speeds of the left and right drive wheels.

The design objective of steering is to obtain a stable control over the steering system with a good phase and gain margin and a fast unit step up response. This is the reason for the utilization of a Galil motion control board. This board has the Proportional Integral Derivative (PID) Controller digital control to provide necessary compensation required in the control of the motor. This system is modeled in Mat lab using Simulink and the three parameters of the PID controller (\( K_P \), \( K_D \), \( K_I \)) are selected using a simulation model to obtain the optimum response.

The Simulink model is given in Figure 5.3 below. In this model, a step-input is fed to the summation block. Values for the PID controller are set with a Mat lab file calculating the analog gains, for the equivalent digital filter used on the actual system. These analog values in the PID controller model adjust the input signal and feed it to the zero order hold. The zero order hold produces a pulse signal. This digital signal is fed to a D/A converter (Digital-to-Analog converter) and then passed through an amplifier. This is the amplified signal that is fed to the load, which are the motor and the steering wheels. The purpose of the encoder is to detect the movement on the wheel and give a signal that is fed back to the summation block for correction. The whole process is
repeated cyclically until the desired values for the PID controller are achieved. The values for the PID controller are tested on the actual vehicle, and are then fine-tuned using the software kit supplied by Galil Motion Inc., WSDK 1000. This software is helpful in estimating the frictional losses in the gearbox and the bearing mechanisms. Bode plots were plotted for the obtained values. To obtain the most dependable values, the actual tests can be carried out under three conditions:

- Steering wheel off the ground
- Steering wheel on the ground with robot moving
- Steering wheel on the ground with robot stationary

In addition to this procedure, another task to be accomplished is the tuning of the amplifiers. This is necessary to ensure that the exact signal goes in to the amplifier and the output is sufficiently amplified to move the drive wheels. Tuning the amplifier parameters, particularly loop gain and the selection of the PID parameters are very important and require iterative adjustments [29].

Fig 5.3: SIMULINK Model for the Steering System Compensation [29]
5.2 Actual Logic

It is known that varying the speeds of the left and right drive wheels helps in robot motion. The way in which the motion of the wheels affects robot navigation is the point of interest in this chapter. Throughout this discussion, the following terminology is used:

\( V_L \) = Speed of the left drive wheel

\( V_R \) = Speed of the right drive wheel

\( V_M \) = Base speed of motion of the robot (desired speed of robot motion)

\( W \) = Width of the robot

Controlling the sum and difference of the two wheel speeds does the control of the vehicle.

\[ V_L + V_R = 2V_M \quad (5.1) \]

\[ V_L - V_R = W \left( \frac{\text{d} \theta}{\text{d}t} \right) \quad (5.2) \]

Substituting the value of \( V_R \) from equation (5.1) in equation (5.2), we have-

\[ V_L - (2V_M - V_L) = W \left( \frac{\text{d} \theta}{\text{d}t} \right) \quad (5.2a) \]

\[ 2V_L - 2V_M = W \left( \frac{\text{d} \theta}{\text{d}t} \right) \quad (5.2b) \]
\[ V_L - V_M = \frac{W}{2} \left( \frac{d\theta}{dt} \right) \]  
\[ (5.2c) \]

\[ V_L = V_M + \frac{W}{2} \left( \frac{d\theta}{dt} \right) \]  
\[ (5.3) \]

Similarly substituting the value of \( V_L \) from equation (5.1) in equation (5.2), we have-

\[ (2V_M - V_R) - V_R = W \left( \frac{d\theta}{dt} \right) \]  
\[ (5.3a) \]

\[ 2V_M - 2V_R = W \left( \frac{d\theta}{dt} \right) \]  
\[ (5.3b) \]

\[ V_M - V_R = \frac{W}{2} \left( \frac{d\theta}{dt} \right) \]  
\[ (5.3c) \]

\[ V_R = V_M - \frac{W}{2} \left( \frac{d\theta}{dt} \right) \]  
\[ (5.4) \]

Assuming a continuous correction control loop, we can confine ‘\( V_M \)’ to a base speed, the optimum value of which was found by experimentation for minimal processing time delay and minimal error. The robot behaved smoothly at \( V_M = 35000 \) to \( 36000 \) counts. For Bearcat II, 2000 encoder counts make one revolution. So 36000 counts is equivalent to 18 revolutions. With \( W/2 \) being a constant at 13.45 inches, the factor to determine the crux of the logic mainly rests with the \( d\theta/dt \) term in equations (5.3) and (5.4).

The ISCAN image-tracking device accomplishes image tracking. This device finds the centroid of the brightest or darkest region in a computer-controlled window, and returns the X, Y coordinates of its centroid and size information of the blob. If no
object is found, a loss of track signal is generated. This information is updated every 16 ms. however, the program must wait 10 ms after moving the window to get new data. The vision algorithm feeds the co-ordinates of the centroid of the blob of two snap-shots alternatively as above. This is the function of the vision system, which is used for obtaining the two-dimensional image co-ordinates on the ISCAN tracking device corresponding to the actual three-dimensional world co-ordinates (refer to Figure 3.3 and Figure 3.4 for the three-dimensional world co-ordinates and the corresponding two-dimensional image co-ordinates on the ISCAN tracking device).

The obstacle avoidance system consists of one rotating ultrasonic transducer. A Polaroid ultrasonic ranging system is used for the purpose of calibrating the ultrasonic transducer [28]. In operation, the system uses a “Time of Flight” (TOF) approach to compute the distance by transmitting sound towards a target and detecting an echo. The elapsed time between the start of the transit pulse and the reception of the echo pulse is measured.

Knowing the speed of sound in air, the system can convert the elapsed time into a distance measurement and hence compute the distance. The range of the system depends on system parameters as well as outdoor operating conditions, but is approximately 40 feet.

Using a closed loop DC servomotor, the transducer is made to sweep an angle depending on the horizon. The control loop is closed by an encoder, which measures position for feedback.

The transducer sweep is achieved by programming the Galil motion control system [28]. By adjusting the Polaroid system parameters and synchronizing them with
the motion of the motor, distance values at known angles with respect to the centroid of the robot are measured. Thus, the inputs to the system are: \( x_1, y_1 \) and \( x_2, y_2 \) from the vision algorithm, sonar-position and sonar-distance from the ultrasonic transducer.

Consider Figure 3.3 and Figure 3.4 again. The following equations can be obtained from mathematical manipulation.

The distance of the line from the centroid of the robot is-

\[
d_1 = \frac{\text{abs}(x_2 + x_1)}{2}
\]  \hspace{1cm} (5.5)

The angle of the line with respect to the Y-axis of the robot is given by:

\[
\dot{\theta} = \tan^{-1}\left(\frac{x_2 - x_1}{y_2 - y_1}\right) \times \frac{180}{\delta}
\]  \hspace{1cm} (5.6)

The time instant \( \Delta t \) is given by:

\[
\Delta t = t_2 - t_1
\]  \hspace{1cm} (5.7)

where ‘\( t_2 \)’ and ‘\( t_1 \)’ are the times at two successive correction loops.
5.3 Rotating Ultrasonic Transducer

It is known that the mobile robot senses its location and orientation using the integrated vision system. A rotating ultrasonic sensor is used to map the location and size of possible obstacles. With these inputs, the fuzzy logic controls the speed and the steering decisions of the robot. The rotating sonar working is represented diagrammatically in Figure 5.4.

Fig 5.4: Rotating Sonar tracking an obstacle (getting its width) [26]
5.4 Extension of the Logic

Figure 5.4 above shows us the way the rotating sonar tracks the obstacle. In order to have an in-depth understanding of the working of the sonar system, look at Figure 5.5. This figure gives an indication of the way the sonar tracks the obstacle in a simpler form.

Fig 5.5: Working of the rotating sonar
The working of the rotating sonar is in such a way that it is assumed to have a pentagon-shaped sweeping area. Let us call it ABCDE. In the figure, ‘A’ is the vertex at which the sonar is present. It sweeps the area ahead of it in which the obstacle is assumed to be present. In the figure above, we assume a rectangular obstacle, but the logic to be discussed in the succeeding parts of this chapter is found to work well for circular obstacles also (this is the shape of the major obstacle in the main competition run). In this figure, the following assumptions are made:

\[ AB = EA \]
\[ BC = DE \]
\[ CD = \text{Width of the robot} + 2 \times 6 \text{ inches (clearance)} \]

Here the second term in the factor CD corresponds to a clearance of 6” on either side of the robot. This helps ensure that the sonar tracks to a distance greater than the robot’s width, which helps in avoiding obstacles that are very near to the robot. Now imagine a line drawn from the point ‘A’ to the mid-point of CD. Let it intersect at point ‘F’. Then we have AF = 110 inches. This is taken as the critical sweep of the sonar. Thus the sonar sweeps the area in front of the robot to a distance of 110 inches. Any obstacle present at a distance less than 110 inches is tracked. This is called the “Critical Sweep Area” of the sonar. Thus the actual sweep of the sonar is a sector with vertex ‘A’ (the point where the rotating sonar is fixed on the robot’s front) and radius 110 inches, which corresponds to the maximum distance the rotating sonar is capable of tracking (for obstacles) in front of the robot.
In this approach, the following three assumptions are made:

- The counter-clockwise direction of rotation of the sonar is taken to be “positive”.
- The position when the sonar is looking straight ahead is taken as zero degrees.
- The sonar makes mirror image stops on each side of the centerline.

From Figure 5.5, we can easily infer that

\[ \angle FAE = -\angle FAB \]

The sonar can then be made to choose any number of positions to stop (we have selected 5 positions for Bearcat II), at each of which it will check for possible obstacles. Keep in mind that the sonar makes mirror stops on each side of the centerline. It is necessary to calculate the number of pulses required to move the sonar to and fro and stop it at these positions. These 5 positions can be thought of as follows:

- Two extreme positions at AB and EA.
- One middle position at AF.
- One position at the midpoint of \( \angle FAE \).
- One position at the midpoint of \( \angle FAB \).

Thus the sonar can be made to stop at these 5 positions and collect data for possible obstacles. This can then give the distance (which can be calculated). Also the value of ‘\( \theta \)’ between the robot and the line is known.

Initially it was thought of as having two stationary sonars mounted on front of the robot. It is known that the sweep area of the sonar is in the form of a ‘V’; the sweep area depends on the specifications of the sonar. Consider the simple representation
below in which the sweep areas of two independently operating stationary sonars are shown (Figure 5.6).

In this case observe the sweep areas. If an obstacle is at a reasonable distance, (say O₁, O₂, O₃), then it is most likely to be sensed by either one of the sonars. But consider obstacle O₄, which lies in a “dead zone”. This obstacle is critical as it is very near to the robot. It is highly possible that the robot will run into this obstacle. This is the drawback of having two stationary sonars. Having a single rotating sonar, however, will alleviate this problem. It is nearly impossible that the robot will miss tracking an obstacle. It is to be noted that the rotating sonar must have a reasonable angle of sweep.

Coming back to some more mathematical relations, if the number of stops the sonar makes in one sweep is Max Stop ‘Sm’ and ‘Sp’ gives the Sonar Position, then the location of the obstacle with respect to the robot is indicated by the factor-

![Fig 5.6: Two Stationary Sonars](image-url)
\[ \dot{a} = \frac{S_m/2 - S_p}{\text{abs}(S_m/2 - S_p)} \]  \hspace{1cm} (5.8)

(Note: The number of stops made by the sonar will always be odd, as the sonar will make mirror stops on the left and right sides and will also take a reading at \( \theta = 0 \)).

This will help the algorithm infer if the obstacle is in the left or center or the right. The factor for the location of the obstacle with respect to the robot is-

\[
\begin{align*}
\alpha' & \text{ is taken as 2 if the obstacle is straight ahead.} \\
\alpha' & \text{ is taken as 1 if the obstacle is in the right.} \\
\alpha' & \text{ is taken as -1 if the obstacle is in the left.} 
\end{align*}
\]  \hspace{1cm} (5.9)

The sonar always returns noise distance even if there are obstacles. Thus it is essential that we have a measure of the criticality of the obstacle. This criticality is measured in terms of the distance of the obstacle from the robot. Depending on the criticality of the obstacle, we need to decide how to navigate the robot. Thus to see if the obstacle is critical or not, we introduce the following factor ‘\( \beta \)’ in the control sequence:

\[
\begin{align*}
\beta' & \text{ is taken as 1 if the distance is less than 100 inches.} \\
\beta' & \text{ is taken as 0 if the distance is less than 100 inches.} 
\end{align*}
\]  \hspace{1cm} (5.10)
Consider Figure 5.7. Here ‘A’ is the current position of the robot and for the left camera. If ‘R’ is the reaction distance tolerated, to go to the point B the robot should turn by $\theta_c$.

Then we have-

$$
\hat{\theta}_c = \cot^{-1} \left[ \frac{R}{H_d - d_1} \right] \cdot \frac{180}{\delta}
$$

(5.11)

If ‘$\theta$’ is the angle of the robot with respect to the lines, introducing the correction for this angle, we have-
\[ \dot{\theta}_c = \cot^{-1} \left[ \frac{R}{H_d - d_1} \right] * \frac{180}{\delta} + \left[ \dot{\theta} + \left( \frac{\text{abs}(H_d - d_1)}{(H_d - d_1)} \right) \right] * 90 \] (5.12)

as \[ \dot{\theta} + \left( \frac{\text{abs}(H_d - d_1)}{(H_d - d_1)} \right) * 90 \] is the correction taking into account the distance error.

Including the correction for the obstacle data, from equations (5.8), (5.9) and (5.10), ‘\( \dot{\theta}_c \)’ becomes:

\[ \dot{\theta}_c = \cot^{-1} \left[ \frac{R}{H_d - d_1} \right] * \frac{180}{\delta} + \left[ \dot{\theta} + \left( \frac{\text{abs}(H_d - d_1)}{(H_d - d_1)} \right) * 90 \right] + \hat{\alpha} * \hat{\alpha} * 15 \] (5.13)

This \( \dot{\theta}_c \) is the instantaneous correction angle for the left camera.

Hence let us introduce a total correction factor for the camera as follows:

\[ \dot{\theta}_c = (-1)^c \left[ \cot^{-1} \left[ \frac{R}{H_d - d_1} \right] * \frac{180}{\delta} + \left[ \dot{\theta} + \left( \frac{\text{abs}(H_d - d_1)}{(H_d - d_1)} \right) * 90 \right] \right] + \hat{\alpha} * \hat{\alpha} * 15 \] (5.14)

where, \( c = 1 \), if left camera is used.

and \( c = 2 \), if right camera is used.
Hence the final expressions for $V_R$ and $V_L$ are-

$$V_R = V_M - \frac{W}{2} \left[ \frac{\dot{e}_C}{(t_2-t_1)} \right] \quad (5.15)$$

$$V_L = V_M + \frac{W}{2} \left[ \frac{\dot{e}_C}{(t_2-t_1)} \right] \quad (5.16)$$

Thus we obtain the values of the speeds of the left and right drive wheels. It is seen that when the speeds are maintained at the obtained values, the robot successfully avoids the obstacles and navigates the track in the most efficient manner.

The number of stops that the sonar makes on each side of the centerline can be set as required. But it must always be an odd number. This is because it makes mirror stops on each side of the center. However, in this case, increasing the number of stops of the sonar will slow down the total speed of the robot. It is seen that having more number of stops will help in tracking the obstacles more effectively at the cost of the reduction of overall speed of the robot. In such a case, the time taken by the DMC controller to place the sonar at the particular position is relatively high. In order to take this into consideration, we have to introduce additional sleep routines in the software.
Chapter 6

The IGRC Competition

6.1 Review

In this chapter, we review the happenings and events at the 1999 International Ground Robotics Competition. The contest was held at the campus of Oakland University in Rochester, Michigan on June 7, 1999. This contest was started in 1991 and has been held each year ever since. The goal of the competition is to build a robotics vehicle that can navigate around an outdoor obstacle course. In addition to avoiding obstacles, the robots have to remain within the track. Though the contest has retained its main event (the navigation around the obstacle course), several other additional contests are also conducted in order to test the teams to the limit.

The basic aim of the navigation event is to complete the obstacle course. Since it is possible that more than one team complete the course without knocking the obstacles over, the time taken to complete the course is another crucial factor. In cases where there is more than one team completing the course, this time factor is taken into account. The purpose of this contest is to offer a design experience that is at the very cutting edge of engineering education. It is multidisciplinary, team implemented, theory-based, hands-on, outcome assessed, and is based on product realization. It
encompasses the very latest technologies impacting industrial development and taps subjects of high interest to students and researchers.

6.2 General Rules of the Contest

The rules of the contest are made clear to all the participating teams much before they arrive at the competition. The important are as under:

- The most important rule is to ensure that the mobile robot stays inside the track at all times of the run. Straying outside the track will result in immediate elimination. Another attempt, if available, will have to be made.

- The number of attempts to navigate the obstacle course is decided by a meeting of the leaders of the participating teams. It is usually 3, although it may vary depending on the circumstances.

- Among all the attempts made by the robots, the best performance will be taken into account; i.e., the maximum distance traveled in any of the attempts is the final distance for the team.

- While staying within the track, the vehicles must also ensure that they avoid obstacles. These obstacles may vary from 5 gallon buckets, construction barrels, a sand trap to anything. In the 1999 contest, a new obstacle the vehicles had to pass through was a ramp inclined at about 15 degrees. In the 2000 contest, an additional feature is the addition of potholes in the track.
• The path for obstacle navigation is usually made as the shape of the state/country from which the team that won the preceding year’s contest originates. But as found out in the 1999 contest, this is not always the case.

• Before the vehicles take an attempt at the main contest track, it has to pass through a qualifying test. Only vehicles that pass the qualifying event are allowed to take a shot at the main track. In this qualifying event, the speed and safety features of the robot are examined. Only vehicles that travel at a speed greater than 5 miles per hour and have a functional emergency stop (both on the vehicle and a remote stop) are passed through. Thus vehicles that satisfy these safety norms are allowed to take the final shot. Any small indication of danger, and the vehicle is immediately disqualified from the contest.

6.3 The 1999 Contest

The 1999 contest had 4 different competitions - Vehicle Performance Competition, Vehicle Design Competition, Road Debris Bonus Event and Follow the Leader Bonus Event.

(a) Vehicle Performance Competition

This is the main event and tests the navigation skills of the autonomous guided vehicle. The objective of this event is to complete the course without colliding with any obstacle as well as stay in track at the same time. The team that achieves this is declared the winner. The vehicle is eliminated from the run if it knocks down any of the obstacle. Any collision with obstacles resulting in their displacement (but not
knocking down) is penalized in terms of distance covered. In cases where no team completes the entire course, the robot that travels the greatest adjusted distance is declared the winner.

The performance of Bearcat II in this event is worth mentioning. On the day preceding the contest, it had the remote E-Stop burnt out. A new remote E-Stop unit was then connected in series with the motor power control circuit, which is controlled by the solenoid. This helped Bearcat II in passing the qualifying test. But before taking a first attempt at the main contest, a problem was detected with the battery output voltage. The problem was then zeroed on to a loose connection of the wires with the battery disconnect switch. It was then rectified and the robot was ready to go. The payload was placed on top of the robot. But the robot ran out of track and was eliminated. In the second attempt however, all the systems worked perfectly well and responded to computer control commands. The Vision System was the best of them all. It responded very well to the commands and the robot always tried to stay in the center of the track, as needed. The rotating sonar element also performed to perfection. It passed through various terrains with ease. Finally it ran out of its track after traveling a distance of 153.3 feet. This performance helped it to secure the fourth place overall in that event. The reason for straying off the track was attributed to a 5 gallon white bucket placed very close to the borderline. But the result was worth the effort put in by the students. A picture of the Bearcat II in action is shown in Figure 6.1.
(b) **Vehicle Design Competition**

The outcome of the design competition is based on the contents of the written design report (limited to 15 pages), a presentation on the features of the vehicle and a visual inspection of the robot by the judges. Although Bearcat II failed to win a place in the top 3, it drew applause from the judges for submitting the most complete report and for the marketing abilities of its leader.

(c) **Road Debris Bonus Event**

The nature of this event was very similar to the real day navigation on a street. In this event, the robot will encounter irregular size small obstacles that could be seen during
regular highway driving. This can include construction barrels, hubcaps, tire treads, tailpipes, barricades, wooden logs etc. Each robot entering the contest had 2 attempts to try and complete the course (around 200-250 feet long).

The performance of Bearcat II in this event was also stunning. It traveled a distance of around 65 feet before it went off the track. The success was once again attributed to the excellent functional performance of the vision, sonar and steering mechanisms and their perfect co-ordination.

(d) **Follow the leader Bonus Event**

This was a new event introduced in the 1999 contest. In this contest, the basic aim was to follow a lawn mower driven by one of the judges. There were two phases: Phase I was named as “Headway Maintenance while following” and Phase II was named as “Headway Maintenance and Free Following”. The objective was to follow the lead vehicle while maintaining a standard distance (3 meters). An omni-directional motion system (Fish Eye lens) was used to determine the angle of the target and the rotating sonar to determine the distance. Bearcat II was placed fourth in this competition.

**6.4 The 2000 Contest: A preview**

Bearcat II is rigorously being tested for the 2000 contest to be held in Orlando, Florida from July 8 – 10, 2000. This year is expected to have 5 events. A new event named “Pothole Problem” is being introduced for the first time. In this event, the track is expected to have potholes at certain regions and the robot is expected to complete the
course without entering into any of these. In addition, other obstacles may be present.
The steering logic is to be modified to take care of the potholes in addition to the
obstacles. An additional camera is being put in place to track potholes. The software is
being checked for possible errors, malfunctioning (it is also being modified) and the
functioning of all the systems is being streamlined for this purpose. The rotating sonar
is mounted on a new sonar mount. In order to increase the overall speed of motion and
keep it above the 5-mph limit, larger drive wheels (18” in diameter) are being used. A
new design outdoor castor is being used. Special attention is being given to tackle the
ramp. Also the cameras are being recalibrated for the contest.
Chapter 7

Conclusions and Recommendations for Future Work

For the accurate functioning of the vision system, a four-point calibration method has been implemented for camera calibration. This is buttressed by the use of a rotating sonar for tracking obstacles. The motion algorithm is written in the form of a C++ program, which makes it easier to be integrated with other software (as all of them are written in C++). The four point calibration is really easy to understand and very effective in terms of functioning. With a good integration between the vision system and the obstacle avoidance system, mobile robot navigation is made easier. Hence the purpose is solved. A stable platform for the purpose of testing has already been designed and constructed. With the values of the velocities for the left and right wheels, the speed and direction of the robot motion can be easily controlled. This logic when incorporated along with the vision and the rotating sonar systems mounted on the Bearcat II has yielded excellent results. Since this logic considers only the width of the robot, its base speed and the inclination of the robot with respect to the track, it is stable enough to accommodate for changes in these values.

As mentioned earlier, the use of a single rotating sonar will be effective only if the angle of sweep of the sonar is quite reasonable. The decision as to what is the optimum angle of sweep is to be determined based on the requirements and the functional capability of the affecting components. Also as discussed conventional sonars follow
the “Time of Flight” (TOF) approach. This method itself has certain inherent
drawbacks. The most important drawback is the possibility of the sonar detecting noise
from the surrounding environment. This may be noise from the surrounding
environment. This in turn may send in a wrong signal to the motion controller
regarding the presence of an obstacle. It would be worthwhile to study this process and
suggest improvements. The present arrangement of equipment is in such a way that the
number of sonar stops is taken as 5. As mentioned earlier, increasing the number of
stops will greatly enhance the obstacle detection capability of the mobile robot. But the
problem then arises with the speed of the robot. The DMC controller will take a long
time to place the sonar at the particular position. A study can be conducted to look into
this problem and suggest improvements. A novel way to improve the obstacle detection
capabilities of the sonar while keeping the effect of this change on the robot speed as
minimum as possible is an important exploration. Also this study discusses the case of a
single rotating sonar. It is also possible to use two rotating sonars mounted on the front
of the robot instead of one. This will be a great leap ahead in the navigation capabilities
of Bearcat II.
References

[1] www.robotics.org


[26] www.maxim-ic.com


Appendix A

Useful Programs

// ROTATSON.CPP (The developed motion control program for steering)

// This program causes the rotation of sonar motor from left to right.
// The sonar was tested at 24khz and reliability was achieved for an
// obstacle distance of 100 inches.
// The angles with respect to the robot was found to be -24,0,24 degrees
// With an encoder count of 2048/rev, this translates to -140,0,140 counts
// The sonar will stop at -140, -70, 0, 70, 140 counts

/* This is the program rotatson.h */

int rotatson();
int initionarmotor();
float normalsweep();

/* End of program rotatson.h */

int positionz;
int rotatson()
{
    positionz = 0;
initionarmotor();
testloop:
normalsweep();

    //test code begins
    if(kbhit())
    {
        if (getch() == 27)
        {
            submitDMC("PA,,0");
            exit(0);
        }
    }
    else
        goto testloop;

    // test code ends

}

int maxstop; //This is the maximum number of stops of the sonar.
initsonarmotor()
{
    maxstop = 4; // Assuming that it gives readings from 5 positions.
    submitDMC("SH");
    submitDMC("AC,.350000");
    submitDMC("SP,.350000");
    submitDMC("DP,.0");
    submitDMC("DP,.0");
}

float normalsweep()
{
    float obsdistance;
    positionz = (70 * sonarposindex)-140;
    positionsonar(positionz);
    gotoxy(1,1);

    //cout << "
    gotoxy (1,1);
    obsdistance = snl();
    cout << "distance = " << obsdistance << " position = " << sonarposindex;

    if (sonarposindex == maxstop)
        sonarposindex = 0;
    else
        sonarposindex++;

    //OBSSIDE = 1 if obstacle is in the right
    //OBSSIDE = 2 if obstacle is in the center
    //OBSSIDE = -1 if obstacle is in the left
    // if OBSPRESENT = 0, there is no obstacle
    // if obspresent =1, there is an obstacle being sensed.

    if (obsdistance <= 95)
    {
        OBSPRESENT = 1;
        if (sonarposindex != maxstop/2)
            OBSSIDE = (maxstop/2-sonarposindex)/abs(maxstop/2-sonarposindex);
        else
            OBSSIDE = 2;
    }
    else
    {
        OBSPRESENT = 0;
    }

    return (obsdistance);
}
// TSTSON.C

// - Procedure -
// 1. turn on PC into Dos C:\
// 2. C:\mode com1:48,n,8,1 (set 4800 baud, no parity, 8 bits, 1 stop bit)
// 3. enter Turbo c, test the following program
// 4. can use MSD DOS command to check above information in your computer

/* This is the program tstson.h */

float tstson(float angle);

/* End of file tstson.h */

#include <ctype.h>
#include<stdio.h>
#include<bios.h>
#include<conio.h>
#include<dos.h>
#include<stdlib.h>
#include "car.h"
#define port1 0x03f8     /* I/O port address declaration */

float sonl(){
    float dis;
    unsigned char buffer[7];
    unsigned char value1,value2;
    unsigned int i,j;
    outportb(port1,'@');     // Send '@' to ask sensor to response distance
    while((value2=inportb(port1))!=12);
    for (i=0;i<6;i++){
        while((inportb(0x03fd) & 0x01) != 0x01);
        buffer[i]=inportb(port1);
    }
    dis=atof(buffer);
    return dis;
}       //End of son

#define port2 0x02f8     /*  I/O port address declaration*/

float sonr(){
    float dis;
    unsigned char buffer[7];
    unsigned char value1,value2;
    unsigned int i,j;
    outportb(port2,'@');     // Send '@' to ask sensor to response distance
    while((value2=inportb(port2))!=12);
    for (i=0;i<6;i++){

while((inportb(0x02fd) & 0x01) != 0x01);
buffer[i]=inportb(port2);
}

dis=atof(buffer);

return dis;
}       //End of son

float tstson( float angle){
float answer;
float sr,sl;
sl = sonl();

if( sl < sr){
   if(sl > 0 && sl <= 40 )
      answer = final_angle( angle, -sl);
   return answer;
}

if( sr < sl){
   if(sr > 0 && sr <= 40 )
      answer = final_angle( angle, sr);
   return answer;
}
return(0);
}       // End of tstson

// MATRIX.CPP (the implementation of the vision program with test data for the right camera)

Author: Sameer Parasnis

//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 multi4_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");
    }
  }
*/

//These are values for calibrati on done on April 28, 2000. The values of $X_0 = 57^\circ$ and $Y_0 = 33.125^\circ$ (distances of the tip of the calibration device from the X and Y axes respectively.

vision_matrix[0][0]= 58.00;
vision_matrix[0][1]= 37.125;
vision_matrix[0][2]= -11.5625;
vision_matrix[0][3]= 1;
vision_matrix[1][0]= 60.00;
vision_matrix[1][1]= 41.125;
vision_matrix[1][2]= -12.5625;
vision_matrix[1][3]= 1;
vision_matrix[2][0]= 62.00;
vision_matrix[2][1]= 39.125;
vision_matrix[2][2]= -11.5625;
vision_matrix[2][3]=1;
vision_matrix[3][0]= 60.00;
vision_matrix[3][1]= 35.125;
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]= 198;
x_image_matrix[1][0]= 191;
x_image_matrix[2][0]= 158;
x_image_matrix[3][0]= 180;
y_image_matrix[0][0]= 85;
y_image_matrix[1][0]= 48;
y_image_matrix[2][0]= 70;
y_image_matrix[3][0]= 112;
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 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:");  

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 = 200.00;
test_y_image = 123.00;

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]);

// VIS_L.C (The vision program for the left camera)

#include "vis_l.h"
//include "wintest1.c"
//include "tstson.h"

int vis_l(void){

#define DELAY1 10
#define DELAY2 10
#define SONAR FALSE
int count = 0;
float obsdist;
int y1_size_c=0;
int y2_size_c=0;
int x1_size_c=0;
int x2_size_c=0;
int vis_count = 0;
int cor3,cor4,cor3P,cor4P;

// Calibration values
struct constant2 {
    float a11;
    float a12;
    float a13;
    float a14,a21,a22,a23,a24;
};
constant2 *con;
con = new constant2;
float angle =0;
float z_coord= 18.75;

//The calibration coefficients as calculated on April 28, 2000
con->a11= -9.500;
con->a12= 2.000;
con->a13= -7.000;
con->a14= 600.5625;
con->a21= 0.8333;
con->a22= -10.6667;
con->a23= -2.000;
con->a24= 374.2083;

float angleC, SonANGLE, angleD, angle1, angle2;
initISCAN();  //initialize iscan
startCAR();
spdx = 7000;
speedCARx(spdx);
spdy = 7000;
speedCARy(spdy);
CAMERA = 2;
wind *win1= new wind;
wind *win2= new wind;
coordinate *crd1=new coordinate;
coordinate *crd2=new coordinate;
coordinate *crd3=new coordinate;
coordinate *crd4=new coordinate;

//define window 1 x,y pos
win1->pos_x=255;
win1->pos_y=100;
//define window 2 x,y pos
win2->pos_x=255;
win2->pos_y=200;

//define windows size
win1->siz_x=win2->siz_x=500;
win1->siz_y=win2->siz_y=10;

float angle_emer=0.0;
float old_angle=0.0;
clscr();
while(!kbhit()){
    kartcount ++;
    gotoxy(10,1);
    cout << "LEFT CAMERA (" << CAMERA << ") ****> Loop counter = " << kartcount << 
"\n\n";
    setgate(win1);
    getdata(crd1);
    gotoxy(1,3);
    cout << "First image co-ordinate: " << "crd1->LOT: " << crd1->LOT;
    gotoxy(1,4);
    cout << "Co-ordinate 1 (X,Y) = (" << crd1->pos_x << "," << crd1->pos_y << ");"
    gotoxy(1,5);
    cout << "Co-ordinate 1 size (X,Y) = (" << crd1->siz_x << "," << crd1->siz_y << ");"
    if (crd1->LOT != 2) return(1);

    setgate(win2);
    sleep(DELAY1);
    getdata(crd2);
    gotoxy(40,3);
    cout << "Second image co-ordinate: " << "crd2->LOT: " << crd2->LOT;
    gotoxy(40,4);
    cout << "Co-ordinate 2 (X,Y) = (" << crd2->pos_x << "," << crd2->pos_y << ");"
    gotoxy(40,5);
    cout << "Co-ordinate 2 size (X,Y) = (" << crd2->siz_x << "," << crd2->siz_y << ");"
    gotoxy(1,5);
    if (crd2->LOT != 2) return(1);

    //Calculating the real world co-ordinates crd3,crd4 from image
    //co-ordinates crd1,crd2...respectively
    float det=(con->a22*con->a11)-(con->a21*con->a12);

    crd4->pos_x=((con->a22*(crd1->pos_x-con->a14)-(con->a13*(-z_coord)))-(con->a21*con->a12))/det;
    crd4->pos_y=((con->a22*(crd1->pos_y-con->a24)-(con->a23*(-z_coord)))-(con->a21*con->a12))/det;
    crd3->pos_y=((con->a11*(crd1->pos_y-con->a24)-(con->a23*(-z_coord)))-(con->a13*(-z_coord)))/det;
    crd3->pos_x=((con->a11*(crd2->pos_x-con->a14)-(con->a13*(-z_coord)))-(con->a12*con->a14))/det;
}
float A = crd3->pos_x - crd4->pos_x;
float B = crd3->pos_y - crd4->pos_y;
float angle = (atan(A/B))*180.0/3.14;
float actual_angle = angle;
float angleC;

// 12 in the next line is a fudge factor
float dist = ((crd3->pos_x + crd4->pos_x)/2.0)+12;

// Softening the angle to minimize stray data
angle = (angle+old_angle)/2;

gotoxy(20,7);
cout << "Calculated angle of line = " << angle;

/* compute distance error */
float d_error = 60.0-dist;

// Rotate sonar and get reading
// normalsweep();

(pwd) (cd ad) (rm ad)

/*******************************************************************/
/* Here begins the logic - to be tested */
/* These few lines actuate sensor data fusion, replacing fuzzy logic
*********************************************************************/
if (dist == 60)
{
    angleC = (angle);//+(OBSPRESENT*OBSSIDE*15));
}
if (dist > 60)
{
    angleC = (atan(10/(60-dist))*(180.0/3.14)+angle+90);
gotoxy(20,14);
cout << "Distance is greater than 60!   Dist: " << dist;
gotoxy(20,15);
cout << "atan(): " << angleC-90-angle << "+90 + " << angle << " = " << angleC;
}
if (kbhit()) exit(0);
if (dist < 60)
{
    angleC = (-90+atan((10/(60-dist))*(180.0/3.141)+angle));
gotoxy(20,14);
cout << "Distance is less than 60!   Dist: " << dist;
gotoxy(20,15);
cout << "90 - atan(): " << angleC-angle << "+angle" << angle << " = " << angleC;
}

/*******************************************************************/
/*Here ends the logic that replaces the fuzzy approach*/
*********************************************************************/
// if ( angleC >= 30.0) angleC = 30.0;
// if ( angleC <= -30.0) angleC = -30.0;
gotoxy(20,8);
cout <<"Corrected angle of line = "<<angleC;
cout<<"\n\n\t	Distance: "<<dist<<"Dist_err: "<<d_error<<"\n";
old_angle = angle;

if (kbhit())
{
    stopCAR();
    exit(0);
}

steerCAR(angleC);
}
//return 5;
stopCAR();
} //end of main

// VIS_R.C (The vision program for the right camera)

#include "vis_l.h"
//#include "wintest1.c"
//#include "tstson.h"

int vis_l(void){

#define DELAY1 10
#define DELAY2 10
#define SONAR FALSE

int count = 0;
float obsdist;

int y1_size_c=0;
int y2_size_c=0;
int x1_size_c=0;
int x2_size_c=0;
int vis_count = 0;
int cor3,cor4,cor3P,cor4P;

// Calibration values
struct constant2 {
    float a11;
    float a12;
    float a13;
    float a14,a21,a22,a23,a24;
};
constant2 *con;
con = new constant2;
float angle =0;
float z_coord = 18.75;
//The calibration coefficients as calculated on April 28, 2000
con->a11 = -9.500;
con->a12 = 2.000;
con->a13 = -7.000;
con->a14 = 600.5625;
con->a21 = 0.833; 3;
con->a22 = -10.6667;
con->a23 = -2.000;
con->a24 = 374.2083;

float angleC, SonANGLE, angleD, angle1, angle2;

initISCAN(); //initialize iscan
startCAR();
spdx = 7000;
speedCARx(spdx);
spdy = 7000;
speedCARy(spdy);

CAMERA = 2;

wind *win1= new wind;
wind *win2= new wind;

coordinate *crd1=new coordinate;
coordinate *crd2=new coordinate;
coordinate *crd3=new coordinate;
coordinate *crd4=new coordinate;

//define window 1 x,y pos
win1->pos_x=255;
win1->pos_y=100;

//define window 2 x,y pos
win2->pos_x=255;
win2->pos_y=200;

//define windows size
win1->siz_x=win2->siz_x=500;
win1->siz_y=win2->siz_y=10;

float angle_emer=0.0;
float old_angle=0.0;
clrscr();
while(!kbhit()){ 
kartcount ++;
gotoxy(10,1);
cout << "t LEFT CAMERA (" << CAMERA << ") ***** Loop counter = " << kartcount << 
\n\n";
setgate(win1);
getdata(crd1);
gotoxy(1,3);
cout << "First image co-ordinate: " << crd1->LOT; 
  gotoxy(1,4);
cout << "Co-ordinate 1 (X,Y) = (" << crd1->pos_x << "," << crd1->pos_y << ");
  gotoxy(1,5);
cout << "Co-ordinate 1 size (X,Y) = (" << crd1->siz_x << "," << crd1->siz_y << ");
  if (crd1->LOT != 2) return(1);

setgate(win2);
sleep(DELAY1);
getdata(crd2);
gotoxy(40,3);
cout << "Second image co-ordinate: " << crd2->LOT; 
  gotoxy(40,4);
cout << "Co-ordinate 2 (X,Y) = (" << crd2->pos_x << "," << crd2->pos_y << ");
  gotoxy(40,5);
cout << "Co-ordinate 2 size (X,Y) = (" << crd2->siz_x << "," << crd2->siz_y << ");
  gotoxy(1,5);
  if (crd2->LOT != 2) return(1);

//Calculating the real world co-ordinates crd3, crd4 from image
  // co-ordinates crd1, crd2...respectively

float det=(con->a22*con->a11)-(con->a21*con->a12);
crd3->pos_x=((con->a22*(crd1->pos_x-con->a14-(con->a13*(-z_coord)))) - ((crd1->pos_y-con->a24-(con->a23*(-z_coord)))*con->a12))/det;
crd4->pos_x=((con->a22*(crd2->pos_x-con->a14-(con->a13*(-z_coord)))) - ((crd2->pos_y-con->a24-(con->a23*(-z_coord)))*con->a12))/det;
crd3->pos_y=((con->a11*(crd1->pos_y-con->a24-(con->a23*(-z_coord)))) - (crd1->pos_x-con->a14-(con->a13*(-z_coord)))*con->a21)/det;
crd4->pos_y=((con->a11*(crd2->pos_y-con->a24-(con->a23*(-z_coord)))) - ((crd2->pos_x-con->a14-(con->a13*(-z_coord)))*con->a21))/det;

float A=crd3->pos_x - crd4->pos_x;
float B=crd3->pos_y - crd4->pos_y;

float angle = (atan(A/B))*180.0/3.14;
float actual_angle = angle;
float angleC;

// 12 in the next line is a fudge factor
float dist= ((crd3->pos_x + crd4->pos_x)/2.0)+12;

// Softening the angle to minimize stray data
// angle = (angle+old_angle)/2;

gotoxy(20,7);
cout << "Calculated angle of line = " << angle;

/* compute distance error */
float d_error = 60.0-dist;

// Rotate sonar and get reading
// normalsweep();
if (dist == 60) {
    angleC = (angle); // (OBSPRESENT*OBSSIDE*15));
}

if (dist > 60) {
    angleC = (atan(10/(60 - dist))*(180.0/3.14)+angle+90);
    gotoxy(20,14);
    cout << "Distance is greater than 60! Dist: " << dist;
    gotoxy(20,15);
    cout << "atan(): " << angleC - 90 - angle + angle << " = " << angleC;
}

if (kbhit()) exit(0);

if (dist < 60) {
    angleC = (-90+atan((10/(60 - dist)))*(180.0/3.141)+angle);
    gotoxy(20,14);
    cout << "Distance is less than 60! Dist: " << dist;
    gotoxy(20,15);
    cout << "90 - atan(): " << angleC - angle + angle << " = " << angleC;
}

// if ( angleC >= 30.0) angleC = 30.0;
// if ( angleC <= - 30.0) angleC = - 30.0;

if (kbhit()) {
    stopCAR();
    exit(0);
}

steerCAR(angleC);

stopCAR();
} // end of main
// CAR.C

// This is the main motion control program. It controls the start,
// stop and speed of the robot.
// Authors: Kalyan, Krishnamohan

/* This is the program car.h */

#include "galil.c"
void speedcar(long int spz);
void speedCARx(long int spx);
void speedCARy(long int spy);
void stopCAR();
void steerCAR(float val);
void auto_steerCAR(float val);
void startCAR();
void dodgeCAR();

/* Endof program car.h */

#include "car.h"
#ifndef CAR
#define CAR
#endif

define CAR

extern time_t first;

void speedCARx(long int spx)
{
    //This command when invoked will set the speed of Bearcat II.
    //It is called with a value between 0 - 250000.

    char inx[11],sendx[16];
gcvt(spx,6,inx);
char *commandx="JG";
strcpy(sendx,commandx);
strcat(sendx,inx);
gotoxy(10,21);
cout<< " Left-motor: ";
gotoxy(24,21);
cout << sendx;
gotoxy(1,23);
cout << "X-motor --> ";

#if(!TEST)
    submitDMC(sendx);
    submitDMC("BGX");
#endif

}

void speedCARy(long int spy){
    //This command when invoked will set the speed of Bearcat II.
    //It is called with a value between 0 - 250000.
char iny[11],sendy[16];
gcvt(spy,6,iny);
char *commandy="JG,;"
strcpy(sendy,commandy);
strcat(sendy,iny);
gotoxy(38,21);
cout<<"Right-Motor: ";
gotoxy(52,21);
cout << sendy;
gotoxy(1,24);
cout<< "Y-motor --> ";

//if(!TEST)
submitDMC(sendy);
submitDMC("BGY");
}

void stopCAR()
{
gotoxy(1,23);
submitDMC("ST");
submitDMC("ST");
submitDMC("MO");
submitDMC("MO");
submitDMC("SH");
gotoxy(1,23);
cout << "Stopped Running":
}

void steerCAR(float val)
{
  double dtime;
  time_t second=time(NULL);
gotoxy (25,17);
cout << "Inside steercar: -- >VAL: " << val << "\n";
  //This function when invoked will put the steering wheel to the absolute
  //angle given. This angle ranges between +20 and -20.
  dtime=difftime(second,first);
cout << "Time data: first = " << first << " second = " << second;
cout " dtime = " << dtime << endl;
  first=second;
dtime = 0.25;

  /*temporary code begins.....*/
  if (val <= -30)
  {
    spdx = 2000;
  }
spdy = 400;
}

if (val >= 30)
{
    spdx = 400;
    spdy = 2000;
}

if (val < 5 && val > -5)
{
    spdx = 8000;
    spdy = 8000;
}

if (val >= 5 && val < 30 || val <= -5 && val > -30)
{
    spdx = 7000 - ((134.5*val)/dtime);
    spdy = 7000 + ((134.5*val)/dtime);
}

if (spdx > 36000) spdx = 36000;
if (spdx < -36000) spdx = -36000;
if (spdy > 36000) spdy = 36000;
if (spdy < -36000) spdy = -36000;

cout << Spdx = " << spdx << "   ";
cout << Spdy = " << spdy << "   ";
cout << Increment = " << (int)((134.5*val)/dtime) << "   ";
speedCARx(spdx);
speedCARy(spdy);
}

void auto_steerCAR(float val)
{
    // This function when invoked will put the steering wheel to the absolute
    // angle given. This angle ranges between +20 and -20.
    char inc[11], send[16];

    // slow car
    int spdx = -.05*abs(val)+COMP_SPEED;
    speedCARx(spdx);

    // steer car
    steerCAR(val);
}

void startCAR()
{
/This function when called will initialize the Galil board.

initDMC();
set_upDMC();

// for(int wait= 0;wait<500;wait++);
// download("c:\galil\speed.dcp");
// submitDMC("XQ");

#endif

#endif
Appendix B

Results of the 1999 IGRC Competition

1) Obstacle Avoidance Event

<table>
<thead>
<tr>
<th>University Team</th>
<th>Best Distance in the 3 Heats</th>
<th>Rank</th>
<th>Prize Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Colorado at Denver – CUGAR</td>
<td>502.75</td>
<td>1</td>
<td>$4,000</td>
</tr>
<tr>
<td>Hosei University, Japan – NECTAR</td>
<td>384.5</td>
<td>2</td>
<td>$600</td>
</tr>
<tr>
<td>Michigan Technological University - VERONICA</td>
<td>193</td>
<td>3</td>
<td>$400</td>
</tr>
<tr>
<td>University of Cincinnati – BEARCAT II</td>
<td>153.3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>University of Colorado Boulder – Robotic Autonomous Transport</td>
<td>115</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Wayne State University – WAYNE ROVER</td>
<td>20.5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Virginia Tech University – ARTEMIS</td>
<td>6.5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>University of Alberta – POLAR BEAR</td>
<td>-1.5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Virginia Tech University – IVAN</td>
<td>Demonstration Only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oakland University – COYOTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>University of Detroit Mercy – WHY2K</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>University of Tulsa – HURRICANE ARIANE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2) **Design Event**

First Place: Virginia Tech – *ARTEMIS* - $1000

Second Place: University of Alberta – *POLAR BEAR* - $600

Third Place: University of Detroit Mercy – *WHY2K* - $400

3) **Follow – the – Leader Event**

First Place: Virginia Tech – *ARTEMIS* - $1000

Second Place: Hosei University – *NECTAR* – No Prize Money

Third Place: University of Colorado at Denver – *CUGAR* – No Prize Money

4) **Road Debris Event**

First Place: Hosei University, Japan – *NECTAR* - $1000

Second Place: University of Colorado at Denver – *CUGAR* - $600

Third Place: University of Cincinnati – *BEARCAT II* - $400

*First to run*

University of Colorado at Boulder - *RAT*
Top Scorers in Design Competition

1) Written Report

<table>
<thead>
<tr>
<th>Category</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conduct of Design Process</td>
<td>Virginia Tech – ARTEMIS</td>
</tr>
<tr>
<td>Completeness</td>
<td>University of Cincinnati – BEARCAT II</td>
</tr>
<tr>
<td>Quality of Writing</td>
<td>University of Alberta – POLAR BEAR</td>
</tr>
<tr>
<td>Innovation</td>
<td>Virginia Tech – Gnat</td>
</tr>
<tr>
<td>Electronics, Software and Integration</td>
<td>University of Alberta – POLAR BEAR</td>
</tr>
<tr>
<td>Safety, Reliability and Durability</td>
<td>University of Detroit Mercy – WH2YK</td>
</tr>
</tbody>
</table>

Overall Winner: University of Alberta – POLAR BEAR

2) Oral Presentation

<table>
<thead>
<tr>
<th>Category</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization</td>
<td>University of Colorado at Denver</td>
</tr>
<tr>
<td>Graphic Aids</td>
<td>Virginia Tech – ARTEMIS *</td>
</tr>
<tr>
<td>Articulation</td>
<td>Virginia Tech – Gnat</td>
</tr>
<tr>
<td></td>
<td>Virginia Tech - Gnat *</td>
</tr>
<tr>
<td></td>
<td>Virginia Tech – IVAN *</td>
</tr>
</tbody>
</table>

Articulation: Virginia Tech – ARTEMIS *
Response to Questions

Virginia Tech – *ARTEMIS*

University of Alberta *

University of Colorado, Denver *

University of Colorado, Boulder *

Salesmanship

University of Cincinnati

---

**Overall Winner**

**Virginia Tech - ARTEMIS**

---

3) **Examination of Vehicle**

Packaging Neatness

Virginia Tech – *ARTEMIS*

Serviceability

Virginia Tech – *ARTEMIS* *

University of Tulsa *

University of Colorado, Boulder *

Ruggedness

University of Alberta *

Michigan Tech *

Safety

Virginia Tech – *ARTEMIS*

Original Content

Virginia Tech – *GNAT*

Style

Virginia Tech – *ARTEMIS* *

University of Alberta *
Overall Winner
Virginia Tech – ARTEMIS *
University of Detroit Mercy *

Grand Winner
Virginia Tech - ARTEMIS