OBSTACLE AVOIDANCE IN UNSTRUCTRUED ENVIRONMENT FOR THE BEARCAT

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

2003

by

Vidyasagar Murty

B.Tech., Mechanical Engineering

Jawaharlal Nehru Technological University

College of Engineering, Anantapur, India, 2000.

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

ABSTRACT

The Center for Robotics Research at University of Cincinnati does extensive research on mobile robot technologies. The center developed Bearcat, an autonomous unmanned vehicle (AUV). The Bearcat can autonomously navigate and avoid only a limited configuration of obstacles in a structured environment. This research provides a solution for the Bearcat to navigate autonomously in an unstructured environment.

A simple wandering algorithm was developed to avoid obstacles in an unstructured environment. The algorithm uses the feedback from the laser scanner. It filters the data received, identifies potential obstacles and accordingly changes its path to avoid the obstacles based on their configuration. This is achieved by sending explicit motion commands to the motion controller which moves the robot.

The algorithm was successfully tested to avoid obstacles in an unstructured environment with certain limitations. The algorithm was also used by the Bearcat in the Autonomous Challenge event at the 11^{th} annual International Ground Vehicles Competition (IGVC) organized by the Association of Unmanned Vehicle Systems international (AUVSI) enabling it to bag the fifth place in the competition.

The research provides a starting point to build and develop a robust algorithm to address complex real life issues where the robots have to generate alternate paths to avoid obstacles in a dynamic and unstructured environment.

ACKNOWLEDGEMENTS

First and foremost I would like to acknowledge the help provided by my advisor Dr. Ernest L. Hall. Without his guidance and support this thesis would not have been possible. His suggestions, feedback and constant encouragement have been the greatest help in completion of this thesis**.**

I would like to thank Dr. Ronald Huston and Dr. Richard Shell for agreeing to serve on the thesis committee. I would also like to thank them for their suggestions and positive feedback.

Last but surely not the least I would like to thank each member of the Robotics Team, both present and past for their help and support they have given me during the period of my graduate study at the University of Cincinnati. Your contribution, support and suggestions have been the single largest factor towards the improvement of my work.

Table of Contents

List of Figures

Chapter 3

Chapter 4

Chapter 5

1. Introduction

1.1 Background

A pressing need for increased productivity and uniform quality of end products gave rise to computer-based automation. Initially manufacturing tasks were preformed by special purpose machines. But the inflexibility and high cost of these machines led to a broad based interest in the use of robots capable of performing a variety of manufacturing functions in more flexible working environment and at lower cost.

Robot Institute of America defines a robot as "A reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks".

The applications of robots evolved ever since, and have taken a big leap. Making progress toward autonomous robots is of major practical interest in a wide variety of application domains including manufacturing, construction, waste management, space exploration, undersea work, medical surgery and assistance for the disabled. Robots have even reached the common man with the advent of lawn-mowing robots and the like.

It is also of great technical interest because it raises challenging and rich computational issues. These new concepts of broad usefulness are emerging. Lately the development of

autonomous vehicles has found increasing applications in military and highway transportation.

Developing the technologies necessary for autonomous vehicles is a formidable undertaking with deep interweaved ramifications in automated reasoning, perception and control.

The Center for Robotics Research at University of Cincinnati has been a regular participant in the annual International Ground Vehicles Competition (IGVC) organized by the Association for Unmanned Vehicle Systems International (AUVSI). The University of Cincinnati's robot, called the Bearcat has won a lot of prizes in the competition.

Over the years, Bearcat has provided an excellent test bed for technologies being investigated and developed at the Center for Robotics Research. One of the major issues involving the development of an autonomous vehicle is successful obstacle avoidance. For the "Autonomous Navigation" event in the IGVC, a simple algorithm has been developed to avoid specific configuration of obstacles when the Bearcat navigates between two white lines which form a track [1].

1.2 Objective

This work primarily focuses on the development and implementation of a new algorithm which addresses the obstacle avoidance for the Bearcat in a broader perspective. Efforts

have been directed to develop an algorithm where the Bearcat will be capable to "wander" in an unstructured environment with simple, static, and random obstacles. The goal was to propose and implement a solution with the existing configuration of the Bearcat.

1.3 Organization of Thesis

Chapter 2 reviews the history of Motion Planning. Chapter 3 discusses the test-bed – its main components and how they help in autonomous navigation. Chapter 4 puts forth the proposed solution using the existing configuration of the Bearcat. It discusses the assumptions, methodology, and the algorithm. Chapter 5 explains the results and discusses the limitations of the algorithm. Finally chapter 6 states how the current algorithm can be improved and developed to overcome the limitations.

2. Motion Planning

2.1 Background

Motion planning can be loosely stated as follows [2]: *How can a robot decide what options to perform in order to achieve goal arrangements of physical objects?* This capability is eminently necessary since a robot accomplishes tasks by moving in the real world.

The research in motion planning can be tracked back to late 60's, during the early stages of development of computer-controlled robots. Nevertheless, most of the research has been done since 80s. Over the last few years, the theoretical and practical understanding of some of the issues has increased rapidly due to the combined work of researchers in Artificial Intelligence, Computer Science, Mathematics and Mechanical Engineering. The research not only produced effective planning methods but also contributed in advancing the understanding of the mathematical structure of the problems and their inherent computational complexity.

Motion planning looks relatively simple, since we deal with it in our daily life without any difficulty. In fact, so is the case with perception, the elementary operative intelligence that people use unconsciously to interact with their environment. This turns out to be extremely difficult to duplicate using a computer-controlled robot. Some simple methods produce impressive results for simple cases, but limitations spring out

when the situation becomes complex. A reliable motion planner needs a lot of nontrivial mathematical and algorithmic techniques.

A widespread view is that motion planning essentially consists of doing some sort of collision checking or collision avoidance. In fact motion planning is much more than that. It involves such diverse aspects as computing collision-free paths among possibly moving obstacles, coordinating the motions of several robots, planning sliding and pushing motions to achieve precise relations among objects, reasoning to build reliable sensory-based motion strategies, dealing with models of physical properties such as mass, gravity and friction, and planning stable grasps of objects.

Therefore, motion planning requires robot to consider geometrical constraints, as well as physicals and temporal constraints. In addition, uncertainty may require that it plan not only motion commands, but also their interaction with sensing. When knowledge at planning time is too incomplete, it may become necessary to interweave planning and execution in order to collect appropriate information through sensing.

Motion planning primarily requires sensors to capture the information about the robot's surroundings. And it requires an intelligent algorithm/strategy to analyze the data and compute a collision-free path to avoid the obstacles and successfully navigate to the target. Different kinds of sensors and algorithms are discussed in subsequent sections.

2.2 Literature Review

This section reviews the different kinds of sensors and navigation strategies that have been employed in robots so far.

2.2.1 Sensors

The function of robot sensors may be divided into two principal categories [4]: *internal state* and *external state*. Internal state sensors deal with the detection of variables such as arm joint position, which are used for robot control. External sensors deal with the variables such as range, proximity, and touch used for robot guidance, object identification, and handling. Sensors are primarily 3 kinds – Tactile, Proximity and Range sensors.

As the name implies, *tactile sensors* detect the object of interest with direct physical contact. They are typically employed on mobile robots to provide a last-resort indication of collisions with surrounding obstructions.

Proximity sensors on the other hand, are non-contact devices that provide advance warning on the presence of an object in close proximity to the sensing element. Proximity sensors fall short to provide the system with sufficient situational awareness to support intelligent movement. This needs acquisition of appropriate information regarding ranges and bearings to nearby objects, and subsequent interpretation of that data.

Range sensors measure the actual distance to target of interest without direct physical contact. They provide information regarding the location and position of the object of interest. There are at least seven different types of ranging techniques employed in such devices. [3]

1. Triangulation
2. Time of flight
3. Phase-shift measurement
4. Frequency modulation
5. Interferometry
6. Range-from-focus
7. Return signal intensity

Triangulation

It is based on the premise of the plane trigonometry that states given the length and two angles of a triangle, it is possible to determine the length of other sides and the remaining angle.

Time of flight

This technique measures the round trip time required for a pulse of emitted energy to travel to a reflecting object and echo back to a receiver. The time, multiplied by the velocity of the pulse gives a value twice the distance of the object as the pulse travels twice the distance.

Phase-shift measurement

This involves continuous wave transmission as opposed to pulses used in time-of-flight. This also gives a measure of the direction and velocity of a moving target.

Frequency modulation

This technique involves transmission of a continuous electromagnetic wave modulated by a periodic triangular signal that adjusts the carrier frequency.

Interferometry

This concept is based on the resulting interfering patterns that occur when two energy waves caused to travel different paths are compared. If the length of one of the paths is changed, the two beams will interact in such a way that clearly visible constructive and destructive interference fringes are produced. The number of fringes gives a measure of the distance of the object. Interferometers measure relative displacement but not actual ranges.

Range from focus

The principle of operation is based on the *Gauss thin lens law*:

$$\frac{1}{r} = \frac{1}{f} - \frac{1}{s}$$

where:

r = distance from lens to object viewed

s = distance from lens to image plane

f = focal length of the lens

Rearranging the terms, the distance to object is:

$$r = \frac{fs}{s - f}$$

Return signal intensity

This technique determines the distance to an object based on the amplitude of energy (usually light) reflected from the object's surface. The inverse square law for emitted energy states that as the distance from a point source increases, the intensity of the source diminishes as a function of the square of the distance.

Proximity, tactile and range sensing techniques discussed above play a significant role in the improvement of a robot performance. A need for more sophisticated mechanism that allows the robot to respond to its environment in an intelligent manner has motivated the use of vision. Vision is recognized as the most powerful sensory capabilities of a robot.

Robot vision may be defined as the process of extracting, characterizing, and interpreting information from images of a three-dimensional world. This is also known as machine vision or computer vision. The process can be classified into 6 stages:

1. Sensing
2. Preprocessing
3. Segmentation
4. Description
5. Recognition
6. Interpretaion

Sensing is the process of image acquisition while preprocessing deals with image enhancement using noise reduction and similar techniques. Segmentation is the process where the image is partitioned into areas of interest based on the needs of the situation.

Description deals with the computation of features like shape and size which help in differentiating objects in the image. Recognition is the stage where the objects are identified and interpretation completes the image processing by giving a meaning to the visual image.

The images from a camera are two-dimensional and extraction of three-dimensional information from such an image is a key issue. Camera Calibration and Stereo Imaging are two techniques which are used to capture the 3D information from a 2D camera image.

A mathematical as well as geometrical transformation occurs via the camera parameters in transforming a 3-D coordinate system to a 2-D system. For known values of three-dimensional world coordinates, the corresponding 2D image coordinates are captured and using a set of computational procedures, the mathematical and geometrical relations are established. Using those relations, the unknown 3D world coordinates are calculated for any given 2D image coordinates. This is referred to as *camera calibration.*

Stereo Imaging is another imaging technique which maps a 3D scene onto an image plane and determines the location of the world point using transformations. Stereo imaging involves obtaining two separate image views of an object of interest from two different cameras. Using the principles of geometry, the image information from both the cameras is used to obtain the missing depth information.

2.2.2 Navigation Control Strategies

A number of different navigational control strategies have been developed as a result of extended research. They can be briefly classified as *global navigation* and *localized navigation*.

Global navigation is a high-level strategy used for planning an optimal path to some desired goal location in world coordinates. Localized navigation involves piloting the robot around unexpected obstructions.

Six general approaches for localized navigation can be stated as:

1. "Wander" routine
2. Circumnavigation
3. Potential fields
4. Certainty grids
5. Motor schema
6. Vector field histogram.

"Wander" routine

This is the most simplistic approach which involves traveling more or less in a straight line until an obstacle is encountered, altering course to avoid impact and then resuming straight-line motion. This can be simply hard-coded or rule-based.

Circumnavigation

Circumnavigation is a collision avoidance behavior in which the robot deflects laterally to "sidestep" an obstruction, while attempting to move in the direction of the goal.

Potential fields

The concept of potential field involves artificial force acting upon the robot, derived from the vector summation of an attractive force representing the goal and a number of repulsive forces associated with the individual known obstacles.

Certainty grids

Certainty grids are a way to construct an internal representation of static environments based on sensor measurements. This method takes into account the uncertainty of sensory data by working with probabilities or certainty values. The robot's environment is represented as a grid of cells, and an occupancy value is attached to each cell. This value represents the certainty of an obstacle occupying the corresponding region. The certainty grid representation can be used directly in robotic motion planning or navigation.

Motor schema [5]

This approach employs a strategy that is very much analogous to potential fields. A *schema* can be defined as "a generic specification of a comparing agent." Schemas are basically parameterized motor behaviors (motor schemas) and their associated perceptual strategies (perceptual schemas).

Each schema represents a general behavior that is instantiated when a copy of the generic specification is parameterized and activated. A collection of such schemas provide the potential family of actions for control of the robot.

For example, a mobile robot has these typical schemas: *move-ahead, avoid-static-obstacle* and *move-to-goal*. The output of a schema is a single velocity vector reflecting the resolution of all potential field influences experienced by the robot at any given location, which is used to compute the desired trajectory in real time.

Vector field histogram [6]

This technique is a combination of both Cartesian and polar representations. In this method a two-dimensional Cartesian *histogram grid* is constructed with data from the range sensors. The *histogram grid* is reduced to a one-dimensional *polar histogram* that is constructed around the momentary location of the robot. It allows a spatial interpretation (called *polar obstacle density*) of the robot's instantaneous environment. This is used for motion planning or navigation.

Obstacle avoidance for mobile robots has presented many challenges and extensive research work is being conducted in this area.

J. Borenstein et. al [7] introduced *histogramic in-motion mapping* (HIMM), a method for real-time map building with a mobile robot in motion. HIMM represents data in a two-dimensional array, called a *histogram grid* that is updated through rapid in-motion sampling of onboard range sensors. Rapid in-motion sampling results in a map representation that is well-suited to modeling inaccurate and noisy range-sensor data, such as that produced by ultrasonic sensors, and requires minimal computational

overhead. Fast map-building allows the robot to immediately use the mapped information in real-time obstacle-avoidance algorithms. The benefits of this integrated approach are twofold: (1) quick, accurate mapping; and (2) safe navigation of the robot toward a given target.

D. Sekimori et. al [8] propose a method of obstacle avoidance and a method of self-localization for mobile robots based upon image of floor region taken with an omni-directional camera mounted on the robot. They suggested the use of omni-directional vision for detecting obstacles and landmarks in a wide area at high speed. Free space is detected by using general labeling function for the floor region divided into many small areas. For self-localization, the floor region is compensated by computing the convex hull of its boundary points. The geometric features of detected floor region and the linearized least square method considering the properties of omni-directional imaging are employed to fit the known floor shape.

D. Fox et. al [9] proposed a hybrid approach to the problem of collision avoidance for indoor mobile robots. The model-based dynamic window approach integrated sensor data from various sensors with information extracted from a map of the environment, to generate collision-free motion. They used a novel integration rule ensuring that with high likelihood, the robot avoids collisions with obstacles not detectable with its sensors, even if it is uncertain about its position. They developed this approach for ill-shaped obstacles which prohibit the use of purely sensor-based methods for collision avoidance.

D. Castro et. al [10] proposed a reactive navigation system for an autonomous non-holonomic mobile robot in dynamic environments. The sensory perception based on a laser range finder system is used for the object detection algorithm and the reactive collision avoidance method.

I. Nourbakhsh et. al [11] presented a passive vision system that recovers coarse depth information reliably and efficiently. This system is based on the concept of depth from focus, and robustly locates static and moving obstacles as well as stairs and drop-offs with adequate accuracy for obstacle avoidance.

S. SOUMARE et. al [12] presented a real-time vision-based obstacle detection and avoidance method in an indoor environment for an autonomous mobile robot. In the paper they proposed a scenario in which 3-Dimensional obstacles in the real world environment are detected by scanning a vertically emitted laser slit paired with an active stereo vision system. They presented a simple and effective obstacle avoidance algorithm.

3. Test Bed – Bearcat

3.1 Background

The Bearcat was first developed in 1992 as a project by students to take part in the International Ground Vehicles Competition. After undergoing many improvements it has evolved to Bearcat Brat [13]. A block diagram of the system is shown in Figure 3.1.It was designed to participate in 3 events:

1. Autonomous challenge
2. Follow the leader
3. Navigation challenge

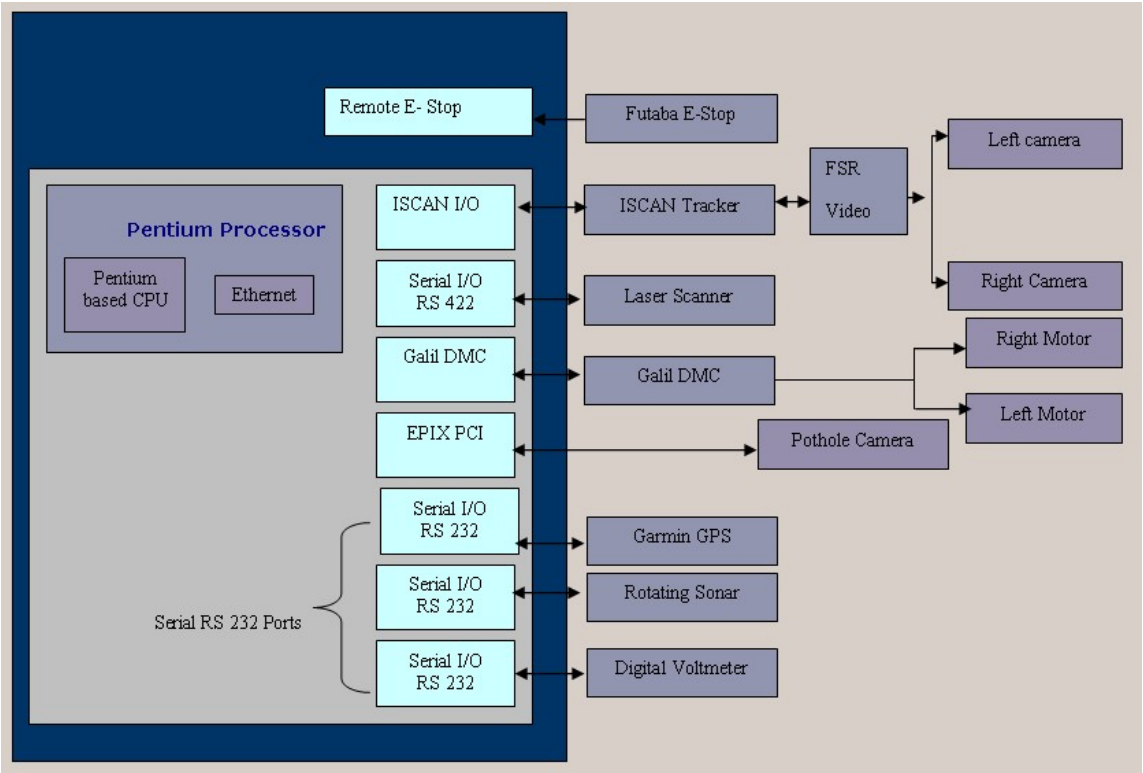

Figure 3.1: Bearcat Brat – Block Diagram

3.2 System Components

The Bearcat has the :

- Vision system,
- Motion control system
- Mechanical system
- Obstacle detection systems (sonar and laser)
- Global positioning system
- Electrical system
- Safety system
- Health monitoring system
- Central controller

Some important systems are discussed below in detail.

3.2.1 Vision System:

The Bearcat's vision system for the autonomous challenge has to track and follow 2 white lines which form a course. The vision system for line following uses 2 CCD cameras and an image tracking device (I-Scan) for the front end processing of the image captured by the cameras. I-Scan tracker processes the image of the line. The tracker processes the two-dimensional image from the camera and it locates the brightest position from the window as shown in the Figure 3.2 and sends the 2 image co-ordinates to the central controller. The central controller uses a four-point calibration system [14] to transform the image co-ordinates back to world co-ordinates for navigation purposes. The objective of the vision system is to identify the line and its location for the central

controller to direct the robot through the motion control system. At any given instant, the Bearcat tracks line either on the right side or left side. If the line is lost on one side the central controller changes the camera input using a video switch.

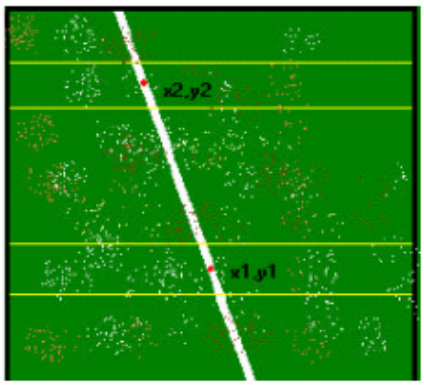

Figure 3.2: Capturing brightest point from 2 windows

3.2.2 Motion Control System:

The motion control system shown in the Figure 3.2 enables the vehicle to move along a path parallel to the track and to negotiate obstacles. Steering is achieved by applying differential speeds to the left and right wheels. Manipulating the speed of the left and right wheels, the velocity and orientation of the vehicle can be controlled at any instant. Two motors power the gear trains. The motor torque is increased by a factor of 40 using a worm gear train. The power to each motor is delivered through an amplifier that amplifies the signal from the Galil DMC motion controller. The data from the obstacle avoidance system works as an input to the central controller to give commands to the motion control system to drive the vehicle.

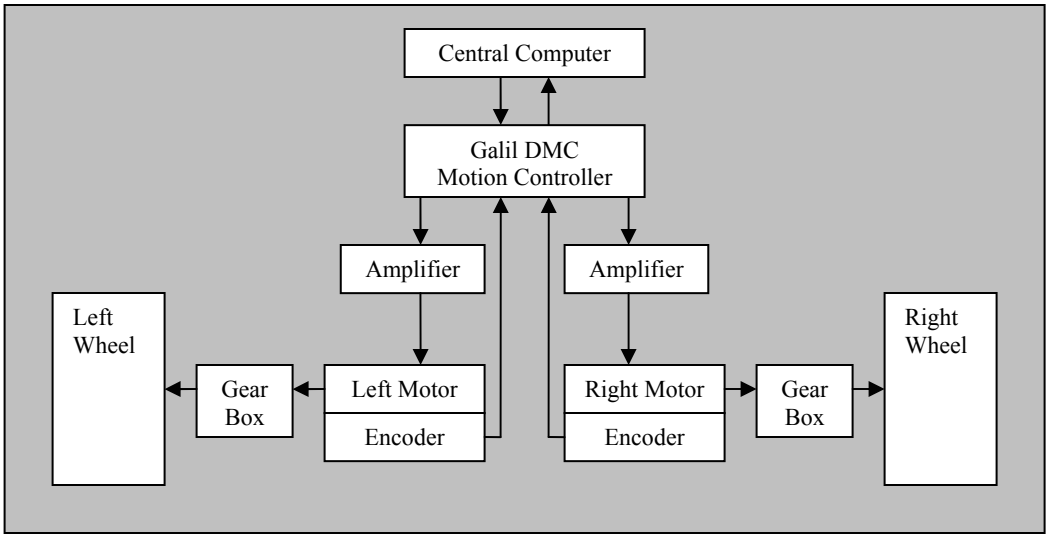

Figure 3.3: Motion Control System - Block Diagram

3.2.3 Motion Control System:

The robot frame is constructed using aluminum extrusions, joining plates, angles and T-nuts from 80/20 Aluminum Industrial Erector Set as shown in the Figure 3.4. The design makes it easy to reconfigure the frame for structural enhancements.

Figure 3.4: Mechanical System

The Bearcat III is designed to be an outdoor vehicle able to carry a payload of 100 pounds.

3.2.4 Laser Obstacle Detection System:

The Bearcat uses the SICK laser scanner (LMS 200) for sensing obstacles in the path. The power is supplied to the unit through a 24Volt-1.8 Amp adapter. The unit communicates with the central controller using a RS232 serial interface card. The maximum range of the scanner is 32 meters. For the current application, a range of 8 meters with a resolution of 1^{o} has been selected. The scanner data is used to get information about the environment of the robot. This is used to determine the profile of the environment and to determine possible obstacles in the path of the robot. The central controller performs the logic for obstacle avoidance and alternative path generation. Figure 3.3 shows the visibility and detection zones seen by the laser scanner.

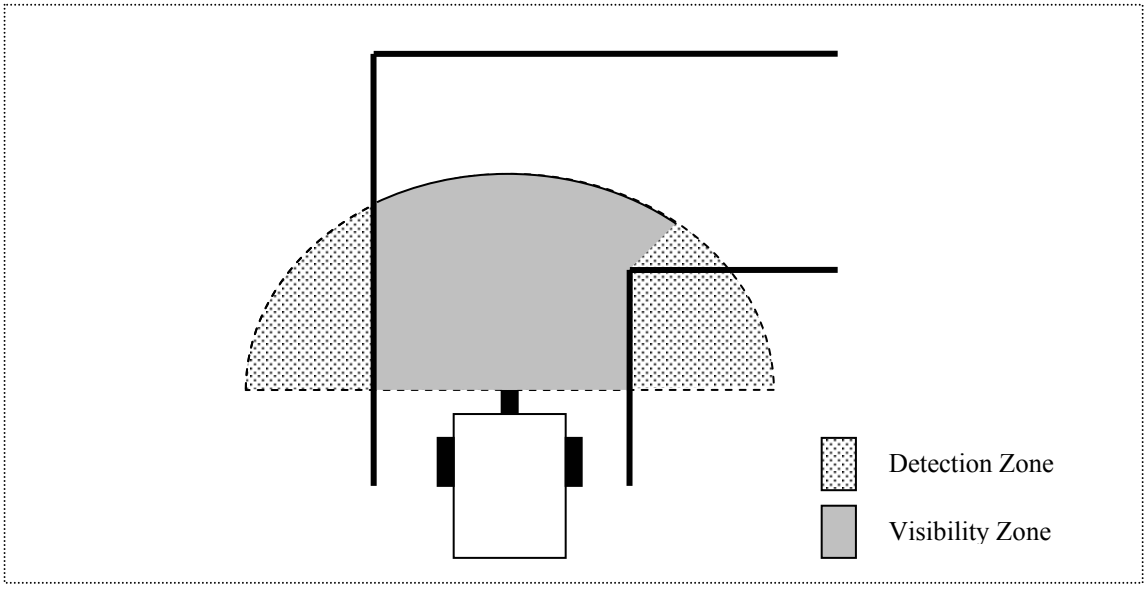

Figure 3.5: Robot navigation in a corridor

3.2.5 Sonar Obstacle Detection System:

The two main components of the ultrasonic ranging system are the transducers and the geared-motor. A 12 Volts DC, 0.5 Amps unit powers the sonar. A "time of flight" approach is used to compute the distance from any obstacle. The sonar transmits sound waves towards the target, detects an echo, and measures the elapsed time between the start of the transmit pulse and the reception of the echo pulse. The transducer sweep is achieved by using a motor and Galil motion control system. Adjusting the Polaroid system parameters and synchronizing them with the motion of the motor permits measuring distance values at known angles with respect to the centroid of the vehicle. The distance of the reflecting object is returned through an RS232 serial port to the central controller. The central controller uses this input to drive the motion control system. The range of this system is 40 feet

3.2.6 Global Positioning System:

The GPS navigation feedback control loop consists of the Garmin 76 GPS unit, the motion control system, laser scanner, and the central controller. Waypoint coordinates are fed to the central controller through a waypoint file during the initialization stage of the program and stored in an array in memory. The central controller sends the command signal, a NMEA message to the GPS unit via the RS 232 port which sets the active target waypoint in the GPS unit's memory. The waypoint coordinate is used by the GPS unit to calculate bearing, track, and range to the target waypoint. The 76 GPS unit transmits ASCII data output via the RS232 port containing the bearing, track, and range to the destination waypoint. The turn angle (angle error) is related to the track angle and bearing angle by the equation: Turn Angle = Track Angle – Bearing Angle.

This equation gives the turn angle in the 0 to 360 degree reference frame but this angle is transformed to 0 to 180 degrees (left turn angle) or 0 to -180 degrees (right turn angle) for the robot turning subroutine. The robot turns to the commanded correction turn angle if the turn angle is greater than 6 degrees or less than -6 degrees and then moves forward until the GPS position data are updated. When the robot arrives within 5 feet of the destination waypoint the next target waypoint is selected and this process is repeated until all targets have been reached. This process defines the discrete feedback control loop algorithm used for the robot GPS navigation course.

3.2.7 Electrical System:

The electrical system consists of a DC battery power system that supports an AC power system through an inverter. Three 12-Volt DC, 130 Amp hours, deep-cycle marine batteries connected in series provide a total of 36 Volts DC, 390 Amp hours for the main electrical power. A 36-Volt, DC input, 600-Watt inverter provides 60 Hz pure sine wave output at 115 Volts AC. The inverter supplies AC electrical power for all AC systems including the main computer, cameras, and auxiliary regulated DC power supplies. An uninterruptible power source (UPS) interfaces the robot main computer with the AC power system. The UPS provides 3 minutes of emergency power to the main computer during AC power system interruptions. The DC system provides 36 volts unregulated DC electrical power to the motors at a maximum of 10 Amps. The total power required by the Bearcat is approximately 735 Watts for the DC systems and 411 Watts for the AC systems. Thus, 1146-Watts total power is required to operate the Bearcat Brat. A loss of 10 percent was estimated for the required power to yield 1261 Watts actually required. A 10 percent loss can also be assumed for power supplied by the batteries to yield 4212-

watt hours available. Based on these estimates the Bearcat Brat power system has an estimated endurance of 3.34 hours at full load. A spare set of batteries is available and will be needed during the contest runs.

3.2.8 Central Controller:

The central controller or the brain of the Bearcat processes the inputs it receives from the sensing system. The motion of the Bearcat is decided by using a series of algorithms, which after processing its sensory inputs define a navigational path to travel.

The control logic is written in C++. A central computer running on both MS-DOS and Windows operating systems hosts the control program.

The fusion of the sensor inputs with the algorithms is achieved through high-level control logic that takes in the data from the sensor and makes high intelligent decisions to determine the path. The data from the obstacle avoidance system (laser scanner) is used to make decisions to control the motion of the robot.

4. Obstacle Avoidance – Proposed Solution

4.1 Introduction

This research is an adaptation from the author's undergraduate thesis [15] where a theoretical solution was put forward with a computer model for robot path planning to avoid a single obstacle in an unstructured environment. In the current research the same idea has been improvised to handle cases with multiple obstacles and has been implemented with modifications to suit the sensor input and the motion control resolution of the Bearcat.

Due to hardware limitations of the Bearcat sensor configuration the following assumptions limit the scope of the solution. The laser scanner detects objects only in a plane. The location of the detection plane depends upon the height at which the laser scanner is mounted. Hence all obstacles are assumed to be high enough for the laser scanner to detect. The same factor may also limit the system from detecting an object as a potential obstacle when the height of the obstacle varies or if the profile is such that there are holes in the object which might make the object invisible at the laser scanner's height. Hence, it is assumed that all the obstacles are perpendicular to the ground with a vertical profile.

The proposed solution has been developed for a static environment. In other words, all the potential obstacles are assumed to be stationary.

4.2 Methodology

The Figure 4.1 shows the components of the obstacle avoidance system and their functions.

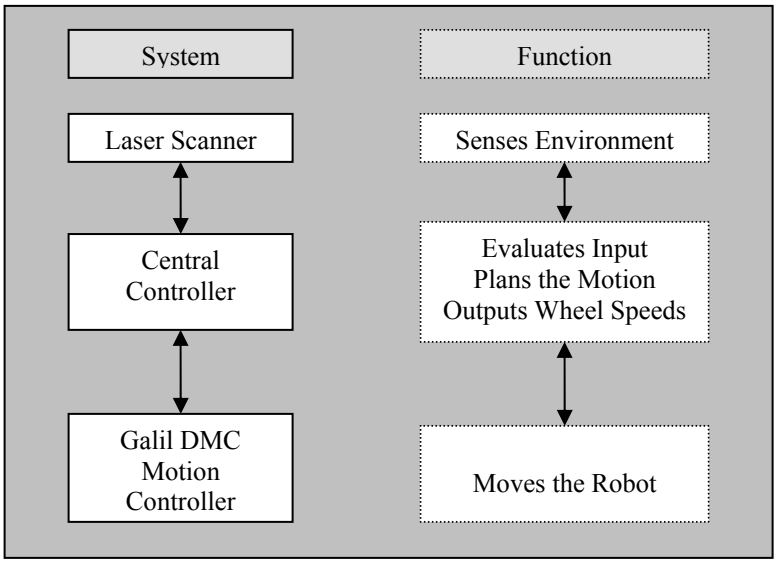

Figure 4.1: System's components and their functions

As explained in section 3.2.3, the laser scanner is set to scan a region of 8 meter radius with 1 degree resolution. With this configuration, at a distance of 8 m, the scanner will miss detecting objects that are as wide as 14 cm as illustrated in Figure 4.2. Under such circumstances, based on such data planning the path may give erroneous results. So that path should be checked for obstacles in immediate vicinity and the path planned accordingly.

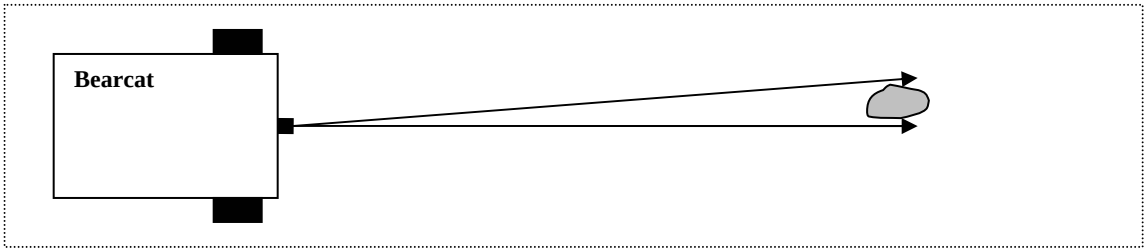

Figure 4.2: Objects at long distance missed due to the gap between successive rays

The feedback from the laser scanner consists of 180 data points. Often, the data is redundant and processing the data might be an overhead on the central controller which can be avoided.

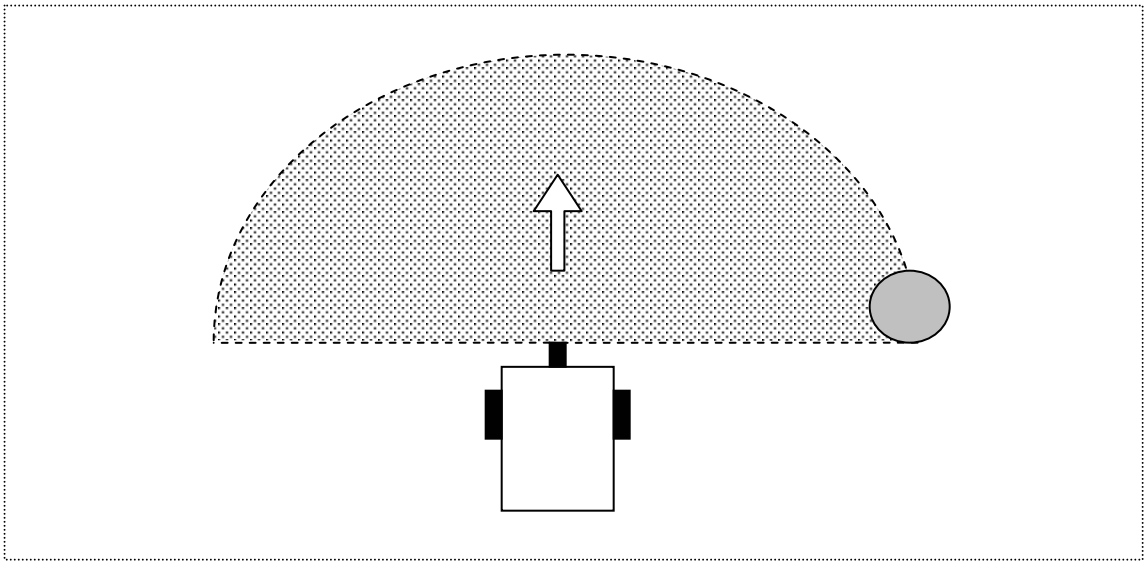

Figure 4.3: Obstacle far away from the robot, and not in the direction of movement

As illustrated in the Figure 4.3, the obstacle is far away from the robot and not in its path. It would be redundant to identify the obstacle to plan the path.

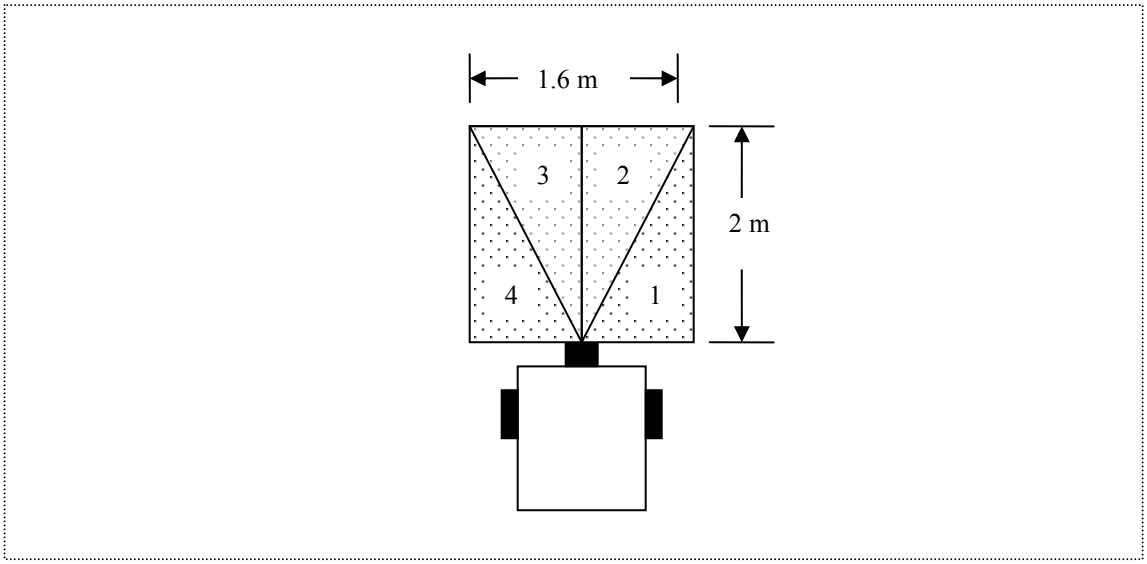

Figure 4.4: Frontal area checked for obstacle presence

For a “wandering” algorithm which is localized navigation, it is practical to filter the data and interpret it as required. Based on the above factors, in the proposed algorithm, the data from the laser scanner is processed only when the obstacle is identified to be a potential obstruction in the path. This is done by checking for presence of obstacle in the frontal area as shown in the Figure 4.4.

The dimensions of the frontal area were arrived at by considering 2 factors – 1) reaction time for the robot to detect and avoid an obstacle, and 2) clearance required by the robot for successfully maneuvering itself around the obstacle.

4.3 Algorithm

The frontal area described in section 4.2 is divided into 4 zones, as shown in Figure 4.4. The algorithm is very simple. It checks for the presence of obstacles in each zone, and according to the feedback it plans the robot’s path.

Every time the program gets a feedback from the laser scanner, a variable called *ob_factor* is initialized to zero and updated according to the following pseudo-code:

```
for(each zone, i = 1 to 4)
{
        if(obstacle is  present in zone)
        {
                ob_factor = (ob_factor*10) + i
        }
}
```

If there are no obstacles in the path, the variable *ob_factor* remains zero. If there are any obstacles, the variable accordingly provides information on the configuration of the obstacles. For instance, consider the configuration of obstacles as shown in Figure 4.5.

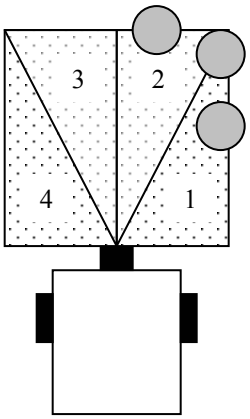

Figure 4.5: Obstacles in zone 1 and zone 2

In this case, for zone 1, *ob_factor*, which is initially *0* updates as follows:

*ob_factor = (0 * 10) + 1 = 1*

For zone 2, *ob_factor*, which is now *1* updates as follows:

*ob_factor = (1 * 10) + 2 = 12*

And it remains unchanged after zones 3 and 4 are processed as there are no obstacles present. The final value ob_factor is 12, which indicates the presence of obstacles in zones 1 and 2.

While checking for the presence of obstacle in each zone, if there is an obstacle present, the program also records three values, *ob_left*, *ob_right*, and *ob_near*. For each zone, the program sweeps the data array received from the laser scanner from the starting angle of the zone to the ending angle of the zone and checks for the presence of obstacle within the radial dimension, Figure 4.6.

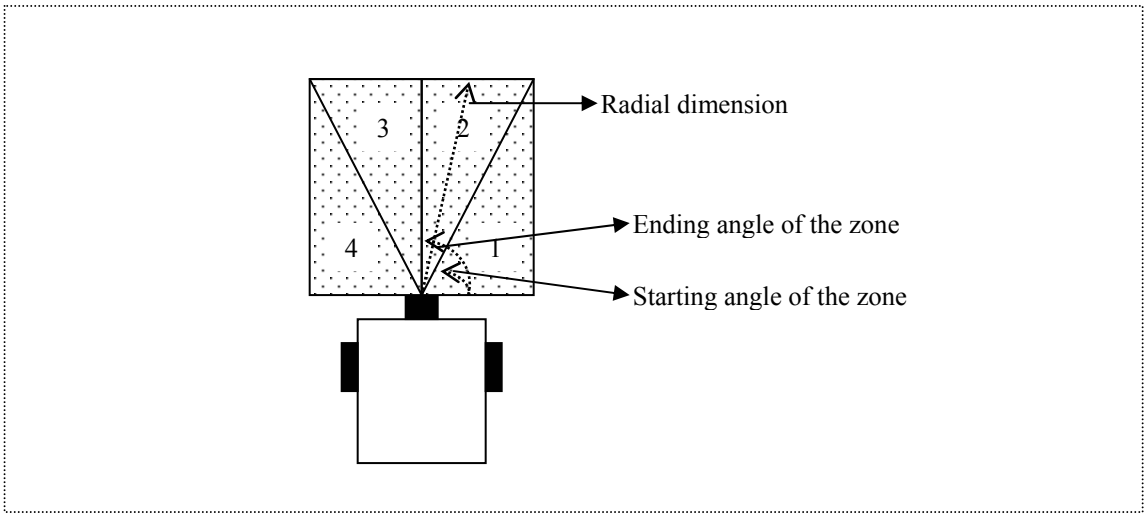

Figure 4.6: Angles and Radial Dimension

If the distance measured is less than that of the radial dimension of the zone, it indicates the presence of an obstacle. The 2 values *ob_left*, *ob_right* recorded indicate the angular points where the obstacle is present in the left and right extremes of the zone respectively. The value *ob_near* indicates the angular point where the obstacle is closest to the robot. The corresponding distances of the obstacle for the three angular points are also recorded for each zone.

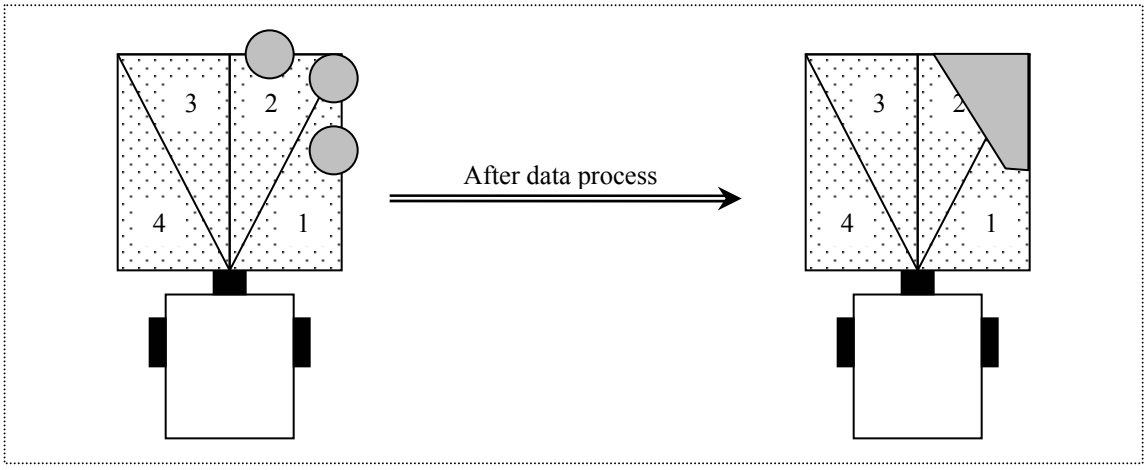

Figure 4.7: The obstacles as seen by the program after data process

Pseudo-code for laser data process in each zone:

```
for( θ from starting angle to ending angle of the zone)
{
        if( laserdata[θ] < radial dimension at θ)
```

```
    {
        if(the loop is entered for the first time)
        {
            ob_right = θ
            ob_right_distance =laserdata[θ]
        }

        if( laserdata[θ] < laserdata[θ-1])
        {
            ob_near = θ
            ob_near_distance =laserdata[θ]
        }

        ob_left = θ
        ob_left_distance =laserdata[θ]

    }
}
```

For the obstacle configuration discussed earlier, Figure 4.7 shows how the program sees the obstacles after the laser data is processed.

At the end of the data process, the program has the *ob_factor* value and the other values as discussed above. For the current algorithm, there are 15 different obstacle configurations possible:

- 4 cases when the obstacles are present in one of the zones

  ob_factor = 1 or 2 or 3 or 4
- 6 cases when the obstacles are present in any 2 zones

  ob_factor = 12 or 13 or 14 or 23 or 24 or 34
- 4 cases when the obstacles are present in any 3 zones

  ob=factor = 123 or 124 or 134 or 234
- 1 case when the obstacles are present in all zones

  ob_factor = 1234

For each case the path is exclusively planned by the program. For cases when the obstacles are present in multiple zones, the values of *ob_right*, *ob_left*, and *ob_near* and corresponding distances for each zone are processed and one set of values are finalized which gives a picture of the obstacle as shown in Figure 4.7.

Pseudo-code for path planning:

```
case(ob_factor among 1,2,12,123)
{
        turn left
}

case(ob_factor among 3,4,34,234)
{
        turn right
}

case(ob_factor among 23)
{
        turn left or right
}

case(ob_factor among 14)
{
        turn left or right or
        re-evaluate the array, check for clearance between the obstacles and proceed
}

case(ob_factor among 13)
{
        turn left or
        re-evaluate the array, check for clearance between the obstacles and proceed

}

case(ob_factor among 24)
{
        turn right or
        re-evaluate the array, check for clearance between the obstacles and proceed
}
```

case(ob_factor among 124,234,1234)
{
re-evaluate the array and decide if it is a deadlock or check for clearance
}

In each case, the angle and radius of turn are governed by the values *ob_right*, *ob_left*, and *ob_near* and their corresponding distances. Figure 4.8 shows special cases where there is a clearance between for the obstacles.

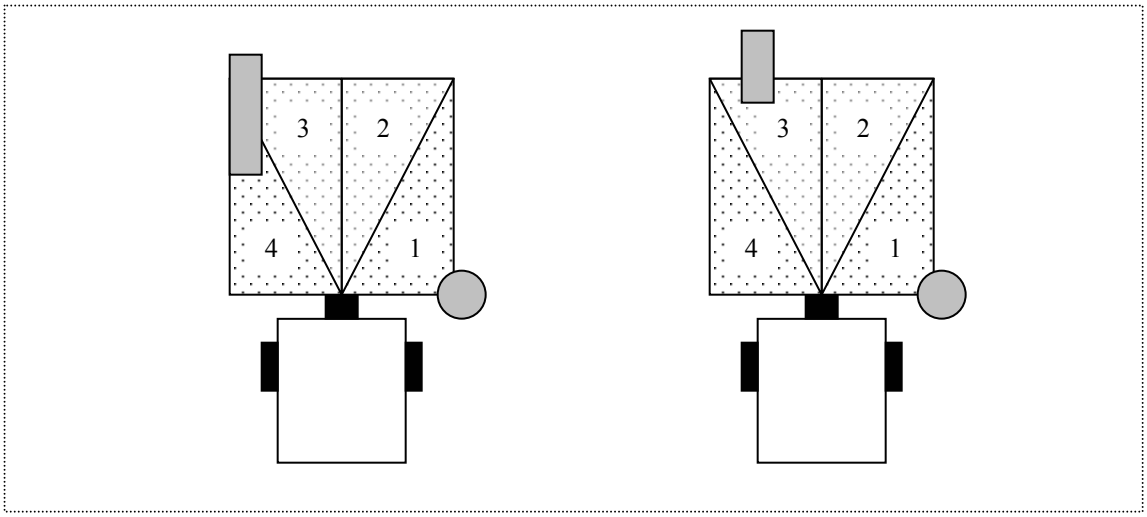

Figure 4.8: Cases with clearance between the obstacles

4. Results

The robot successfully avoided simple obstacles placed randomly in an open ground. It was successfully tested for all the cases discussed in section 4.3. However it failed under some situations where the robot, while steering to avoid an obstacle, gets itself into a position too close to another obstacle which was not in its view earlier. If the distance is too small for the robot to identify the obstacle and react, it hits the obstacle.

The solution was also tested and used at the 11th annual International Ground Vehicles Competition (IGVC) organized by the Association of Unmanned Vehicle Systems international (AUVSI). In the autonomous challenge event the robot has to follow a course formed by 2 white lines and avoid obstacles in the path. This becomes a structured environment and the Robot is confined to boundaries. The robot has to identify obstacles and avoid them with an alternative path and still remain in the course without crossing the lines. With little modifications the algorithm was used as part of the obstacle avoidance logic to generate alternative paths. The robot successfully avoided the obstacles in its path but it failed in a special case where it had to negotiate a ramp. As shown in the Figure 5.1, the robot failed to distinguish the ramp treating it as an obstacle and it crossed the boundaries to avoid hitting the ramp.

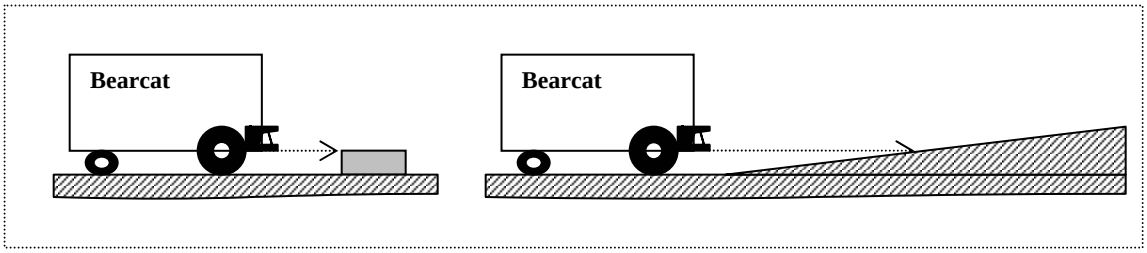

Figure 5.1: Failure to differentiate a ramp from a short obstacle

The points discussed in the introduction of the previous chapter (Section 4.1) also become limitations of the system. The limitations are not inherently due to the algorithm but some of them are due to hardware configuration limitations as discussed below.

The Bearcat has only one laser scanner. It scans a single plane at the mounted height limiting it from detecting obstacles which are either short or overhung as shown in the Figure 5.2. If the laser scanner is mounted at a shorter height, it will detect shorter obstacles but will lead to a different problem. When the terrain is uneven the scanner will detect the ground as an obstacle giving rise to misinterpretation of the environment.

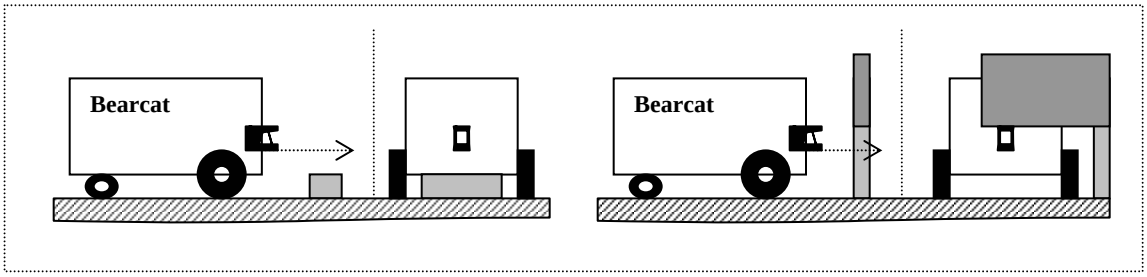

Figure 5.2: Failure to detect short and overhung obstacles

The algorithm fails to handle the case when the situation becomes dynamic. It will change its path to avoid all the obstacles it sees every time it scans the surrounding. It fails to take into account the trajectory of the moving obstacles. The alternate path taken by the robot may result in hitting the obstacle if both the robot and the obstacle move in the same direction.

The algorithm also fails when the configuration of obstacles becomes complex such that it wont have a way to go further. A simple case is illustrated in the Figure 5.3. The algorithm does not have any kind of memory mapping to retrace its path, so the robot comes to a stop when it enters a deadlock.

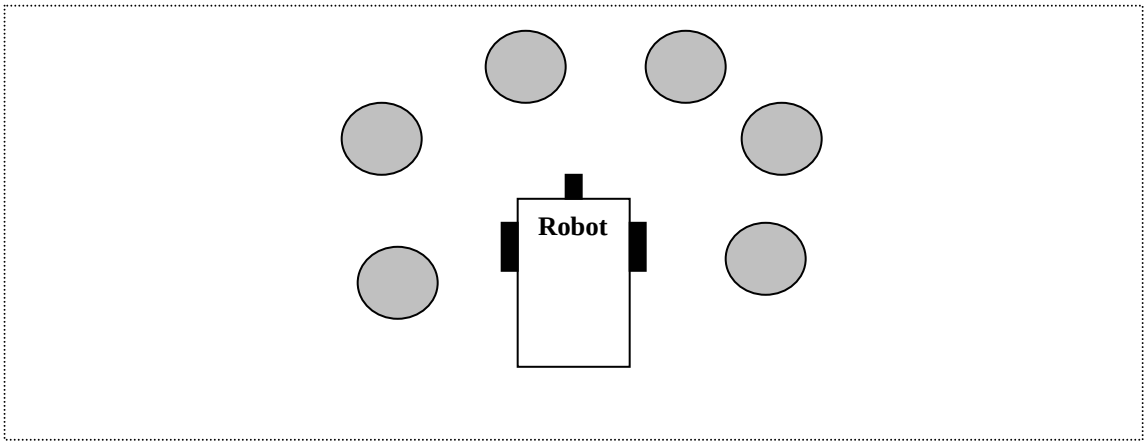

Figure 5.3: Robot in a deadlock

5. Future Direction

5.1 Conclusions

Obstacle avoidance is a challenging problem for mobile robots. Though the laser scanner is a state-of-the-art sensor which provides substantial input about the environment at appreciable rates, it fails to address the third dimension. The laser scanner is capable of scanning objects at long distances, but its resolution negates the benefits. Different kinds of sensors have unique advantages and also disadvantages at the same time. It will be very productive to integrate different kinds of sensors to design a system which will enable to overcome the limitations of the current obstacle avoidance system on the Bearcat.

5.2 Recommendations

This work provides a basis for developing the algorithm to overcome the limitations and give Bearcat the capability of global navigation.

Memory mapping can be implemented where the robot has a memory of the environment and it plans the path based on its current surroundings and the region it already navigated.

Memory mapping also enables the robot to get out of deadlock situations. An alternative solution would be using multiple sensors located around the robot. This will be possible to navigate out of deadlock situations without any kind of memory mapping.

A system with multiple sensors located at different heights can also be used to detect overhung obstacles. Differentiating a short obstacle from a ramp will still be a challenge. Stereo vision is a very good solution to evaluate the terrain. Information from a single stereo vision sensor can also detect overhung obstacles making the design and sensor integration easy.

Point to point navigation can be developed by *dead reckoning*. Dead reckoning is a simple mathematical procedure for determining the present location of the vehicle by advancing some previous position through known course and velocity information over a given length of time. The most simplistic implementation of *dead reckoning* is termed as *odometry*.

Heading information can be: 1) indirectly derived from an onboard steering angle sensor, 2)supplied by a magnetic compass or gyro, or 3) calculated from differential odometry. Incremental displacement along the path is broken up into X and Y components, either as a function of elapsed time or distance traveled. For straight-line motion, periodic updates to vehicle-position co-ordinates are given by:

$$x_{n+1} = x_n + V.t.\sin\theta$$
$$y_{n+1} = y_n + V.t.\cos\theta$$

where:

V = velocity of the robot

t = time of the motion

θ = robot heading

Potentiometers, Synchros and Encoders are some types of common sensors used for *odometry* sensing. The wheels of the Bearcat are equipped with encoders and the encoder feedback can be used for achieving point to point navigation. However, these rotational displacement sensors derive navigational parameters directly from the wheel rotation and are subject to give problems due to slippage, tread wear, tire imbalance, and improper tire inflation.

The algorithm can also be improvised to use the information from existing Global Positioning System (GPS) and navigate through waypoints based on latitude and longitude information.

Bibliography

1. M. Saxena, “Obstacle Avoidance Using Laser Scanner for Bearcat III”, Masters Thesis, University of Cincinnati, 2001.
2. J.C. Latombe, “Robot Motion Planning”, Kluwer Publishers.
3. H.R. Everett, “Sensors for Mobile Robots – Theory and Application”, A K Peters Ltd., 1995.
4. K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, “Robotics: Control, Sensing, Vision and Intelligence”, McGraw-Hill Book Company, 1987.
5. R.C. Arkin, “Motor-schema Based Mobile Robot Navigation”, International Journal of Robotic Research, Vol. 8, No. 4, 1989, pp. 92-112.
6. J. Borenstein, and Y. Koren, “The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots”, IEEE Journal of Robotics and Automation, Vol. 7, No 3, June 1991, pp. 278-288.
7. J. Borenstein, and Y. Koren, “Histogram In-Motion Mapping for Mobile Robot Obstacle Avoidance”, IEEE Journal of Robotics and Automation, Vol. 7, No 4, 1991, pp. 535-539.
8. D. Sekimori, T. Usui, Y. Masutani, and F. Miyazaki, “High-speed Obstacle Avoidance and Seld-localization for Mobile Robot Based on Omni-directional Imaging of Floor Region”, IPSJ Transactions on Computer Vision and Image Media Abstract, Vol. 42, No. SIG 13, 2001.

9. D. Fox, W. Burgard, S. Thrun, A.B. Cremers, “A Hybrid Collision Avoidance Method For Mobile Robots”, Proceedings of the IEEE International Conference on Robotics & Automation, 1998.
10. D. Castro, U. Nunes and A. Ruano, “Obstacle Avoidance in Local Navigation”, Proceedings of 10th Mediterranean Conference on Control and Automation - MED2002, Lisbon, July 2002.
11. I. Nourbakhsh, D. Andre, C. Tomasi, and M. Genesereth, “Mobile Robot Obstacle Avoidance via Depth from Focus”, Robotics and Automation Systems, Vol. 22, June 1997, pp. 151-158.
12. S. Soumare, A. Ohya and S. Yuta, “Real Time Obstacle Avoidance by an Autonomous Mobile Robot using an Active Vision Sensor and a Vertically Emitted Laser Slit”, The 7th International Conference on Intelligent Autonomous Systems (IAS-7), California, March 25-27, 2002.
13. Center for Robotics, University of Cincinnati, “Bearcat Brat Design Report”, 11th International Ground Vehicles Competition, June 2003.
14. S. Parasinis, “Four Point Calibration and a Comparison of Optical Modeling and Neural Networks for Robot Guidance”, Master’s Thesis, University of Cincinnati, Ohio, 1999.
15. V. Murty, “Alternative Path Planning for a Robot”, Undergraduate Thesis, Jawaharlal Nehru Technological University, India, 2000.

Appendix

Source Code

```
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include "nav.h"
#include "sicklms2.h"

//#include <constrea.h>

#define DMC_Status 1001
#define port1 0x03f8
#define port2 0x02f8

#define RADIAL 300

#define D_FACTOR 58
#define A_FACTOR 511
#define PI 3.14
#define RSPEED 4000

//RSPEED is the robot turning speed
//D_FACTOR = 1600 if distance is given in inches
//D_FACTOR = 630  if distance is given in cm

//Initializing functions

void submitDMC(char str1[15]);
void sendDMC(char str1[]);
void writeDMC(char a);
int getDMCValue();
int galilIsReady();
void steerCAR(float val);
void speedCARx(long int spx);
void speedCARy(long int spy);

void movexy(long int gdist, int speed);
void rotxy(long int g_angle, int speed);
void stopCAR();

int tracktarget(int a, int b, int aa, int bb);

void sensor();

SickLMS laser;
int datamm[361];


void main()

{
```

```
    int i;
    int datacm[181];
    int diff[180],filter1[180],filter2[180],filter3[180],filter4[180];
    int filter3_set;

    float cos_t1,sin_t1,cos_t2,sin_t2;

    sensor();
    filter1[0]=1;
    filter2[0]=1;
    filter3[0]=1;
    filter4[0]=1;
    filter3_set=0;
    datacm[0]=datamm[0]/10;

cout<<"\nAng"<<"\t"<<"cm"<<"\t"<<"diff"<<"\t"<<"filter1"<<"\t"<<"filter
2"<<"\t"<<"filter3"<<"\t"<<"filter4"<<"\n\n";

    for (i=1;i<180;i++)
    {
        datacm[i]=datamm[i]/10;
        diff[i]=datacm[i]-datacm[i-1];
    }

//************** FOR LOOP FOR FILTERS
    for (i=1;i<180;i++)
    {

/****IF LOOP for CALC****/
        if (i<=90)
        {
            cos_t1=(cos(i*PI/180.0));
            cos_t2=(cos((i-1)*PI/180.0));
        }
        else if (i>90)
        {
            cos_t1=(cos((180-i)*PI/180.0));
            cos_t2=(cos((179-i)*PI/180.0));
        }
/****IF LOOP Ends****/

        sin_t1=(sin(i*PI/180.0));
        sin_t2=(sin((i-1)*PI/180.0));

//        hor[i]= (datacm[i]*cos_t1)-(datacm[i-1]*cos_t2);
//        ver[i]= (datacm[i]*sin_t1)-(datacm[i-1]*sin_t2);

/****IF LOOP for FILTER1****/
        if(diff[i]<11 && diff[i]>-11)
        {
            filter1[i]=0;
        }
        else
        {
            filter1[i]=diff[i];
        }
/****IF LOOP Ends****/
```

```
/****IF LOOP for FILTER2 & FILTER3****/
        if(diff[i]<0)
        {
                filter2[i]=-1;
            if(filter1[i]==0)
            {
                if(filter3_set==0)
                {
                    filter3[i]=-1;
                }
                else
                {
                    filter3_set=0;
                }
            }
            else
            {
                filter2[i]=filter1[i];
                filter3[i]=datacm[i];
                filter3[i+1]=datacm[i+1];
                filter3[i-1]=datacm[i-1];
                filter3_set=1;
            }
        }
        else if(diff[i]>0)
        {
                filter2[i]=1;
            if(filter1[i]==0)
            {
                if(filter3_set==0)
                {
                    filter3[i]=1;
                }
                else
                {
                    filter3_set=0;
                }
            }
            else
            {
                filter2[i]=filter1[i];
                filter3[i]=datacm[i];
                filter3[i+1]=datacm[i+1];
                filter3[i-1]=datacm[i-1];
                filter3_set=1;
            }
        }
        else if(diff[i]==0)
        {
            filter2[i]=filter2[i-1]/abs(filter2[i-1]);
                if(filter3_set==0)
                {
                    filter3[i]=filter2[i];
                }
                else
                {
```

```
                    filter3_set=0;
                }
        }
/****IF LOOP Ends****/

    }

    for(i=1;i<180;i++)
    {
        if((filter3[i]/filter3[i-1])==(-1))
        {
            filter3[i]=datacm[i];
        }
    }

//********* FILTERS FOR LOOP ENDS

//*****************************
//*****************************

steerCAR(0);
while(!kbhit())
{

        sensor();
        datacm[0]=datamm[0]/10;
        for (i=1;i<180;i++)
        {
            datacm[i]=datamm[i]/10;
        }

        cout <<"\n Laser Data Acquired";


//********** CHECKING OPEN GAPS

int max_limit_no=0;
int fltr_flag,fltr_count,gap_size,gap_start,gap_end;
int max_start=0,max_end=0;
float gap_hor,gap_ver,gap_width;
float gap_hor1,gap_ver1,gap_width1;
float gap_hor2,gap_ver2,gap_width2;
int min_val;
float max_hor,max_ver,max_width=0.0;
fltr_flag=0;
fltr_count=0;
gap_size=0;

    for (i=1;i<180;i++)
    {
        filter4[i]=-1;
        if(datacm[i]>=RADIAL)
        {
            if(fltr_flag==0)
            {
                fltr_flag=1;
                fltr_count++;
```

```
                gap_start=i;
            }
            gap_end=i;
            gap_size++;
            filter4[i]=(gap_size*10)+fltr_count;
            max_limit_no++;
        }
        else
        {
            if(fltr_flag==1)
            {
                fltr_flag=0;
                if(gap_size>1)
                {
                    gap_hor =
(cos(gap_start*PI/180.0)*datacm[gap_start])-
(cos(gap_end*PI/180.0)*datacm[gap_end]);
                    gap_ver =
(sin(gap_start*PI/180.0)*datacm[gap_start])-
(sin(gap_end*PI/180.0)*datacm[gap_end]);
                    gap_width = sqrt ((gap_hor*gap_hor) +
(gap_ver*gap_ver));
                    gap_start--;
                    gap_end++;
                    gap_hor1 =
(cos(gap_start*PI/180.0)*datacm[gap_start])-
(cos(gap_end*PI/180.0)*datacm[gap_end]);
                    gap_ver1 =
(sin(gap_start*PI/180.0)*datacm[gap_start])-
(sin(gap_end*PI/180.0)*datacm[gap_end]);
                    gap_width1 = sqrt ((gap_hor1*gap_hor1) +
(gap_ver1*gap_ver1));

                    if(gap_width<gap_width1)
                    {
                        gap_hor2=gap_hor;
                        gap_ver2=gap_ver;
                        gap_width2=gap_width;
                        gap_start++;
                        gap_end--;
                    }
                    else
                    {
                        gap_hor2=gap_hor1;
                        gap_ver2=gap_ver1;
                        gap_width2=gap_width1;
                    }
                }
                if ( gap_width2>=160.0 )
                {
                    if(max_width==0.0)
                    {
                        max_width=gap_width2;
                        max_start=gap_start;
                        max_end=gap_end;
                        max_hor=gap_hor2;
                        max_ver=gap_ver2;
```

```
                    }
                    else
                    {
                        if(max_width<gap_width2)
                        {
                             max_width=gap_width2;
                             max_start=gap_start;
                             max_end=gap_end;
                             max_hor=gap_hor2;
                             max_ver=gap_ver2;
                        }
                    }
                }
                gap_size=0;
            }
        }
    }

//********** OPEN SPACE CHECK ENDS *************
//*********************************************************

        cout<< "\n MAX LIMIT :" << max_limit_no;               //no of
array elements with values greater than RADIAL
        cout<< "\n Open Gaps :" << fltr_count;
        cout<< "\n Open Gap Hor  :" << max_hor;
        cout<< "\n Open Gap Ver  :" << max_ver;
        cout<< "\n Open Gap Width:" << max_width;
        cout<< "\n Open Gap Start:" << max_start;
        cout<< "\n Open Gap End  :" << max_end;

int int_ob_flag=0,int_ob_pos=0;
float int_pi_angle=0;

//*********************************************************
//*******CHECKING FRONTAL AREA STARTS*****
//*********************************************************

    for (i=0;i<=68;i++)
    {
        int_pi_angle=i*PI/180.0;
        if (datacm[i] <= 80/(cos(int_pi_angle)))
        {
            int_ob_flag=1;
            cout << "\n RIGHT Zone:   BOOM BOOM!!! THERE IS AN OBSTACLE
\n";
            cout << "Dist \t Angle \n";
            cout << datacm[i] << "\t"<< i;
            int_ob_pos=i;
            break;
        }
    }

    for (i=69;i<=111;i++)
    {
        int_pi_angle=i*PI/180.0;
        if (datacm[i]< 198.0/(sin(int_pi_angle)))
        {
```

```
            int_ob_flag=(int_ob_flag*10)+2;
            cout << "\n MIDDLE Zone:  BOOM BOOM!!! THERE IS AN OBSTACLE
\n";
            cout << "Dist \t Angle \n";
            cout << datacm[i] << "\t"<< i;
            int_ob_pos=i;
            break;
        }
    }

     for (i=112;i<=179;i++)
     {
         int_pi_angle=(180-i)*PI/180.0;
         if (datacm[i]<80/(cos(int_pi_angle)))
         {
             int_ob_flag=(int_ob_flag*10)+3;
             cout << "\n LEFT Zone:   BOOM BOOM!!! THERE IS AN OBSTACLE
\n";
             cout << "Dist \t Pos Angle \n";
             cout << datacm[i] << "\t"<< i;
             int_ob_pos=i;
             break;
         }
     }

//*********************************************************
//********CHECKING FRONTAL AREA ENDS********
//*********************************************************


        if(int_ob_flag==0)
        {
            cout<<"\n ******************************Moving Straight";
            steerCAR(0);
            //steerCAR( (max_start+max_end)/2.0 - 90.0);
        }

        if(int_ob_flag==1)
        {
            //stopCAR();
            cout<<"\n ******************************Steering 10";
            steerCAR(10);
            int_ob_flag=0;
        }

        if(int_ob_flag==12)
        {
            //stopCAR();
            cout<<"\n ******************************Steering 15";
            steerCAR(15);
            int_ob_flag=0;
        }

        if(int_ob_flag==123)
        {
            stopCAR();
```

```
            cout<<"\n ******************************Obstacle Flag
:123";
            exit(0);
        }

        if(int_ob_flag==23)
        {
            //stopCAR();
            cout<<"\n ******************************Steering -15";
            steerCAR(-15);
            int_ob_flag=0;
        }

        if(int_ob_flag==3)
        {
            //stopCAR();
            cout<<"\n ******************************Steering -15";
            steerCAR(-15);
            int_ob_flag=0;
        }

        if(int_ob_flag==2)
        {
            //stopCAR();
            cout<<"\n ******************************Obstacle only in
middle";
            steerCAR( (max_start+max_end)/2.0 - 90.0);
            int_ob_flag=0;
        }

} //While Loop ends

} //End Main

//DMC Functions//

void submitDMC(char str1[15])
{
//This function when invoked will send the DMC a command, and if the
//command is legal and there are no errors it will say so.
sendDMC(str1);
};

void sendDMC(char str1[])
{
//This function when invoked will send a string to the DMC.
//Add a null character to the string
        char send[15];
        char *command=str1, *null = NULL;
        strcpy(send,command);
        strcat(send,null);


   if(galilIsReady())
   {            //Send character by character to DMC

                int i;
```

```
                for(i=0; send[i]; ++i) {writeDMC(send[i]);};
                writeDMC(13);
   };
}


void writeDMC(char a)
{
        outp(1000,a);
}

int galilhasInfo()
{
    return ((inp(DMC_Status)&32)==0);

}


int getDMCValue()
{
    /*
    while(galilhasInfo())
        cout<<inp(1000);
      */
      int j;
      while((j=int(inp(1000))) != '3')
      {
       //cout<<"did not get the value u wanted"<<endl;
       //getchar();
      }
      //cout<<j<<endl;
      return 0;
}



int galilIsReady()
{
        return((inp(DMC_Status)&16)==0);
}

//*********************************

// Functions to move CAR

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[10],sendx[15];
        gcvt(spx,6,inx);
        char *commandx="JG";
        strcpy(sendx,commandx);
        strcat(sendx,inx);
        system("gotoxy 10 21");
        cout<< " Left-motor: ";
        system("gotoxy 24 21");
```

```
        cout << sendx;
        system("gotoxy 1 23");
        cout << "X-motor --> ";
//if(!TEST)
        submitDMC(sendx);
        submitDMC("BGX");
}

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[10],sendy[15];
        gcvt(spy,6,iny);
        char *commandy="JG,";
        strcpy(sendy,commandy);
        strcat(sendy,iny);
        system("gotoxy 38 21");
        cout<<"Right-Motor: ";
        system("gotoxy 52 21");
        cout << sendy;
        system("gotoxy 1 24");
        cout<< "Y-motor --> ";
        //if(!TEST)
        submitDMC(sendy);
        submitDMC("BGY");
}


void steerCAR(float val)
{

int base_speed = 15000;
int spdx,spdy;

      if (val <= -30)
        {
                spdx = 10000;
                spdy = 4000;
        }

        if (val >= 30)
        {
                spdx = 4000;
                spdy = 10000;
        }

        if (val < 5 && val > -5)
        {
                spdx = base_speed;
                spdy = base_speed;
        }
        else
        /*if (val >= 5 && val <= 30 || val <= -5 && val>=-30)*/
        {
                spdx=base_speed-((134.5*val)/1);
                spdy=base_speed+((134.5*val)/1);
        }
```

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

        cout << "\t\tspdx = " << spdx  << "   ";
        cout << "spdy = " << spdy << "   ";

        speedCARx(spdx);
        speedCARy(spdy);
}


void movexy(long int g_dist, int speed)
{

        char sp[10],sendsp[15],sp1[10];
        char dist[10],sendun[15];
        int speY=20000;
        int speX=20000;
        gcvt(speX,6,sp);
        gcvt(speY,6,sp1);
        gcvt(g_dist,8,dist);

        char *commands="JG ";
        char *commandd="AD ";

        strcpy(sendsp,commands);
        strcat(sendsp,sp);
        strcat(sendsp,",");
        strcat(sendsp,sp1);

        strcpy(sendun,commandd);
        strcat(sendun,dist);

        cout << sendsp;
        cout << sendun;

        submitDMC("DP 0,0");
        submitDMC(sendsp);
        cout << "Sent the speed command";
        submitDMC("BG");
        submitDMC(sendun);
        cout << "Sent the limit command";
        submitDMC("ST");
        cout << "Sent stop command";
        submitDMC("AM;V=3;V=");
        getDMCValue();
        cout << "GOT DMC VALUE";
}


void rotxy(long int g_angle, int speed)
{

        int ispeed;
```

```
        char sp[10],isp[10],sendsp[15];
        char angle[10],sendun[15];

        ispeed=-speed;

        gcvt(speed,6,sp);
        gcvt(ispeed,6,isp);
        gcvt(g_angle,8,angle);

        char *commands="JG ";
        char *commanda="AD ";

        strcpy(sendsp,commands);
        strcat(sendsp,sp);
        strcat(sendsp,",");
        strcat(sendsp,isp);

        strcpy(sendun,commanda);
        strcat(sendun,angle);

        cout << sendsp;
        cout << sendun;

        submitDMC("DP 0,0");
        submitDMC(sendsp);
        cout<<" Sent the speed command";
        submitDMC("BG");
        submitDMC(sendun);
        cout<< "Sent the limit command";
        submitDMC("ST");
        cout<< "Sent stop command";
        submitDMC("AM;V=3;V=");
        getDMCValue();
        cout<< "GOT DMC VALUE";


}


void stopCAR()
{
        submitDMC("ST");
        submitDMC("ST");
        submitDMC("MO");
        submitDMC("MO");
        submitDMC("SH");
        cout << "\t\t\t\t\t\t\t\t\t\t\t\t\t    Stopped
Running\t\t\t\t\t\t\t\t\t\t\t\t";
}

//*******************************

// ****** Function to track target


tracktarget(int a, int b, int aa, int bb)
{
```

```
int p;

if(a<aa && b==bb)
p=1;

if(a<aa && b<bb)
p=2;

if(a==aa && b<bb)
p=3;

if(a>aa && b<bb)
p=4;

if(a>aa && b==bb)
p=5;

if(a>aa && b>bb)
p=6;

if(a==aa && b>bb)
p=7;

if(a<aa && b>bb)
p=8;

return(p);
}

//***********************************

void sensor()
{
        int key = 0;
        laser.readData();
        laser.getdistancearray(datamm);
//        laser.printData();
}
```