Navigation system for a mobile robot incorporating trinocular vision for range imaging

Kalyan Chakravarthy Pattisam
University of Nevada, Las Vegas
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
NAVI GATION SYSTEM FOR A MOBILE ROBOT
INCORPORATING TRINOCULAR VISION
FOR RANGE IMAGING

by

Kalyan C. Pattisam

Bachelor of Engineering
University of Mysore, India
1995

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Department of Mechanical Engineering
University of Nevada Las Vegas
May 1998
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

MARCH 27, 1998

The Thesis prepared by
KALYAN PATTISAM

Entitled
NAVIGATION SYSTEM FOR A MOBILE ROBOT INCORPORATING TRINOCULAR VISION FOR RANGE IMAGING

is approved in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN MECHANICAL ENGINEERING

George Tarr
Examination Committee Chair

Penry Clay
Dean of the Graduate College

W.B. Staff
Examination Committee Member

Allan
Examination Committee Member

Graduate College Faculty Representative

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABSTRACT


By

Kalyan C. Pattisam

Dr. Georg F. Mauer, Examination Committee Chair
Professor of Mechanical Engineering
University of Nevada, Las Vegas

This research focuses on the development of software for the navigation of a mobile robot. The software developed to control the robot uses sensory data obtained from ultra sound, infra red and tactile sensors, along with depth maps using trinocular vision. Robot navigation programs were written to navigate the robot and were tested in a simulated environment as well as the real world. Data from the various sensors was read and successfully utilized in the control of the robot motion. Software was developed to obtain the range and bearing of the closest obstacle in sight using the trinocular vision system. An operator supervised navigation system was also developed that enabled the navigation of the robot based on the inference from the camera images.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABISTRACT</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
</tr>
<tr>
<td>1.1 Review of past research</td>
</tr>
<tr>
<td>CHAPTER 2 MERCATOR - THE MOBILE ROBOT</td>
</tr>
<tr>
<td>2.1 Ultrasound sensors</td>
</tr>
<tr>
<td>2.2 Infrared sensors</td>
</tr>
<tr>
<td>2.3 Tactile sensors</td>
</tr>
<tr>
<td>2.4 Vision System</td>
</tr>
<tr>
<td>2.5 Communication</td>
</tr>
<tr>
<td>CHAPTER 3 NAVIGATION AND MOTION PLANNING - THEORY</td>
</tr>
<tr>
<td>3.1 Navigation</td>
</tr>
<tr>
<td>3.2 Motion planning</td>
</tr>
<tr>
<td>CHAPTER 4 DEVELOPMENT OF NAVIGATION SYSTEM</td>
</tr>
<tr>
<td>4.1 Operator Supervised Navigation</td>
</tr>
<tr>
<td>4.2 Robot Programming</td>
</tr>
<tr>
<td>4.3 Managing the project with make</td>
</tr>
<tr>
<td>CHAPTER 5 PATH PLANNING AND MOTION CONTROL ALGORITHM</td>
</tr>
<tr>
<td>5.1 Simulated and Experimental Results</td>
</tr>
<tr>
<td>5.2 Simulation I</td>
</tr>
<tr>
<td>5.3 Simulation II</td>
</tr>
<tr>
<td>5.4 Simulation III</td>
</tr>
<tr>
<td>5.5 Experimental Results</td>
</tr>
<tr>
<td>CHAPTER 6 IMPLEMENTATION OF THE VISION SYSTEM</td>
</tr>
<tr>
<td>6.1 Theory of Triangulation</td>
</tr>
<tr>
<td>6.2 Computation of Depth using Stereo Images</td>
</tr>
<tr>
<td>6.3 Range Image results</td>
</tr>
<tr>
<td>CHAPTER 7 CONCLUSION</td>
</tr>
<tr>
<td>CHAPTER 8 RECOMMENDATIONS</td>
</tr>
<tr>
<td>APPENDIX A Robot Motion Control Commands</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Map conversion algorithms</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>The iterative approach control scheme</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>The control architecture</td>
<td>7</td>
</tr>
<tr>
<td>Figure 1.4</td>
<td>Computational architecture and subsystems used for MARS</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Mercator - The B14 Mobile robot</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Sonar numbering on B-14 Mobile robot</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Three-camera module</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Camera module configuration</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Camera module specification</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Camera interface specification</td>
<td>19</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Camera software specification</td>
<td>19</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>Camera Performance</td>
<td>20</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Classification of environments</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Multi-process environment for operator controlled motion</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Flow chart Operator controlled navigation system</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Robot program function main() - general structure</td>
<td>35</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Maze constructed using Simulator Simulation of the Robot - path using hard coded program</td>
<td>42</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Simulation of the Robot path - using hard-coded program</td>
<td>43</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Simulation of the Robot path - using sensor based navigation</td>
<td>44</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Simulation of the Robot path - using sensor based navigation</td>
<td>46</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Actual x,y plot, and simulated robot path</td>
<td>48</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Stereo vision system using the triangulation principle</td>
<td>51</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>Coordinate system for the depth mapping</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>Structure of the main() function</td>
<td>54</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>Structure of the grabStereo() function</td>
<td>55</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>Top View of the robot with vision system showing dx and dy offset</td>
<td>59</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>Flow chart for highest disparity search</td>
<td>60</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>Images Obtained From The Right, Left And Top Cameras And The Corresponding Disparity Image In The Robotics Lab</td>
<td>64</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>Images Obtained From The Right, Left And Top Cameras And The Corresponding Disparity Image Of The Hallway</td>
<td>65</td>
</tr>
<tr>
<td>Figure 6.9</td>
<td>Original and Rectified Images of the Hallway</td>
<td>66</td>
</tr>
<tr>
<td>Figure 6.10</td>
<td>Original and Rectified Images of the Robotics Lab</td>
<td>67</td>
</tr>
<tr>
<td>Figure 6.11</td>
<td>Range and Bearing of test object in the Robotics Lab</td>
<td>68</td>
</tr>
<tr>
<td>Figure 6.12</td>
<td>Actual scene and 2D local map in floor plan style</td>
<td>69</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

At the outset, I would like to thank Dr. Georg F. Mauer Ph.D. for his invaluable support and guidance throughout this project. I would like to thank Dr. Woosoon Yim, Dr. Mohamed Trabia, and Dr. Evangelos Yfantis for their willingness to be part of my thesis committee.

My sincerest gratitude to all the professors and staff of the Mechanical Engineering Department of UNLV for educating me in the caring and considerate environment that they have provided for students over the years.

I dedicate this humble effort to my parents, whose motivation brought me this far and my entire family, and friends for their admirable support.
CHAPTER 1

INTRODUCTION

In recent years, the area of mobile robots has become very active in both research and development. Since its inception in 1960, robotics has generally focused on the design and control of robot manipulators. To a large extent, this has been motivated by the needs of the industry. Robots were merely powerful manipulators, rapid and precise, blind and deaf, living in extraordinarily ordered worlds. Recently however, robots have been liberated from their cages and new mutations have appeared with wheels, legs and wings. Research has come to a point where a mobile robot has been used to explore the surface of the planet MARS.

Mobile robots operating in such large, unstructured domains must be able to cope with high uncertainty in the position and identity of objects. In fact, the uncertainty is such that one of the most challenging activities for a mobile robot is simply going from point A to point B, a relatively trivial task for an industrial manipulator. To cope with uncertainty the robot must have the following two basic capabilities;

- **Sensory interpretation:** The robot must be able to determine its relationship to the environment by sensing. A wide variety of sensing technologies is available: odometry; ultrasonic, infrared and laser range sensing; and monocular, binocular and
• trinocular vision. These data need to be interpreted to understand what the sensor signal is telling us about the environment.

• **Reasoning:** The robot must be capable of deciding what actions are required to achieve its goals in a given environment. This may involve decisions ranging from what paths to take to what sensor to use.

  Machine perception, i.e., the recognition of objects is an important condition for achievement of robot intelligence [1]. As is true in humans, vision capabilities endow a robot with a sophisticated sensing mechanism that allows the robot to respond to its environment in an intelligent and flexible manner. Much like a trained worker, a robot should be capable of planning a work sequence from a high-level command, a prior knowledge of its work environment and real-time sensory information. The robot should plan and execute the sequence of operations autonomously, while adjusting in real time to unpredicted changes in its surroundings.

  A major task in robot navigation in industrial environment is collision avoidance and coping with uncertainty. In most manufacturing environments several different machines are involved, humans and other robots share a common work area. Information about the location of various machines, and area not traversible by a mobile robot are usually known in advance. Hence, the problem of autonomous navigation reduces to creation of a map/template of the work area and plan the motion of the robot based on its location at a given instant of time, and the target to be reached.

  Thus, developing a system that can effectively incorporate the various sensors and the vision system would be very useful for the development of a robust autonomous navigation system.
REVIEW OF PAST RESEARCH

Research in the field of robot navigation and mobile robotics has been developing over the years. Intensive research is in progress at various universities and research organizations throughout the world. The recent mission of exploring the surface of Mars using the "pathfinder" is a direct reflection of the progress in the field of mobile robotic research.

The Jet Propulsion Laboratory (NASA)-a government organization, Real World Interface-a non-government organization and schools like Brown university, Massachusetts Institute of Technology's Artificial Intelligence program, Carnegie Mellon University, Stanford University, University of California at Berkley, University of Edinburgh, UK, Seoul National University, Korea, Korea Institute of Science and Technology and many other organizations are involved in the research of navigation systems.

Research in the field of navigation is very wide, ranging from Fuzzy and Neural networks to simple numerical algorithms using environment world maps.

Canhg, Tsai-Yu et al., implemented a navigation system for mobile robots in dynamic environments using motion planning[6]. They divided motion planning into global path planning and local reactive navigation. The former uses genetic algorithms to find a collision-free path; the latter is implemented using neural network techniques. They found that the system could find a reasonably good path and achieve its goal. Neural networks have the ability to learn, but with some neural networks, knowledge representation and extraction are difficult [9]. Godjevac, Jelena and Steele Nigel studied the adaptive neuro-fuzzy controller for navigation of mobile robots. Though the fuzzy
systems were able to treat uncertain and imprecise information they had the drawbacks of difficulty in defining accurate membership functions and lack of a systematic procedure for the transformation of expert knowledge into the rule base [9].

Another approach to navigation is the use of mapping. Matsumoto, Tsutomu and

![Diagram of map conversion algorithms](image)

Figure 1.1 Map conversion algorithms [11]

Yuta, Shin'ichi proposed two levels of map information, i.e., a world map and a route map. The route map includes the geometric information on the path from the starting point to the destination [7]. Many conventional methods for map generation are time consuming, error-prone and necessary to transform the map into information available for the given task. Nakamura, Takayuki et al., researched the implementation of statistical methods for map generation. A graph representation of the environment was

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
constructed in which nodes and arcs correspond to the local structures and transition possibilities between them [10]. John A., Horst, at the Intelligent systems Division, developed algorithms that allow an intelligent system to dynamically convert between two representations of spatial occupancy, namely, certainty grids and object boundary curves. Sensory processing and world modeling components of an intelligent system for functionality of map conversion algorithms is shown in the figure. The author suggests work on executing the algorithms on image processing hardware and the integration of a local obstacle boundary map into the global obstacle boundary map [11]. Efficient, iterative, sensor based 3-dimension map building using rating functions in configuration space was researched at the Technical University of Braunschweig, Germany [12]. Kruse E. et al., used the "planning-sensing-updating" cycle for the exploration of an unknown three-dimensional environment. They assumed that the environment was not time varying and that the world is initially completely unknown. Robot motions and sensor measurements were planned to iteratively explore the unknown area. The iterative, incremental approach with the current world model as input is shown in the figure as the control scheme.

The inaccuracies and uncertainty problems faced in the navigation systems using mapping was overcome to a certain extent by the implementation of landmark based navigation. In this approach, sensory landmarks, such as wires buried under the floor, acoustic reflectors, laser reflectors, and ceiling landmarks are used to continually update the position of the robot with reference to a world. These systems actually reduced the flexibility of the system and made them impossible to navigate in new environments.
One such system based on the ceiling landmark recognition for autonomous mobile robot was developed at the Nagoya University, Japan [13]. Toshio Fukuda et al. applied the fuzzy and neural network algorithms for template matching and visual recognition. The algorithm employs a fuzzy template matching with the target object as well as the similar objects in the environment. The robot verifies the tentative recognition by comparing the likelihood of plus elements (target objects) with
that of minus elements (similar objects) [13]. To integrate this algorithm into the robot navigation system, a hierarchical control architecture was proposed. The control architecture is as shown in the figure.

![Diagram of control architecture](image)

**Figure 1.3 The control architecture [13]**

Franklin, David and Firby, James researched on integrating range and object data for robot navigation. The sensors were broadly divided into range sensors, which determine the distance, and, object-based sensors, that locate the objects. Information from the two types of sensors was integrated and implemented to navigate a robot through an office [8].

Exact knowledge of the position of a vehicle is a fundamental problem in mobile robot applications. Borenstein, j. et al., reviewed the relevant mobile robot positioning

Vision based systems is yet another approach to the development of an autonomous navigation system. A major contribution to this approach of research was the development of the vision based autonomous cart at the Stanford University. The Stanford Cart used a single camera to take nine pictures, spaced along a 50-cm track, and used the Moravec interest operator to pick out distinctive features in the images.

Running with a remote time-shared computer as its "brain", the Stanford Cart took about five hours to navigate a 20 meter course, with 20 percent accuracy at best,
lurching about one meter every ten to fifteen minutes before stopping again to take pictures, think, and plan a new path [16, 17]. A later development was the stereo vision based navigation for mobile robots in buildings. The Stanford mobile robot, known as the Mobile Autonomous Robot Stanford (MARS) was a 3-degree-of freedom omnidirectional configuration built by Unimation West. A stereo pair of cameras was mounted on a pan/tilt head, giving it two degrees of freedom. The computational architecture and subsystems used in this research is schematically shown in figure 1.4.

**Project Objective:** The navigation systems researched in the past and those being researched at present show that the quality and quantity of sensory signals used as input for the motion control and path planning is of considerable importance. With the development of new technologies in vision systems and sensing, an effective autonomous navigation system could be developed by integrating a wide range of aspects. In this research, we incorporate a trinocular vision system into the motion controller to obtain a visual depth map of the environment in the form of a floor plan, and also the range and bearing to the closest obstacle in view. The stereo data along with the range and bearing data obtained from the infrared and ultra-sound sensors are used for navigation in a completely unknown environment. This approach of using stereo range data along with the conventional sensors would have a significantly higher resolution in perceiving the environment as compared to a sonar map.
CHAPTER 2

MERCATOR - The Mobile Robot

The mobile robot here at the University of Nevada Las Vegas is named "MERCATOR" after the famous German Cartographer. Real World Interface Inc., located in New Hampshire, USA, commercially manufactures the robot. This is a highly capable yet compact mobile robot platform designed for robotics and Artificial Intelligence research.

The Base is part of the Mobile Robot where the motors, belts, wheels, gears, batteries and motion control electronics are located. The Base is 172 mm high and 340 mm in diameter. It is fully surrounded by a protective skin of six Smart Panels, which sense bumps, and provide protection for internal components [3].

Two motors provide translation and steering. Two 12 volt 8 Amp/hour battery packs provide the power to the entire robot. The base controller registers the motor current draw, encoder position and the battery voltage [3].

The Enclosure houses the sensors, controls and the main CPU of the Mobile Robot. The enclosure is mounted on the base with four screws and can be easily removed for transporting. There are 16 ultrasonic, 16 infrared and 16 tactile sensors mounted on the Smart panels. Integrated to the top of the enclosure is the console where most of the controls are located. The console is also designed to physically provide
mounting for the three cameras of the stereo vision system. Located at the center of the enclosure is the 233 MHz Pentium Pro CPU. The CPU communicates with the enclosure...
sensor via high speed Access bus and communicates with the Base via RS-232 serial. Two Multi-Sensor Processors (MSPs) mounted in the Enclosure control all of the sonar, infrared and tactile sensors.

ULTRASOUND SENSORS

*SONAR*, an acronym for Sound Navigation And Ranging, models the contours of an environment based on how it catches and throws back sound waves. The sender generates a sonic, or sound, wave that travels outwards on an expanding cone, and listens for an echo. The characteristic of the echo can help the listener locate objects [3]. Sonar uses the time of flight (TOF) technique for ranging. Time-of-flight ranging involves calculating the time it takes for a signal to reach and return from an object. Since distance equals the product of velocity and time, the range of an object must be:

\[ R = \frac{V \cdot t}{2} \]

Where;

\[ R = \text{range from the ranging device} \]
\[ V = \text{velocity of transmitted signal} \]
\[ t = \text{time it takes the signal to reach and return from object} \]

The robot is equipped with an array of 16 sonar transducers about 50 cms above the floor, around the perimeter of the upper enclosure. For the purpose of identification by the application program, the sonars are numbered; sonar zero is at the left rear of the robot; numbers ascend clockwise around the perimeter. The robot reads its sonars about three times per second. For each reading, the total time between the generation of the
ping and the receipt of the echo, coupled with the speed of sound in the robot's environment, generated an estimate of the distance of the object.

Figure: 2.2 Sonar numbering on B-14 Mobile robot

INFRARED SENSORS

Infrared light - is that part of the electromagnetic spectrum of radiation above 0.75 millimeters in wavelength and invisible to human eyes. The operating principle of the infrared sensors is the same as the sonars. In this case, infrared light instead of sound waves is used. The IR sensors are mounted along the lower perimeter of the enclosure. The IR sensors are numbered for the purpose of identification in programs in the same way as the sonars. IR zero is located at the rear left, ascending clockwise. The robot "reads" these sensors about ten times per second [3].

The infrared sensors have a short range - about half meter - and the reflectance value is heavily dependent on the color of the object detected. Mirrors, glass walls and solidly black objects tend to confuse the IR sensor. An integer represents the reflectance
value. This is the actual measurement of the IR's perception of the obstacle's reflectance. Values range from zero (no reflectance) to 100 (full reflectance) [3].

TACTILE SENSORS

Tactile, or touch sensing is required when intelligent robots must perform delicate operation and to protect the equipment on board in case an obstacle is encountered while navigating. The upper enclosure of the robot is equipped with Smart Panels, containing "bump switches", 4 on each door located more or less at the corners. There are 16 tactile sensors on the upper enclosure and 24 on the base. The robot senses pressure on any of these points such as firmly bumping against a person or an object. The tactile sensors return a value of zero or one. One means it has been touched and a zero means it has not.

VISION SYSTEM

Vision is an important aspect in "Machine Intelligence". Vision provides the most enhancement to an intelligent machine's abilities. Truly, intelligent machines, like robots, must be able to see if they are to perform human like tasks such as assembly, inspection and navigation. In addition, vision provides a means whereby an intelligent machine can acquire information and learn from its own environment, rather than being limited to the knowledge base provided by a programmer. Vision involves the transformation, analysis and understanding of light images. Image transformation involves the conversion of image features, such as brightness, color, and texture, into digital signals that can be stored and analyzed by a computer [5].
Figure 2.3 Three-camera module
Image analysis must provide enough information to the subsequent image understanding process so that it does not have to access any of the original image data. The first step in image analysis is to remove noise using hardware filters and software smoothing techniques. Gray scale, color, and texture data are then analyzed to find edge points, lines and regions. Edge detection and region analysis are the primary image-analysis techniques, which are used to define shape of objects.

The final task of a computer vision system is to interpret the information obtained from the image-analysis phase. Range finding and navigation are particularly important for robotics. The method of triangulation is used to find depth of objects using stereo vision.

The vision system mounted on the mobile robot is a triclops three-camera module that produces stereo vision. The three-camera module consists of three gray-scale 1/3" CCD cameras. The cameras are attached to an aluminum back plate so that the positions of the cameras relative to each other do not change. The camera module has electronics that synchronizes the three cameras such that images from all three cameras are obtained at the same time. The triclops system is distributed with a Matrox Meteor RGB frame grabber. The frame grabber is a standard PCI bus card and it is installed similar to other PCI cards [4]. A video cable is used for connecting the camera module to the computer. The three camera module is as shown in the figure 2.3. The diagram also shows the naming convention for the three cameras.
Figure 2. 4 Camera Module Configuration
VISION SYSTEM SPECIFICATIONS

<table>
<thead>
<tr>
<th>IMAGE SENSOR:</th>
<th>1/3&quot; CCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution:</td>
<td>512x492 Lines</td>
</tr>
<tr>
<td>Focal Length:</td>
<td>3.8 mm</td>
</tr>
<tr>
<td>HFOV:</td>
<td>71.5 degrees</td>
</tr>
<tr>
<td>Baseline:</td>
<td>10 cm</td>
</tr>
<tr>
<td>Size:</td>
<td>14x14 cm</td>
</tr>
<tr>
<td>Sync:</td>
<td>59.9 Gen-locked</td>
</tr>
<tr>
<td>Illumination:</td>
<td>50 lux, minimum 0.3 lux</td>
</tr>
<tr>
<td>Power:</td>
<td>9V @ 500mA</td>
</tr>
<tr>
<td>Weight:</td>
<td>450g</td>
</tr>
</tbody>
</table>

Figure 2.5 Camera module specification [Point Grey Inc.]
Interface Specifications

<table>
<thead>
<tr>
<th>Interface</th>
<th>Matrox Meteor single slot RGB PCI frame grabber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>RS170, 640x480</td>
</tr>
</tbody>
</table>

Figure 2.6 Camera interface specification [Point Grey Inc.]

Software System

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SAD - Sum of Absolute Difference Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Intel Pentium® or Pentium II® with MMX</td>
</tr>
<tr>
<td>Operating Systems</td>
<td>Linux 2.0, Windows 95/NT</td>
</tr>
</tbody>
</table>

Figure 2.7 Camera software specification [Point Grey Inc.]

TRICLOPS PERFORMANCE

The following performance chart shows the capabilities of the Triclops system running on a Pentium® MMX 166MHz and Pentium® II MMX 266MHz:
<table>
<thead>
<tr>
<th>RESOLUTION</th>
<th>INPUT AND RANGE IMAGE DIMENSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disparity range</td>
<td>Number of search steps during correlation</td>
</tr>
<tr>
<td>Bandpass filtering</td>
<td>Smoothing and normalizing images</td>
</tr>
<tr>
<td>Validation</td>
<td>Verifying correlation matches to improve robustness</td>
</tr>
<tr>
<td>Frequency</td>
<td>Running speed in frames/second</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Disparity Range</th>
<th>Bandpass/Validation</th>
<th>Frequency Pentium II 266 MHz</th>
<th>Frequency Pentium II 266 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>160 x 120</td>
<td>8</td>
<td>On</td>
<td>15</td>
<td>10.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Off</td>
<td>15</td>
<td>12.3</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>On</td>
<td>10</td>
<td>8.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Off</td>
<td>15</td>
<td>10.0</td>
</tr>
<tr>
<td>320 x 240</td>
<td>16</td>
<td>On</td>
<td>6.5</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Off</td>
<td>6.7</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>On</td>
<td>4.5</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Off</td>
<td>4.5</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Figure 2. 8 Camera Performance [Point Grey Inc.]

COMMUNICATION

Communication plays a vital role in the development of the navigation system for a mobile robot. Communication in a mobile robot can be mainly divided into two types:

- Communication between the CPU of the robot and its various sensing devices, and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
• Remotely accessing the computer on board the robot to perform all the required computations and control the motion of the robot while it is in motion. Obviously, a cable going into the robot would be a hindrance.

The CPU of the B-14 robot is built with six RS-232 serial ports (four are provided by an adapter board) and one parallel port, Access.bus controller, and Ethernet as standard equipment. The parallel port has a short ribbon cable that plugs into the front of the console PBC.

**Communication Busses**

The B-14 uses several different types of communications busses. A list of them follows:

**RS-232 asynchronous serial.** Each CPU comes with two RS-232 ports. The CPU uses /dev/cua0 to communicate with the base MCP (motion control processor). /dev/cau1 in most cases is used to communicate with the pan tilt unit.

Ethernet network. The robot CPU is equipped with an Ethernet adapter, which will allow it to communicate at a high speed (10Mbits/sec) with other computers on-board or off-board the robot. There are three popular ways of Ethernet cabling: 10baseT, 10base5, and 10base2. The B-14 is configured for the 10base2, which is called the "thinnet", and uses coaxial cable to daisy chain CPUs together. There are several rules about 10base2 Ethernet wiring that must be observed:

Network devices should be separated by a minimum of one meter of cable. The network must be terminated at both ends with terminating resistors. "T" connectors must...
be plugged directly into the devices on the network. No branches leading off the network at any point are allowed.

**Access.bus fast multi master asynchronous serial:** The Access.bus (A.b) is an extension of the I2C hardware. On the B14 A.b is used to link seven distributed MSPs (multi sensor processors) to the main CPU and to each other. Since the A.b is a multimaster bus any device on the bus may send a message at anytime with the arbitration being handled in hardware and software. An example of this is when the sonar sensors are fired MSP #20 synchronizes the two other MSPs on the enclosure allowing 4 of the 16 sonars on the robot to fire simultaneously. An A.b adapter card is installed inside the main CPU. On the B14 the standard 4 pin A.b cable is converted to a 16-pin ribbon cable with +5 volts, kill and reset added.

**Parallel port.** The CPU installed on the B14 has a parallel port, which controls the following;

1. Reading of the four colored function buttons on the top of the console.
2. Control of the four LEDs next to the four buttons.
3. Light control in emergency stop switch.

**Sonar sensor bus.** The sonar sensor bus is a ten-pin ribbon cable that connects all the discrete sensor boards on the enclosure doors to the MSPs that control them. The bus uses 10-pin ribbon cable. Four pins are used for addressing, four pins carry +5 volts & ground, and the remaining two pins fire and read the sonars.

**IR sensor bus.** The IR sensor bus is a ten-pin ribbon cable that connects the discrete sensor boards on the enclosure and base to the MSPs that control them. Three
pins are used for addressing, four pins carry +5 volts and ground, and the remaining three pins fire and read the IR sensors.

Communication between the workstation and the computer on-board the robot is established either by a RS232 radio modem or a wireless ethernet. The B14 is equipped with an RS232 radio modem operating in the frequency range of 900MHz. One radio modem connects to one of the free serial ports on the B14 CPU. The second radio modem is connected to the host computer and an external power source. For communication both radio modems should be set to "send" and set to the same channel either "A" or "B".
CHAPTER 3

NAVIGATION AND MOTION PLANNING - THEORY

NAVIGATION

The problem of navigation can be summarized as the following three questions:

- *Where am I?*
- *Where am I going?*
- *How should I get there?*

To answer the first question the robot should have a global picture of its surroundings and should also know its location with the world reference. The global picture and the location, would be based on what the robot sees using its various sensors. The second and the third questions can be answered using the motion planning and mapping [16].

The navigational capabilities of an autonomous robot can be divided into five sub-systems [2]:

*Localization:* refers to the means by which the position and orientation of the robot might be monitored.

*Environmental Modeling:* Capture and integration of sensory data and visual images to determine the occupancy of the room and the free area available for the robot to move to its destination.
Path Planning: This refers to the path that the robot will take in order to reach its destination based on the environment model and localization. The path would be dynamically changing based on collision avoidance.

Motion Control: This refers to the means by which and action devised by the path planner is actually executed by the robot using the various actuators. For wheeled robots, the control of rotation velocity and acceleration would be set based on the strategy planned.

Communication: All the analysis is performed by the use of a computer either on board or off board the robot or both. In such a case communication between the robot and the workstation controlling the robot is of prime importance. In addition, communication between the processing unit and the various sensors and visual devices is to be established without conflict between processes.

MOTION PLANNING

Planning future actions is one of the fundamental components for an intelligent mobile robot to carry out its tasks. Developing a plan is defined as finding a sequence of actions to accomplish a specified goal. The capability of motion planning in dynamic domains is important for autonomous robots. The robot is to perform tasks by itself without the need to specify every action that is to be taken. To realize such an autonomous robot, it is necessary to obtain information from the outside world using tactile, visual and auditory sensors. It needs to design an appropriate plan to handle unexpected events that arrive from the outside world. A dynamic environment can be characterized as either static or dynamic, and completely known, partially known or
unknown. In terms of this classification the environment can be categorized into six groups as shown in figure 3.1. Much of the prior work on motion planning is concerned with the case of completely known static environments -- the environment that contains

only stationary obstacles whose locations and sizes are precisely known ahead of time. Realistically, however, the environment may be partially known and dynamic, i.e., it can change over time. A robot that can deal with such dynamic obstacles will be capable of performing much larger and more complex classes of tasks.
CHAPTER 4

DEVELOPMENT OF NAVIGATION SYSTEM

Navigation systems for mobile robots are not trivial, and cannot be developed without extensive research. There has been considerable research in the field of mobile robot navigation. However, relatively few complete and working systems have been reported. To be mobile, a robot must be able to safely navigate within its environment and reliably get from A to B. In this research, the objective is to develop a navigation system using the state of the art technology. The fields of interest range from obtaining information from the various sensors to obtaining information from a 3D stereo depth image.

The first step was to be acquainted with the available resources and the style of robot programming. As described in chapter 2, the robot and its operating were carefully studied. The robot has a built-in collision avoidance server that would help navigate the robot in an unknown environment and be able to avoid obstacles. This was done using the sonars, infrared and the tactile sensors only. The robot lacked the visual perception in this mode of operation. The motion of the robot was planned and controlled by an operator. Our first goal in this research was to improve the visual perception of the robot system. This was achieved by incorporating a stereo vision system that enabled the robot and the operator, perceive the robot's surroundings. A detailed physical description of the
stereo vision system is presented in chapter 2. The theory and the analysis involved in stereo imaging will be described in the subsequent chapter.

OPERATOR SUPERVISED NAVIGATION

The next task was to develop an operator supervised navigation system using the camera system and the software supplied with the robot. The task was divided into two major parts, one is to obtain a real-time view of the robot's path and the obstacles in its vicinity. The second part was to enable the operator to command the robot to perform actions based on the camera image from a remote console. The camera system and the robot control software are independent of each other. The camera image system was first developed based on the application program interface provided by the Point Grey Research Inc., Canada. The objective was to have a close to real-time image of the robot's surroundings, that is a view of the area ahead of the robot in its direction of travel. A Matrox meteor frame grabber obtains the images from the cameras. The grabbed frame was then displayed on the monitor using programs written in C++ and Xwindows. The application program is set up to initialize the frame grabber and allocate memory for the image data. The frame is now grabbed, digitized and stored in the allocated memory space. This process of grabbing the image, digitizing, and saving the data in its allocated memory space is repeated in a cycle at a rate of five frames or images every second. The stored digitized image is now displayed on the visual display unit in a window. The process of grabbing the image frames and displaying them on the visual display unit is continuous or repeated infinitely, that required a dedicated shell. This created the need
for a multi-process and multi-shell environment. The schematic diagram of this multi-shell environment is as shown in the figure 4.1.

Figure 4.1 Multi-Shell, Multi-Process Environment For Operator Supervised Motion
The flow chart for the process in the operator-controlled navigation of the mobile robot is as shown in figure 4.2. The operator keeps a close observation on the camera image generated at a rate of 5 frames per second. The surroundings and the environment ahead of the robot are studied. The closest obstacle and its distance from the center of the robot along with its bearing are also available for the operator, upon which the corrective action...
could be based. The operator then commands the robot using the colliserver, a Beesoft robot motion management software that operates in guarded motion or collision avoidance mode.

The collision avoidance package uses only the sonars and has the capability to perform local navigation. Local navigation is the robot's "Sense as you go" mode of travel in which it pays attention only to the latest few readings coming from its sensors, rather than depending on any previous knowledge of the contours of its environment.

A set of commands to control the robot motion in the collision avoidance mode is listed in appendix-A along with a description for each command. The robot accepts a command and will execute it until it reaches the specified target position. If an obstacle occupies the target location, the robot will continue maneuvering to try to reach that point. A detailed procedure for the operator-controlled motion of the mobile robot is included in the appendix-B.

**ROBOT PROGRAMMING**

"Training" a robot to perform a task -- that is, programming it to accurately observe and reliably interact with its environment -- seems, at first, just a "trivial matter of programming". However, the embedded software must operate in noisy, messy, unpredictable, rapidly changing environments, rather than being safely enclosed within the abstract and relatively static cocoon enjoyed by programs that are more conventional. The robot programmer cannot follow the time-tested algorithm: "Observe, act and if no errors are reported exit and pursue the next task in line". The robot programmer must take special care to try to increase the likelihood that the action requested, does in fact occur.
A more reliable style of programming that uses an iterative and heuristic algorithm is implemented here. Observe, act, observer again to discern the results of acting, act again to correct inaccuracy in the previous action, observe again...." and so on, is embedded into the Beesoft software. The Beesoft Software is supplied with APIs (Application Program Interface) help in building programs that are reliable and efficient.

The first task was to understand the setup of the robot software and the purpose of the various servers. In any program the objective was to control the robot using the various API's provided by the software. The steps followed in achieving this were as follows:

1. Write the function to control the robot motion.
2. Write a module for the above function to be invoked.
3. Write code to:
   • Initialize the program
   • Create all beesoft and user modules needed, and
   • Start up the beesoft scheduler.

Once the programming is done, the next step is to compile and debug it. Since the robot software is an embedded system and uses multiple servers and modules to control the various sensors and motors, it is necessary to use a 'Makefile'.

MANAGING THE PROJECT WITH make

Make is a command generator. Using a description file and some general templates, it creates a sequence of commands for execution by the UNIX shell. These commands commonly relate to the maintenance of files comprising of a software project.
"Maintenance" refers to a whole array of tasks, ranging from status reporting and the purging of temporary files, to building the final, executable version of a complex group of programs[18].

Make is most naturally used to sort dependency relations among files. Even relatively small software projects typically involve a number of files that depend upon each other in various ways.

The 'makefile' describes the relationships among the files in a project and provides commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files.

Once a suitable makefile exists, each time some source files are changed, a simple shell command,

```
make
```
suffices to perform all the necessary recompilations. The make program uses the makefile database and the last modification times of the files to decide which of the files need to be updated. For each of those files, it issues the commands recorded in the database. The version of make used in this project is called the GNU make from the Free Software Foundation Inc., implemented by Richard Stallman and Roland McGrath [17]. A detailed description of the makefile is included in appendix-C and a copy of the Makefile is included in the accompanying disk. Once this is done the program is ready to be compiled and checked for errors.

The main() function of the C-program for any of the robot application is schematically shown in the flow chart of figure 4.3. The required servers and modules are first started up and then verified by the program to establish communication between
the application program and the various software modules that control the hardware. Once this is done the various programs that are linked to the application through the API's would be available for use. The next step before starting the application is to make sure the system exits the program without any conflict and in case of errors or unforeseen events shuts down all the servers running and the program itself. This is done by creating the RaiShutdown() module. Now the module for the application can be called and executed to perform the required action. The application program is in the form of a polled function that is continuously executed based on the polling interval specified. The motion control commands for the robot are scheduled by a scheduler. The scheduler helps to queue the tasks assigned to the robot and provides sufficient time for a given task to be completed before the new task is assigned.
Infinite loop will call raiShutdown() only in case of error or if user enters control-c

Figure 4.3 Robot Program's Function Main() - General Structure
CHAPTER 5

PATH PLANNING AND MOTION CONTROL ALGORITHM

The path planning and motion control algorithm used in developing a navigation system was mainly based on sensor data. The path planning decisions were made based on the range and the bearing of the closest obstacle detected. The robot path was corrected and made to move away from the detected obstacle. The motion controls for executing the decisions made by path planning were implemented using two methods. The first method was to control the robot motion by varying the rotate and translate velocities. The second method was to control the robot motion by translating finite distances and rotating finite angles.

Velocity based motion control: The control logic for the velocity based control method is as follows:

If on top of obstacle, stop. This is normal when just starting and all obstacles are initialized to range 0

\[
\text{if} \ (\text{range} < .01) \ \{ \\
\text{if} \ (\text{limp}) \ \{ \\
\text{rotateLimp}(); \\
\text{translateLimp}(); \\
\text{\} } \\
\text{else} \ \{ \\
\text{This function is used for setting the required translate and rotate velocities, the first}
\]

36
number is for translation and the second for rotation

```c
bSetVel(0.0, 0.0);
return;
```

The translate velocity is set to 0.0 if an obstacle is sensed and the range is equal to the safe distance.

```c
if (range==safeDistance) {
    transSpeed = 0.0;
}
```

If range is less than safe distance and, the bearing is less than 60 degrees, the translate velocity is negative, enabling the robot to move away to a safe distance from obstacle.

```c
else if (range<safeDistance) {
    if (fabs(bearing) < 60.0*M_PI/180.0) {
        transSpeed = -0.5*(safeDistance-range);
    }
    else {
        transSpeed = 0.0;
    }
}
```

The general equation for computing the translate velocity if none of the above cases are satisfied. The acceleration and safe distance are constant and the velocity is directly proportional to the range

```c
transSpeed = sqrt(2.0*(transAccel/2.0)*(range-safeDistance));
```

Translate velocity is limited to the maximum speed of the robot

```c
if (transSpeed>maxSpeed) {
    transSpeed=maxSpeed;
}
```

Assign the computed translate speed to translate velocity and initialize rotate velocity to 0

```c
transVel = transSpeed;
rotVel = 0;
```

// Computation of the rotate velocity
```c
if (range<10*safeDistance) {
```
if (bearing==0) {
    bearing=0.01;
}

Computation of the rotate velocity.

\[ \text{rotVel} = -30.0 / \text{bearing} / \text{range}; \]

Limit the maximum rotate velocity between \(-\pi/2\) and \(+\pi/2\) radians

\[
\begin{align*}
    &\text{if (rotVel} > \pi_{\text{by}}2) \{
        &\text{rotVel} = \pi_{\text{by}}2; \\
    \}
    \\
    &\text{if (rotVel} < -\pi_{\text{by}}2) \{
        &\text{rotVel} = -\pi_{\text{by}}2; \\
    \}
    \\
    &\text{if (rotVel} > \text{rotSpeed}) \{
        &\text{rotVel} = \text{rotSpeed}; \\
    \}
    \\
    &\text{if (rotVel} < -\text{rotSpeed}) \{
        &\text{rotVel} = -\text{rotSpeed}; \\
    \}
\end{align*}
\]

Finite distance and angle based motion control: The second method of motion control was by using the finite distances and angles. The decisions were based on the input sonar information. The base register was used for triggering the various sets of code for motion control. The control logic used was as follows:

The variables right, left, ahead, midright and mid left are the sensor readings obtained by averaging two adjacent reading 180 degrees in front of the robot as shown in figure 5.1.

If range on right side is greater than 3 meters turn right

\[
\begin{align*}
    &\text{if (right}>=3000) \\
    &\text{rotateRelativePos(0x100);}
\end{align*}
\]
If range on left side is greater than 3 meters turn left

```c
if(left>=3000)
    rotateRelativeNeg(0x100);
```

If the range ahead is less than 5.5 meters translate only 3.0 meters

```c
if((ahead <= 550) && (ahead >= 260))
{
    translateRelativePos(300);
    break;
}
else
{
    case SQE_TRAN_HALT:
```

Check for the sonar reading on the three sides i.e., front, left and right

```c
if(((right >= 3000) && (midright >= 2000)) || ((left <= 200) && (midleft <= 200))
{
    rotateRelativePos(0x80);
    break;
}
if(((left >= 3000) && (midleft >= 2000)) || (right <= 100) && (midright <= 100))
{
    rotateRelativeNeg(0x80);
    break;
}
if(midright<200)
{
    rotateRelativeNeg(0x35);
```
break; }
if (midleft<200)
{
    rotateRelativePos(0x35);
    break;
}

Check if ahead is greater than left and right if yes, move ahead by a meter

    if((ahead >=right) && (ahead >= left))
    {
        translateRelativePos(500);
        break;
    }

Else check if right is greater than left if yes turn right else turn left

    if(right>=left)
    {
        rotateRelativePos(0x55);
        break;
    }
    else
    {
        rotateRelativeNeg(0x55);
        break;
    }

break;
}

    case SQE_ROT_HALT:

        translateRelativePos(1000);
        break;

    default:
    translateLimp();
    rotateLimp();
    break;

The translation units are in millimeters and the rotation units are encoder pulses in hexa-
decimal. The relation between pulses and angle is that 512 pulses equal 180 degrees.

SIMULATED AND EXPERIMENTAL RESULTS

Before implementation of any new design, the best way to study the performance is to simulate using a computer. The various programs that were written for the control of the robot were first studied by simulation. The Beesoft motion management software

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
provides a built in Simulator that enables the programmer to visually study the performance of the control code. The simulator here uses the same baseServer that actually commands the base of the robot in a simulation mode. That is, the base of the robot does not actually move but the commands are used for moving a spot on the computer display, which represents the robot.

The simulator allowed studying the performance of three methods of controlling the robot motion. The first method was to use only hard coded values for linear distances and rotation angles based on the environment. For this a virtual maze was generated using the software, and the robot was navigated from the start point and back along the maze. The second and third methods of navigation were totally autonomous using the sonar, infra red and tactile sensors. In this method the robot would actually sense the objects around it and decide on the next action or motion to be performed. The first method of control is not autonomous and would not be applicable in new environments totally or partially unknown. The distances and angles need to be accurately measured before the values are hard coded into the program based on a fixed origin.

SIMULATION I

The first experiment was to control the robot using the hard coded method. For this purpose, a maze was created with the dimensions as shown in figure 5.2. The corners and the sides were labeled in a sequence so that it would enable the robot to know its position with reference to the global frame of reference and decide on what action needs to be performed. The control commands were triggered using the API provided by the Beesoft software that registers the status of the base at regular intervals. The interval at which the base register stack is read could be varied as required by the program. Two
states of the base were utilized in this program, one is the 'base translate halt' and the second is the 'base rotate halt'. These two states are registered in a stack by the base whenever the base performs a translation and stops or whenever the base performs a rotation and stops.

Figure 5.2 Maze Constructed Using Simulator

whenever the base performs a translation and stops or whenever the base performs a rotation and stops.
The maze was constructed using solid rectangles that acted as walls in the Beesoft simulator. The maze and its dimensions are shown in figure 5.2. The labels C1 through C14 denote the corners in the maze and S1 through S14 denote the sides. The origin of the robot was set at the top left corner of the maze.

The robot motion was simulated using the hard coded method and the results of the simulation are as shown in figure 5.3.

Figure 5.3  Simulation Of The Robot - Path Using Hard Coded Program

SIMULATION II

The second method of controlling the robot motion was by using the ultra-sound, infrared and tactile sensors. In this style of programming, the robot uses the value input from
the various sensors to perceive its surroundings and the open or free area around the robot. The program reads all of its sensors and obtains the range and bearing of the object that is closest. This data is then used for updating a target with reference to the position and orientation of the robot at that instant of time.

![Simulation of the Robot Path - Using Sensor Based Navigation](image)

**Figure 5.4** Simulation of the Robot Path - Using Sensor Based Navigation

The angle is obtained from the sensor based on the sensor numbering as discussed in chapter 2. The target point is then reset based on the new data obtained from all the sensors. The closest object sensed by either one of the three sensing devices namely the sonars, infrared and the tactile, takes precedence over the rest. This information is now utilized to set a target point. The robot motion is now controlled using the rotate velocity.
and the translate velocity based on the safe distance and travel time. The rotate and translate velocity are computed based on the range and bearing of the target. The rotate velocity is large if the bearing is small. The translate velocity is large if the range is large. This is because the safe distance and a safe travel time of two seconds is used for calculating the translate velocity. The basic mode of control in this method is to continually change the velocity of rotation and translation. The robot program was simulated, and the results of the simulation are as shown in figure 5.4.

SIMULATION III

A third method of controlling the robot motion was now studied. This was based on the range reading obtained from the sonars. The mode of controlling the robot motion was to use appropriate translate distance and a rotation angle based on the sonar readings. The translate and rotate velocities were constant. The motion decisions were a direct reflection of the range values in the east, west, north, northeast, and northwest directions of the robot. North corresponds to the direction ahead of the robot. That is, the sonar readings from the front 180 degrees were read at intervals of 45 degrees.

This program also enabled the robot to follow the nearest wall as a reference and maintain a safe distance from it by correcting its bearing at regular intervals. This was a basic solution to the problem of drift encountered in the hard coded motion control program. The performance of the program was studied by simulating various cases of hallways incorporating ' T ' and ' L ' junctions. The robot was started at different points with reference to the turns and in random orientations. The simulated paths taken by the robot in the various cases are as shown in figure 5.5. The robot was found to successfully navigate through the hallway and decide on its path.
EXPERIMENTAL RESULTS

The control programs for navigation of the robot were very successful in the simulation. The same control programs were now used to navigate the robot. The robot navigation using the hard-coded values of the environment was inefficient and lost its orientation on traveling distances in greater than two meters. The problem encountered was the errors generated by the robot itself, and the surroundings like the defects in the

Figure 5.5  Simulation of the Robot Path - Using Sensor Based Navigation
floor etc. The robot tends to drift as it moves larger distances and the drift is considerably high rendering the hard coded control method inapplicable.

The navigation of the robot using the second method that is, by updating the translate and rotate velocities based on the new target was very effective and the practical implementation was found to be very much similar to the simulation. The error and drift in the robot were continually corrected using sonar and infrared sensors that made the motion control more practical and applicable to real world environments. This program can be modified to incorporate the range and bearing values obtained from the stereo vision system.

The navigation of the robot in the hallway using the sensors to determine its direction and magnitude of travel was also studied. The control logic was based on the range values obtained from the ultrasound sensors. The readings of the sensors were averaged in pairs of two and the range in the east, west, north, northeast and northwest were obtained. The error in the drift of the robot was corrected using the wall as a reference and changing the orientation of the robot by 10 degrees for every meter using the walls as reference. This method was successful in navigating and maintaining a safe distance from the wall and at the same time incorporated a correction for the drift generated. The performance was very similar to the results obtained from the simulation.

**Experiment:** Robot Negotiating A 90° Bend In The Hallway Of The Engineering Building. The actual X and Y values of the robot position were plotted to generate the path traced. Figure 5.6 shows a comparison of one such x-y plot obtained while negotiating a 90° bend and the results from the simulation.
Experiment: Robot Negotiating a 90° Corner

Simulation of Robot path using simulator

Plot of Actual x and y for Robot Motion

Figure 5.6 Actual and simulated robot paths for autonomous robot motion around a 90° bend in the hallway of the Engineering Building
CHAPTER 6

IMPLEMENTATION OF THE VISION SYSTEM

A robust and sophisticated vision system is essential for a mobile robot to operate in the high levels of uncertainty posed by the modern industrial and research applications. Our robot is equipped with a trinocular vision system capable of depth imaging. The depth imaging enables the robot to perceive the ranges of the objects in view and plan its path accordingly. The depth imaging may also be used for object and pattern recognition and other tasks that need to be performed by the robot. The robot is now capable of a three dimensional space perception, which is more realistic as compared to a two dimensional space in the conventional camera vision. The third dimension involves determining the distance, or range, of all the points that define a scene, the results being a range image that complements the two dimensional image.

Two fundamental techniques are available for determining the third dimension, one is triangulation and the other is the time of flight technique. In this research we are using the triangulation technique for the depth imaging and the time of flight technique is used for range sensing with the ultrasound and infrared sensors.

THEORY OF TRIANGULATION

A vision system can employ two different triangulation techniques for measuring
the distance, or range to an object. One technique, passive triangulation, uses two image devices or cameras [5]. The other technique is referred to as active triangulation and employs a single imaging device, and a controlled or structured light source [5]. The vision system used in this research is based on the passive triangulation, and uses three imaging devices with one as a reference for the other two.

PASSIVE TRIANGULATION

Passive triangulation is also referred to as stereo or binocular vision. Two imaging devices are placed a known distance apart, as shown in figure 6.1. Two elements in the system are known, the distance between the cameras, \( d \), and the focal length of the camera, \( f \). The idea is to calculate the range, \( r \), from the cameras to a given point \( P \) in the scene or the object. Both cameras scan the scene and generate a picture matrix. Given any point in the scene such as a point \( P \), there will be two pixels representing that point. One pixel is in the left camera image and the other is in the right camera image.

Each pixel is located a given distance from the center of its image. Let \( x_1 \) be the distance that the left camera image pixel is located from the center of its image. Likewise, let \( x_2 \) be the distance that the right camera pixel is located from the center of its image. On overlapping the two images, the two image points, \( x_1 \) and \( x_2 \), will not coincide. Rather, there will be a certain distance between them. This distance is calculated using the Sum of Absolute Differences algorithm (SAD). The resulting distance is called the disparity between the two image points. The essential element in determining the depth maps is the disparity. In practice, the possible values of disparity are limited:
Because the observed points are necessarily in the half-space situated in front of the image planes and the optical centers of each camera.

- By the dimensions of the image
- By the dimensions of the observed scene, when finite.

The range, \( r \), from the cameras to the image point is inversely proportional to the disparity between \( x_1 \) and \( x_2 \). For instance, as the disparity approaches zero the range becomes infinite.

![Figure 6.1 Stereo Vision System Using the Triangulation Principle [5]](image)

Given the stereo system in figure 6.1, the range of any point can be approximated by the following equation:
Range = r = \frac{d \sqrt{f^2 + x_1^2 + x_2^2}}{|x_1 - x_2|}

Where,

- \( r \) is the range
- \( d \) distance between the lens centers (camera baseline)
- \( f \) focal length of the lens
- \( x_1 \) distance of point from center of image 1
- \( x_2 \) distance of point from center of image 2
- \(|x_1 - x_2|\) disparity

COMPUTATION OF DEPTH USING STEREO IMAGES

The baseline stereo method described in the previous section takes advantage of the redundancy contained in multi-stereo pairs, resulting in a straightforward algorithm that is appropriate for hardware implementation. The Triclops camera system implements this, and produces the output as a disparity depth map or an array of disparity values. To use the disparity values for computing the depth or range values of the objects in sight an appropriate coordinate system was established. The objective was to measure the distances from the center of the robot to a particular object. The coordinates used are as shown in figure 6.2. The measured distances are local distances, that is, with reference to the robots current orientation and location. To obtain the distances from the disparity image, computer programs were developed based on the software supplied by Triclops Stereo Vision system and the API's for the various functions.
Figure 6.2 Coordinate System for the Depth Mapping
The first task is to define the size of the images to be grabbed and stored. The following lines of code define the number of rows and columns in the images:

```
int nRows = 120+120;
int nCols = 160+160;
int triclopsRows = 120+120;
int triclopsCols = 160+160;
```

---

**Figure 6.3 Structure of the main( ) function**
Figure 6.4 Structure of the grabStereo( ) function
The next task is to define the call back functions that need to be called from the main() function. The structure of the main function is as shown in figure 6.3, incorporates creation of the various X applications and the box Widgets to hold the images. It also initializes and sets up communication with the function 'grabStereo()' that needs to be polled for grabbing the images and finally starts the X application. Once the grabStereo() function is initialized, it is ready to be polled at a specified rate to grab the images. The memory for the images that are grabbed and digitized is allocated during the first execution of the grabStereo() function and is done only once. The structure and components of the grabStereo() function are as shown in figure 6.4.

The GNU C++ compiler was used for compiling the C++ programs. Athena Widgets and X11 windows libraries were used for generating the screen windows to display the images. Calling predefined functions from the C++ programming language and the other Triclops and X-window libraries does this. The other file

```c
/*===============================================
 *= System Includes ==
*===============================================*/

#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <assert.h>
#include <time.h>
#include <values.h>
#include <unistd.h>

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Box.h>
```
essential for this project was the system file supplied as part of the Triclops Vision System software. This file was included for using the various API built into the system such as controlling the frame grabber card, initializing, capturing the required images, rectification, disparity-mapping etc.

The various functions used in this project were as follows:

**FindClosestObstacle()**

The inputs passed to this function are:

- dx: Offset in the X-axis between the robot center and the Camera unit
- dy: Offset in the Y-axis between the robot center and the Camera unit
- disp: The disparity image saved by the grabStere()

The 'dx' and the 'dy' offsets measurements are as shown in figure 6.5.

```c
void
FindClosestObstacles( TriclopsImage *disp,
                        int       closest[] )
{
    int nrows = disp->numRows;
    int ncols = disp->numCols;
    unsigned char *above, *center, *below;

    // make all closest disparities for each column = -1
    // which is the 'not checked' value

    for ( int c = 0; c < ncols; c++ )
    closest[c] = -1;
```
for ( int r = 1; r < nrows-1; r++ )
{
    above = &(disp->data[(r-1) * ncols]);
    center = &(disp->data[(r) * ncols]);
    below = &(disp->data[(r+1) * ncols]);

    unsigned char ref = *above;
    if ( ref == *center &&
        ref == *(center+1) &&
        ref == *below &&
        ref == *(center-1) )
    {
        // closest[c] = the closest (i.e. the maximum) disparity in
        // the disparity image in column ’c’

        if ( ref > closest[c] && // largest yet
            ref < INVALID_DISP && // not invalid code
            ref > floor ) // not the floor
            closest[c] = ref; // save this as the
        closest
    }
    center++; above++; below++;
}

The closest obstacle is the pixel with the maximum disparity in the image. The search for
the highest disparity value starts from the top left corner of the image and works down
and to the right. If there is a large blob of the same disparity which is the closest
obstacle, the point found will be the upper left corner of this blob. Another way to do this
is to take the centroid of the blob or the centroid of the pixels at that disparity. The first
step would be to initialize the maxdisp and the maxdispindex. 'maxdisp' is the maximum
disparity value. The maxdispindex is the column of the closest obstacle. The flowchart
for finding the maximum disparity is as shown in figure 6.6.
Figure 6.5 Top View of the Robot with Vision System Showing Dx and Dy Offset

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
For each pixel in the image $P(r,c)$

If disp is valid & greater than current maxdisp

No

Yes

set *col=c
set *row=r
set maxdisp=P(r,c)

Figure 6.6 Flow Chart For Highest Disparity Search

// find the closest disparity in the entire image
// (i.e. the maximum disparity in closest)
maxdisp = 0;
maxdisp_index = -1;
for ( int c = 0; c < disp->numCols; c++ )
{
    if ( closest[c] < INVALID_DISP &&
    closest[c] > maxdisp )
    {
        maxdisp = closest[c];
        maxdisp_index = c;
    }
}

Now that the maximum disparity and its coordinates are obtained, the range and bearing can now be calculated. This is done by calling the function GetRangeAndBearing( ).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
**GetRangeAndBearing( )**

The inputs to this function are the column value for the closest pixel or the highest disparity and the center column. The center column is the column that indicates the zero position on the X-axis. For a normal image, this is the half of the total number of columns. The disparity value of the closest pixel that is the value obtained from the search. Finally, the ’dx’ and ’dy’ offset from the robot center to the camera center or reference.

The outputs from the function are distance to the closest obstacle in millimeters and the angle in radians, with reference to the robot center and the obstacle, based on the coordinate system.

Next, find the point location in the world coordinate frame XYZ as shown in figure 6.2 and apply the correction ’dx’ and ’dy’ from figure 6.5. According to this system, bearing is off the X-axis and 90 degrees is straight ahead. The functions for the above-described procedure are as follows:

```c
void
GetRangeAndBearing( int col,       // in pixels
                      int centerCol,  // in pixels
                      int disparity,  // in pixels
                      double dx,      // in mm
                      double dy,      // in mm
                      double *range,  // in mm
                      double *bearing ) // in radians
{
    float X, Z;
    if ( disparity > 64 )
    {
        // illegal disparity... shouldn’t happen
        *range = 0.0;
        *bearing = 0.0;
        return;
    }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[ Z = \text{disparityToZ}(\text{disparity}); \]
\[ X = \text{colToX}(Z, \text{col}, \text{centerCol}); \]

// apply the \( dX \) \( dY \) correction to the measurement
\[ X -= dX; \]
// here it is +, because my \( X \) is in the opposite
// direction
\[ Z += dY; \]

// range is the absolute value of the vector \(<X, Z>\)
\[ \text{range} = \sqrt{X^2 + Z^2}; \]

// bearing in radians
\[ \text{bearing} = \text{atan2}(Z, -X); \]
return;
}

The function \text{disparityToZ}(\ ) is the function that converts the disparity values into
the distance along the \( Z \)-axis with reference to the robot coordinate system. Similarly the
\text{colToX}(\ ) computes the \( X \) value with reference to the robot coordinate system using the
column of highest disparity and the center column. The functions are as follows:

\text{inline float}
\text{disparityToZ}(\text{int} \ d) \{
\text{if (} \ d > 0 \text{) }
\{ \text{return (focalLengthX} * \text{cameraBaseline) / (float)} \ d; \}
\text{else if (} \ d == 0 \text{) }
\text{return 20000.0; // 20 m... close to infinity :) }
\text{else }
\text{return 40000.0; // 2 * infinity }
\}

\text{inline float}
\text{colToX}(\text{float} \ Z, \text{int} \ \text{col}, \text{int} \ \text{center}) \{
\text{return (float)} \ (\text{col} - \text{center}) * Z / \text{focalLengthX};
\}
The constants used for the above two functions are

```c
const float gridSize = 10.0; // mm per pixel
const float cameraBaseline = 100.499; // mm
const float focalLengthX = 133.178044; // mm
```

Fundamentally, the camera images all conform to the pinhole camera model. This means, for a location in the image plane \((u, v)\), the following relationships hold:

\[
Z = \frac{f \cdot b}{d}
\]
\[
X = \frac{Z \cdot u}{f}
\]
\[
Y = \frac{Z \cdot v}{f}
\]

Where \((X, Y, Z)\) is the location of a point in the 3-D world, \(f\) is the focal lengths in the X and the Y direction, \(b\) is the camera baseline, and, \(d\) is the disparity. The focal lengths in the X and the Y direction are 133.178044 mm for our model. The camera baseline, that is the distance between the centers of the two cameras is 100.499 mm. The right camera is the reference camera. The top camera is aligned with the reference camera, only shifted by the base along the Y-axis of the reference camera. The left camera is similarly aligned and shifted along the X-axis of the reference camera. Thus, the computation is based on the simple geometry and the two camera equations above. The results obtained from the range sensing using trinocular vision are as shown below. The accuracy depended mainly on the intensity of the image and the texture.
RANGE IMAGE RESULTS

The results obtained from the range images are as follows. There are three images, one each from the right, left, and top cameras, and the corresponding disparity image. The disparity image has a gray scale mapping, where white corresponds to the

![Disparity image](image1)
![Top camera image](image2)

![Left camera image](image3)
![Right camera image](image4)

Figure 6.7 Images Obtained From The Right, Left And Top Cameras And The Corresponding Disparity Image In The Robotics Lab

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 6.8 Images Obtained From The Right, Left And Top Cameras And The Corresponding Disparity Image Of The Hallway

Closest object and black corresponds to the farthest object. This is due to the fact that the object closest has the highest disparity value and the one farthest has the lowest disparity value. The original image grabbed by the frame grabber is of 480 by 640 pixels in size. This image is then rectified to obtain the 120 by 160 pixel image. The original and the rectified images are as shown in the figure. The original image clearly shows that the straight edges appear curved before rectification.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 6.9 Original and Rectified Images of the Hallway
Figure 6.10 Original and Rectified Images of the Robotics Lab
The range and bearing results for the following image were obtained as follows:

![Image of test object](image)

**Figure 6.11 Range and bearing of test object**

Range $= 337.047084$ cms and bearing $= 107.328477$ degrees. The closest obstacle was the black chair in the image. The actual range and bearing were 330.0 cms and 100 degrees.

The stereo data obtained was condensed to a single row and the range and bearing of each of these pixels was determined. A floor plan style of local map was generated using the 'x' and the 'z' values obtained from the single row disparities. The local map obtained is with reference to the center of the robot and is as shown in figure 6.12.
Figure 6.12 Actual scene and 2D local map in floor plan style

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER 7

CONCLUSION

This thesis developed software as part of the development of a navigation system using the ultra sound, infra red and tactile sensors, along with depth maps obtained from the trinocular vision system. The goals achieved by this thesis were incorporation the vision system and establish software for operator-controlled navigation and ranging using stereo images. Various methods of motion control for navigation was developed.

The operator-controlled navigation system was designed and tested. Three methods of navigation were developed and studied for autonomous navigation. One was to hard code the range and angle values of the environment into the program and base the robot motion on them. The second method was to use the sonar, infrared and tactile sensors to update the translate and rotate velocity based on the latest target position. The third method of navigating the robot was to use the range data obtained from the sonars and control the translation distance and the rotation angle.

The trinocular vision system was incorporated and the range and bearing of the closest obstacle was determined. It was found that the values obtained were very much dependent on the intensity and texture of the object in view.

The vision-based navigation could not be developed as the vision system and the robot motion control system were independent and there was no communication between.
the two processes. The communication was required since the range and the bearing world of multi process communication. Some of the available choices for multi-process communication are:

- Sockets
- Pipes
- Shared memory
- Message queues.

The programming language used was C and C++, which is object oriented, and development of specific applications can be based on the software developed. This project would be useful for further development of the navigation system and any specific application.
CHAPTER 8

RECOMMENDATIONS

The achievements of this thesis would enable the development of an autonomous navigation system. As of now, the vision system and the robot motion control programs are independent processes and run in separate shells. An inter process communication needs to be established that can utilize the data obtained from the range imaging in the control of the robot motion and path planning.

A global map of the environment could be created and used for path planning and navigation. The trinocular vision system can be used not only for navigation but also in the object recognition and pattern matching applications and research.

Since the research involves extensive programming an interdepartmental project should be planned and a team of students and faculty is required for developing a robust autonomous navigation system.

A mobile robot with range imaging and navigation capabilities has a wide range of applications in the real world. Specific applications could be developed based on the requirements.
APPENDIX A

ROBOT MOTION CONTROL COMMANDS

BASE Commands:

RP -- Rotation possible
RB distance -- RollBack
HOME -- Rotation home
BQ -- Battery query
STAT -- Status report

RH -- Rotate halt
RA acceleration -- Set rotation acceleration
RV velocity -- Set rotation velocity
R+ -- Rotate clockwise
R- -- Rotate anticlockwise
RR degrees -- Rotate degrees

TH -- Translate halt
TA acceleration -- Set translation acceleration
TV velocity -- Set translation velocity
T+ -- Translate forward
T- -- Translate backward
TR cm -- Translate cm SONAR Commands:
SLS maskno -- Start loop to get sonarReadings (use maskno)
SLE -- End loop to get sonarReadings
SLI no -- Set number of RT commands to be sent at once
SCM maskno -- Change maskno for loop sequence

COLLISION AVOIDANCE Commands:

CSM -- Set mode for target trajectory

0: DEFAULT_MODE
1: FAST_TRAVEL_MODE
2: FIND_DOOR_MODE
3: SERVO_MODE
4: ARM_OUT_MODE
5: RANDOM_MODE
6: APPROACH_OBJECT_MODE
7: APPROACH_TRASH_BIN_MODE
8: ARM_OUT_RANDOM_MODE

CTR -- Translate relative (forward, sideward)
C+ -- Go forward to target
C- -- Go backward to target
CDUMP -- Write collision information
CMARK -- Write a mark in the dump file
CGNU -- Write gnuplot file of evaluation function
D -- Debug: set base debug on
ND -- No Debug: set base debug off
STOP -- Stop movement until next command
? -- Help
. -- Exit
APPENDIX B

PROCEDURE FOR OPERATOR SUPERVISED NAVIGATION

The operator-controlled navigation is designed for controlling the robot manually using input about the environment from the camera images. This is done using the programs designed to display the camera image and the robot control commands from Appendix-A. The camera system and the robot control software are independent of each other. Hence, the camera image system was first developed based on the software provided by the TRICLOPS VISION SYSTEM. A copy of this directory is also included in the accompanying disks. The files in this directory require the configuration file from Triclops called "config.cfg" located in the directory

```
~/bee/triclops/export/config/config.cfg
```

To establish a symbolic link to the config files and to avoid copying them over the following command could be used:

```
$> ln -s ~/bee/triclops/export/config/config
```

this would create a link in the current directory which is represented as "config@". All files that have the "@" as their last character are symbolic links. Once the symbolic links are set up, it is necessary to set the environment for the display. This is done using the bash shell command:

```
$> VARIABLE=value
```
$> \textit{export \texttt{VARIABLE}}$

for this particular setup it is:

$> \texttt{TRICLOPS\_CONFIG\_PATH=\texttt{config/config.cfg}}$

$> \textit{export \texttt{TRICLOPS\_CONFIG\_PATH}}$

If the environment variable were not set the vision project would not be able to access the configuration files from the Triclops software and thus will not be able to grab any images or compute the disparities. Since the environment variable needs to be set every single time the session is started, it is easier to write a shell script for this. Once the project is compiled using the 'make' based on the relationships in the makefile an executable or target is created. The shell script can now be written as a new file containing the following lines;

```
#!/bin/csh
setenv TRICLOPS_CONFIG_PATH config/config.cfg
./distance
```

Since the computer on board is running on a Linux operating system, the working shell is changed from bash to \texttt{csh}, that is a C shell. The line \texttt{./distance} is the command for executing the target file generated from compiling the project. The above file should have the executable mode enabled by setting the 'chmod' to 775 using the command:

$> \textit{chmod 775 \texttt{vision}}$

where \texttt{vision} is a file containing the shell script.

\textbf{ROBOT MOTION CONTROL COMMANDS}

The robot motion control commands are built in the beesoft software. The robot is operated in guarded motion or collision avoidance mode. The collision avoidance
package requires the tcxServer and the baseServer to be running. Once these are started up, the colliServer can be started. To start the colliServer type

\$> \textit{colliServer -noindex -fork=no}

The \textit{noindex} argument simply keeps the robot from starting to move immediately. Without it, the robot will begin indexing, that is, rotating about its axis to establish orientation between its upper enclosure and its base. Also note that the commands need to be typed exactly as shown with mixed case since linux is case sensitive. Now the robot is ready to accept any commands listed in Appendix-A. All the commands typed in the robot control window should be in upper case. Now that the procedure to operate the camera system and the motion control system are explored, the operator controlled navigation can now be used. The procedure is as follows:

First, since the vision system and the motion control system are two independent processes they need to be run in two separate shells. For this we require at least two windows. Log on to the local host workstation with you login and password. Open a new shell or window. In window \#1 telnet to the computer on board the robot by typing

telnet mercator.me.unlv.edu

enter your login and password

repeat the same with window \# 2.

Now in window \#1 change into the directory containing the project for the vision system that is distance. Type \textit{cd distance}. To run the vision system type \textit{vision}. In window \#2 the motion control commands are typed. For this, start up the tcxServer, baseServer and the colliServer by typing

TcxServer
BaseServer

ColliServer -noindex -fork=no

Now press the caps-lock and type the motion commands in this window. The vision system will show two images on the environment ahead of the robot in a new display window, and the window #1 will have the range and the bearing of the closest obstacle sensed by the trinocular vision. The operator is now ready to navigate the robot. To stop the vision system just type control -C. And to stop executing the motion commands type a period (.)
APPENDIX C

THE MAKEFILE

The following Makefile includes the Beesoft supplied Configuration file called "Makefile.conf" and the rules file called "Makefile.rules". "Makefile.conf" contains a description of the system configuration with respect to the software, i.e., it describes the various directories, variables, compilers etc., that are to be used by the programs for compilation and execution. The makefile also defines the directories in which the various libraries are located for the compiler to search and find during compilation. The variable TARGETS in the file defines the target or the executable version of the project. The variable LIBS defines the various libraries that need to be linked for compilation and use by the various API's. The file "Makefil.depend" contains all the directories and paths for the included files with reference to the various programs written for robot programming.

Hello! How can I assist you today? Do you need help with understanding the content of this makefile? Let me know if you have any specific questions or if you need further clarification on any part of the makefile.
FFIND := \n    LIST="$(BEE_HOMES)" ; \n    for i in $$LIST ; do \n        if [ -f $$i/$$FILE ] ; then \n            echo $$i/$$FILE; \n            exit 0; \n        fi; \n    done; \n    exit 1

CONF = $(shell FILE=Makefile.conf ; $(FFIND))
RULES = $(shell FILE=Makefile.rules ; $(FFIND))

###
### System config variables
###

include $(CONF)

###
### Variables
###

TARGETS += project1

DEPENDENCIES += *.c
DISTCLEAN_FILES += $(TARGETS)

###
### local build variables
###

LIBS += -lbaseClient -lspeechClient -lpantiltClient
LIBS += -lcolliClient -larmClient
LIBS += -lraiclient -lrai -ltcx -lbUtils -lm $(OS_LIBS)

###
### Compulsory rules
###

all:
    $(MAKE) export $(DEPEND)
    $(MAKE) build

build: $(OBJ_DIR) $(TARGETS)

export:
clean:
    $(RM) *.o *.a *~ core a.out obj obj_sun obj_solaris
$(CLEAN_FILES)

distclean: clean
    $(RM) $(DEPEND) Makefile.depend* \ 
    $(NAME)-src.tgz $(NAME)-bin.tgz
$(DISTCLEAN_FILES)

##
## local rules
##

$(OBJ_DIR) $(LIB_DIR) $(BIN_DIR):
    $(SILENT) $(MKDIR) $

$(TARGETS): %: $(OBJ_DIR) $(OBJ_DIR)/%.
    $(ECHO) " ---- Creating %"
    $(SILENT) $(CC) -o % $(CFLAGS) $(OBJ_DIR)/%.
$(LIB_DIRS) $(LIBS)

##
## Standard suffix, version, packaging and dep rules
##

include $(RULES)

The Makefile for TRICLOPS

The following file is the Makefile used for the Camera system or the TRICLOPS vision system. This file defines the compiler for the vision or image programs to be GNU C++ since the codes are written in C++.

#******************************************************************************
# Name : Kalyan C. Pattisam
# Program : Makefile for vision system
# Date : October 15, 1997
# This file is based on the template makefile provided
# by Triclops Inc.,
# This is used for compiling, linking and creating an
# executable file for map.cc and radar.cc
#
#******************************************************************************

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
TOP   = ../

CXX   = g++
CXXFLAGS += -Wall -m486
CXXFLAGS += -O3 -g -fno-strength-reduce
CXXFLAGS += -I$(TOP)/include

CPPFLAGS += -DLINUX=1 -DM86=1
CPPFLAGS += -I$(TOP)/include
CPPFLAGS += -DMMX=1
CPPFLAGS += -DNDEBUG
CPPFLAGS += -DUNIX=1

LDFLAGS  += -L$(TOP)/lib

#
# universal rules....
#

%.o: %.cc
    $(CXX) $(CXXFLAGS) $(CPPFLAGS) -c $< -o $@

LDFLAGS  += -L/usr/X11/lib -L/usr/X11R6/lib
LIBS  += -ltriclops -lXt -lXaw -lXext -lX11

EXEC = distance
SRC = radar.cc map.cc

all: $(EXEC)

$(EXEC): $(SRC:%.cc=%.o)
    $(CXX) $(CXXFLAGS) $(LDFLAGS) -o $@ $^ $(LIBS)

clean:
    -rm -f *.o \  
    $(LIBRARY) $(EXEC)
APPENDIX D

DISPARITY TO DISTANCE TABLE

<table>
<thead>
<tr>
<th>Image size 160x120</th>
<th>Image size 320x240</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance (in m)</td>
<td>resolution (in mm)</td>
</tr>
<tr>
<td>6.4500</td>
<td>3546.7</td>
</tr>
<tr>
<td>4.4333</td>
<td>1520.0</td>
</tr>
<tr>
<td>3.8250</td>
<td>844.4</td>
</tr>
<tr>
<td>2.6600</td>
<td>537.4</td>
</tr>
<tr>
<td>2.2167</td>
<td>372.0</td>
</tr>
<tr>
<td>1.9000</td>
<td>272.8</td>
</tr>
<tr>
<td>1.6625</td>
<td>208.6</td>
</tr>
<tr>
<td>1.4778</td>
<td>164.7</td>
</tr>
<tr>
<td>1.3300</td>
<td>133.3</td>
</tr>
<tr>
<td>1.2091</td>
<td>110.1</td>
</tr>
<tr>
<td>1.1083</td>
<td>92.5</td>
</tr>
<tr>
<td>1.0231</td>
<td>78.8</td>
</tr>
<tr>
<td>0.9500</td>
<td>67.9</td>
</tr>
<tr>
<td>0.8867</td>
<td>59.2</td>
</tr>
<tr>
<td>0.8313</td>
<td>52.0</td>
</tr>
<tr>
<td>0.7824</td>
<td>46.1</td>
</tr>
<tr>
<td>0.7389</td>
<td>41.1</td>
</tr>
<tr>
<td>0.7000</td>
<td>36.9</td>
</tr>
<tr>
<td>0.6650</td>
<td>33.3</td>
</tr>
<tr>
<td>0.6333</td>
<td>30.2</td>
</tr>
<tr>
<td>0.6045</td>
<td>27.5</td>
</tr>
<tr>
<td>0.5783</td>
<td>25.2</td>
</tr>
<tr>
<td>0.5542</td>
<td>23.1</td>
</tr>
<tr>
<td>0.5320</td>
<td>21.3</td>
</tr>
<tr>
<td>0.5115</td>
<td>19.7</td>
</tr>
<tr>
<td>0.4926</td>
<td>18.3</td>
</tr>
<tr>
<td>0.4750</td>
<td>17.0</td>
</tr>
<tr>
<td>0.4586</td>
<td>15.8</td>
</tr>
<tr>
<td>0.4433</td>
<td>14.8</td>
</tr>
<tr>
<td>0.4290</td>
<td>13.8</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
This table represents the distance measurement that corresponds to a given disparity (displacement of the same feature between images) and the associated measurement resolution.

| 0.4156 | 13.0 | 32  | 0.8313 | 26.0 | 32  |
| 0.4090 | 12.2 | 33  | 0.8061 | 24.4 | 33  |
| 0.3912 | 11.5 | 34  | 0.7824 | 23.0 | 34  |
| 0.3800 | 10.9 | 35  | 0.7600 | 21.7 | 35  |
| 0.3694 | 10.3 | 36  | 0.7389 | 20.5 | 36  |
| 0.3595 | 9.7  | 37  | 0.7189 | 19.4 | 37  |
| 0.3500 | 9.2  | 38  | 0.7000 | 18.4 | 38  |
| 0.3410 | 8.7  | 39  | 0.6821 | 17.5 | 39  |
| 0.3325 | 8.3  | 40  | 0.6650 | 16.6 | 40  |
| 0.3244 | 7.9  | 41  | 0.6488 | 15.8 | 41  |
| 0.3167 | 7.5  | 42  | 0.6333 | 15.1 | 42  |
| 0.3093 | 7.2  | 43  | 0.6186 | 14.4 | 43  |
| 0.3023 | 6.9  | 44  | 0.6045 | 13.7 | 44  |
| 0.2956 | 6.6  | 45  | 0.5911 | 13.1 | 45  |
| 0.2891 | 6.3  | 46  | 0.5783 | 12.6 | 46  |
| 0.2830 | 6.0  | 47  | 0.5660 | 12.0 | 47  |
| 0.2771 | 5.8  | 48  | 0.5542 | 11.5 | 48  |
| 0.2714 | 5.5  | 49  | 0.5429 | 11.1 | 49  |
| 0.2660 | 5.3  | 50  | 0.5320 | 10.6 | 50  |
| 0.2608 | 5.1  | 51  | 0.5216 | 10.2 | 51  |
| 0.2558 | 4.9  | 52  | 0.5115 | 9.8  | 52  |
| 0.2509 | 4.7  | 53  | 0.5019 | 9.5  | 53  |
| 0.2459 | 4.6  | 54  | 0.4926 | 9.1  | 54  |
| 0.2418 | 4.4  | 55  | 0.4836 | 8.8  | 55  |
| 0.2375 | 4.2  | 56  | 0.4750 | 8.5  | 56  |
| 0.2333 | 4.1  | 57  | 0.4677 | 8.2  | 57  |
| 0.2293 | 4.0  | 58  | 0.4586 | 7.9  | 58  |
| 0.2254 | 3.8  | 59  | 0.4508 | 7.6  | 59  |
| 0.2217 | 3.7  | 60  | 0.4433 | 7.4  | 60  |
| 0.2180 | 3.6  | 61  | 0.4361 | 7.1  | 61  |
| 0.2145 | 3.5  | 62  | 0.4290 | 6.9  | 62  |
| 0.2111 | 3.4  | 63  | 0.4222 | 6.7  | 63  |

The above table was obtained from Pointgrey Research Inc as part of the Triclops camera system specifications.
REFERENCES


2. Ray Jars, "Distance transform based path planning for robot navigation, Recent trends in mobile robotics" 1993.


15. Franklin, David; Firby, R. James, "Integrating range and object data for robot navigation", Proc. International Conference on Autonomous Agents, p185-192 1997


23. Olivier Faugeras; Bernard Hotz; Herve Mathieu; Thierry Vieville; Zhengyou Zhang; Pascal Fau; Eric Tjeron; Laurent Moll; Gerard Berry; Jean Vuillemin; PatriceBertin; Catherine Proy," Real-time correlation-based stereo; algorithm, implementations and applications", Institute National De Recherché En Informatique Et En Automatique, INRIA, France.

VITA

Graduate College
University of Nevada, Las Vegas

Kalyan C. Pattisam

Home Address:
7913, Canoe Lane
Las Vegas, NV 89128.

Degrees:
Bachelor of Engineering, Mechanical Engineering, 1995
University of Mysore, India

Dissertation Thesis Title:

Dissertation Thesis Examination Committee:
Chairperson, Dr. Georg F. Mauer, Ph. D.
Committee Member, Dr. Mohamed B. Trabia, Ph. D.
Committee Member, Dr. Woosoon Yim, Ph. D.
Graduate Faculty Representative, Dr. Evalgelos A. Yfantis, Ph. D.