IMPLEMENTATION OF A SENSOR FUSION-BASED OBJECT-DETECTION COMPONENT FOR AN AUTONOMOUS OUTDOOR VEHICLE

By

DAVID KEITH NOVICK

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2002
Copyright 2002

by

David Keith Novick
I would like to dedicate this dissertation to my wife, Robin, for being the light at the end of the tunnel, and especially helping to dig me out during cave-ins.
ACKNOWLEDGMENTS

I would like to express my thanks to my entire committee - Dr Carl Crane, Dr. Antonio Arroyo, Dr. Christopher Niezrecki, Dr. Jörg Peters, and Dr. John Schueller, especially to those who found out at the last minute they were on my committee, this include Dr. Erik Schwartz for sitting in for Dr. Arrory.

Special thanks goes out to Carl Crane who has been my advisor these many long years, and to Dave Armstrong, the project manager, who as always kept me on my toes by constantly changing the name of what I was working on. Also thanks to the gang at the Air Force Research Lab at Tyndall for their support in the ongoing research at University of Florida.

To my wife Robin, who has supported me all along with way. Initially with emotional help, then branching into computer support by developing the user interface used to express graphically the interactions and goings on of my research. And for the use of his laptop, Pete Vinch, both during my defense, and in converting my dissertation into electronic format.

And finally to anyone else who I’ve used to bounce ideas off of when working on my research.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Background</td>
<td>2</td>
</tr>
<tr>
<td>Simulation</td>
<td>7</td>
</tr>
<tr>
<td>Real World</td>
<td>9</td>
</tr>
<tr>
<td>2 LITERATURE REVIEW</td>
<td>20</td>
</tr>
<tr>
<td>Ultrasonic Transducer</td>
<td>20</td>
</tr>
<tr>
<td>Planar-structured Light</td>
<td>23</td>
</tr>
<tr>
<td>Laser</td>
<td>24</td>
</tr>
<tr>
<td>Camera</td>
<td>28</td>
</tr>
<tr>
<td>Shape from Shading</td>
<td>29</td>
</tr>
<tr>
<td>Stereo Vision</td>
<td>30</td>
</tr>
<tr>
<td>Sensor Fusion</td>
<td>32</td>
</tr>
<tr>
<td>3 RESEARCH GOALS</td>
<td>36</td>
</tr>
<tr>
<td>Statement of Problem</td>
<td>36</td>
</tr>
<tr>
<td>Assumptions</td>
<td>37</td>
</tr>
<tr>
<td>Research Requirements</td>
<td>37</td>
</tr>
<tr>
<td>Contribution of Current Work</td>
<td>38</td>
</tr>
<tr>
<td>4 LOCAL GRID MAP APPROACH</td>
<td>39</td>
</tr>
<tr>
<td>Local Grid Map Update Methods</td>
<td>39</td>
</tr>
<tr>
<td>Nonhomogenous Markov Chain Approach</td>
<td>48</td>
</tr>
<tr>
<td>Simulation</td>
<td>51</td>
</tr>
</tbody>
</table>
5 LOCAL GRID MAP IMPLEMENTATION ........................................... 55
Local Grid Map Representation ................................................. 55
Sensor Representation ................................................................. 57
Fill Techniques ........................................................................ 59
Tunable Parameters ................................................................... 74
Numerical Example .................................................................... 76

6 JAUGS ...................................................................................... 79
Overview .................................................................................. 79
Problem Statement .................................................................... 81
JAUGS Purpose .......................................................................... 82
Domain Model ........................................................................... 84
Reference Architecture .............................................................. 85
Object Detection Component ....................................................... 88

7 RESULTS AND CONCLUSIONS .............................................. 94
Hardware/Software ..................................................................... 94
Results ...................................................................................... 99
Conclusions .............................................................................. 120

8 FUTURE WORK ........................................................................ 123
Mapping and Data Structures ..................................................... 123
Intersecting Polygons ................................................................ 129

REFERENCES ........................................................................... 136

BIOGRAPHICAL SKETCH ............................................................. 147
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Concept of local grid map and mapping polygons</td>
<td>1</td>
</tr>
<tr>
<td>1-2</td>
<td>Sensor’s physical characteristics</td>
<td>3</td>
</tr>
<tr>
<td>1-3</td>
<td>Area/sensor tessellation</td>
<td>4</td>
</tr>
<tr>
<td>1-4</td>
<td>Simulated local grid map</td>
<td>5</td>
</tr>
<tr>
<td>1-5</td>
<td>Simulated environment</td>
<td>8</td>
</tr>
<tr>
<td>1-6</td>
<td>Simulated sensor</td>
<td>8</td>
</tr>
<tr>
<td>1-7</td>
<td>Navigation Test Vehicle</td>
<td>9</td>
</tr>
<tr>
<td>1-8</td>
<td>Sonar’s geometric errors. (a) - Sonar reflection error. (b) - Sonar offset error. (c) - Sonar range error.</td>
<td>13</td>
</tr>
<tr>
<td>1-9</td>
<td>Actual terrain errors. (a) - Hill seen as an obstacle. (b) - Ground seen as an obstacle.</td>
<td>18</td>
</tr>
<tr>
<td>3-1</td>
<td>Object detection &amp; Object mapping component</td>
<td>37</td>
</tr>
<tr>
<td>4-1</td>
<td>Probability density function</td>
<td>40</td>
</tr>
<tr>
<td>5-1</td>
<td>Local grid map setup</td>
<td>56</td>
</tr>
<tr>
<td>5-2</td>
<td>Sensor’s physical properties</td>
<td>58</td>
</tr>
<tr>
<td>5-3</td>
<td>Bresenham line algorithm</td>
<td>60</td>
</tr>
<tr>
<td>5-4</td>
<td>Polygon and scan line</td>
<td>64</td>
</tr>
<tr>
<td>5-5</td>
<td>Sorted edge table</td>
<td>67</td>
</tr>
</tbody>
</table>
5-6 Active edge table for scan lines. (a) Scan line 9. (b) Scan line 10. ................................. 68
5-7 Polygon fill style. (a) - Polygon filled horizontally. (b) - Polygon filled vertically. ................................. 72
5-8 Sensor overlap ........................................ 75
6-1 Scope of the JAUGS document ......................... 81
6-2 JAUGS topology ...................................... 86
6-3 Object detection component without position component ................................. 89
6-4 Object detection component with position Component ................................. 89
7-1 Sonar and laser range scanner mounted on vehicle ................................. 94
7-2 Typical beam pattern at 50 KHz .................................. 96
7-3 Graphical representation of local grid map. (a) - Shown without position information. (b) - Shown with position information. ................................. 97
7-4 Typical startup of the local grid map .................................. 100
7-5 Two sets of trees without position component ................................. 102
7-6 Trapezoid arrangement of palm trees without position component ................................. 103
7-7 Wall with columns without position component ................................. 104
7-8 Two sets of trees with position component ................................. 106
7-9 Trapezoid arrangement of palm trees with position component ................................. 107
7-10 Wall with columns with position component ................................. 108
7-11 Distance between pillars while stationary ................................. 109
7-12 Distance between pillars while moving ................................. 110
7-13 Distance between two sets of trees ................................. 110

viii
7-14 Distance between 4 trapezoid palm trees .......................... 111
7-15 Realistic representation of environment. (a) Gate swung away from the vehicle. (b) Gate swung toward the vehicle. (c) Gate folded against the fence. .......................................................... 112
7-16 First tree occluding the second. (a) Trees as seen by vehicle. (b) Second occluded tree. (c) Detection sequence. .............................. 113
7-17 Contribution of sonar and laser using hedges and palm trees .... 115
7-18 Contribution of sonar and laser using shrubbery and larger tree ................................................................. 116
7-19 Effects of the tunable parameters on the local grid map ............ 118
7-20 Results of different cutoff percentages on the local grid map ....... 119
7-21 Hill seen as an obstacle ..................................................... 121
8-1 Collective structure .......................................................... 123
8-2 Combining bubbles and polygons ........................................ 131
8-3 Bentley-Ottmann sweepline algorithm .................................. 132
8-4 Intersection of two polygons. ............................................. 134
Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

IMPLEMENTATION OF A SENSOR FUSION-BASED OBJECT-DETECTION COMPONENT FOR AN AUTONOMOUS OUTDOOR VEHICLE

By

David Keith Novick

May 2002

Chairperson: Dr. Carl D. Crane III
Major Department: Mechanical Engineering

Nothing in life is certain. However the more information you have, and the better prepared you are, the more favorable the outcome. To this extent, for an autonomous vehicle to operate in the real world, it must have good information about its surroundings. For a vehicle to avoid obstacles, it must first be able to detect them. Generated data are only as good as the sensors that penetrate the vehicle’s environment.

This doctoral research is the extension into the real world of the work started in Takao Okui’s doctoral dissertation, Development of a Multi-Layered Map Management System Utilizing the Nonhomogeneous Markov Chain Approach. In his research, the area around the vehicle is tessellated into a local grid (the local grid map), and the range sensors are characterized by two methods. The first uses the physical properties to determine which grid cells should be updated. The second uses fuzzy modeling of the
sensor’s uncertainty to actually update the grid cell using a nonhomogeneous Markov chain.

Undertaking what Okui started in simulation, the techniques are implemented on a single board computer that runs on a Kawasaki Mule 500. Unlike a simulation, the constraints of the real world cannot be ignored. These include limits of storage, computational efficiency, maintainability, and the ease of adding additional, and diverse sensors.

Finally, this research provides the first implementation of the Object Detection Component (ODC) in the Department of Defense Joint Architecture for Unmanned Ground Vehicles (JAUGS). The goal of JAUGS is to present a unified architecture for the military’s unmanned ground vehicles. The architecture is currently a work-in-progress, and this research will help to develop the interface and the information passed between the obstacle detection component and the requesting subsystem.
CHAPTER 1
INTRODUCTION

For a robot to maneuver within an environment, many elements must come together. Sensors must be chosen that provide adequate information about the surroundings. In addition, the information must be presented in a timely manner as useable intelligence for navigation. Much of the current research in mapping outdoor environments consists of building up maps from successive range images. The only information about the terrain is what was obtained from the robot's sensors as it passed. This research uses a unique way of combining techniques for locating obstacles and free space local to the robot, and suggests ways of representing these same items for global map management.

This work is an implementation in the real world and extension of the work that Ph.D. graduate Takao Okui began in simulation. The method he developed involves using a local grid map that remains centered on the robot to represent either occupied or free space. As the robot moves, the values within the grid map shift relative to the robot.

Figure 1-1 Concept of local grid map and mapping polygons

When values reach the boundary of
the map, they are converted to polygons for more efficient storage in a global contour map, shown in Figure 1-1.

This paper introduces these two techniques in the Background section. The Simulation section describes the work Takao Okui has done developing these techniques within a simulated environment. The final section, Real World, describes how the simulated research will be applied and adapted to an actual robotic vehicle, and the various complexities that need to be addressed when moving to a genuine environment.

**Background**

Much research has addressed the problem of accurately representing an environment in which a robot traverses. One method uses a grid map to parcel the territory into smaller, more manageable cells. Each cell represents a small area within the specified boundary and contains a value to represent the degree of certainty that an obstacle is within that cell. As the robot travels around the environment, values are built up in each cell containing an obstacle until it reaches a threshold value.

While this method is adequate for a limited indoor scenario, it is impractical for an unrestricted outdoor environment, since the grid would require a large amount of memory for storage. In an outdoor environment, the mapping of obstacles has been broken into two steps. The first step uses a grid map (here, called a local grid map), that surrounds the robot. The second uses a polygon representation of obstacles and free space, called a global contour map, to reduce the amount of memory required.

**Local Grid Map**

The area directly surrounding the robot is tessellated into uniform square cells, similar to a checkerboard. This grid map remains local to the robot, and always maintains
a North-East orientation. As the robot travels around, the robot may assume any orientation within the center cell. However, if the robot crosses the boundary of the center cell, the entire grid map shifts to maintain a robot-centered local grid map.

**Fuzzy modeling**

A map should have information about obstacles and free space. Hence, the environment around a robot is characterized by two states: obstacle and free space. The construction of a suitable sensor model is one of the most crucial issues for increasing the accuracy of the map. Range sensors are first characterized by their physical characteristics, shown in Figure 1-2. This includes the minimum and maximum ranges, and the beam width expressed in angular form. Data from the sensor are either shown as a range that indicates the closest distance between the sensor and the obstacle, or the maximum range of the sensor indicating there are no detectable objects within its field. However, sensor signals have uncertainty such as randomness caused by noise and fuzziness caused by poor resolution. Because of these errors, fuzzy sets have been chosen to represent the data instead of representing the data as crisp events. This approach is more accurate because a slight degree of error in the degree of membership will be less consequential when using a fuzzy set than when a crisp event has an all-or-nothing representation. Two fuzzy sets are used, one indicating the empty area, EMP, and one indicating the occupied area, OBS.
The process of determining values for degree of membership is a subjective process and it is not always necessary to obtain exact values for the degree of membership. The fuzzy sets are constructed from statistical data obtained from a stationary robot. This approach is not optimal to find values for degree of membership since the data are collected using stationary sensors, while during actual operation of the robot, the sensors will be in motion. However, it is a good approach to estimate and produce values for range sensors because it matches the actual sensor data. Since it more closely parallels reality, this method is preferable over the use of a mathematical approximation of the geometry of the beam, which also might be more difficult to obtain. The values are defined as a frequency distribution by collecting imprecise data from the actual sensors.

The sensor field is tessellated as shown in Figure 1-3. Cells in the obstacle portion have a fuzzy set OBS and cells in the free space have a fuzzy set EMP. After producing a fuzzy distribution, a fuzzy set is projected onto the local grid map and is given to a cell as input. The cells are initialized with a value of 50%, representing an unknown condition. When the probability of an obstacle increases, the value in the cell rises, signifying the presence of an obstacle. When the probability of free space increases, the value in the cell falls, signifying the presence of free space (and the absence of an obstacle). This
continues until the probability reaches an upper or lower threshold value, and it is certain that the cell is either occupied or empty.

**Figure 1-4** shows the simulated local grid map. Peaks represent a high probability of an obstacle, while troughs represent a high probability of free space.

**Nonhomogeneous Markov chain**

Various methods for updating the local grid map were investigated by Okui [OKU99]. These included Probability Theory, Bayes's Theorem, Histogram Method, Fuzzy Logic, and Markov chain. They all have their merits, however the nonhomogeneous Markov chain proved to have a low sensitivity to noise, and good performance in the simulated indoor and outdoor environments; and the final probability is independent of the vehicle's approach to an obstacle.

In the previous section, fuzzy modeling was used to represent uncertainty such as randomness caused by noise and fuzziness caused by poor performance of sensors. Regarding the sequence of fuzzy sets as a time series, the nonhomogeneous Markov chain is applied to the stochastic process using a fuzzy model to reduce uncertainty. The fuzzy set EMP or OBS is presented to a cell as input, and a nonhomogeneous Markov chain is used to update the value of that particular cell in the local grid map.

**Global Contour Map**

As a robot moves around in the outside environment, a local grid map, that has a limited size, moves step-wise with the robot at a resolution of one cell in the grid map.
As the robot moves in one direction, the grid map moves in the opposite direction to maintain a robot-centered local grid map. Eventually, cells within the local grid map, that contain probabilities of empty and occupied space, reach the border of the local grid map. To store them in a global map, they are converted to the global coordinate system with a transformation from the grid model to a boundary representation that is a probability contour of the obstacle, and traversed space. Data that falls out of a local grid cell are considered a prime element, a bubble. When a robot returns to a prior location, the local grid map is initialized using the values in the global contour map that intersect the local grid map.

**Bubble**

A cell crossing out of the boundary of the local grid map is called a bubble and each vertex in a bubble is numbered in a counterclockwise fashion starting from an upper right vertex. A cell \((x, y)\) in the local grid map is transformed into a square polygon, a bubble, by calculating the vertices based on the position of the local grid map, and the size of the grid. Bubbles are merged together to form larger polygons.

**Polygon**

Once the bubbles are generated from the local grid map, each one is checked for its union with the current global contour map polygons. When a bubble is encapsulated by a polygon, it is eliminated. When it is outside the polygons, it is checked for its connectivity with the other polygons in the database. When a bubble has some connectivity with a polygon, it is fused with the polygon, creating a larger polygon. When a bubble has no connectivity with any other polygons, it is regarded as a new polygon and added to the database.
The global contour map is also used to initialize the local grid map for areas where the robot has already traversed, and polygons exist that identified obstacles and traversed space. The polygons are fused into the local grid map, initializing the cells with the values contained within the polygons.

Keep in mind that, although the generation of bubbles at the boundary of the local grid map and the fusion of polygons into the local grid map were discussed as two separate events, they are not. Both actions are happening concurrently. New polygons are being generated and fused with current ones while the polygons that are absorbed into the local grid map are being refined by new sensor values.

**Simulation**

Simulations are used when a task is expensive or dangerous to perform, or when an algorithm needs to be verified. Simulations have the advantage of being able to isolate the critical components necessary for testing and to focus on their interactions. There is also the advantage of replaying a scenario and altering just a few parameters. A simulation is the first step toward solving a real-world problem. It must first work in simulation before attempting a real-world solution. Okui used the limited simulation to help prove his theory.

**Position**

The terrain in the simulation is pre-generated. The roll and pitch for the robot is determined based on its position within the stimulation. The orientation and speed of the robot are known, so its position at the next time step can be calculated. Knowing the position, we can determine the tangent to the planar terrain segment, and hence the
orientation of the robot within the simulation. The position and orientation are considered accurate.

**Obstacles**

To the simulated sensors, the obstacles are unknown and various techniques are used to simulate the sensor, and then to detect and determine the range to the obstacles. Obstacles are not limited in number or size, but all are positive and stationary. While some resemble natural objects, they are all made from simple primitives that include boxes, cones, cylinders, and spheres (see Figure 1-5).

**Sensors**

General range sensors are generated using a picking action in OpenInventor. The picking action finds objects along a ray from the camera through a point on the near plane of the viewing volume. The ray is specified by the coordinates of a window-space pixel. In the simulation, rays are distributed uniformly from 0° to 180° in front, as show in Figure 1-6, and the number of rays can be changed. Uniform-distributed noise is added to the general range sensors and produces two kinds of errors. One is an error in distance caused by the
environment-obstacle interaction, and the other is a false detection caused by the sensor performance. The user can change the level of noise. The sampling times for each sensor can be specified. As seen in Figure 1-6, the sensors have a central origin, that also corresponds to the robot coordinate system.

**Real World**

The various techniques described earlier, and tested through simulation were adapted and implemented on an actual robot. Complications in the real world include inaccuracies in position and sensor data. Management of resources must also be considered. This research uses the base laid by Okui’s work along with adjusting other algorithms to tackle a real environment.

**Navigational Test Vehicle**

The Navigational Test Vehicle, NTV, is a Kawasaki Mule all-terrain vehicle, and is shown in Figure 1-7. The vehicle is completely computer-controlled, and has been used to evaluate code for path planning and field sweeping. It has also been used as a test bed to examine various sensors and techniques for sensor integration. While the simulation assumes accurate position and sensor information (aside from randomly generated noise), the real sensors carry various errors that contribute to inaccuracies in position, obstacle detection, and range determination.

*Figure 1-7  Navigation Test Vehicle*
Position

The NTV currently uses two different systems to determine its position on the Earth. The first is the Inertial Navigation System (INS). Sensing the accelerations of the vehicle, the unit is able to output the various pose parameters of the vehicle. The second is the Global Positioning System (GPS). By obtaining signals from at least four satellites, the unit is able to determine the robot's position and velocity. Each has its advantages, but inaccuracies are inherent in each system.

Inertial navigation system

The Inertial Navigation System (INS) used is the H-726 Modular Azimuth Positioning System (MAPS) created by Honeywell, Inc. The MAPS is a completely self-contained system. Given only an initial position, it uses three ring-laser gyros and three accelerometers to determine position, angular orientation, velocities and angular rate. Output includes azimuth (yaw), roll, pitch, azimuth rate, latitude, longitude, elevation, time stamp of the data, velocity in the north, east, and up directions, and a status message. These values are made available to the controlling computer at a rate of 10 Hz.

A problem encountered with the INS is that the position solution tends to drift over time from the actual position due to measurements and system errors. In order to reduce the amount of drift, the MAPS can make use of zero-velocity update (ZUPT) at a determined interval, typically every four minutes.

Global positioning system

The Global Positioning System, GPS, uses radio frequency signals from 24 orbiting NAVSTAR satellites to accurately determine position and velocity data. The
satellites' constellation comprises six equally spaced orbit planes orbiting the earth at approximately 10,900 nautical miles. The planes are tilted approximately 55° to the equator.

The GPS delivers absolute position to within 10 meters and velocity to within 0.1 m/s. While this level of accuracy would exclude it from use in absolute positioning, there are methods available to improve the results.

Differential GPS (DGPS) is a method of eliminating errors in a GPS receiver to make the output more accurate. This process is based on the principle that most of the errors seen by GPS receivers in a local area will be common errors. Two receivers are used, and one receiver is placed in a pre-surveyed location called the base station. The second receiver is placed on the vehicle, and is called a rover. The base station, using the known and accepted knowledge of its position, can determine the systematic or bias errors from the incoming signal. The corrections for the bias errors are then transmitted to the remote vehicle via a radio modem.

The Global Positioning System used consists of two Z-12 receivers marketed by Ashtech Inc. The Z-12 receiver is designed to make full use of the NAVSTAR Global Positioning System. The system uses carrier phase differential mode, providing centimeter level accuracy. It has twelve independent channels and can track all of the satellites in view automatically. Data from the Z-12 is made available to the controlling computer at a rate of 1 Hz. These data include a time stamp of the data, latitude, longitude, elevation, and number of satellites currently locked on.
Sensors

Various sensors can be used to perceive the robot's surroundings. Active sensors probe the scene with certain forms of energy to determine ranges. Sensors that acquire data without introducing any form of energy into the environment are considered passive. A popular active sensor, because of its low cost, is the ultrasonic transducer, or SONAR (SOund Navigation And Ranging). Structured light and a camera is another active sensor that can be used to determine shapes and ranges. A more costly active sensor is the LASER (Light Amplification by Stimulated Emission of Radiation) Range Scanner (LRS). For the passive sensor system, the camera is mainly used. One advantage of the camera over the active system is the time required to acquire the image. Using various filters and algorithms, detailed information about the scene can be extracted. With the camera, the first pixel is captured at the same moment in time as the last pixel in the image. For an active system, this is not the case. An active system requires time to calculate ranges. In the case of the sonar, a pulse is emitted, and the transducer must wait for the return pulse (if any) to calculate the range. The Laser Range Scanner generates a two dimensional spherical array of range points (pan, tilt, and distance). Through a series of mirrors, the scanning laser is physically positioned from point to point within the array. If the LRS isn't stationary during this process, inconsistencies between the beginning and ending scan points will exist.

Ultrasonic transducer

In an ultrasonic transducer, a short acoustic pulse is first emitted from a transducer. If a return echo is sensed, the range is determined by time-of-flight measurement. When using this sensor, there are three geometric properties of the
ultrasonic transducer that reduce its effectiveness in calculating an accurate range. The first is due to the "large" wavelength of ultrasonic energy compared to surface roughness of the object. At an angle at or greater than 30°, the energy is reflected off the object and away from the sensor, effectively rendering the object invisible to the sensor, see Figure 1-8(a). Another geometric effect is due to the beam width. The acoustic energy leaves the sensor in a cone of about 30°. Because of this, it is impossible to determine where, within the cone of energy, the object is. The object can be offset from the centerline of the sensor, but because of the beam width, it is almost impossible to determine the size and location of the object. Therefore the object is assumed to be along the centerline of
the sensor and there is a similar problem with the side lobes of the energy pattern, shown in Figure 1-8(b). The last geometric effect addresses the problem of an object slightly at an angle to the sensor. The closest range between the object and the sensor would be calculated instead of the centerline distance, as is shown in Figure 1-8(c).

**Planar-structured light**

If a plane of light is projected into a scene, range values can be determined. The 3D location of an illuminated point relative to the sensor reference frame determines the sensor 2D location of that point on the image plane of the camera used, because all illuminated points are in the plane of the projected light. The reverse of this transformation from 3D location in the scene to 2D location in the image provides the x,y,z location from the image coordinates of a point on a viewed stripe. To scan a scene, the light plane is swept through the volume of interest. Resolution is a function of camera optics and sensor geometry.

**Laser measurement system**

Similar to the ultrasonic transducer, the laser scanner emits a pulsed laser beam. If it strikes an object, the reflection is received by the scanner, and the time-of-flight measurement is calculated. The pulsed laser beam is deflected by an internal rotating mirror so that a fan-shaped scan is made of the surrounding area. Since the beam-width of a laser is much smaller than that of an ultrasonic transducer, the laser does not have the same geometric problems associated with the ultrasonic transducer. However, the energy still can be reflected by objects that have a high degree of smoothness. Because of the smaller beam-width, the laser is more susceptible to false echos due to smaller particles. A dust storm could register as an obstacle to the laser scanner.
A laser range scanner operates similar to conventional radar. Electromagnetic energy is emitted into the space to be observed and reflections are detected as return signals from the scene. The scene is scanned with a tightly focused beam of amplitude-modulated, infrared laser light. As the laser beam passes over the surface of objects in the scene, some light is scattered back to a detector that measures both the brightness measurements and the phase of the return signal. The brightness measurements are assembled into a 2D intensity image. The phase shift between the outgoing and return signals are used to produce a 3D range image. Geometry affects the ranging image of the laser range scanner, producing range "rings" on an otherwise featureless, flat wall. Aliasing is also a problem in a large scene, and is a function of the amplitude-modulation. If the range on the laser range scanner were set to 40 meters, and an object that was 41 meters away returned some of the energy, the amplitude of the returned energy would be identical to that of an object that was only 1 meter, and a distance of 1 meter would be calculated.

**Camera**

Since the human animal uses vision as a primary sense, it is natural to try to use a similar sensor for robots. As processor speed increases, the use of vision becomes a more viable option, reducing computation times from 15 minutes to seconds. A scene can be analyzed for shapes by performing edge detection, and for movement by calculating image flow. A disadvantage of passive sensors is the reliance of sufficient energy within the scene. There must be adequate lighting for a camera to be able to display useful images.
Shape from shading

Objects cast shadows when illuminated by a strong light source, because of their height above the ground. Therefore, a strategy for image analysis is to use the relationship between cast shadows and the associated man-made structures. The use of shadows to perform height determination in aerial imagery has a long tradition in photogrammetry and manual photo-interpretation. Given the sun angle, shadow length, and image scale parameters, a good estimation of building height can be generated.

Stereo vision

Stereopsis uses the disparity between images taken simultaneously by two cameras (or sequentially by one) to find the distance to points visible in both images, provided the geometry and optics of the sensor are well known. Corresponding points from the left- and right-eye view are found. The angular discrepancy between their locations is used in conjunction with the optical parameters, locations, and orientations of the cameras to calculate the position of that point relative to the camera's reference frame in 3D space. The disparity is inversely proportional to distance. Therefore, the distance of near objects can be measured fairly accurately, while that of far objects cannot. Disparity is directly proportional to the baseline distance. Thus, increasing the baseline increases the accuracy of depth determination. Unfortunately, as the baseline distance increases, the images become less similar.

Resources

Anytime resources are involved, they must be managed properly to be sure that they are available when required. In this application, the resources include the actual
amount of storage (hard disk drive and RAM), and the CPU time required to calculate the various equations involved.

**Limited storage**

The simulation ran on a Silicon Graphics Indy2 computer. This computer contains a 200 MHz processor with 128 Mbytes of memory. The simulation is able to take advantage of this Workstation's resources, using floating point representation of numbers for the local grid map, and not worrying about storage for the global contour map.

However, the CPU used for the module on the vehicle (the Detection Mapping System [DMS]), is entirely different, with a limited set of resources. The single-board computer contains a AMD-K6II-E-333MHz Socket 7 CPU with 256Mbytes of DIMM memory. Since this is a limited system, various techniques must be used to conserve valuable space. For instance, the percentages for the local grid map, that are stored as floats in the simulations, are stored as byte values.

**CPU time**

Unlike the simulation, in which time is a relative measurement, the actual system must perform its calculations in a timely fashion. In the simulation, the rest of the simulation is delayed while waiting for the results from the simulated sensors. Nothing else proceeds until the values from the sensors are returned. In the real system, all of the various procedures happen in parallel. While the sensors are gathering data, the robot remains in motion. Therefore, it is important to perform all of the calculations in an expedient fashion. This is accomplished by performing almost all of the calculations for
updating and maintaining the local grid map in integer arithmetic, and pre-multiplying and reducing all equations that are used repeatedly.

**Actual Terrain**

One of the areas neglected in the simulation was the inclusion of the actual terrain in determining the presence of obstacles. In the real world, as the vehicle approaches a change in terrain elevation, the ground can be detected as an obstacle. If not accounted for, the path could become impassable due to false ground detections. An example of this is shown when the robot is approaching a traversable hill, see **Figure 1-9**. As the robot is approaching the hill, it could be detected as an obstacle, and try to circumvent it. However, if the robot happens to ascend the hill, it could detect the ground as an obstacle while it's descending the hill. With the ground as an obstacle, the robot is effectively trapped on the hill.

While the simulation tried to incorporate a realistic sense, nature is not so uniform. Trees come in various shapes. Even within the same species, there are variations that include varying trunk girth and straightness, and assorted canopy types. Geometric figures with flat faces usually aren't found in nature either. Rocks and other commonly found objects contain many facets. These are all positive obstacles, or obstacles that protrude upward from the ground. Missing from the simulation are
negative obstacles, or ones that recede into the ground. These include ditches, ravines, and other sorts of chasms.

**JAUGS Message**

The Department of Defense is in the process of standardizing a software architecture for Unmanned Ground Vehicles (UGV). The Joint Architecture for Unmanned Ground Systems (JAUGS) purpose is to standardize the interfaces, protocols, and message formats in the software of the UGV family. The objectives of JAUGS are to allow interoperability, reduce life cycle costs, and to provide for easier technology insertion. The University of Florida is taking an active role in determining the specifications for the message interface. A final portion of this work is the determination of a messaging interface for the Object Detection Component (ODC). This system accepts JAUGS compliant messages, and return pertinent information. This information includes known or determined objects in the path of the robot for obstacle avoidance.
CHAPTER 2
LITERATURE REVIEW

The previous chapter discussed various sensors that could be used to detect objects in the environment. This chapter will present the various methods in which the sensors are used, and the algorithms developed to accomplish the tasks.

Ultrasonic Transducer

Elfes [ELF89] uses occupancy grids to represent the spatial information gained from ultrasonic sensors. The occupancy grid is a multidimensional random field that maintains stochastic estimates of the occupancy state of the cells in a spatial lattice. To construct a sensor-derived map of the robot's world, the cell state estimates are obtained by interpreting the incoming range readings using probabilistic sensor models. Probabilistic sensor models support the development of agile and robust sensor interpretation, handling uncertainty, multi-sensor composition, and spatial reasoning. The occupancy grid is used to map the robot's environment and aid in navigation.

By using a confidence-based map, Oskard, Hong and Shaffer [OSK90] were able to update the terrain map of a lake bottom. Confidence values are incremented or decremented from an initially-assigned base value as confirming or conflicting information is received. This information is integrated with an a-priori multi-resolution depth model stored in a quadtree format. The quadtree is an efficient method for storing 2D information by successfully decomposing non-uniform space into four quadrants: NE, NW, SW, and SE, until a quadrant contains only like values.
A system has been developed by Lim and Cho [LIM93] that uses sonar range data to build a map of the robot's surroundings. The mapping and navigation system is based on certainty grids. The range data from the sonar sensors are integrated into a probability map that is composed of two dimensional grids that contain the probabilities of being occupied by the objects in the environment. A Bayesian model is used to estimate the uncertainty of the sensor information and to update the existing probability map with new range data. Gourley and Trived [GOU94] used an occupancy grid to build a low level map and a detailed location of pipes to be decommissioned. A graphical interface is used to make the interaction with the system simple and user friendly.

Histogramic in-motion mapping (HIMM) is a method that Borenstein and Koren [BOR91] originated for real-time map building with a mobile robot in motion. The HIMM represents data in a two-dimensional array, called a histogram grid, that is updated through rapid in-motion sampling of onboard range sensors. Rapid in-motion sampling results in a map representation that is well-suited to modeling inaccurate and noisy range-sensor data, such as that produced by ultrasonic sensors, and requires minimal computational overhead. The HIMM uses a two-dimensional Cartesian histogram grid for obstacle representation. Like the certainty grid, each cell in the histogram grid holds a certainty value that represents the confidence of the algorithm in the existence of an obstacle at that location. However, the HIMM method increments only one cell in the histogram grid for each range reading.

Schneider, Wolf and Holzhausen [SCH94] used ultrasonic transducers onboard the vehicle to generate a world model using statistical methods and transforms the digital map into a topographical map suitable for display on a computer console.
Reconnaissance and Obstacle Surveillance in Unknown Environments (ROSE) also contains software to guide the vehicle autonomously through the obstacle field using a real time path planner on the basis of the Virtual Force Field (VFF) method. The operator can select this topographical map or a visual simulation of the environment. The visual simulation is displayed in real time on the console.

A system of one ultrasonic transmitter with 2 receivers is used to determine the normal of the surface. Ohya, Nagashima and Yuta [OHY94] uses this sensor to include information about the reflection characteristics of objects. A vector map is used to reconstruct the environment.

Fuzzy logic is used by Oriolo, Vendittelli and Ulivi [ORI95] to solve the fundamental processes of perception and navigation. The robot collects data from its sensors, builds local maps and integrates them with the global maps so far reconstructed, using fuzzy logic operators. The inputs to the map building phase are the range measurements coming from the ultrasonic sensors. Its outputs are two updated fuzzy maps. Both convey information about the risk of collision at each point of the environment.

A Triangulation technique is presented by Wijk, Jensfelt and Christensen [WIJ98] which is used for filtering data to obtain an improved grid map of the environment. The technique relies on triangulation of sonar readings recorded from different positions for estimation of the location of structures in the environment. The intersection between the arcs defined by the readings specify the location of the common target. Multiple sensor readings are used to define the location of a sensed target. When a new hypothesis is generated by the algorithm the corresponding occupancy grid cell is set to a measure of
belief that the cell is occupied. Cells on the line from the target to the sensor position are set to zero.

Yi, Khing, Seng, and Wei [YI00] used the Dempster-Shafer evidence in sensor fusion with a specially designed sensor model to reduce uncertainties in ultrasonic sensor. The conflict value in Dempster-Shafer evidence theory is used to modify the sensor model dynamically.

**Planar-structured Light**

Using structured light to test an algorithm, Asada [ASA90a] defines a method for building a three-dimensional world model for a mobile robot from sensory data derived from outdoor scenes. The obstacles are classified into artificial objects or natural objects according to their geometrical properties such as slope and curvature. The local maps generated by the sensor are integrated into the larger global map.

Little and Wilson [LIT96] developed usable methods to rapidly build world models of real world workspaces. To accomplish this, 3D range sensors are deployed to capture surface data within the workspace. This data is then transformed into surface maps, or models. The primary hardware additions are video cameras and structured lighting. Selected range or distance measurements are used in updating and registering existing CAD models. Where traditional CAD modeling is not appropriate, dense range data is collected over an area and transformed directly into surface maps, or models. As additional areas are scanned and modeled, they are added together to form a 3D world model of the target workspace. Filtering is used to reduce data count. Delaunay triangulation is used to connect the points, and then a decimation method is used to
maintain areas of high detail, while allowing low detail surfaces, such as flat areas between edges, to be reduced.

**Laser**

Gowdy and Stentz [GOW90] builds a Cartesian Elevation Map (CEM) from a series of Environmental Research Institute of Michigan (ERIM) laser range scanner images. The elevation map is quantified in a two dimensional array with a resolution of 10 cm per element, the contents of each element being the height at that point. Data far from the sensor is sparse and is filled with an interpolation method. Various scanner images are fused by using the x and y position of the inertial navigation system, and comparing the average elevation values over a patch of terrain from each of the elevation maps that correspond to the same path of terrain in the real world. A terrain pyramid is then built to represent the terrain in a more efficient way. In the terrain pyramid, the bottom layer is simply the elevation map. The next layer up is the maximum and minimum elevation values over a group of four elements. This step is repeated for each level in the terrain pyramid until the top is reached. The planner requests the range of a feature over a bounding box area. The bounding box is evaluated, the appropriate level of the terrain pyramid is accessed, and the maximum and minimum are returned.

Color video and a laser range scanner was used by Olin and Hughes [OLI91] in developing planning software that used digital maps to plan a preferred route, and then as the vehicle traversed that route obtain scene descriptions by classifying terrain and obstacles. Techniques were developed to detect and avoid obstacles in cross-country terrain, with minimum delay between sensing and acting. The scanned image is converted
to a Cartesian elevation map, and subsequent images are fused using a rigid-body motion
recovery algorithm.

Nashashibi, Devy and Fillatreau [NAS92] were able to build a rough geometric
model of a 3D indoor terrain using a laser range scanner. The terrain model relies on two
grid-based representations: the Local Elevation Map and the Local Navigation Map. The
Local Elevation Map is built from the direct fusion of the laser range scanner data. The
Local Navigation Map is a symbolic representation of the terrain built from the Local
Elevation Map. The terrain is partitioned into homogeneous regions corresponding to
different classes of terrain difficulties, according to the robot locomotion capabilities.
The two main elements of the classification are the average slope of the terrain, and the
local slope and height discontinuities.

Multiple sensors are used by Kweon and Kanade [KWE92] for incrementally
building an accurate 3D representation of rugged terrain. The locus method is used to
model the rugged terrain. The locus method exploits sensor geometry to efficiently build
a terrain representation from multiple sensor data. Cartesian elevation maps are used to
represent the data obtained from an ERIM laser scanner. The locus method uses a model
of the sensor and works in the image space.

Rugged natural shapes are represented, analyzed, and modeled using fractal
graphology by Arakawa and Krotkov [ARA93]. Fractal Brownian functions are used to
estimate the fractal dimensions. The method is applied to a Perceptron scanning laser
rangepfinder, and is extended to accommodate irregularly sampled data.

Klein [KLE93] builds a Cartesian elevation map from a ERIM laser range scanner
images on the Navlab II traveling at speeds from 4.02 MPH (1.8 m/s) while avoiding
obstacles to 10.07 MPH (4.5 m/s) while checking for obstacles. Motion distortion is removed by tagging each image with a series of vehicle poses sampled at a regular time interval during digitization. Interpolation is used to fill in the unknown data regions. Range shadows are eliminated by computing the ray in space emanating from the range image pixel. The range value associated with each ray is direct evidence that the terrain is no higher then the ray is until the ray intersects the terrain. The CEM is fused by matching the vehicle's pitch, roll and altitude.

Quantitative models are constructed by Krotkov and Hoffman [KRO94] of surface geometry from Ambler, a walking robot. The accuracy of the constructed maps enables safe, power-efficient locomotion over the natural, rugged terrain found on planetary surfaces. The mapping system acquires range images with a laser rangefinder and constructs Cartesian elevation maps from them at arbitrary resolutions, in arbitrary reference frames. Because of the locus algorithm that uses a model of the sensor to interpolate at arbitrary resolution without making any assumptions on the terrain shape other than the continuity of the surface, the mapping system is one that can handle extremely rugged terrain and exhibits a real-world robustness because of its aggressive detection of imaged-based errors and in its compensation for time-varying errors.

Stuck, Manz, Green and Elgazzar [STU94] describes a navigation system that unites three important capabilities. It enables a mobile robot to avoid obstacles, map the environment, and plan local paths around more complex obstacles while navigating. This system extends and integrates the histogram grid representation and the modified A* algorithm from a BIRIS laser range scanner. The navigation system also creates and updates a histogram grid map for path planning.
Three navigation modes are used by Lacroix, Chatila, Fleury, Herrb and Simon [LAC94] to perform cross-country autonomous navigation. The navigation mode is adapted to the nature of the terrain: A 2D planned navigation mode when the terrain is mostly flat; a 3D planned navigation mode when an uneven area has to be crossed; and a reflex navigation mode. It is believed that aiming at building a "universal" terrain model that contains all the necessary information is extremely difficult and not efficient, and therefore chose to endow the robot with a multi-layered heterogeneous terrain modeling capacity.

Castellanos and Tardós [CAS96] presents a technique to segment the data obtained by a laser range finder mounted on a mobile robot navigating in a structured indoor environment. Using this segmentation and an a priori map of the environment the localization of the mobile robot is found by application of a constraint-based matching scheme. Feature-based methods are of interest, in which a set of features are extracted from the sensed data and then matched against the corresponding features in the model. The mobile robot localization is calculated by matching observed segments, obtained by a laser rangefinder, and model segments, stored in a database, representing the structure of the environment.

Hancock, Hebert and Thorpe [HAN98], present a method for obstacle detection for automated highway environments. They show that laser intensity, on its own, is sufficient (and better) for detecting obstacle at long ranges. At the long look ahead distances and grazing angles typical of high-speed travel, horizontal surfaces should provide very weak (or nonexistent) laser returns. Vertical surfaces (obstacles), however, should result in stronger signals.
A time-of-flight (TOF) laser ranging systems is used by Mázl and Přeučil [MÁZ00] is combined with vehicle odometry to generate a 2D polygonal approximation of an indoor environment. The obtained range measurements are in the form of isolated points, and the processing requires input data segmentation into point sets belonging to particular boundaries based on a set of heuristic rules. The new segments discovered in the sensor data are merged with the existing world map.

Plaza, Prieto, Dávila, Población and Luna [PLA01] describe an implementation of an obstacle detector system based on a laser pointer and a phototransistor receptor. The idea consists of receiving the reflected power of an emitted laser beam from an obstacle and estimating the distance based on the received power.

**Camera**

A method for representing a global map consisting of local map representations and relations between them has been developed by Asada, Fukui and Tsuji [ASA90b]. Sensor maps viewed at locations close to each other are integrated into a local map representation in a Cartesian coordinate system fixed to an object. First 3D information of the edges on the floor is obtained at each sensor map by assuming the camera model and the flatness of the floor. A reliable feature is selected as a reference in a sensor map, which has important roles in finding the correspondence between the current sensor map and the following ones and in building a local map with these sensor maps. Finally, the relation between local maps that represents the relative orientation and the approximate distance from the previous local map to the current one is included in the global map.

Hoover and Olsen [HOO00] describe a novel mobile robot system that uses an environment-based sensor network of camera providing a powerful third-person
perspective on the environment. The idea is that mobile robots working in the area tune in to broadcasts from the video camera network (or from environment-based computers processing the video frames) to receive sensor data. The occupancy map is a two-dimensional raster image, uniformly distributed in the floor-plane. Each map pixel contains a binary value, signifying whether the designated floorspace is empty or occupied. A spatial frame of the occupancy map is computed from a set of intensity images, one per camera, captured simultaneously. During calibration a background image is acquired for each camera, in which all the relevant floorspace is cleared. A binary mask is created for each background image, denoting which pixels are floorspace, and the algorithm detects the differences between the background image and the live image for each camera.

**Shape from Shading**

McKewon [MCK90] has developed a method of extraction of buildings from monocular views, the detection and delineation of road networks, scene registration and matching for stereo analysis, and knowledge-based scene interpretation. Building information is extracted in two ways. One method, Built-up Area Building Extraction, BABE, analyzes intensity edges, estimates the shadow intensity and illumination direction, and produces a set of building hypotheses. Another technique uses an estimate of shadow intensity produced by BABE to find shadow regions and to hypothesize the buildings that cause those shadows. Road extraction is performed in three steps. First the whole area of interest is processed, and sequences of possible road points are found. Road finding algorithms generate road seed hypotheses that are used as initial starting points for road tracking. Road tracking takes the center lines defined by the road seeds
and attempts to extend the road by tracking the road surface and road boundary. The result of road tracking using multiple starting points results in a set of completely and partially tracked roads. These road hypotheses are examined for overlaps and intersections, and a graph that represents the overall network structure is constructed.

Shao, Chellappa and Simehony [SHA91] suggests several algorithms for recovering depth and orientation maps of a surface from its image intensities. They combine the advantages of stereo vision and shape-from-shading methods (SFS). The SFS algorithm for a single image is extended to use stereo images. The correspondence over stereo images is established simultaneously with the generation of surface depth and orientation. Finally an algorithm is presented to combine sparse depth measurements with an orientation map to reconstruct a surface.

**Stereo Vision**

Waxman, LeMoigne, Davis, Srinivason, Kusher, Liang and Siddaligaiah [WAX87] developed a system that consists of vision modules performing image processing, three-dimensional shape recovery, and geometric reasoning, as well as modules for planning, navigating, and piloting. The system has been implemented as a set of concurrent communicating modules and used to drive a camera over a scale model road network on a terrain board. Navigation is broken into three spatial scales, long-, intermediate-, and short-range navigation. Long-range navigation concerns the decomposition of the environment into regions that share common properties such as uniform visibility of landmarks and navigability of the terrain. Intermediate-range navigation is invoked to select a corridor of free space through which the vehicle will
next travel. Short-range navigation is responsible for selecting the actual path to be traversed through the established corridor of free space.

Using outdoor stereo image sequences, Leung and Huang [LEU91] developed an integrated system for 3D motion estimation and object recognition. The scene contains stationary background with a moving vehicle. The goals are to obtain the 3D motion description and the identification of the vehicle. The wheels of the vehicle are extracted, and from that information, the vehicle is identified.

Adding another camera, Zhang and Fangeras [ZHA92] uses this trinocular stereo system to build a local map about the environment. A global map is obtained by integrating a sequence of stereo frames taken when the robot navigates in the environment. A Kalman filter is used to merge matched line segments. An important characteristic of the integration strategy is that a segment observed by the stereo system corresponds only to part of the segment in space, so the union of the different observations gives a better estimate on the segment in space.

Zhu, Xu, Chen and Lin [ZHU94] describes a new framework for detection of dynamic obstacles in the unstructured outdoor road environment by purposely integrating binocular color image sequence. Hue, saturation, and intensity are used to separate the pavement of the road from neighboring areas, and objects on the pavement. Image sequences are used to determine the motion of a dynamic object.

Krotkov and Herbert [KRO95] demonstrates practical, effective approaches to outdoor mapping and positioning, and presents results from systems implemented for a prototype lunar rover. A binocular head provides images to a normalized correlation matcher, that intelligently selects what part of the image to process, and subsamples the
images without subsampling disparities. The mapping software takes as input a stereo pair and outputs arrays of the three coordinates X, Y, and Z.

A real-time approach to obstacle detection is presented by Li and Brady [LI98] for an active stereo vision system based on plane transformations. If the transformation matrix for the ground plane could be computed in real time, it can be used to check if the corresponding points are from the ground plane and hence are not from obstacles. They derived a model for ground plane transformation and developed an approach to identify the related system static parameters. The ground plane transformation is computed analytically, is very fast, and hence can be used in real-time systems.

Haddad, Khatib, Lacroix and Chatila [HAD98] present an approach that relies on a particular probabilistic obstacle detection procedure that describes the area perceived by a pair of stereo cameras as a set of polygonal cells. Attributes are computed for each cell, and are used to label the cells in terms of navigation classes, thanks to a supervised Bayesian classifier. On a perfectly flat ground corresponding to the reference plane, the number of points that belong to a cell - i.e. whose vertical projection coordinates are bounded by the cell’s border - is equal to a constant nominal density. On the other hand, a cell covering an obstacle area contains much more points than the normal density. A supervised Bayesian classification procedure is used to label each cell as either flat or obstacle.

**Sensor Fusion**

McKerrow [MCK95] discusses the application of a four-level data fusion software architecture to ultrasonic mapping with a mobile robot. Perception is defined as a four step process: detection, recognition, discrimination and response. The first level is the
physical transducer and electronics that sense and amplify the echo. The second level includes signal processing and algorithms for extracting acoustic features from the range values derived from the echos. The third level includes techniques to transform acoustic features into spacial features. The forth, and last, level aims to obtain a symbolic description of the environment and to reason about it.

Jörg [JÖR95] uses heterogenous information provided by a laser-radar and a set of 24 sonar sensors to achieve reliable and complete indoor world models for both real-time collision avoidance and local path planning. He does this by creating a Radar Map (RM) from the laser-radar and incrementally building up a grid-based representation of the environment called the Accumulative Grid Map (AGM) from the sonar. The AGM is a local map which is defined relative to the robot. New range information it integrated into the AGM using an asscoiated weight and frequency measure. The weight expresses the degree of belief that a cell is actually occupied by an obstacle. The frequency measure counts the number of update cycles that have passed since the cell's weight has been incremented last.

Akbarally and Kleeman [AKB96] present a novel method of combining sonar and visual data to create a 3D sensing combination that models structured indoor environments. The sensor combination is intended for autonomous mobile robots operating indoors on problems such as localization and map building. Data from a CCD camera represents a 2D projection of 3D features in the environment. Conversely, a sonar sensor is intrinsically a detector of range. A sonar sensor has been developed to localize and classify targets into 16 different target types, and the visual data is obtained using a grayscale CCD camera and is processed using a Hough transform to extract a set of
equations of all lines that occur in the image. At first, the processed sonar data is used to create an initial map, then the corresponding visual parts are introduced, and finally the environmental map is expanded by iteratively combining the remaining visual data.

Langer and Jockem [LAN96] describe an integrated radar and vision sensor system for autonomous on-road navigation. The radar sensor is integrated with a vision based lane keeping system to accurately detect and classify obstacles with respect to the danger they pose to the vehicle. Range and angular information of targets from the radar are obtained by Fast Fourier transform. Detected targets are kept in an object list which is updated by successive data frames from the radar sensor. Target information is fused with road geometry to assess if the object is in the vehicle’s lane.

Visual data obtained by a binocular active vision system is integrated with 24 ultrasonic range sensors by Silva, Menezes, and Dias [SIL97] in the development of an obstacle detection and avoidance system based on a connectionist grid. Sensor integration is accomplished by a cellular automata. This can be pictured as a cellular lattice, where each cell is connected to its nearest neighbors. The cells’ values depend on the information about the environment provided by the sensing devices along with its neighbors values.

Two little used components of the Dempster-Shafer theory are exploited by Murphy [MUR98] for the purpose of sensor fusion. The weight of conflict metric is used to measure the amount of consensus between different sensors. Enlarging the frame of decomposition allows a modular decomposition of evidence. Dempster-Shafer theory is used for sensor fusion at the symbol level in the Sensor Fusion Effects (SFX) architecture. Sensor fusion in SFX consists of three distinct activities: configuration,
uncertainty management, and exception handling. The configuration activity is concerned with using the task goals of a robot to generate expectations of precepts and to predict what features of the percept will be observable to which sensors and the significance of their contribution. The uncertainty activity collects observations and computes the total belief in the percept using a Dempster-Shafer framework. The percept is then used by the motor behavior to control the robot. Missing or abnormal observations trigger exception handling.

Information from an omni-directional sonar and omni-directional vision was fused by Yata, Ohya, and Yuta [YAT00] for indoor environment recognition. The new omni-directional sonar provides accurate reflecting points that include distance and angle, but those points are sparse. The omni-directional vision provides direction of edges, but they do not include distance information. By fusing those data, environmental features are extracted.
CHAPTER 3
RESEARCH GOALS

Statement of Problem

Given the following:

• A vehicle capable of movement.

• The position and orientation of the vehicle measured with respect to a global coordinate system.

• Various sensors used in the detection of objects, which can include sensors for positive objects, such as tree and rocks, and sensors for negative objects such as holes and ditches.

• Outdoor terrain.

Develop:

• A system capable of detecting objects and maintaining their positions relative to the vehicle for the task of avoidance.

• This system consists of various non-homogeneous sensors and a local grid map. The sensors are chosen for their ability to detect distinct characteristics in the outdoor environment. They can be active, by projecting energy out into space, and waiting for the return, or they can be passive and measure the energy in space. Through fuzzy modeling, the sensors are homogenized and projected onto the local grid map. With the knowledge of position of the vehicle, the objects can be maintained within the local grid map once they exit from the sensor’s field of view. This allows the vehicle to avoid currently detected objects, and ones that have been previously detected.

• This research will implement the fusion of non-homogeneous sensors into a local grid map on an actual robotic vehicle. The code will run on a single-board computer and will be self-contained. It will be validated by running the vehicle outside.
Assumptions

- There is enough memory on the single-board computer to implement the local grid map and to run the various threads used to interface to the sensors and update the local grid map, and the computer is fast enough to handle all tasks in 'real-time'.

- The sensors can be characterized by specifying a minimum and maximum range and a beam angle width.

Research Requirements

The conceptualized structure is shown in Figure 3-1. The goal of this research is to develop the upper half of the structure, the Object Detection Component. Fully realized, the Detection Component takes input from two sources. The primary source is from the multiple sensors located around the vehicle used to detect unknown objects within the vehicle’s path. The second source is input from the position component that keeps track of the vehicle’s location. This is used to maintain the position and orientation of obstacles once they are outside the sensors field of view.

The detection component takes the data from the different sensors and homogenizes it using the physical characteristics of the sensor plus the fuzzy modeling of

![Figure 3-1: Object detection & Object mapping component]
the sensors performance. This sequence of data is used in local map management by updating values in each grid cell of the local grid map. Uncertainty is reduced, and objects and free space spring from the collection of data.

**Contribution of Current Work**

The main contribution of this research is the implementation and extension into the real world of Takao Okui’s research started in simulation. Unlike a simulation, the complexities of a real environment cannot be ignored. Therefore extensive research was focused on finding efficient algorithms that could be adapted for use in converting the theory into reality. The Object Detection Component, that uses the local grid map derived from Okui’s research, includes the following:

- A small and resourceful storage mechanism for each grid cell in the local grid map.
- Efficient fill algorithms that use integer arithmetic to update the local grid map in the most timely manner.
- An easy method to add new and different sensors to the system.
- A JAUGS interface to query object information from the local grid map.

The main benefactor of this research would be anyone who wishes to provide a vehicle with the capability of object detection. With an easy interface, various types of sensors can be examined to determine which complement each other. Object detection can be used by autonomous vehicles to discover and avoid unknown obstacles. For teleoperation, be it local or located on another planet, the object detection component can present obstacles to the operator to determine the best course of action. In the commercial realm, object detection can be used for vehicle navigation and intelligent highway systems.
CHAPTER 4
LOCAL GRID MAP APPROACH

In his work, Okui [OKU99] investigated various methods for use in updating the Local Grid Map. The techniques evaluated included Probability Theory, Bayes's Theorem, The Histogram Method, Fuzzy Logic, and finally Nonhomogenous Makrov Chains. This chapter will present the background and the results of Okui's simulation.

Local Grid Map Update Methods

Probability Theory

The poor performance of a sensor is regarded as randomness in certainty. A probability density function can be used to model a sensor beam by the following definition:

Definition:

Suppose that an experiment is associated with a sample space \( S \). To every event \( A \) in \( S \) (\( A \) is subset of \( S \)) we assign a number, \( P(A) \), called the probability of \( A \), so that the following axioms hold:

Axiom 1: \( P(A) \geq 0.0 \)

Axiom 2: \( P(S) = 1.0 \)

Axiom 3: If \( A_1, A_2, A_3 \ldots \) for a sequence of pair-wise mutually exclusive events in \( S \), then \( P(A_1 \cup A_2 \cup A_3 \cup \ldots) = \sum P(A_i) \)

A range reading is interpreted as providing information about space volumes that are either empty or occupied. Two regions in the sensor beam are defined in a sensor
coordinate system based on geometry of the beam and the spatial sensitivity pattern.

These two regions represent the empty probability density function for a point $P$ inside the sensor beam and the corresponding occupied probability density function.

After producing the probability density function for every sensor, shown in **Figure 4-1**, they are projected onto the cell of a grid map defined in a global coordinate system. For a map update, probability theory was used regarding uncertainty of a sensor as randomness. Each cell has two random variables, the probability of free space, $P_{\text{free}}(X,Y)$, and the probability of obstacles $P_{\text{obstacle}}(X,Y)$, where $X,Y$ are the position in a global coordinate system. $P_{\text{free}}(X,Y)$ and $P_{\text{obstacle}}(X,Y)$ are updated by an additive law of probability using both $P_{\text{free}}(X,Y)$ and $P_{\text{obstacle}}(X,Y)$ from all sensors.

Utilizing the additive law, it was assumed that probabilities in the previous map and probabilities from sensors are independent from each other, $P(A|B) = P(A)$, and the sensor probabilities are not modified by the information in the previous map. The evidence combination proceeds along the following steps:

- **Reset**: The whole map is set to unknown by making $P_{\text{free}}(X,Y):=0$ and $P_{\text{obstacle}}(X,Y):=0$
- **Superposition of empty space**: For every sensor reading, modify the emptiness information over its projection by
  
  $P_{\text{free}}(X,Y) := P_{\text{free}}(X,Y) + P_{\text{free}}(X,Y) - P_{\text{free}}(X,Y) \cdot P_{\text{free}}(X,Y)$
• Superposition of occupied area: For every reading, shift the occupied probabilities around in response to the combined emptiness map using:

\[
P_O(X,Y) := P_O(X,Y) \cdot (1 - P_{Emp}(X,Y))
\]

\[
P_O(X,Y) := P_O(X,Y) / \sum P_O(X,Y)
\]

\[
P_{O_{ce}}(X,Y) := P_{O_{ce}}(X,Y) + P_O(X,Y) - P_{O_{ce}}(X,Y) \cdot P_O(X,Y)
\]

• Thresholding: The final occupation value attributed to a cell is given by a thresholding:

\[
\begin{cases} 
P_{O_{ce}}(X,Y) \text{ if } P_{O_{ce}}(X,Y) \geq P_{Emp}(X,Y) \\
-P_{Emp}(X,Y) \text{ if } P_{O_{ce}}(X,Y) < P_{Emp}(X,Y)
\end{cases}
\]

**Bayes's Theorem**

Instead of using two random variables to update a map, one random variable was used to express the probability of occupancy for each cell. A value close to 0.0 indicates low probability of occupancy, which is a high probability of emptiness. The value 0.5 is used as unknown and a value close to 1.0 indicates high probability of occupancy.

To use Bayes’s theorem for a map update, it was assumed there are two states in an environment; obstacle, \( \{A_o\} \), and free space, \( \{A_f\} \). A probabilistic sensor model was regarded as a conditional probability density function \( P(\text{sensor reading } R \mid \text{state of environment, } \{A_o\} \text{ or } \{A_f\}) \). For additional simplification, it was assumed that \( P(R\mid \{A_f\}) = 1.0 - P(R\mid \{A_o\}) \). Let \( S = \{A_f\} \cup \{A_o\} \) and \( \{A_f\} \cap \{A_o\} = \Phi \), from Bayes’s theorem:

\[
P(\{A_o\} \mid R) = \frac{P(\{A_o\}) \cdot P(R\mid \{A_o\})}{P(\{A_o\}) \cdot P(R\mid \{A_o\}) + P(\{A_f\}) \cdot P(R\mid \{A_f\})}
\]  \hspace{1cm} (4-2)
The initial value of $P(\{A_0\})$ is 0.5 (unknown) and $P(\{A_o\})$ is assigned to the previous value of $P(R|\{A_0\})$. $P(R|\{A_o\})$ is given by the probabilistic sensor model. Using the complement and the simplification, the equation is rewritten as

$$P_{\text{occ}}(X, Y) = \frac{P_{\text{occ}}(X, Y) \cdot P(X, Y)}{P_{\text{occ}}(X, Y) \cdot P(X, Y) + (1 - P_{\text{occ}}(X, Y))(1 - P(X, Y))} \quad (4-3)$$

**Histogram Method**

To avoid real number calculations for real-time execution by an on-board computer, the denominator was separated from the probability density function by regarding it as a maximum certainty value. By changing the value, the probability density function also changes. Therefore, it is possible to tune the sensor model according to the performance of the range sensors and the condition of the environment. The relationship between the probability density function and the histogramic probability distribution is as follows:

$$\begin{cases} P_\text{O}(X, Y) = \text{(Increment step value) / (Maximum certainty value)} \\ P_\text{E}(X, Y) = \text{(Decrement step value) / (Maximum certainty value)} \end{cases} \quad (4-4)$$

At a range, $R$, all cells preceding $R$ are considered empty, and decremented by $P_\text{O}(X, Y)$. At $R$, the cell is occupied and is incremented by $P_\text{O}(X, Y)$. Two variables, $P_{\text{occ}}(X, Y)$ for occupancy and $P_{\text{imp}}(X, Y)$ for emptiness, are used to update the map. It was assumed that two pervious cell values, $P_{\text{occ}}(X, Y)$ and $P_{\text{imp}}(X, Y)$, and the two step values from the sensor, $P_\text{O}(X, Y)$ and $P_\text{E}(X, Y)$, are not only independent but also mutually exclusive. Therefore, utilizing the additive law of probability, the update formula is:
\[
\begin{align*}
\begin{aligned}
P_{\text{OCC}}(X, Y) &= P_{\text{OCC}}(X, Y) + P_{\gamma}(X, Y) \\
P_{\text{Emp}}(X, Y) &= P_{\text{Emp}}(X, Y) + P_{\text{\varepsilon}}(X, Y)
\end{aligned}
\end{align*}
\]

(4-5)

Where the final occupation value attributed to a cell is given by:

\[
P_{\text{OCC}}(X,Y) - P_{\text{emp}}(X,Y).
\]

**Fuzzy Logic**

For the previous methods, it was assumed that uncertainty of range data is random and sensor modeling and map update procedures are based on probability theory. However, the uncertainty of range data can also be attributed to vagueness of the boundary of the sensing field and fuzzy logic can be used. In sensor modeling, a membership function is used instead of a probability density distribution.

Membership functions are projected to the grid map using the position and yaw angle of the vehicle and the direction of the main axis of sensors. To update the map, a degree of certainty is increased non-linearly using a fuzzy union operation similar to the increment in the histogram method. The pieces of information concerning the empty and the occupied cells are separately collected during the measuring process. The associative property of fuzzy union allows using two fuzzy sets \(\mu_{\text{Emp}}(X,Y)\) and \(\mu_{\text{OCC}}(X,Y)\) as accumulators. For the membership function for emptiness, \(\mu_{\text{Emp}}(X,Y) = \cup \mu_{\text{E}}(X,Y)\) and for the membership function for occupancy, \(\mu_{\text{OCC}}(X,Y) = \cup \mu_{\text{O}}(X,Y)\). For the accumulation of

\[
u_{\lambda}(\mu_{\text{A}}(x), \mu_{\text{B}}(x)) = \frac{1}{1 + \left[ \left( \frac{1}{\mu_{\text{A}}(x)} - 1 \right)^{-\lambda} + \left( \frac{1}{\mu_{\text{B}}(x)} - 1 \right)^{-\lambda} \right]^{-\lambda}}
\]

(4-6)
the degree of certainty, Dombi’s union operation, whose strength can be varied by tuning a parameter $\lambda$, was used.

The chosen union family reaches certainty asymptotically and the operator produces larger union sets as $\lambda$ is decreased; that is, weaker unions are obtained for small value of $\lambda$.

**Nonhomogenous Markov Chain**

The final method for updating the Local Grid Map, and the one chosen to implement was the Nonhomogenous Markov Chain. Unlike the other methods that used a form of a probability density function, or membership functions in the case of Fuzzy Logic, a new method of modeling the sensors had to be developed. This new method uses a fuzzy approach to describe uncertainty in range sensors.

The construction of a suitable sensor model is one of the most crucial issues for overall performance to reduce uncertainty. A typical sensor model has a distribution of higher probabilities near the center line axis of the sensor and relatively low probabilities on either side. It describes the uncertainty caused by the poor performance of sensors, but range data are also influenced by noise and sensor-environment interaction. Some models are modified by adding adjustable parameters like the maximum certainty value in the histogram method approach and weights in the fuzzy logic approach to produce a suitable model. However, it is very difficult to justify a particular probability profile considering the sensor has to work in varied, unknown environments.

**Fuzzy modeling**

A sensor can either return no data or a range that indicates the closest distance between a vehicle and obstacles within the sensing field. The value of range data, $R$,
exists between minimum range, $R_{\text{min}}$ and maximum range, $R_{\text{max}}$. Information regarding
free space in range data is either the entire sensing field when no data is returned or the
area between the obstacle and the sensor when there is a range value (since beyond that
value is unknown). Sensor signals have uncertainty such as randomness caused by noise
and ambiguity caused by poor resolution.

To describe uncertainty in range data, three prime elements are defined as a frame
of discernment $\Theta$.

$$\Theta = \{ A_O, A_F, A_N \}$$

where

$A_O$ : Obstacle in an environment

$A_F$ : Free Space in an environment

$A_N$ : Noise in a sensor

Taking the power set of $\Theta$, the following set of all subsets of $\Theta$ is obtained.

$$X = \{ \emptyset, \{ A_O \}, \{ A_F \}, \{ A_N \}, \{ A_O, A_F \}, \{ A_O, A_N \}, \{ A_F, A_N \}, \{ A_O, A_F, A_N \} \}$$

removing the contradictions of $\emptyset$ (emptyset), $\{ A_{\text{Ni}} \}$ (no environment and noise in a
sensor), $\{ A_O, A_F \}$ (obstacle and free space and no noise in a sensor), and $\{ A_O, A_F, A_N \}$
(obstacle, free space and noise in a sensor), $X$ is reduced to:

$$X = \{ \{ A_O \}, \{ A_F \}, \{ A_O, A_N \}, \{ A_F, A_N \} \}$$

and is used as the universe of discourse.

Definition

Let $X$ be a nonempty set considered as the universe of discourse, whose
generic elements are denoted $x$. A fuzzy set $A$ in $X$, that is a subset of $X$
that has no sharp boundary, is characterized by a membership function $\mu_A$:

$$X [0,1] \text{.} \text{ The closer the value of } \mu_A(x) \text{ is to } 1, \text{ the more } x \text{ belongs to } A \text{.}$$

$A$ is completely characterized by the set of pairs:

$$A = \{ (x, \mu_A(x)), x \in X \}$$
Sensor data contains information of the empty and occupied area. Instead of using them as crisp events, Okui defined the empty area as the fuzzy set EMP and the occupied area as the fuzzy set OBS. That is, a fuzzy set EMP indicates that the sensor says this is free space and a fuzzy set OBS indicates the sensor says this is an obstacle.

The set \( \{ A_O, A_O, A_N \}, \{ A_F, A_N \} \) is selected out of subsets of \( X \) and defined as a core of a fuzzy set OBS. The set \( \{ A_F, A_O, A_N \}, \{ A_F, A_N \} \) is selected out of the subsets of \( X \) and defined as a core of a fuzzy set EMP. Therefore, fuzzy sets OBS and EMP are expressed as:

\[
OBS = \{(x, \mu_{OBS}(x)), x \in X\}
\]

\[
OBS = \frac{\mu_{OBS}(\{A_O\})}{\{A_O\}} + \frac{\mu_{OBS}(\{A_F\})}{\{A_F\}} + \frac{\mu_{OBS}(\{A_O, A_N\})}{\{A_O, A_N\}} + \frac{\mu_{OBS}(\{A_F, A_N\})}{\{A_F, A_N\}} \tag{4-7}
\]

where

\( \mu_{OBS}(\{A_O\}) \): The grade of membership for a situation of obstacle and no noise in a sensor when a sensor says obstacle area.

\( \mu_{OBS}(\{A_F\}) \): The grade of membership for a situation of free space and no noise in a sensor when a sensor says obstacle area.

\( \mu_{OBS}(\{A_O, A_N\}) \): The grade of membership for a situation of obstacle and noise in a sensor when a sensor says obstacle area.

\( \mu_{OBS}(\{A_F, A_N\}) \): The grade of membership for a situation free space and noise in a sensor when a sensor says obstacle area.

\[
EMP = \{(x, \mu_{EMP}(x)), x \in X\}
\]

\[
EMP = \frac{\mu_{EMP}(\{A_O\})}{\{A_O\}} + \frac{\mu_{EMP}(\{A_F\})}{\{A_F\}} + \frac{\mu_{EMP}(\{A_O, A_N\})}{\{A_O, A_N\}} + \frac{\mu_{EMP}(\{A_F, A_N\})}{\{A_F, A_N\}} \tag{4-8}
\]

where

\( \mu_{EMP}(\{A_O\}) \): The grade of membership for a situation of obstacle and no noise in a sensor when a sensor says free space area.
\( \mu_{\text{EMP}}(\{ A_f \}) \): The grade of membership for a situation of free space and no noise in a sensor when a sensor says free space area.

\( \mu_{\text{EMP}}(\{ A_o, A_s \}) \): The grade of membership for a situation of obstacle and noise in a sensor when a sensor says free space area.

\( \mu_{\text{EMP}}(\{ A_f, A_s \}) \): The grade of membership for a situation of free space and noise in a sensor when a sensor says free space area.

The process of determining values for degree of membership is a subjective process and it is not always necessary to obtain exact values for the degree of membership. A slight error in the degree of membership will be less consequential than when a crisp event has an all-or-nothing representation. Instead of a mathematical approximation of the geometry of the beam, Okui chose an approach of fuzzy sets constructed from statistical data. These values are distributed all over the entire sensing field because of the randomness of noise, and are defined as follows:

In the presence of an object,

\[
\mu_{\text{los}}(\{ A_o \}) = \frac{\text{Number of obstacle signals: } |R_o - R| \leq \varepsilon}{\text{Total number of samples}} \tag{4-9}
\]

\[
\mu_{\text{emp}}(\{ A_o, A_s \}) = \frac{\text{Number of freespace signals: } R_o - R > \varepsilon}{\text{Total number of samples}} \tag{4-10}
\]

\[
\mu_{\text{los}}(\{ A_o, A_s \}) = \frac{\text{Number of obstacle signals: } |R_o - R| > \varepsilon}{\text{Total number of samples}} \tag{4-11}
\]

By removing the object from the sensing field,

\[
\mu_{\text{emp}}(\{ A_o, A_s \}) = \frac{\text{Number of freespace signals: } R_o - R > \varepsilon}{\text{Total number of samples}} \tag{4-12}
\]
\[ \mu_{\text{los}} = \{A_r\} = \frac{\text{Number of freespace samples}}{\text{Total number of samples}} \]  

(4-13)

\[ \mu_{\text{los}}(\{A_r, A_o\}) = \frac{\text{Number of false obstacle signals}}{\text{Total number of samples}} \]  

(4-14)

With no noise,

\[ \mu_{\text{los}}(\{A_r\}) = \mu_{\text{los}}(\{A_o\}) = 0.0 \]

Fuzzy sets OBS and EMP obtained by the way discussed above are stored as a sensor database. A sensing field is defined by the characteristics of a range sensor. Each cell has two fuzzy sets, OBS and EMP, whose values for degree of membership are obtained experimentally during calibration. When a sensor returns a reading, the fuzzy distribution is generated. Cells in the obstacle area have a fuzzy set OBS and cells in the free space have a fuzzy set EMP.

After producing a fuzzy distribution, a fuzzy set is projected onto the local grid map and input to a cell. As notations, fuzzy sets OBS and EMP after a transformation from a sensor coordinate system to a local grid map coordinate system are expressed as OBS(X,Y) and EMP(X,Y) where X and Y are the position of a cell in a local grid map coordinate system.

**Nonhomogenous Markov Chain Approach**

In the previous section, fuzzy modeling is discussed to represent the uncertainty such as randomness caused by noise and fuzziness caused by poor performance of sensors. Regarding a sequence of fuzzy sets as a time series, the nonhomogeneous Markov chain is applied to the stochastic process using a fuzzy model to reduce
uncertainty. For the local grid map management, two states in an environment are considered and the state space is defined as $\{A_0\}, \{A_r\}$. If the state space of a stochastic process is countable and finite and the process can be called a chain.

The movement of the process for the probability of occupancy among the members of the state space is determined by conditional probabilities. Different from a conditional probability in Bayes's theorem that produced $P(R|\{A_0\})$ from a range sensor directly as a crisp event, the following conditional probabilities are defined as transition probabilities for the chain using fuzzy sets, OBS and EMP. The probability of $X_{n+1} = \{A_0\}$ given the occurrence of $X_n = \{A_0\}$ is defined as

$$P(x_{n+1} = \{A_0\} | x_n = \{A_0\}) = \max \left[ \begin{array}{c}
P(x_n = \{A_0\}), \\
\mu_{OBS}(\{A_0\}) \times \min \left( \frac{n_{OBS}}{n}, m(n) \right), \\
\mu_{OBS}(\{A_0, A_s\}) \times \min \left( \frac{n_{OBS}}{n}, m(n) \right), \\
\mu_{EMP}(\{A_0, A_s\}) \times \min \left( \frac{n_{EMP}}{n}, m(n) \right) \end{array} \right]$$

and the probability of $X_{n+1} = \{A_r\}$ given the occurrence of $X_n = \{A_r\}$ is defined as

$$P(x_{n+1} = \{A_r\} | x_n = \{A_r\}) = \max \left[ \begin{array}{c}
P(x_n = \{A_r\}), \\
\mu_{EMP}(\{A_r\}) \times \min \left( \frac{n_{EMP}}{n}, m(n) \right), \\
\mu_{EMP}(\{A_r, A_s\}) \times \min \left( \frac{n_{EMP}}{n}, m(n) \right), \\
\mu_{OBS}(\{A_r, A_s\}) \times \min \left( \frac{n_{OBS}}{n}, m(n) \right) \end{array} \right]$$
where
\[ P(.) \] - previous probability in a cell
\[ n_{\text{OBS}} \] - number of OBS from sensor modeling
\[ n_{\text{EMP}} \] - number of EMP from sensor modeling
\[ n \] - total number of fuzzy sets: \( n = n_{\text{OBS}} + n_{\text{EMP}} \)
\[ m(.) \] - weighting function at the initial stage of sequence.
\[ P(X_{n+1},...) \] is the maximum value of four values.

A nonhomogeneous Markov chain is completely described by a starting vector and a sequence of transition matrices. A starting vector is the initial distribution of probabilities of the Markov chain and is defined as \( \pi_0 = (P(X_0 = \{A_o\}), P(X_0 = \{A_f\})) \). A cell is initialized as unknown, that is \( \pi_0 = (P(X_0 = \{A_o\}) = 0.5, P(X_0 = \{A_f\}) = 0.5) \). The transition matrix consists of conditional probabilities, and contain all the relevant information regarding the movement of the process among the states in state space, and is defined as

\[
P_n = \begin{bmatrix}
P(X_{n+1} = \{A_o\} | X_n = \{A_o\}) & 1 - P(X_{n+1} = \{A_o\} | X_n = \{A_o\}) \\
1 - P(X_{n+1} = \{A_f\} | X_n = \{A_f\}) & P(X_{n+1} = \{A_f\} | X_n = \{A_f\})
\end{bmatrix}
\]

(4-18)

Ergodicity of the nonhomogenous Markov chain is used to see if the limiting vector is independent of the choice of the starting vector. There is weak ergodicity and strong ergodicity. When a limiting vector is independent of the starting vector, the effect of the starting vector is lost after a long time. This is called a loss of memory. For strong ergodicity, the behavior of the process is convergence and loss of memory. Weak ergodicity is the loss of memory without convergence. A weak ergodicity was used to prove strong ergodicity. When the formula has strong ergodicity, the limiting probability of obstacles is independent of the initial probability of a cell. Therefore, when the vehicle approaches an obstacle in different orientations and at a different times, the final
probability must converge to the same values as long as a same sensor database is used. It is then possible to increase the certainty by revisiting an obstacle. Okui went on to prove strong ergodicity for the nonhomogenous Markov chain. This strong ergodicity indicates the limiting probability value in a cell is independent of the initial probability.

Simulation

The map management strategies were developed and demonstrated in simulation. Grid model techniques are simulated and evaluated in an indoor environment, and are also investigated as a possibility for application in an outdoor environment. Simulation programs were written in C++ with Open Inventor and Motif used to develop the computer graphics and graphical user interface. The dimension of the local grid map is 41x41 cells and each cell represents a real world square of size 0.5m x 0.5m. Uniform distributed noise is added to general range sensors and produces two kinds of errors. One is an error in distance caused by the environment-obstacle interaction, and the other is a false detection caused by the sensor performance. A car-like vehicle was used in the simulation. It is assumed that no wheel slippage occurs and that no acceleration and deceleration are considered.

Assumptions

The environment and characteristics of information used in this research are specified by the performance of the sensors used. The following assumptions are made:

- Accurate position and orientation data can be obtained for the vehicle. To obtain accurate position and orientation, many researchers who study navigation use a variety of sensors and methods. For an outdoor vehicle, it is common to use a Differential Global Positioning System. The sensor-related positioning errors are not considered here.
• Obstacles are unknown, multiple, positive, and natural shaped objects in an outdoor environment. They are also stationary. Unknown objects are objects that are not known at the start of the simulation. Multiple objects are just several objects in one field-of-view of the sensor. Positive objects are convex objects such as trees and rocks extending upward from the ground. Negative objects like ditches and holes are not considered.

• A limited sensing field of view, different sampling times, and resolutions are considered as performance parameters of ultrasonic and laser range sensors.

• The vehicle is operated in a simulated environment that consists of three dimensional terrain and positive objects. It is controlled by speed and steering angle commands.

**Conclusions**

Okui’s [OKU99] map management strategy involved the development of a multi-layered map management system. The system consists of a local grid map and a global contour map. The local grid map deals with obstacle-free space detection using range data from a variety of sensors. The global contour map reduces the size of data from the local grid map. The system maintains the consistency between two maps. The management strategy was implemented and demonstrated in simulation.

The local grid map is represented as a grid model and the management strategy is based on a concept of uncertainty. Uncertainty was classified under one of three categories; randomness, fuzziness, and indeterminacy.

The probability theory approach regards a poor performance of a sensor as randomness and models the sensor by an empty probability density function and an occupied probability density function. To update a grid map, the additive law of probability was applied, assuming the value of previous map and sensor data are independent from each other. The results obtained in simulation indicate this approach is not suitable for an outdoor environment because of the high sensitivity to noise.
The Bayes’s theorem approach also regards poor performance of a sensor as randomness and describes uncertainty by a probabilistic distribution of occupancy. To update a grid map, a probability produced by range data was regarded as a conditional probability and Bayes’s theorem was used. The results obtained in simulation show this approach can be used for an indoor environment but is not suitable for the outdoor environment because of the high sensitivity to noise.

The histogram method approach regards a poor performance of a sensor as randomness as well, and implements a simplified sensor model using a histogramic probability distribution. To update a map, the additive law of probability was modified to increase and decrease a cell value by an integer number, assuming a value of a previous map and sensor data are not only independent but also mutually exclusive. The results obtained in simulation indicate this approach requires repeatability of sensing and it is very difficult to use in both indoor and outdoor environments when a vehicle moves more than 0.8 m/s.

The fuzzy logic approach regards noise and poor performance of a sensor as fuzziness and models a sensor by a membership function. To update the map, the fuzzy union operator was used as a non-linear accumulation. The results obtained in simulation show this approach also requires the repeatability of sensing and it is difficult to use when a vehicle moves faster than 0.8 m/s.

The results obtained for previous grid map techniques conclude that they are not adequate for an outdoor environment. Hence, Okui developed a significant theoretical contribution in the work. He classified uncertainty and described uncertainty of a range sensor caused by noise and poor performance. The real world was modeled using a
subset of three elements; obstacle, free space, and sensor noise. A fuzzy model approach was adopted and sensor output was regarded as fuzzy sets. Experimental statistical data are used to assign the degree of membership to a focal element of fuzzy sets and uncertainty was represented in this way. To update the map, a sequence of fuzzy sets was regarded as a stochastic process and nonhomogeneous Markov chain was applied. The new formula was proved to be strongly ergodic and that the final probability is independent from the vehicle's approach to an obstacle. The results in a simulated outdoor environment support the validity of this work.
CHAPTER 5
LOCAL GRID MAP IMPLEMENTATION

What was introduced in the previous chapter was the background theory used for describing the sensors and updating the Local Grid Map (LGM). This chapter covers how all the theory is put into practice. The simulation helped prove the theory, however extensions were necessary to implement it in a real environment. First addressed is how the LGM is represented, followed by how the sensor’s field of view is discretized into a polygon. Then the techniques used to merge the sensors information into the LGM will be covered, and a discussion of the tunable parameters follow. Finally, a numerical example is presented to clarify the computational background of the theory.

Local Grid Map Representation

To facilitate the modification of the range, size, and probability of the local grid map, a text file is read at runtime to set all of these parameters, as shown below.

```
# Grid Map config file
# Grid map cutoff value for empty or obstacle (percentage 0.0 - 1.0).
0.8

# Grid resolution and range (in meters).
0.25, 8.0

# Rollover, Empty Increment, Obstacle Increment
50, 1, 10
```

Grid map configuration file

Comments are marked by a ‘#’, the empty/obstacle percentage is expressed as a real number from 0.0 to 1.0, and the resolution and range of the grid is expressed in meters.
The range value stored in the file is the distance from the center of the grid to the edge of the grid, as shown in Figure 5-1. The resolution of the grid is the distance from one edge to another of a grid cell. The entire grid map is square and created with an odd number of square grid cells. This creates a center (0,0) grid in which the vehicle is located. The final set of numbers are used to fine tune the system. The values are used to adjust the refresh rate of the grid cell (Rollover), and the increments for updating a cell with empty or obstacle. These tunable parameters are discussed in more detail later in the chapter.

Internally, the Local Grid Map is stored as a dynamically allocated 2D array. The size of the array is computed based on the resolution and range of the input file, based on the following formula:

$$\text{temp} = \frac{\text{range}}{\text{resolution}}$$
$$n = 2 \times (\text{temp} + \text{roundUp}(\text{temp})) + 1$$

$$\text{roundUp}(x) = \begin{cases} 
0 & x = (\text{int})x \\
1 & x > (\text{int})x 
\end{cases}$$

(5-1)

This creates an odd array size with n by n members. To reduce the storage size of the grid, the amount of data stored within each grid cell was minimized as shown in the structure below. The smallest data type, an unsigned char was chosen to store the number of times the cell is updated as an empty/obstacle. The probabilities that the cell is
typedef struct _gridMapCell {
    unsigned char nObs, nEmp, obsProb, empProb;
} gridMapCell_t;

Grid cell structure

an obstacle or empty space is also stored as an unsigned char. The decimal fraction percent is converted to an unsigned char by multiplying the number by 200 and truncating the result. This results in a resolution of approximately 0.5%. When nObs or nEmp equal the maximum value, the lower value is reduced to a minimum, and the higher value is computed based on the probability. This eliminates the dilemma of rollover error, maintaining the computed probability percentages.

Sensor Representation

As with the grid map, the parameters for each sensor are contained within their own configuration file. The configuration file, as shown, contains all of the sensor’s physical properties along with the fuzzy representation of the sensor’s uncertainties. As with the grid configuration file, comments are marked with a ‘#’. The position of the sensors is expressed from the sensor coordinate system with the major axis of the sensor’s range expressed along the X-axis. The file contains the transformation from the vehicle’s coordinate system to the sensor coordinate system, the physical properties of the sensor, the fuzzy representation of the sensors’ uncertainty, the number of sensors, and finally the transformation of the sensor within the sensor’s coordinate system.

The sensor’s physical characteristics determine the type of polygon used to represent the sensors field-of-view, as shown in Figure 5-2. The values Δx and Δy are computed based on $R_{\text{max}}$ and the beam angle, Θ. If Δy is less than or equal to the grid resolution, then a line is used to represent the sensor. If Δy is greater than the grid
# Sonar config file
# Offset from vehicle coordinate system
# X, Y, Z (in meters), thetaX, thetaY, thetaZ (in degrees).
1.20650, 0, 0, 0, 180, 90

# Sensor field: Rmin, Rmax (meters), angular resolution (degrees).
#0.1524, 5, 30
0.1524, 7, 30

# Sensor fuzzy sets:
# OBS({Ao}), OBS({Ao, An}), OBS({Af, An}), EMP({Af}), EMP({Ao,An}),
EMP{Af,An})
1, 0, 0, 1, 0, 0

# Number of Sensors
16

# Sensors positions from sensor coordinate syste (SCS)
# X, Y, Z (in meters), thetaX, thetaY, thetaZ (in degrees).
# Starting from far right, firing order:
# 9
# 14 11 16 13 2 7 4 1
0.34765, 0.05398, 0, 0, 0, 35
0.22065, 0.17620, 0, 0, 0, 57
0.05555, 0.23495, 0, 0, 0, 80.5
0.30798, 0.10160, 0, 0, 0, 41.5
0.16985, 0.20320, 0, 0, 0, 67
0, 0.23970, 0, 0, 0, 90
0.26827, 0.13970, 0, 0, 0, 48
0.10795, 0.22382, 0, 0, 0, 75
0, 0.23970, 0, 0, 0, 90
-0.16985, 0.20320, 0, 0, 0, 114
-0.30955, 0.10160, 0, 0, 0, 137.5
-0.05237, 0.23495, 0, 0, 0, 98
-0.22065, 0.17463, 0, 0, 0, 122.5
-0.34607, 0.05555, 0, 0, 0, 144
-0.11430, 0.22225, 0, 0, 0, 107.5
-0.26670, 0.13970, 0, 0, 0, 130

Sensor’s physical characteristics

resolution, and Δx is less than or equal to the grid resolution, then a 3-sided polygon is used to represent the sensor.

Finally, if Δy is greater than the grid resolution, and Δx is greater than the grid resolution, then a 4-sided polygon is used to represent the sensor.

Figure 5-2 Sensor’s physical properties
Fill Techniques

Since the greatest amount of time would be spent updating the local grid map with sensor information, it was desirable to find the most efficient method to do so. The local grid map resembles a computer screen with the grid cells representing the pixels on the screen. Therefore, the techniques for drawing lines and filled polygons on the computer screen were investigated and adapted. The advantage of these algorithms was the exclusive use of fast integer arithmetic.

Line Drawing

A scan-conversion algorithm for lines computes the coordinates of the pixels that lie on or near an ideal, infinitely thin straight line imposed on a 2D raster grid. In principle, the sequence of pixels should lie as close to the ideal line as possible and to be as straight as possible. Consider a 1-pixel-thick approximation to an ideal line; what properties should it have? For lines with slopes between -1 and 1 inclusive, exactly 1 pixel should be illuminated in each column; for lines with slopes outside this range, exactly 1 pixel should be illuminated in each row. All lines should be drawn as rapidly as possible. Since the line is defined on a integer grid, the endpoints have integer coordinates.

Bresenham, [FOL90], developed a classic algorithm that is attractive because it uses only integer arithmetic, and allows the calculation for \((x_i, y_i)\) to be performed incrementally - that is, by using the calculation already done at \((x, y)\). Bresenham showed that his line and integer circle algorithms provide the best-fit approximations to true lines and circles by minimizing the error (distance) to the true primitive.
Consider the line in Figure 5-3, where the previously selected pixel appears as a black circle and the two pixels from which to choose at the next stage are shown as unfilled circles. Assume that the pixel \( P \) at \((x_p, y_p)\) has just been selected and now must choose between the pixel one increment to the right (called the east pixel, \( E \)) or the pixel one increment to the right and one increment up (called the northeast pixel, \( NE \)). Let \( Q \) be the intersection point of the line being scan-converted with the grid line \( x = x_p + 1 \). In Bresenham’s formulation, the difference between the vertical distances from \( E \) and \( NE \) to \( Q \) is computed, and the sign of the difference is used to select the pixel whose distance from \( Q \) is smaller as the best approximation to the line. In the midpoint formulation, it is observed on which side of the line the midpoint \( M \) lies. It is easy to see that, if the midpoint lies above the line, pixel \( E \) is closer to the line; if the midpoint lies below the line, pixel \( NE \) is closer to the line. The line may pass between \( E \) and \( NE \), or both pixels may lie on one side, but in any case, the midpoint test chooses the closest pixel. Also, the error - that is, the vertical distance between the chosen pixel and the actual line - is always \( \leq \frac{1}{2} \).

What is needed is a way to calculate on which side of the line the midpoint lies. By representing the line as an implicit function with coefficients \( a, b, \) and \( c \): \( F(x,y) = ax + by + c = 0 \). If \( dy = y_i - y_o \), and \( dx = x_i - x_o \), the slope-intercept form can be written as
\[ y = \frac{dy}{dx} + B \] (5-2)

therefore,

\[ F(x, y) = dy \cdot x - dx \cdot y + B = 0 \] (5-3)

Hence \( a = dy \), \( b = -dx \), and \( c = B \cdot dx \) in the implicit form.

It can easily be verified that \( F(x,y) \) is zero on the line, positive for points below the line, and negative for points above the line. To apply the midpoint criterion, \( F(M) = F(x_p + 1, y_p + \frac{1}{2}) + c \) need only be computed, and the sign tested. A decision variable, \( d = F(x_p + 1, y_p + \frac{1}{2}) \), is defined. By definition, \( d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c \). If \( d > 0 \), choose \( NE \); if \( d < 0 \), choose \( E \); and if \( d = 0 \), choose either, so pick \( E \).

The location of \( M \) and therefore the value of \( d \) must be updated for the next grid line; both depend, of course, on whether \( E \) or \( NE \) is chosen. If \( E \) is chosen, \( M \) is incremented by one step in the \( x \) direction. Then,

\[ d_{new} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c \] (5-4)

but

\[ d_{old} = a(x_p + 1) + b(y_p + \frac{1}{2}) + c \] (5-5)

Subtracting \( d_{old} \) from \( d_{new} \) to get the incremental difference, \( d_{new} = d_{old} + a \).

The increment to add to \( d \) after \( E \) is called \( \Delta_E \); \( \Delta_E = a = dy \). In other words, the value of the decision variable can be derived at the next step incrementally from the value at the current step without having to compute \( F(M) \) directly, by merely adding \( \Delta_E \).

If \( NE \) is chosen, \( M \) is incremented by one step each in both the \( x \) and the \( y \) directions. Then,
\[ d_{\text{new}} = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c \]  \hspace{1cm} (5-6)

Subtracting \(d_{\text{old}}\) from \(d_{\text{new}}\) to get the incremental difference, we write

\[ d_{\text{new}} = d_{\text{old}} + a + b \]  \hspace{1cm} (5-7)

The increment to add to \(d\) after \(NE\) is called \(\Delta_{NE}\): \(\Delta_{NE} = a + b = dy - dx\).

Recapping, at each step, the algorithm chooses between 2 pixels based on the sign of the decision variable, \(d\), calculated in the previous iteration; then, it updates the decision variable by adding either \(\Delta_E\) or \(\Delta_{NE}\) to the old value, depending on the pixel choice.

Since the first pixel is simply the first endpoint \((x_0, y_0)\), we can directly calculate the initial value of \(d\) for choosing between \(E\) and \(NE\). The first midpoint is at \((x_0 + 1, y_0 + \frac{1}{2})\), and

\[
F(x_0 + 1, y + \frac{1}{2}) = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c \\
= ax_0 + by_0 + c + a + \frac{b}{2} \\
= F(x_0, y_0) + a + \frac{b}{2} \]  \hspace{1cm} (5-8)

But \((x_0, y_0)\) is a point on the line and \(F(x_0, y_0)\) is therefor 0; hence, \(d_{\text{star}}\) is just \(a + b/2 = dy - dx/2\). Using \(d_{\text{star}}\), the second pixel is chosen, and so on. To eliminate the fraction in \(d_{\text{star}}\), the original \(F\) is redefined by multiplying it by 2; \(F(x, y) = 2(ax + by + c)\). This multiplies each constant and the decision variable by 2, but does not affect the sign of the decision variable, which is all that matters for the midpoint test. The arithmetic needed to evaluate \(d_{\text{new}}\) for any step is simple addition. No time-consuming multiplication is involved.
The adaptation of this method to the research is rather simple. First, determine if the range value presented represents the existence of an obstacle (\( R < R_{\text{max}} \)). If no obstacle is present, update the entire range from \((x_{\text{start}}, y_{\text{start}})\) to \((x_{\text{end}}, y_{\text{end}})\) with \textit{Empty}. However, if an obstacle is present, update the range from \((x_{\text{start}}, y_{\text{start}})\) to the point just before \((x_{\text{end}}, y_{\text{end}})\) with \textit{Empty}, and update the last point, \((x_{\text{end}}, y_{\text{end}})\) with \textit{Obstacle}. The full implementation of the algorithm is shown below.

```c
void fillLine( polygonPoint_t *point, int isObstacle,
    sensorData_t *sensorData, gridMap_t *gridMap )
{
    void updateGridCell( cellType_t cellType, int x, int y,
        sensorData_t *sensorData, gridMap_t *gridMap )
    {
        int x1, y1, x2, y2 ;
        int dx, dy, xInc, yInc ;
        cellType_t cellType ;
        x1 = point[0].x ; y1 = point[0].y ;
        x2 = point[1].x ; y2 = point[1].y ;
        dx   = abs( x2 - x1 ) ;
        dy   = abs( y2 - y1 ) ;
        xInc = (x1 > x2) ? -1 : 1 ;
        yInc = (y1 > y2) ? -1 : 1 ;
        if ( dx >= dy ) {
            int dPr   = dy<<1 ;
            int dPru  = dPr - (dx<<1) ;
            int P     = dPr - dx ;
            for ( ; dx>=0 ; --dx ) {
                cellType = Empty ;
                if ( isObstacle && (x1 == x2) && (y1 == y2) )
                    cellType = Obstacle ;
                updateGridCell( cellType, x1, y1, sensorData, gridMap ) ;
                if ( P > 0 ) {
                    x1 += xInc ;
                    y1 += yInc ;
                    P  += dPru ;
                } else {
                    x1 += xInc ;
                    P  += dPr ;
                }
            }
        } else {
            int dPr   = dx<<1 ;
            int dPru  = dPr - (dy<<1) ;
            int P     = dPr - dy ;
            for ( ; dy>=0 ; --dy ) {
                cellType = Empty ;
                if ( isObstacle && (x1 == x2) && (y1 == y2) )
                    cellType = Obstacle ;
                updateGridCell( cellType, x1, y1, sensorData, gridMap ) ;
                if ( P > 0 ) {
                    x1 += xInc ;
                    y1 += yInc ;
                    P  += dPru ;
                } else {
                    x1 += xInc ;
                    P  += dPr ;
                }
            }
        }
    }
    int x1, y1, x2, y2 ;
    int dx, dy, xInc, yInc ;
    cellType_t cellType ;
    x1 = point[0].x ; y1 = point[0].y ;
    x2 = point[1].x ; y2 = point[1].y ;
    dx   = abs( x2 - x1 ) ;
    dy   = abs( y2 - y1 ) ;
    xInc = (x1 > x2) ? -1 : 1 ;
    yInc = (y1 > y2) ? -1 : 1 ;
    if ( dx >= dy ) {
        int dPr   = dy<<1 ;
        int dPru  = dPr - (dx<<1) ;
        int P     = dPr - dx ;
        for ( ; dx>=0 ; --dx ) {
            cellType = Empty ;
            if ( isObstacle && (x1 == x2) && (y1 == y2) )
                cellType = Obstacle ;
            updateGridCell( cellType, x1, y1, sensorData, gridMap ) ;
            if ( P > 0 ) {
                x1 += xInc ;
                y1 += yInc ;
                P  += dPru ;
            } else {
                x1 += xInc ;
                P  += dPr ;
            }
        }
    } else {
        int dPr   = dx<<1 ;
        int dPru  = dPr - (dy<<1) ;
        int P     = dPr - dy ;
        for ( ; dy>=0 ; --dy ) {
            cellType = Empty ;
            if ( isObstacle && (x1 == x2) && (y1 == y2) )
                cellType = Obstacle ;
            updateGridCell( cellType, x1, y1, sensorData, gridMap ) ;
            if ( P > 0 ) {
                x1 += xInc ;
                y1 += yInc ;
                P  += dPru ;
            } else {
                x1 += xInc ;
                P  += dPr ;
            }
        }
    }
}
```
updateGridCell(cellType, x1, y1, sensorData, gridMap);

if (P > 0) {
    x1 += xInc;
    y1 += yInc;
    P  += dPru;
} else {
    y1 += yInc;
    P  += dPr;
}
} /* end of fillLine()... */

**Polygon Filling**

The general polygon scan-conversion algorithm, [FOL90] described next handles
both convex and concave polygons, even those that are self-intersecting or have interior holes. It operates
by computing spans that lie between left and right edges of the polygon. The span extrema are calculated by
an incremental algorithm that computes a scan line/edge intersection from the intersection with the previous scan line. Figure 5-4, which illustrates the basic polygon scan-conversion process, show s a polygon and one scan line passing through it. The intersections of scan line 8 with edges FA and CD lie on integer coordinates, whereas those for EF and DE do not. The intersections are marked in the figure by vertical tick marks labeled a through d. It must be determined which pixels on each scan line are within the polygon, and those pixels (in this case, spans from x=2 through 4 and 9 through 13) must be set to their appropriate values.
A straightforward way of deriving the extrema is to use the midpoint line scan-conversion algorithm (Bresenham’s algorithm) on each edge and to keep a table of span extrema for each scan line, updating an entry if a new pixel is produced for an edge that extends the span. Note that this strategy produces some extrema pixels that lie outside the polygon; they were chosen by the scan-conversion algorithm because they lie closest to an edge, without regard to the side of the edge on which they lie - the line algorithm has no notions of interior and exterior. It is obviously preferable to draw only those pixels that are strictly interior to the region, even when an exterior pixel would be closer to the edge. With this criterion, a polygon does not intrude (even by a single pixel) into the regions defined by other primitives.

As with the original midpoint algorithm, an incremental algorithm is used to calculate the span extrema on one scan line from those at the previous scan line without having to compute the intersections of a scan line with each polygon edge analytically. In scan line 8 of Figure 5-4, for instance, there are two spans of pixels within the polygon. The spans can be filled in by a three-step process:

1. Find the intersections of the scan line with all edges of the polygon.

2. Sort the intersections by increasing x coordinate.

3. Fill in all pixels between pairs of intersections that lie interior to the polygon, using the odd-parity rule to determine that a point is inside a region: Parity is initially even, and each intersection encountered thus inverts the parity bit - draw when parity is odd, do not draw when it is even.

Step 1 in the procedure, calculating intersections, must be done cleverly lest it be slow. In particular, the brute-force technique of testing each polygon edge for intersection with each new scan line must be avoided. Very often, only a few of the edges
are of interest for a given scan line. Furthermore, many edges intersected by scan line $i$ are also intersected by scan line $i+1$. The edge coherence occurs along an edge for as many scan line as intersect that edge. As the scan lines advance, the new $x$ intersection of the edge can be computed on the basis of the old $x$ intersection, just as the next pixel was computed from the current pixel in the midpoint line scan conversion, by using

$$x_{i+1} = x_i + \frac{y}{m} \quad (5-9)$$

where $m$ is the slope of the edge. In the midpoint algorithm for scan converting lines, fractional arithmetic was avoided by computing an integer decision variable and checking only its sign to choose the pixel closest to the mathematical line; here, integer arithmetic will be used to do the required rounding for computing the closest interior pixel.

Consider lines with a slope greater that $+1$ that are left edges; right edges and other slopes are handled by similar arguments, and vertical edges are special cases. Draw a pixel at the $(x_{min}, y_{min})$ endpoint. As $y$ is incremented, the $x$ coordinate of the point on the ideal line will increase by $1/m$, where $m = (y_{max} - y_{min})/(x_{max} - x_{min})$ is the slope of the line. This increase will result in $x$ having an integer and a fractional part, which can be expressed as a fraction with a denominator of $y_{max} - y_{min}$. As this process is iterated, the fractional part will overflow and the integer part will have to be incremented. For example, if the slope is $5/2$, and $x_{min}$ is $3$, then the sequence of $x$ values will be $3, 3 \ 2/5, 3 \ 4/5, 3 \ 6/5 = 4 \ 1/5$, and so on. When the fractional part of $x$ is zero, draw the pixel $(x,y)$ that lies on the line, but when the fractional part of $x$ is nonzero, round up in order to get a pixel that lies strictly inside the line. When the fractional part of $x$ becomes greater than $1$, increment $x$, and subtract $1$ from the fractional part; also move $1$ pixel to the right.
Fractions can be avoided by keeping track only of the numerator of the fraction and observing that the fractional part is greater than 1 when the numerator is greater than the denominator. Keep track of successive additions of the numerator until it “overflows” past the denominator, when the numerator is decremented by the denominator and \( x \) is incremented.

To take care of step 2, sorting intersections, a \textit{scan-line algorithm} is developed. This algorithm takes advantage of this edge coherence. It also maintains the set of edges each scan lines intersects and the set of intersection points in a data structure called the \textit{active-edge table} (AET), which is sorted by their \( x \) intersections. As we move to the next scan line at \( y+1 \), the AET is updated. First, edges currently in the AET but not intersected by this next scan line (i.e., those whose \( y_{max} = y \)) are deleted. Second, any new edges intersected by this next scan line (i.e., those edges whose \( y_{min} = y+1 \)) are added to the AET. Finally, new \( x \) intersections are calculated, using the preceding incremental edge algorithm, for edges remaining in the AET.

To make the addition of edges to the AET efficient, a global \textit{edge table} (ET) is initially created containing all edges sorted by their smaller \( y \) coordinate.

Within each \( y \) coordinate, edges are kept in order of increasing \( x \) coordinate of the lower endpoint.

Each entry in the ET contains the \( y_{max} \) coordinate of the edge, the \( x \) coordinate of the bottom endpoint

![Figure 5-5 Sorted edge table](image)
(x_m), and the x increment used in stepping from one scan line to the next, 1/m. Figure 5-5 shows how the six edges from the polygon of Figure 5-4 would be sorted, and Figure 5-6 shows the AET at scan line 9 and 10 for that polygon. Once the ET has been formed, the processing steps for the scan-line algorithm are as follows:

1. Set y to the smallest y coordinate that has an entry in the ET.
2. Initialize the AET to be empty
3. Repeat until the AET and ET are empty:
   a. Move from ET y to the AET those edges whose \( y_{min} = y \) (entering edges), then sort the AET on x.
   b. Fill in desired pixel values on scan line y by using pairs of x coordinates from the AET.
   c. Remove from the AET those entries for which \( y = y_{max} \) (edges not involved in the next scan line).
   d. Increment y by 1 (to the coordinate of the next scan line).
   e. For each nonvertical edge remaining in the AET, update x for the new y.

This algorithm uses both edge coherence to calculate x intersections and scan-line coherence (along with sorting) to calculate spans. For purposes of scan conversion, triangles and trapezoids can be treated as special case of polygons, since they have only
two edges for any scan line (given that horizontal edges are not scan-converted explicitly).

And finally step 3, the span-filling strategy is discussed. In Figure 5-4, the sorted list of x coordinates for the scan line is (2, 4.5, 8.5, 13). However, four more questions must be addressed before the polygon is filled (with step 3 reiterated for clarity):

3 Fill in all pixels between pairs of intersections that lie interior to the polygon, using the odd-parity rule to determine that a point is inside a region: Parity is initially even, and each intersection encountered thus inverts the parity bit - draw when parity is odd, do not draw when it is even.

3.1 Given an intersection with an arbitrary, fraction x value, how to determine which pixel on either side of that intersection is interior?

3.2 How to deal with the special case of intersections at integer pixel coordinates?

3.3 How to deal with the special case in 3.2 for shared vertices?

3.4 How to deal with the special case in 3.2 in which the vertices define a horizontal edge?

To handle case 3.1, if we are approaching a fractional intersection to the right and are inside the polygon, round down the x coordinate of the intersection to define the interior pixel; if we are outside the polygon, round up the x coordinate to be inside.

Handle case 3.2 by applying the criterion used to avoid conflicts at shared edges of rectangles: If the leftmost pixel has integer x coordinate, define it to be interior, if the rightmost pixel has integer x coordinate, define it to be exterior. For case 3.3, count the \( y_{min} \) vertex of an edge in the parity calculation but not the \( y_{max} \) vertex; therefore, a \( y_{max} \) vertex is drawn only if it is the \( y_{min} \) vertex for the adjacent edge. Vertex \( A \) in Figure 5-4, for example, is counted once in the parity calculation because it is the \( y_{min} \) vertex for edge \( FA \) but the \( y_{max} \) vertex for \( AB \). Thus, both edges and spans are treated as intervals that are
closed at their minimum value and open at their at their maximum value. Clearly, the opposite rule would work as well, but this rule seems more natural since it treats the minimum endpoint as an entering point, and the maximum as a leaving point. When treating case 3.4, horizontal edges, the desired effect is that bottom edges are drawn but top edges are not. This happens automatically if the horizontal edges’ vertices are not counted, since they are neither \( y_{\text{min}} \) nor \( y_{\text{max}} \) vertices.

Apply these rules to scan line 8 in Figure 5-4, which hits no vertices. Pixels are filled from point \( a \), pixel (2,8), to the first pixel to the left of point \( b \), pixel (4,8), and from the first pixel to the right of point \( c \), pixel (9,8), to 1 pixel to the left of point \( d \), pixel (12,8). For scan line 3, vertex \( A \) counts once because it is the \( y_{\text{min}} \) vertex of edge \( FA \) but the \( y_{\text{max}} \) vertex of \( AB \); this cause odd parity, so we draw the span from there to 1 pixel to the left of the intersection with edge \( CB \), where the parity is set to even and the span is terminated. Scan line 1 hits only vertex \( B \); edges \( AB \) and \( BC \) both have their \( y_{\text{min}} \) vertices at \( B \), which is therefore counted twice and leaves the parity even. This vertex acts as a null span - enter at vertex, draw the pixel, exit at the vertex. Although such local minima draw a single pixel, no pixel is drawn at a local maximum, such as the intersection of scan line 9 with the vertex \( F \), shared by edges \( FA \) and \( EF \). Both vertices are \( y_{\text{max}} \) vertices and therefore do not affect the parity, which stays even.

The modifications to the polygon fill algorithm are more involved than for the previous case of scan converting a line segment. Only concave polygons are used, which simplifies some aspects, however the complications require the necessity of maintaining the ‘obstacle’ line segment and the orientation of the polygon for optimum filling of the ‘obstacle’ edge.
The advantage of using only concave polygons is that there are only two active edges at one time. This simplifies the implementation of the AET, since there will only be two elements at any time. And since there are, at most, four edges for the ET, both the ET and the AET are implemented using doubly linked lists. Because there are so few elements, a bucket sort is unnecessary. They are sorted in the ET, first by the minimum step (either the minimum x, or the minimum y, depending on the orientation), and then by slope. When the AET is updated, it always contains two elements, and therefor the parity used for determining which way to round the point is unneeded. The first edge must be rounded up to be contained within the polygon, while the last edge must be rounded down for the same reason.

When there is no obstacle within the field of view of the sensor, filling the polygon becomes the same as any other polygon, start from the bottom and fill to the top with *EMPTY*. Add an obstacle to the mix, and the complexity increases. There is now an edge that has importance over the others. It must be kept track of, so that when it’s active, the point that falls on that edge fills the corresponding grid cell with *OBSTACLE*. This is particularly true with horizontal edges. As with the normal algorithm, regular edges (*EMPTY*), are dropped, since they are picked up by the surrounding edges. When the edge is a *OBSTACLE*, it too is dropped, but a flag is set which fills the entire span with *OBSTACLE*.

Unlike the original algorithm, the orientation of the polygon is important when there is an ‘obstacle’ edge. When the major axis of the polygon is oriented close to horizontal, filling horizontally works well, since the edge which is an obstacle is oriented approximately vertical. However, orientate the polygon close to vertical, and the
horizontal fill leaves gaps in the obstacle’s edge, see Figure 5-7. Therefore, the orientation is taken into account when filling the polygon. When the major axis is close to horizontal, fill the polygon from bottom to top. When the major axis is close to vertical, fill the polygon from left to right. All that’s required is to swap the references of the x and y variables. By doing so, the obstacle edge is more uniformly filled. The fully implemented algorithm is shown below

```c
void
fillPolygon( int nPoints, polygonPoint_t *point, int isObstacle,
             sensorData_t *sensorData, gridMap_t *gridMap )
{
    void          activeEdgeIncX( doubleList_t *aet ) ;
    int           activeEdgeUpdate( doubleList_t *et, doubleList_t *aet,
                                    int y, int *nextY, et_t *etD ) ;
    polygonFill_t edgeTableCreate( int nPoints, int isObstacle,
                                    polygonPoint_t *point,
                                    doubleList_t *edgeTable ) ;
    void          updateGridCell( cellType_t cellType, int x, int y,
                                   sensorData_t *sensorData,
                                   gridMap_t *gridMap ) ;
    int           printEdgeData( generic_ptr* ) ;

doubleList_t edgeTable, activeEdgeTable ;
et_t etData ;
et_t *aetData0=NULL, *aetData1=NULL ;
polygonFill_t fill ;
in i, step, *x, *y, nextStep ;
in full, startIbs=FALSE, endObs=FALSE, firstHalf=1,
   startI, endI, tmpI ;
/* Create the edge table */
doubleInitList( &edgeTable ) ;
doubleInitList( &activeEdgeTable ) ;
fill = edgeTableCreate( nPoints, isObstacle, point, &edgeTable ) ;
/* Create the active edge table, and start-a-filling */
/* Since the polygons are convex, there will be only */
/* two edges active at the time (except for a horizontal */
/* edge, which is a special case anyway, damn them) */
/* "Point" to the correct values */
if ( fill == Horizontal ) {
    x = &i ;
    y = &step ;
}
```

**Figure 5-7** Polygon fill style. (a) - Polygon filled horizontally. (b) - Polygon filled vertically.
else {
  x = &step ;
  y = &i ;
}

while( !doubleListEmpty(edgeTable) ) {

  /* Initialize y and nextY */
  step     = ( (et_t *)DATA( edgeTable ) )->stepMin ;
  nextStep = ( (et_t *)DATA( edgeTable ) )->stepMax ;
  full     = activeEdgeUpdate( &edgeTable, &activeEdgeTable, 
                               step, &nextStep, &etData ) ;

  if ( doubleListEmpty(edgeTable) )
    ++nextStep ;

  while( step < nextStep ) {

    if ( full ) { /* Horizontal edge */
      startI = etData.I ;
      endI   = startI + etData.errorSumAdv ;
    } else {
      if ( NEXT(activeEdgeTable) == NULL )
        return ;

      aetData0 = (et_t *)DATA( activeEdgeTable ) ;
      aetData1 = (et_t *)DATA( NEXT(activeEdgeTable) ) ;

      if ( firstHalf==2 ) {
        if ( aetData0->errorSum == 0 ) {
          aetData0->errorSum += aetData0->errorSumAdv ;
          aetData0->errorSum -= aetData0->errorReset ;
        }

        if ( aetData1->errorSum == 0 ) {
          aetData1->errorSum += aetData0->errorSumAdv ;
          aetData1->errorSum -= aetData0->errorReset ;
        }
      }

      firstHalf = 3 ;
    }

    startObs = aetData0->obsEdge ;
    startI   = aetData0->I ;
    endObs   = aetData1->obsEdge ;
    endI     = aetData1->I ;

    if ( startI > endI ) {
      tmpI   = endI ;
      endI   = startI ;
      startI = tmpI ;
    }

    for ( i=startI ; i<=endI ; ++i ) {

      cellType = Empty ;
      /* Entire line is obstacle edge */
      if ( full )
        cellType = Obstacle ;

      /* Starting or ending line is obstacle edge */
      else {
        if ( i==aetData0->I && startObs )
          cellType = Obstacle ;

        if ( i==aetData1->I && endObs )
          cellType = Obstacle ;
      }

      updateGridCell( cellType, *x, *y, sensorData, gridMap ) ;
    }
  }
}

else {
Tunable Parameters

After observing the system, a set of values that could modify the behavior appeared. These variables fall into two categories. The first is the top end for the stored counting value, the rollover parameter. The second is the increment used to update the empty and obstacle values.

Rollover

The variables used to store the counts for the number of times a particular cell is updated as an empty or obstacle cell are stored as byte values, unsigned char. Left unchecked, they would rollover once they reached a value of 255. In order to preserve the percentages calculated, when one of the variables reaches 255, the lower of the two variables is set to a low end value, and the new higher value is calculated based on the percentages. For example, if the empty/obstacle percentages are 10%/90%, then the new value for empty could be 5, and the new value for obstacle is 5(0.9/0.1)= 45.

While having a value of 255 for the upper end is fine in a static environment, it poses interesting dilemmas within a dynamic one. When the environment is unchanging, cells updated as empty, remain empty, and cells update as obstacle, remain obstacle. Within a dynamic one, a cell that was once empty could now be targeted with obstacle, and vise versa. How quickly should a cell switch from one to the other? If a cell has only
been updated with empty, and say it has a value of 200, it would take 200 hits of obstacle just to bring the to 50%/50%. Clearly, the higher the value of the rollover variable, the longer the delay from switching from one type of cell to the other. While a smaller value would cause a faster switch, it may also cause spurious events. By placing the values within a text file, the variables can be tuned for the environment.

**Increments**

Initially, the increments for the empty and obstacle were both set to one.

However, sensory overlap, which is caused by placing sensor’s with wide field-of-views in close physical spacing, causes intersections, as seen in **Figure 5-8**.

With this overlap, one cell can have it’s obstacle variable increment once, while one, two, three, or more other sensors increment it’s empty variable. As such, only large obstacles that engage multiple sensors will be detected. To counter this, the increments for empty and obstacle can be individually set. By setting the obstacle increment higher than the empty, the effects of overlap can be countered and smaller obstacle will be detected. This also enhances the sensitivity to dynamic elements. The empty increment can be seen as a decay rate for detecting dynamic obstacles, the higher the value, the faster the decay rate for dynamic objects. However, as with rollover, the increased response may cause spurious events to happen, which is why they are also placed in a text file to be tuned to the environment.
Numerical Example

This last section will step through an numerical example to help clarify the techniques used. Starting from the sensor, we’ll step through the entire process, and
discuss the methodology used.

Before beginning to update the LGM, the grades of membership ( $\mu_{\text{obs}}(\{A_o\})$, $\mu_{\text{obs}}(\{A_f\})$, $\mu_{\text{emp}}(\{A_o, A_N\})$, $\mu_{\text{emp}}(\{A_f\})$, $\mu_{\text{emp}}(\{A_f, A_N\})$ ) for the fuzzy sets OBS and EMP are determined experimentally, a suggestion of how is described in chapter 4. These determine the ‘confidence’ one has with the sensor and it’s interaction with the environment. A good sensor will have few, if any, false readings. The noise inherent with the sensor will not generate either misinterpretations of range to an obstacle (either farther or shorter that actual), will not generate an obstacle which does not exist (free space detected as an obstacle), or miss an obstacle (obstacle classified as free space).

When a sensor returns a range, the algorithm translates this range into a polygon representation of the sensor’s field-of-view. For sensor’s with a narrow field-of-view, the range is converted into a line, with the start point set to the sensor’s minimum value, and the end point set to the sensor’s maximum value. For any other sensor, the range is converted into a three or four sided polygon, with the range determining the farthest
points, and the beam width determining the expanse if the polygon. These set of points are transformed from the sensor coordinate system, to the vehicle coordinate system, and finally into the LGM coordinate system using a series of 3D transformations specified in the sensor’s configuration file.

Once the points are in the LGM coordinate system, they are used to fill the grid with information about obstacles and free space. For this example, say the membership
values are \( \mu_{\text{obs}}(\{A_0\}), \mu_{\text{obs}}(\{A_f\}), \mu_{\text{obs}}(\{A_0, A_N\}), \mu_{\text{emp}}(\{A_0\}), \mu_{\text{emp}}(\{A_f\}), \mu_{\text{emp}}(\{A_f, A_N\}) \) = (0.9, 0, 0.2, 0, 1, 0) such that the sensor’s noise affects the true range reading, but it accurately reports free space. The empty and obstacle increments are set to one, and note that the probabilities for the cells are both initialized to 50%. On the first pass, with OBS being presented to the cell, use equation 4.16 and the Markov model evaluates to:

\[
\begin{align*}
 n_{\text{emp}} &= 0; n_{\infty} = 1 \\
 P(x_{n+1} = \{A_f\}) &= \max \left\{ \frac{P(x_n = \{A_f\})}{\mu_{\text{obs}}(\{A_f\}) \times \min \left( \frac{N_{\text{obs}}}{n}, m(n) \right)} \right\} \\
 P(x_{n+1} = \{A_f\}) &= \max \left\{ \frac{0.5}{0.9 \times \min \left( \frac{1}{1-0.01}, 0 \times \min \left( \frac{0}{1-0.01} \right) \right)} \right\} = 0.5 \\
 P(x_{n+1} = \{A_f\}) &= 1.0 - P(x_{n+1} = \{A_0\}) = 1.0 - 0.8 = 0.5
\end{align*}
\]

So, after the first step, the probabilities remain unchanged. This is due to the weighting function \( m(n) \) which is defined as follows:

\[
m(n) = \begin{cases} 
  n < 10 & \frac{n}{100.0} \\
  n \geq 10 & 1.0
\end{cases} \tag{5-11}
\]

while \( n \) is less than 10, the value returned is very small, once \( n \) is equal to or greater than ten, 1.0 is returned. The weighting function is used to temper the initial values of a cell. Without it, a single first value would cause the probabilities to reach their maximum. For this example, if after initialization the cell is incremented with an OBS, the equation \( n_{\text{obs}}/n \) would evaluate to \( 1/1 \) and set the probability to 0.9 causing the cell to be reported as
an obstacle. With the weighting function, the cell must be updated at least ten times before the probabilities reflect the actual state of the cell.

Knowing this, let’s step farther along such that \( P(x_n = \{A_r\}) = 0.11 \) and \( P(x_n = \{A_o\}) = 0.89 \) and update with EMP with equation 4.17:

\[
\begin{align*}
n_{\text{emp}} &= 2; n_o = 11 \\
P(x_{n+1} = \{A_r\} | x_n = \{A_r\}) &= \max \left\{ \frac{P(x_n = \{A_r\}) \times \min \left( \frac{n_{\text{emp}}}{n}, m(n) \right)}{\mu_{\text{emp}}(\{A_r\}) \times \min \left( \frac{n_{\text{emp}}}{n}, m(n) \right)} \right\} \\
&= \max \left\{ 0.11 \times \min \left( \frac{2}{13}, 1.0 \right) \right\} = 0.15
\end{align*}
\]

\( P(x_{n+1} = \{A_o\}) = 1.0 - P(x_{n+1} = \{A_r\}) = 1.0 - 0.15 = 0.85 \)

Now update with OBS with equation 4.16:

\[
\begin{align*}
n_{\text{obs}} &= 2; n_o = 12 \\
P(x_{n+1} = \{A_o\} | x_n = \{A_o\}) &= \max \left\{ \frac{P(x_n = \{A_o\}) \times \min \left( \frac{n_{\text{obs}}}{n}, m(n) \right)}{\mu_{\text{obs}}(\{A_o\}) \times \min \left( \frac{n_{\text{obs}}}{n}, m(n) \right)} \right\} \\
&= \max \left\{ 0.85 \times \min \left( \frac{12}{14}, 1.0 \right) \right\} = 0.85
\end{align*}
\]

\( P(x_{n+1} = \{A_r\}) = 1.0 - P(x_{n+1} = \{A_o\}) = 1.0 - 0.85 = 0.15 \)
CHAPTER 6
JAUGS

Communication is important when trying to express information, and speaking the same ‘language’ is key. The Joint Architecture for Unmanned Ground Systems (JAUGS) tries to establish this ‘language’ for use with unmanned ground systems within the military community. This starts from the final user requesting a system (the domain model) all the way down to the interface between two computer subsystems (the reference architecture). This chapter describes JAUGS, and how this research contributes to the development of the Object Detection System and it’s messaging scheme.

Overview

The Joint Architecture for Unmanned Ground Systems (JAUGS) is the architecture defined for use in the research, development, and acquisition of Department of Defense (DoD) unmanned ground vehicle systems (UGVS). JAUGS is defined in three separate volumes. Volume I is the JAUGS Domain Model (DM). Volume II is the JAUGS Reference Architecture (RA). Volume III is the JAUGS Configuration Management Plan (CMP).

Volume I: Domain Model

The JAUGS domain model is the model of the operational requirements, both known and potential, that may be requested by a Combat Developer (CBTDEV). The CBTDEV is the final user of the system to be acquired. The Material Developer
(MATDEV) is the organization that specifies the system to be acquired and contracts with the prime contractor to have the system built. The MATDEV uses the domain model to model the operational requirements. The tech base community uses the domain model to understand future capabilities that may be desired by the CBTDEV. The domain model is a tool used by the MATDEV and the tech base to understand the needs of the user.

**Volume II: Reference Architecture Specification**

The JAUGS reference architecture is the performance specification to implement the JAUGS domain model. While the domain model is written in the language of the CBTDEV, the reference architecture is written in the language of scientists and engineers. Each message and service (capability) defined in the reference architecture can be directly traced to a domain model requirement. The JAUGS reference architecture defines a common set of message and a message passing system to support the capabilities defined in the domain model.

It is the intent of JAUGS that the domain model be used to model all requirements, known and anticipated, of UGV systems. The reference architecture will only implement those requirements in the domain model that have gone through a technical evaluation process. The evaluation could be performed by the tech base, academia, or by industry. Therefore, all messages defined in the reference architecture can be traced back to a domain model service, but all domain model services may not trace forward to an reference architecture message.

The domain model is used to model the CBTDEV’s requirements to provide a framework for research, development, and acquisition. The reference architecture is used to specify the MATDEV’s or the tech base’s requirements to the system prime contractor.
The reference architecture is the standard against which compliance by the prime contractor shall be assessed. **Figure 6-1** shows the intended audiences for the domain model and reference architecture documents.

**Volume III: Configuration Management Plan**

The JAUGS Configuration Management Plan is an independent document that defines the process to update the JAUGS domain model and reference architecture. The CMP establishes the charter and organization for the JAUGS Configuration Control Board (CCB). The JAUGS CCB is the governing body that is authorized to modify all JAUGS documentation. The JAUGS configuration management plan also defines the process to validate compliance with the reference architecture.

**Problem Statement**

In order for the different UGVSSs to communicate with each other (interoperability) or to potentially share subsystems (intraoperability), a common set of functional modules must be defined along with the interfaces to those modules. A set of functional modules and their interfaces is called an architecture.

The Department of Defense and the Department of the Army have both released their technical architectures. The technical architectures mandate a set of information
technology standards and interfaces for use in all military systems. These standards address information transport, information modeling, information processing, human-computer interfaces, and information system security.

Several architectures are currently in use on existing UGVs prototypes. However, none is in use across all UGVs. The military robotics community must establish a common architectural framework that is applicable to all classes of robotic vehicles. This framework must incorporate all militarily relevant features, including autonomy and multiple vehicle control, without limiting technological enhancements. As the first UGVs emerge, a common architecture must be inserted early in the development phase to ensure UGVs interoperability. JAUGS is designed to be that common architecture.

**JAUGS Purpose**

The purpose of JAUGS is to support the acquisition of UGVs by providing a mechanism for the reduction of life cycle costs and by providing a framework for technology insertion. The scope of JAUGS is all military unmanned ground systems. The variance within this domain is quite broad. Systems range from small, man-portable vehicles to tanks. The sophistication of these systems range from basic tele-operation (Teleop) to complex semi-autonomous and autonomous operations. JAUGS defines a set of reusable components and their interfaces. Reusable components reduce the maintenance costs of a system as well as the development costs of any follow-on system. It also allows a component developed for one UGV to be ported to another.

Technology insertion is achievable when the architecture is designed to be scalable. JAUGS defines components for all classifications of UGVs from remote control to autonomous as well as numerous types of missions.
Technical Constraints

The technical constraints on JAUGS are self-imposed to ensure that the architecture is applicable to the entire domain of UGVs. To effectively support all UGV systems, JAUGS must meet the following constraints:

- Vehicle platform independence
- Mission independence/isolation
- Computer hardware independence
- Technology independence

Vehicle platform independence

Analysis has shown that UGVs will be based on a variety of vehicles. Those vehicles to date have ranged from small all-terrain vehicles (ATV) to armored tanks to custom vehicles. In order for JAUGS compliant components to be interoperable, no assumptions about the underlying vehicle should be made.

Mission independence/isolation

Analysis has also shown that the current inventory of UGVS prototypes perform a wide variety of missions. Furthermore, a large set of potential missions have been identified that have not been demonstrated on any UGVS prototype. Since the growth in the number and types of missions has the potential to be large, JAUGS must isolate mission specific functions from the generic UGVS functions. This allows the architecture to be independent of any specific mission or set of missions.

Computer hardware independence

The growth in the computer industry has been enormous over the past 20 years and there are no indications that the growth will slow down. Future UGVSs must be able
to capitalize on commercial advancements in computing and sensor technology. The issue for UGVs is two-fold. First, a single UGV must be able to evolve over its approximate 20 year life cycle to accommodate new missions and greater degrees of autonomy. An architecture that imposes a specific hardware implementation reduces that opportunity to take advantage of future advancements. Second, each UGV prime contractor should have the flexibility to design a computer hardware architecture that meets that particular system’s requirements. Computer hardware that is appropriate for one UGV may not be appropriate for another. JAUGS must maintain computer hardware independence in order to be applicable to all UGVs.

**Technology independence**

The final technical constraint is technology independence. This constraint is similar to computer hardware independence but focuses more on the techniques or approaches instead of the commercial hardware. For example, a UGV may use a visual system for numerous purposes. However, there are other systems that may provide similar benefits. Active LADAR (Laser raDAR) is one possibility. The point is that there are multiple solutions to a problem. An architecture that is built around a particular technology solution may eliminate a superior solution.

**Domain Model**

A *domain* is a set of systems that perform similar missions. For example, the F-15 Eagle, the Sopwith Camel, and the Mitsubishi Zero all belong to the domain of fighter aircraft. A *model* is an abstraction of something developed for the purpose of understanding before building. Therefore, a *domain model* (DM) is an abstraction of the systems within a domain built for the purpose of understanding its requirements,
information, and process. For JAUGS, the domain is defined as military UGVs. Thus, the JAUGS domain model is a model of the requirements, information, and processes performed on military UGV systems.

The JAUGS domain model will model the capabilities for all UGVs, both near-term and far-term, with the DoD. Since the domain model focuses on the capabilities, it becomes a tool for defining a system’s requirements. The JAUGS domain model is a tool used by the material developer to model the operational requirements developed by the combat developer. The JAUGS domain model also defines far-term capabilities that may never have been demonstrated on prototype UGVs. By working with the combat developer, the domain model becomes a “road map” used by the material developer and the tech base to focus research and development efforts to support potential future requirements.

For the JAUGS domain model to be effective in modeling operational requirements, it must be written in the language of the combat developer. Engineering and technology terminology should be eliminated unless it is relevant to the military domain. Any terminology that does not have the same interpretation to all four branches of the military should also be avoided.

**Reference Architecture**

The reference architecture is not a design, but rather a framework for a design. Given the broad scope, there are sections of the specification that may not apply to every system. For instance, in the future, this specification will likely define the interface to an obstacle detection component. If a system does not use obstacle detection, then that portion of the reference architecture does not apply to it. The benefit of this specification,
though, is that if the system were to add obstacle detection, the interface would be known. Furthermore, the potential would exists to incorporate an existing obstacle detection component.

In its simplest view, JAUGS is an architecture for distributed computing systems. Computing resources reside within the system, and these resources communicate. The communication paths, the types of communication channels, and the tactical implication governing communication vary greatly.

The reference architecture specifies methods of communication and the data formats used to communicate between computing resources. The reference architecture forms the basis for a design, but does not eliminate system engineering.

**System Topology and Nomenclature**

The definition of the JAUGS system topology is shown in Figure 6-2. A System is composed of one or more Operational Subsystems. Each operational subsystem is composed of one or more nodes. A node contains either a single processor or a tightly coupled set of processors. There is no exact definition of tightly coupled, but processors that have access to a common memory area and/or share a common backplane bus would normally be considered tightly coupled. Computer resources linked by Ethernet, 1553, CAN bus, USB, FDDI, RS232, etc., are not tightly coupled and form separate nodes. A node is composed of one or more components. The
component is the lowest level abstraction in the JAUGS reference architecture. A component is a cohesive functional unit, usually implemented in software that provides a predefined set of services.

Definition of components, along with their interfaces, are the primary areas specified by this architecture. Node-to-node communication is also specified, which combined with the component interfaces forms the basis for node integration within the architecture.

**Architecture Style**

JAUGS is based on message passing between components, which reside on various nodes and operational subsystems within the system. The system configuration contains the components that exist within the system, the messages they support, and their locations (operational subsystems, node). This version of the reference architecture assumes that the configuration of a system is known a priori. This architecture specifies a “Data Upon Request” style. Data upon request means that data flow between nodes is not specified in the system configuration. Data flow is accomplished either by a query-inform message set, an event setup-notification, or by establishing a periodic flow of data.

**Messages**

JAUGS is a component-based, message-passing architecture. Component interfaces are described by the set of messages a component can send and receive. Messages are also defined for specific interactions between the messaging system of each node.

Internodal communication is done through a Message Routing Service (MRS) that exists on each node. Every JAUGS node must have an MRS. When components on
different node communicate, the messages are handled by the MRS on each node. System configuration information is contained within each MRS. The MRS consults the system configuration information to route the message to the appropriate node, and/or operational subsystem.

Within a node, the method of implementing the message interface is not currently specified. Intranodal messages can be implemented as procedure or function call, as long as the parameters of the function/procedure correspond to the message data specification.

Messages are queued. The queue mechanism within a node is not specified. Messages coming into a node are received by the MRS on that node, and are delivered to the appropriate component.

**Object Detection Component**

One element necessary for an autonomous vehicle to exhibit true autonomy is the ability to detect and avoid unknown objects encountered on the vehicle’s path. It is the responsibility of the Object Detection Component (ODC) to perform this task. Many sensors exist that can accomplish this undertaking, and each exhibits strengths and weaknesses. As per the JAUGS doctrine, there are no sensors directly specified, however the outgoing message sent when quarried for obstacles remains generalized to expressed the objects detected.

**Description**

The object detection component maintains the location of all objects detected by the various sensors in use. In order to provide sensor independence, the sensor’s physical and uncertainty characteristics are used to homogenize all sensors, as described in the previous chapters.
Figure 6-3 shows the ODC without the benefit of a position system to provide a constant update of the vehicle’s position. The output of the sensors are fed into the ODC, which uses the information to update the local grid map (LGM). However, since the orientation of the vehicle is unknown, only the portion of the LGM which is within the field-of-view of the sensors is used, and the only objects known, are with ones that are within the sensors range.

However, add position input, as show in Figure 6-4, and the capabilities of the component greatly increase. Similar to the previous example, the ODC uses the sensors output to update the LGM, but with position input, the orientation and the movement of the vehicle can be incorporated into the grid. This enhances the grid in two ways, multiple update on a object, and a history of objects detected.

The first enhancement permits multiple updates on an object as the vehicle approaches. Say an object is detected a distance away. As the vehicle approaches this object, the object approaches the vehicle in the LGM perspective. When the new range is returned from the sensors,
the object’s corresponding grid cells are updated as *OBSTACLE*. This improves the
detection and sizing of the object, since the grid cells associated with the object are
updated numerous times.

While the first enhancement improves object detection, the second maintains the
objects when they exit from the sensors field-of-view. As described above, without
position, the only objects detected are the ones that the sensors can ‘see’. When the
orientation and position of the vehicle are known, the objects can be located correctly
within the grid. They no longer disappear from the grid when the vehicle passes them.

The addition of the position component also allows an Object Mapping
Component (OMC). In the future implementation of the OMC, it can take input from an
external database, and translate it into the internal representation of the global map.
However, the most pertinent input is from the ODC. The obstacles that ‘fall off’ the edge
of the grid as the vehicle moves are converted into polygons. These obstacle polygons,
along with traversed-space polygons (the entire edge of the LGM) are presented to the
OMC. The OMC takes this information and maintains a Global Polygon Map (GPM) of
traversed-space and obstacle polygons. In can also take, as input, polygons from other
OMCs. In this way, multiple vehicles can explore a large area, and a total overall map
can be created from the pieces generated from each vehicle. To complete the cycle,
obstacle polygons from the mapping system can be used by the object detection system to
seed the LGM with obstacles as the vehicle navigates in the environment.
Messaging

Since JAUGS is a message passing architecture, appropriate messages must be defined for input to and output from the ODC. The messages (both input and output) must be able to handle any currently perceived usage, and any potential future usage.

The ODC has only one true output message, since the sole recipient of the edge polygons is the OMC. The message contains the series of obstacles represented by range, r, heading from the vehicle, Θ (bearing), the inclination from the vehicle, Φ (elevation), and the calculated probability of the object. This is essentially a spherical coordinate system centered on the vehicle. The objects are presented in the vehicle coordinate system (VCS), since this accommodates an ODC without and with a position system. Without a position component, the objects are, by default, in the vehicle coordinate system. By knowing the true positions of the objects and the orientation of the vehicle, the objects can be translated into the VCS. Also possible additions include a class and type that will be used to classify the object. These are place holders for future work on identifying the object. For example, an object can be classified as a tree of type pine. The final item is the time stamp of when the data was obtained from the grid, see Table 6-1 for the layout.

The main input message is querying for object data. The other inputs are either defined by the other systems, for the case of the position component, or the OMC, or are hardware dependent, in the case of the interface to the sensors. Currently, the only parameter necessary is the range at which to scan for obstacles, essentially the radius from the center outward. A value less than the maximum range will result in a bounding box at that range around the vehicle. Objects within the bounding box are returned, objects
outside are ignored. This allows the calling program to obtain pertinent information without having the ODC process the entire grid. A value that is less than zero, or equal to or greater than the maximum range will return all detected objects within the grid.
Table 6-1: Current Object Detection Component message set.

<table>
<thead>
<tr>
<th>Field #</th>
<th>Name</th>
<th>Type</th>
<th>Units</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Presence Vector</td>
<td>Byte</td>
<td>N/A</td>
<td>Selection of fields to include</td>
</tr>
<tr>
<td>2</td>
<td>Time</td>
<td>Unsigned Integer</td>
<td>N/A</td>
<td>Bits 0-9: milliseconds, range 0.999</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bits 10-15: Seconds, range 0.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bits 16-21: Minutes, range 0.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bits 22-26: Hour, range 0.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Bits 27-31: Day, range 1.31</td>
</tr>
<tr>
<td>3</td>
<td>Number of Points</td>
<td>Byte</td>
<td>N/A</td>
<td>1..255</td>
</tr>
<tr>
<td>4</td>
<td>Range</td>
<td>Integer</td>
<td>Meters</td>
<td>Scaled 0.0..10,000.0</td>
</tr>
<tr>
<td>5</td>
<td>Range Error</td>
<td>Integer</td>
<td>Meters</td>
<td>Scaled 0.0..1,000.0</td>
</tr>
<tr>
<td>6</td>
<td>Bearing</td>
<td>Integer</td>
<td>Radians</td>
<td>Scaled -(\pi)..(\pi)</td>
</tr>
<tr>
<td>7</td>
<td>Bearing Error</td>
<td>Integer</td>
<td>Radians</td>
<td>Scaled 0..(\pi)</td>
</tr>
<tr>
<td>8</td>
<td>Elevation</td>
<td>Integer</td>
<td>Radians</td>
<td>Scaled -(\pi)..(\pi)</td>
</tr>
<tr>
<td>9</td>
<td>Elevation Error</td>
<td>Integer</td>
<td>Radians</td>
<td>Scaled 0..(\pi)</td>
</tr>
<tr>
<td>10</td>
<td>Confidence</td>
<td>Byte</td>
<td>N/A</td>
<td>0-Low, 255-Highest</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td>Repeat of Fields 4 through 10 for each point.</td>
</tr>
</tbody>
</table>
CHAPTER 7
RESULTS AND CONCLUSIONS

What is a theory without presenting results that authenticate it. This chapter is the culmination of all the work presented in the previous chapters. A graphical user interface is used to visually present the Local Grid Map (LGM). The results section presents the actual hardware in use on the vehicle, an explanation of the graphical display, how the lack of a position component and having a position component effects the outcome, the accuracy of the component, the consequences the tunable parameters have on the map generation, and finally an error that the environment can generate with the LGM.

Hardware/Software

The Object Detection Component (ODC) is run on a WinSystems EBCTX-Plus, which is a small, high-performance, embeddable computer system. The two sensors that are integrated, and then fused together are an array of 16 SONAR transducers and a Laser Range Scanner (LRS). They are mounted on the front of the vehicle, and are shown in Figure 7-1.

Computer

The single-board computer integrates a number of popular I/O options including VGA, 10/100 Mbps Ethernet, Solid-State Disk,
and 48 bi-directional TTL digital I/O. Four PC compatible serial ports are standard, as are the floppy, hard disk, and parallel printer interfaces. The EBCTX-Plus is populated with an AMD-K6II-E-333MHz Socket 7 CPU with 256 Mbytes of DIMM memory. A full 16-bit PC/104 expansion bus is provided for further expansion to an entire industry of add-on peripherals. The operating system installed is Red Hat Linux 7.1, and the code is written in the C programming language.

**Sonar**

The TRC SonaRanger provides a mobile robot with the means for sensing its environment. Each interface board can obtain range information from up to eight sonar transducers, and binary (presence/absence) object detection is available via eight reflective infrared transmitter/receiver channels. Up to three interface boards can be connected to a single controller board, giving a total possible sensor configuration of 24 ultrasonic and 24 infrared channels.

The on-board control software provides a simple way for a host computer to control the SonaRanger. Several commands are available to configure sensor firing order, set timeout distances for each sensor, name partners, and read data from the system. The Subsystem uses RS-232 ASCII characters for communication. The SonaRanger is monitored and controlled by a host computer via its RS-232 serial port. The interface protocol uses a byte-count header designed to support a future network implementation.

The instrument grade ultrasonic transducers are intended for operation in air. The typical beam pattern at 50 KHz is shown in **Figure 7-2**. From the figure it can be seen that at 15° from the centerline of the sonar, in both directions the response drops off
drastically, thus producing a 30° cone. The range is accurate from 6 inches to 35 feet, with a resolution of ±1%.

**Laser**

The SICK Laser Measurement Scanner 200 (LMS) [SIC01] is an optical sensor that scans its surroundings with infrared laser beams two-dimensionally similar to a laser radar. The sensor operates on the principle of reflex light time measurement. It emits very short light pulses. At the same time a timer/counter is initialized. If the light encounters an object, it is reflected back to the sensor. From the time between sending and receiving, the sensor calculates its distance from the object. In the sensor there is also a uniformly rotating mirror, that deflects the light pulses so that they sweep a semicircular arc. By determining the mirror angle, the LMS detects in which direction the object is located.

An RS232 interface is used to communicate with the sensor. The 180° field-of-view can be broken down into 0.5° or 1° increments. Range can be set to either 8 meters or 80 meters with a resolution of 1 cm or 10 cm respectively.

**Description of Display**

Shown in Figure 7-3 is the graphical display of the LGM. JAVA is used as the programming language, and is written to handle a series of files (Series), or continually reopen and redisplay the same file (Realtime). In Series mode, the files are displayed as a movie, however the user is given the opportunity to stop, and increment forward or backward within the ‘scenes’. In Realtime mode, the specified file is continually checked.
Figure 7-3  Graphical representation of local grid map. (a) - Shown without position information. (b) - Shown with position information.
for a change in it’s date/time. Once detected, it is reopen and redisplayed, allowing the
user to watch the LGM in sudo-realtime. In either mode, the user is able to check
distances by clicking and dragging the mouse from one point to another. Using a
conversion between pixels and the grid resolution, the distance between the two points
can be calculated.

The left-hand side is for user input. The user can input a file to display in the
upper section. The middle section allows the user to select the file type (usually detected
automatically), shows the current file visible (for Series), and the RMS, an error associate
with the position data. The higher the RMS, the worse the error in position. Usually
remaining under one, it can change with intermittent GPS. The bottom panel contains
buttons which allow the user to Start and Stop displaying the LGM, along with pausing,
resuming the movie, and stepping forward or backward through a sequence of images.

The right-hand side displays the LGM. The file used for input to the graphical
LGM is written twice a second and contains the number of rows and columns along with
the percentage associate with an obstacle, the heading of the vehicle (expressed in
degrees), the grid resolution in meters, and the RMS value from the position component.
The graphical display reads in these values along with the entire grid map (row by col)
which is in integer values of percentage. Any value equal to or over obstacle percentage
is drawn as a dark red (obstacle). Any value equal to or less than (100 - obstacle
percentage) is drawn as a dark green (free space). Any values between these are
considered unknown. The unknown section is divided into thirds. The upper third,
which is tending toward obstacle, is drawn as a lighter red. The lower third, which is
tending toward free space, is drawn as lighter green. The center third, that is really still unknown, is drawn as gray.

The center of the grid is marked with a ‘+’ and an arrow-head is formed from the center to the front of the vehicle. The arrow-head is drawn in the orientation of the vehicle to signify the direction of movement. Figure 7-3(a) shows the display without position information. The ‘vehicle’ is drawn pointing north, and the only portion of the LGM being used is that which is in the field-of-view of the sensors. Figure 7-3(b) shows the display with position information. The entire grid is used, and the ‘vehicle’ is drawn based in the orientation of the actual vehicle.

A typical startup for the LGM is shown in Figure 7-4. Each ‘scene’ represents approximately ½ second in time. Starting from all unknown Figure 7-4(1), the object is detected, and the center section is filled first with free space (due to the large amount of overlap in the front of the vehicle between the two sensors) Figure 7-4(2). In Figure 7-4(3)-(5) the fringes are filled, until in Figure 7-4(6) the entire field-of-view of the sensors is accounted for. Notice the information behind the tree remains unknown, since no sensor has penetrated that area yet.

**Results**

The following sections describe the outcomes obtain from this research. First presented are the results from the LGM without a position component. The next section covers the same ground (literally) but with all the benefits of using a position component. Following that, the accuracy of the output is address, and how each of the sensors contributes to generating the map. Finally, the tunable parameters are manipulated and the various outcomes discussed. For all the results presented, the vehicle is traveling
approximately 5 mph, unless otherwise noted. The grid map range is set to 7 meters, the resolution set to 0.2 meters, and the percentage for free space/obstacle set to 80%. For the tunable parameters, rollover is set to 50, the empty and obstacle increments are set to one and ten respectively, unless otherwise noted.

**Local Grid Map Without Position**

The first sequence shows the vehicle driven between two groups of nearly parallel sets of trees, **Figure 7-5** (see also np_22trees.avi [4850 KB]). In the first image,
Figure 7-5(a), the vehicle is stationary and is detecting the trees located off to each side. As the vehicle is driven through the first set, Figure 7-5(b)-(c), the trees create a streak as the values from their old position decay, and their newer positions are detected by the sensors. Again, as the vehicle approaches the second set, Figure 7-5(d)-(f), the first tree detected, the one on the left, creates a streak, and then the second streak appears, and moves off on the right. While the trees are detected during the movement, their size is hardly represented by the objects detected.

Sure large trees are detected, how well are smaller trees detected? Figure 7-6 (see also np_4trees.avi [2386 KB]) show a trapezoid arrangement of small palm trees. Again the vehicle is stationary and the sensors are detecting the two smaller trees on each side, Figure 7-6(a). The vehicle begins it’s movement, and the trees start to streak downward in Figure 7-6(b), the shrubbery on the left-hand side is also detected. In Figure 7-6(c), as the closer left-hand tree exit from the image, the two further trees have been detected, and more of the shrub is displayed. The movement continues in Figure 7-6(d)-(e), where the last two trees exit. Like the previous example, the objects are detected during the movement. Comparatively these detected objects are smaller than the previously detected objects in Figure 7-5, and therefore represent a smaller physical objects.

The final sequence is an example of a wall with columns, Figure 7-7 (see also np_meWall.avi [3042 KB]). Observe the four white columns protruding from the side of the wall (the fifth at the far end is obstructed by shrubbery). Similar to the first two examples, the vehicle is stationary and the first column is visible in the map, Figure 7-7(a). As the vehicle progresses along, the remaining columns and the shrubbery at the
Figure 7-5  Two sets of trees without position component
Figure 7-6  Trapezoid arrangement of palm trees without position component
Figure 7-7  Wall with columns without position component
end of the run are detected, Figure 7-7(b)-(e). While difficult to visualize, the columns can be barely discerned by the horizontal ‘noise’ that appears at periodic increments.

**Local Grid Map With Position**

Starting again with the two groups of nearly parallel sets of trees, Figure 7-8 (see also p_22trees.avi [5154 KB]), the vehicle, starting from a stationary position, Figure 7-8(a), proceeds west through the first set of trees, Figure 7-8(b)-(c), then heads south, Figure 7-8(d), and through the second set of trees, Figure 7-8(e)-(f). Remember, since position information is available, the vehicle is able to rotate within the center cell of the map. This causes the objects to be placed in the map in their correct orientation in the North-East orientation of the LGM. In this example, the first set of trees is parallel with North-South Drive (that runs North-South), and therefore the trees are orientated along the North-South line. Similarly, the second set of trees is perpendicular to the first, and are orientated East-West. Unlike the example without position information, there is no streaking, since the objects move in the map in relation to the movement of the vehicle. Therefore, the objects will be better defined, since the sensors will detect them a multitude of time. This also improves the clarity of free space, since there is no decay associated with the previous location of the object.

The second image, the trapezoid of palm trees, is show in Figure 7-9 (see also p_4trees.avi [2706 KB]). These sequences also show a dramatic improvement over the previous set without position information. Starting from a stationary position, Figure 7-9(a), the vehicle travels south-southeast through the trees, Figure 7-9(b)-(e). Again, the trees, shrubbery, and free space is much better defined then shown previously without
Figure 7-8  Two sets of trees with position component
Figure 7-9 Trapezoid arrangement of palm trees with position component
Figure 7-10  Wall with columns with position component
position information. Similarly to the sequence without position, these trees are mapped smaller in size then in the previous example.

The most dramatic example of the benefits of using the position of the vehicle are show in Figure 7-10 (see also p_meWall.avi [3238 KB]). Same wall with columns, although with this generated map, the columns clearly appear. As the vehicle progresses westward, Figure 7-10(a)-(d), the columns are well defined, as well as the shrubbery at the finish, Figure 7-10(d)-(e).

Accuracy

The maps generated by the LGM using position information appear acceptable, but how accurate are they? Is the information reliable enough to use for object avoidance? How well do they represent the environment being detected? What happens with objects which are occluded? These questions will be answered in this section.

Accurate distances

Are the relative distances of objects within the LGM representative of the actual objects? Knowing the distance in pixels, a true distance can be calculated using the following formula:

$$\text{dist} = \text{pixel distance} \times \frac{1 \text{ box}}{\text{num pixels}} \times \frac{\text{box dist}}{1 \text{ box}}$$  \hspace{1cm} (7-1)

Starting with a simple example, the vehicle is oriented toward the wall and images are taken while the vehicle remains motionless, Figure 7-11. The

Figure 7-11 Distance between pillars while stationary
centerline distance between the pillars is 18 feet. Choosing two points on the graph, a distance of 17.99 feet is obtained. While fine for a static object, the more important question is how well the objects are mapped while the vehicle is in motion. Referring to Figure 7-12, the same pillars are used to check the distances of the objects that the LGM has mapped while in motion. The two distances picked off are 17.98 feet and 17.60 feet for the pillars. Since the resolution of the grid is 0.2 meters (0.65 feet), the values obtained place both distances within the tolerance of the system.

The final two images shown, Figure 7-13 and Figure 7-14, are the now familiar images of the two groups of nearly parallel sets of trees, and the trapezoid set of 4 palm trees. In Figure 7-13, the distances between the two sets of trees are 13.4 feet and 21.2 feet, and the calculated distances are 13.41 feet and 19.81 feet. The first distance well within the tolerance of 0.65 feet. However, the second falls outside the tolerance, this can be
Figure 7-14  Distance between 4 trapezoid palm trees

accounted for by considering the fact that the second tree had lower hanging branches which the sonar picked up, as seen by the elongated representation of the tree. In Figure 7-14, the greatest error in distance is 0.17 feet, again within the tolerance.

Realistic environment representation

The last section answered the question of accuracy with the relative distance of objects within the LGM, but how realistic are the obstacles compared to the actual objects? A tree trunk has limited complexity, so in these examples, the vehicle travels alongside a chain link fence. The fence separates a dormitory from a large open field. At
certain points, there are gated openings in the fence with a back fence spanning the opening, see Figure 7-15. In Figure 7-15(a) the gate is positioned away from the vehicle and is touching the back fence. In the LGM, the opening in the gate can be seen along with the back fence. There is also a “bridge” between the front fence and back fence which corresponds to the gate. The gate is opened toward the vehicle in Figure 7-15(b). Once again, the opening and back fence can be seen along with the open gate. Finally in Figure 7-15(c), the gate is pushed back almost fully against the front fence. Again the opening and back fence can be clearly seen, and where the gate is against the fence, the LGM depicts that portion as thicker than the rest.

**Figure 7-15** Realistic representation of environment. (a) Gate swung away from the vehicle. (b) Gate swung toward the vehicle. (c) Gate folded against the fence.
Occlusion

It has been established that the objects relative distances are accurately positioned and that the objects themselves are realistically portrayed, but what happens when one object occludes another. As the vehicle traverses around the first, is the second detected? In this example, the vehicle is lined up parallel to two trees, where the first blocks the second, see Figure 7-16(a)-(b) (see also occlusion.avi [5958 KB]). The

![Figure 7-16](image1.png)

**Figure 7-16** First tree occluding the second. (a) Trees as seen by vehicle. (b) Second occluded tree. (c) Detection sequence.

sequence of events is shown in Figure 7-16(c). At first the second tree is unknown, but as the vehicle proceeds, the second occluded tree starts to emerge in the LGM. As the vehicle continues on, the second tree becomes as well defined as the first tree.

**Contributions of Sensors**

In this section, two different sequences of images are used to show how each of the sensors contributes to the generation of the grip map and the detection of objects. For
a scene, the LGM is operated with just the sonar, just the laser, and then with both. The images are stored approximately twice a second.

The first figure, Figure 7-17, shows the interaction of the sonar and laser with palm trees and hedges. The sonar, being a imprecise sensor, only detects the palm tree to the left, and the palm trees-hedge assemblage on the right. The laser, being a refined sensor, can detect multiple objects within the scene. When combined, the imprecision of the sonar helps to bled together some of the lasers objects that belong to the same physical object, while the laser’s precision helps to polish the sonar’s gross detection of objects. It this way, the two sensors complement each other.

The second figure, Figure 7-18, shows the interaction of the sonar and laser with shrubbery and a large tree. Because of the location of the sonar and it’s wide field-of view, it detects the foliage of the shrubbery and the trunk of the tree. However, since the sonar lacks resolution, the information is blurred together. Because of the laser’s location and resolution, only low hanging branches and the trunks of the shrubbery and tree are detected. Blur and refinement occurs when both are combined, and a better representation of the objects in the scene emerges.

**Tunable Parameters**

The final section of results discusses the role of the tunable parameters. These include the three which are labeled as tunable; turnover, empInc and obsInc; but also the percentage for determining free space/obstacle. can also be adjusted for the environment. For the tunable parameters, the values for each of the three variables is changed drastically to accentuate the results. The percentage can be adjusted to suit the environment, lowered to increase the detection, or raised to eliminate false obstacle.
Figure 7-17  Contribution of sonar and laser using hedges and palm trees
Figure 7-18  Contribution of sonar and laser using shrubbery and larger tree
Turnover, empty increment and obstacle increment

For *turnover*, the value is set to either 50 or 200. For the obstacle increment, the value is either 1 or 10. Due to sensory overlap, it is not required to adjust the empty increment, nor is it recommend to set the empty higher than the obstacle increment. A dynamic object is used to test the parameters since it is the most difficult to detect. Dynamic environments are similar to using the LGM without position information. If an object is introduced once the LGM has reached steady state, the percentage of the cell must ‘flip’ from it’s current state to the other. This leaves a streak, as seen when using the LGM without position. The slower the LGM responds, the longer it takes to detect an object, and the longer it takes for the values to decay once the object has left the cell. For this test, a person walked from stage-left across the front of the vehicle, stopped directly in front for ten seconds, and then exited stage-right. Shown in the sequence is the first time the object disturbs the LGM, any transition to steady state, the best image of object detection at steady state (when the person was stationary in front of the vehicle), and any transitions exiting from steady state.

*Figure 7-19* (see also tune200_1_1.avi [176 KB], tune200_1_10.avi [186 KB], tune50_1_1.avi [166 KB] and tune50_1_10.avi [196 KB]) shows the permutations of the parameters and the associated outcome. The figure is sectioned off into transitions to steady state (left), steady state (center), and transition from steady state (right). Starting at the top, with the worst-case scenario of the *rollover* set to 200 (slow flip rate) and the *empty* and *obstacle increment* set to one, it can be seen that the objects is never detected. Next, the *obstacle increment* is set to ten. This improves the objects detection, since a streak is seen on the left-hand side, and the object is actually detected in front of the
Figure 7-19  Effects of the tunable parameters on the local grid map vehicle. Setting the *rollover* to 50 (fast flip rate), and the *increments* back to one, improves the flip as compared to the scenario of 200-1-1, since the object disturbs the LGM in transition to steady state, and the object is actually detected. The final row is the best-case scenario of 50-1-10, resulting in a fast flip rate, and a weighted *obstacle increment*. In the sequences, the object disturbs the LGM as it enters the grid, and is detected in transition to steady state. The steady state image produced the largest object, and it was also detected exiting stage-right.

**Cutoff percentage**

The other parameter that can be modified is the cutoff percentage used to move from ‘unknown’ to either free space or obstacle within the LGM. Obviously, when the percentage is lowered the transition from unknown to free space or obstacle happens
Figure 7-20  Results of different cutoff percentages on the local grid map
more quickly. Conversely, raising the percentage takes longer to generate definite free space and obstacle. **Figure 7-20** shows the effect of setting the percentage to 70, 80 and 90%. The vehicle is driven past a tree on the left, and then through two trees. Comparing the three different percentages, there is much more uncertainty around the obstacles as the percentage is increased. The fence is less defined and the trees are smaller with each increased step in percentage.

**Environmental Error**

As discussed in chapter 1, a hill can pose problems with trying to detecting obstacles. When approaching a hill, it can be detected as an obstacle since it can reflect energy back to the range sensor. Clearly the same is true while traveling down a hill and encountering level ground. The hill seen as an obstacle is show in **Figure 7-21** (see also hill.avi [5154 KB]). Starting on the top row and moving across, as the vehicle approaches the hill, a ‘contour obstacle’ appears due to the hill. As the vehicle transitions from the level ground to the hill, this contour obstacle continues to appear in front of the vehicle. Once the vehicle has established itself on the hill, the contour obstacle disappears, and the sensors can once again register true obstacles.

**Conclusions**

The complexity of combining existing simulation work with adapted algorithms was undertaken. The result is a functioning component able to handle a real world environment. It is capable of functioning without a position component, but proves to be extremely beneficial when coupled with position information. However, there are two details to consider; the accuracy of the position information, and the speed of the vehicle verses the size of the objects to detect.
Figure 7-21  Hill seen as an obstacle

The work Takao Okui started in simulation was extended and Bresenham’s Line algorithm and the general polygon scan-conversion algorithm were adapted. This collection of work was combined to form the Object Detection Component that is capable of handing any range sensor. Without position information, the component is capable of limited object detection. By adding position information is was shown that the precision of the detected objects increases. Both in accuracy in the size of the object, and the accuracy in the position of the object. The component is used as groundwork in implementing a JAUGS message for object detection.
The accuracy of the LGM is only as good as the accuracy of the position of the vehicle, as the old adage says, “Garbage in - garbage out”. The map can only be as precise as the information presented. Position errors will cause the map to jump and shift, and inconsistencies with object placement will start to appear.

Finally, the size of the object that can be detected is determined by the speed of the vehicle and two parameters of the sensor: The scanning frequency and maximum range. Keeping all other parameters fixed, a smaller object requires a slower vehicle speed, or a faster scanning frequency, or a farther maximum range, to be detected. Any of these increases the number of times the object is detected by the sensor. The larger the object, the easier it becomes to detect, since it’s being scanned by numerous sensors.
CHAPTER 8
FUTURE WORK

Because of the complexity of extending and adapted Okui’s research, only the first half of the collective structure of the Object Detection Component (ODC) and the Object Mapping Component (OMC), was implemented, see Figure 8-1. However extensive investigation has gone into the development of the OMC. This background research includes various mapping formats that can be employed for both input into the component, and as a format for output. Methods using computational geometry to merge polygons where also evaluated, since the OMC is essentially a polygonal database.

Mapping and Data Structures

Whether you are traveling to a local store, and preforming landmark navigation, or driving to another state and consulting a road atlas, maps play an important part in our everyday lives. Mapping is also important for autonomous agents. With an a-priori map, the robot can plan a safe route from start to goal avoiding all known obstacles. The map can also be updated and supplemented as the robot executes the route. However, in an
unknown environment, a map can play a significant role, allowing the robot to re-plan within a known, mapped environment when faced with an impassible avenue.

Maps can store information in two formats. The first is a 2D representation, or planar map, and is suited for structured indoor environments or when elevation, or slope, is of no concern. The second representation is 3D, or terrain map. This representation is well suited for complex environments such as the unstructured outdoors.

**Two-Dimensional Representation**

**Quadtree**

The term quadtree is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. The decomposition may be into equal parts on each level (i.e. regular polygons and termed a regular decomposition), or it may be governed by the input. The resolution of the decomposition (i.e. the number of times that the decomposition process is applied) may be fixed beforehand, or it may be governed by properties of the input data.

In the tree representation, the root node corresponds to the entire array. Each son of a node represents a quadrant of the region represented by that node. The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary.

**Certainty grid**

The certainty grid representation uses a 2D array to store values associated with the occupancy of the cell. Given a sensor reading, all the values in the cells up to the reading are decremented, since it is certain that there is free space up the obstacle. At the cells corresponding to the sensor reading (usually, some uncertainty is included), the values within the cells are incremented. After a certain threshold, the cell is considered
occupied. This method is not affected by dynamic objects, since multiple reading within the same area will remove the presence of the objects within the database.

**Occupancy grid**

The occupancy grid representation employs a multidimensional (typically 2D or 3D) tessellation of space into cells, where each cell stores a probabilistic estimate of its state. Formally, an occupancy field $O(x)$ is a discrete-state stochastic process defined over a set of continuous spatial coordinates $x=(x_1, x_2, ..., x_n)$, while the occupancy grid is a lattice process, defined over a discrete spatial lattice. The state variable $s(C)$ associated with a cell $C$ of the occupancy grid is defined as a discrete random variable with two states, occupied and empty, denoted OCC and EMP. Consequently, the occupancy grid corresponds to a discrete-state binary random field. Since the cell states are exclusive and exhaustive, $P[s(C)=OCC] + P[s(C)=EMP] = 1$.

To interpret the range data obtained from a given sensing device, Elfes [ELF89] defined a stochastic sensor model by a probability density function of the form $p(r | z)$, that relates the reading $r$, to the true parameter space range value, $z$. This density function is subsequently used in a Bayesian estimation procedure to determine the occupancy grid cell state probabilities. Finally, a deterministic world model can be obtain by using optimal estimators such as the maximum a posteriori (MAP) decision rule to assign a discrete state to the cells, labeling them occupied, empty, or unknown.

**Three-Dimensional Representation**

**Octree**

The region quadtree is easily extended to represent three-dimensional binary region data and the resulting data structure is called a region octree. The octree is based
on the successive subdivision of an object array into octants. If the array does not consist entirely of similar values, it is subdivided into octants, suboctants, and so on. This subdivision process is represented by a tree of degree 8 in which the root node represents the entire object and the leaf nodes correspond to those cubes of the array for which no further subdivision is necessary.

**Voxel**

As 2D space can be divided into unit squares, 3D space can be divided into unit cubes or voxels. These voxels can be built up to form arbitrary volumetric shapes. [ROT89] developed an algorithm that uses dense range data from multiple viewpoints in an environment to refine a 3D voxel-based volumetric model of that static environment. The voxels are assigned three possible values: Void, for empty voxels (representing an open piece of space); Full, for occupied voxels; and Unknown, for voxels for which no meaningful information has yet been obtained.

The desired representation of the environment for Triumala, Schunck and Jain [TIR95], is in the form of a finite-resolution 3D grid of voxels. Each voxel within the grid is assigned a binary value corresponding to its occupancy state. An approach is presented for multi-sensory depth information assimilation based on the Dempster-Shafer theory for evidential reasoning. This approach provides a mechanism to explicitly model ignorance which is desirable when dealing with an unknown environment.

**Cartesian elevation map**

A Cartesian Elevation Map is the simplest 3D map method. It consists of a two dimensional array, the size determined by the amount of area to cover and the resolution of each sector. The value at each cell contains the height at that sector. Based on the
sensor used, there are sometimes two maps that are generated. The first map is
sometimes called the Local Elevation Map, and is generated directly from the sensor.
Because it is developed from the sensor, it is usually fixed in size and resolution. This
local map is then integrated into a larger, dynamic, global map.

**Triangulation**

Triangular meshes have been recognized as attractive tools for a wide variety of
applications because of their simplicity and flexibility. Recently, the use of these meshes
has become more common in robotics-related fields because of their efficiency in
representing large sets of scattered points and complex objects. The triangulation,
denoted by M, needs to satisfy the following requirements:

1. M should be topologically and geometrically correct (i.e., triangles should not
   intersect).

2. The quality of M should be as high as possible (i.e., M should contain as few
   badly shaped triangles as possible.

3. The boundary nodes of M should be positioned on the model’s edges and faces.

4. M should be boundary conforming (i.e., triangles should not intersect the object’s
   boundary).

Garcia [GAR94] presents the application of a geometrical modeling technique to
Computer Vision in order to reconstruct smooth surfaces from arbitrary triangulations of
scattered 3D points. These points are considered to be noisy as a result of a sensory
acquisition process. The reconstruction problem is transformed into one of surface
approximation over arbitrary triangular meshes. The reconstructed surface is composed
of a collection of triangular patches. Since these patches are parametric functions,
arbitrary topologies of any genus can be represented.
A fast generation of 3D triangular meshes from range images is described by Garcia [GAR95] that is an efficient technique avoiding optimization procedures. The proposed method has two stages. First, the vertices of the mesh are computed through adaptive randomized sampling of the range image based on curvature estimations. Then, the mesh is generated by triangulating the sampled vertices through an efficient 2½ D Delaunay algorithm. The sampling process concentrates points in areas of large curvature and tends to preserve surface and orientation discontinuities.

**Digital elevation model**

Digital Elevation Model (DEM) data consist of an array of regularly spaced elevations. The United States Geological Survey (USGS) data contain 7.5-minute, 15-minute (Alaska only), and 1-degree units. The USGS has used four production methods to collect DEM data. Currently, interpolation from vectors or digital line graph (DLG) hypsographic and hydrographic data, is used for 7.5-minute DEMs and other series DEMs. All DEM data are similar in logical data structure and are ordered from south to north in profiles that are ordered from west to east. However, they differ in geographic reference systems and sampling intervals. The 7.5-minute data set is the only one considered because of the grid spacing.

DEM data in 7.5-minute units consists of regular arrays of elevation collected on the North American Datum of 1927 (NAD27) or NAD83 horizontal datum. These data are stored as profiles with a Universal Transverse Mercator (UTM), 10- or 30-meter grid spacing, along and between each profile. The accuracy of DEM data depends on the source and resolution of the data samples. DEM data accuracy is derived by comparing linear interpolation elevations in the DEM with corresponding map location elevations.
and computing the statistical standard deviation or root-mean-square error (RMSE). The RMSE is used to describe the DEM accuracy. For 7.5-minute DEM's derived from photogrammetric source, 90 percent have a vertical accuracy of 7-meter RMSE or better and 10 percent are in the 8- to 15-meter range. For 7.5- and 15-minute DEM's derived from vector or DLG hypsographic and hydrographic source data, an RMSE of one-half contour interval or better is required.

**Vector product format**

The Vector Product Format (VPF) is a standard format, structure, and organization for large geographic databases that are based on a geo-relational data model and are intended for direct use. VPF is designed to be compatible with a wide variety of applications and products. VPF allows application software to read data directly from computer-readable media without prior conversion to an intermediate form. VPF uses tables and indexes that permit direct access by spatial location and thematic content and is designed to be used with any digital geographic data in vector form that can be represented using nodes, edges, and faces. VPF defines the format of data objects, and the geo-relational data model provides a data organization within which software can manipulate the VPF data objects. A product specification corresponding to a specific database product determines the precise contents of feature tables and their relationships in the database. In this context, each separate product or application is defined by a product specification and implemented by VPF structures.

**Intersecting Polygons**

The responsibility of the Object Mapping Component (OMC) is to maintain the global contour map, a polygonal representation of the objects detected and the space
traversed that has reached an edge of the local grid map. Any new polygon must be inserted into the database. New polygons, known as bubbles, entering the database must be evaluated. Ones that are fully contained within another polygon are discarded, ones that intersect polygons must be combined, and ones that stand alone are added to the database.

In original method developed by Okui [OKU99], a data structure is produced to check connectivity of a bubble and a polygon. The data structure contains the following elements:

- match_count - number of matched vertices between a bubble and a polygon
- match_polygon - pointer of a matched polygon
- match_bvertex[i] - ith matched vertex number of a bubble
- match_pverteix[i] - ith matched vertex node pointer of a polygon
- online_count - number of online vertices in a bubble
- online_polygon - point of a polygon whose side has a vertex of a bubble
- online_bvertex[i] - ith online vertex number of bubble
- online_pvertex[i][0] - pointer to ith start vertex of an onlined side of a polygon
- online_pvertex[i][1] - pointer to ith end vertex of an onlined side of a polygon

The possible options between a bubble and a polygon are that the bubble is contained within a polygon, and is discarded, that the bubble does not intersect any of the polygons, and becomes a new polygon added to the database, or the bubble intersects the polygon. Since position in the simulation is considered absolute and without any error, there are only two considerations when a bubble intersect a polygon; either the bubble and polygons share vertices, expressed by the first half of the data structure, or the bubble’s vertices line on only of the edges of the polygon, expressed by the second half of the data structure. When there is connectivity between a bubble and a polygon, they are fused together by a series of 47 different rules used to express possible combinations of bubble and polygon intersection.
Intersection of Segments

The problem with the previous method is that it only considers a bubble sharing a vertex and/or an edge with a polygon. There is no consideration of a bubble crossing an edge of a polygon, as shown in Figure 8-2. However by using computational geometry, a simpler method to combine bubbles and polygons can be used by finding the intersection between polygons. This method covers all the rules generated by Okui [OKU99] and handles the general case of a bubble intersecting a polygon because of errors generated in the position of the vehicle.

![Diagram of combining bubbles and polygons]

**Figure 8-2** Combining bubbles and polygons

To find the intersection between two polygons, one must first find the intersection of line segments. This is accomplished by the Bentley-Ottmann Plane sweep [DEB00] algorithm, in which a horizontal line \( L \) is swept downward over a collection of polygon edges. It only remains to search for cord intersections in the local neighborhood of a vertex hit by \( L \).

Imagine sweeping a line \( L \) over a collection of segments \( S=\{s_0, s_1, \ldots, s_n\} \). Let \( x=s_i \cap s_j \) be an intersection point between two segments. Just before \( L \) reaches \( x \), \( L \) pierces both \( s_i \) and \( s_j \), and they are adjacent along \( L \): No other segment is between them on \( L \).
Thus, at some time prior to every intersection “event” (when $L$ crosses an intersection point), the intersection segments are adjacent on $L$. This produces the sought-for locality: Computing intersections between segments adjacent on $L$ suffices to capture all intersection points. Some of these adjacent segments do not in fact intersect, be the “wasted effort” is small.

To keep focused on the main idea, let’s make several simplifying assumptions:

Assume no segment is horizontal and no three segments pass through one point. The plan is to sweep $L$ over the segments, stopping at events of three types:

4. The top endpoint of a segment is hit,
5. The bottom endpoint if a segment is passes, or
6. An intersection point between two segments is reached.

All three of these events cause the list $\mathcal{L}$ of segments pierced by $L$ to change: A segment is inserted, deleted, or two adjacent segments switch places, respectively. With each change, intersections between newly adjacent segments must be computed.

Although segments must become adjacent in $\mathcal{L}$ prior to their point of intersection $x$, it is not guaranteed that $x$ is the next intersection event when it is computed. Rather, the intersection events must be placed in a queue $Q$ sorted by height, along with the segment endpoints. An illustration should make the algorithm clear. Consider the set of segments $S=\{s_0, s_1, \ldots\}$ show in Figure 8-3. Let $a_i$ be the upper endpoint of segment $s_i$, 

![Figure 8-3](image)
and \( b \), its lower endpoint. Then the event queue is initialized to \( Q = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, b_2, \ldots) \), all the segment endpoints sorted top to bottom. When \( L \) reaches \( a_1 \) and \( a_2 \) (position 1), \( s_0 \) and \( s_1 \) become newly adjacent, and their intersection point \( x_{01} \) is added to the queue after \( b_2 \). \( s_1 \) and \( s_2 \) are also newly adjacent but do not intersect. Note that the higher intersection point \( x_{56} \) has not yet been constructed. At position 2, \( L \) hits \( a_5 \); the newly adjacent segments \( s_3 \) and \( s_0 \) do not intersect. At this point \( \mathcal{L} = \{s_3, s_0, s_1, s_2\} \). At position 3, \( L \) hits \( a_4 \), and intersection point \( x_{34} \) is added to \( Q \) at its appropriate location. Note that three intersection event “between” \( s_3 \) and \( s_4 \) will be encountered before \( x_{34} \) is reached. By the time \( L \) reaches the first intersection event at position 6, all the endpoints above have been processed, and \( \mathcal{L} = (s_5, s_3, s_0, s_4, s_0, s_1) \). This event causes \( s_5 \) and \( s_6 \) to switch places in \( \mathcal{L} \), introducing new adjacencies that result in \( x_{56} \) and \( x_{45} \) being added to \( Q \). \( Q \) now contains all the circled intersection points shown in the figure.

The algorithm needs to maintain two dynamic data structures: one for \( \mathcal{L} \) and one for \( Q \). Both must support fast insertions and deletions in order to achieve an overall low time complexity. A balanced binary tree is sufficient to permit both \( \mathcal{L} \) and \( Q \) to be stored in space proportional to their number of elements, \( m \). Recall the earlier worry about “wasted effort.” However, the number of intersection calls is at most twice the number of events, because each event results in at most two new segment adjacencies: an inserted segment with its new neighbors, two new neighbors when the segment between is deleted, and new left and right neighbors created by a switch at an intersection event.

**Intersection of Nonconvex Polygons**

It is not difficult to alter the Bently-Ottmann sweepline algorithm to compute the intersection of two polygons. Let the two polygons be \( A \) and \( B \), with vertices labeled \( a_i \),
and \( b_1 \) respectively. The main idea is similar to that used by scan-line algorithms for filling (painting) a polygonal region on a graphics screen. One maintains along the length of the sweep line \( L \) a “status” indicator, that has the following value:

\[
\varnothing: \text{exterior to both polygons} \\
A: \text{inside } A, \text{ but outside } B \\
B: \text{inside } B, \text{ but outside } A \\
AB: \text{inside both } A \text{ and } B
\]

The status is recorded for the span between each two adjacent segments pierced by \( L \); clearly it is constant throughout each span.

Consider the example shown in Figure 8-4. When \( L \) is at position 2 (event \( b_1 \)), the left-to-right status line is (\( \varnothing, A, B, AB, B, \varnothing \)). This information can be easily stored in the same data structure representing \( L \).

Starting from the beginning, at position 0, when \( L \) hits \( a_0 \), the fact that both \( A \)-edges are below \( a_0 \) indicated that an \( A \)-span is being intersected. At position 1, a \( B \)-span is inserted. Just slightly below \( b_0 \), an intersection event open up an \( AB \)-span, easily recognized as such because the intersecting segments each bound \( A \) and \( B \) from opposite side, with \( A \) and \( B \) below. At position 3, intersection event \( x \), the opposite occurs: the intersecting segments each bound \( A \) and \( B \) above them. Thus an \( AB \)-span disappears, replaced by an \( \varnothing \)-span.
between the switched segments. At a2 (position 4), the inverse of the a0 situation is encountered: The A-edges are above, and an A-span is engulfed by the surrounding B-spans. Although precise rules were not provided, it should be clear that the span status information may be maintained by the sweepline algorithm.

Boolean operations between polygons may be constructed with variants of the Bentley-Ottmann sweepline algorithm. If you want to compute intersection $A \cap B$, extract the faces that are labeled AB. If you want to compute the union $A \cup B$, extract the faces that are labeled $A$ or $B$. And if you want to compute the difference $A \setminus B$, extract the faces that are labeled with $A$ and not with $B$. These Boolean operations are the heart of man CAD/CAM software systems, which, for example, construct complex parts by subtracting one shape from another, joining shapes, slicing away part of a shape, etc.
REFERENCES


BIOGRAPHICAL SKETCH

As a young child, David Keith Novick was always taking things apart. When he was able to put them back together again, and they still worked, he knew what he wanted to be. He wanted to be an engineer. But which kind?

During high school, he took all the relevant course work, and found he enjoyed programming on computers, working with electronics, and designing mechanisms. After graduation, he decided to attend the University of Florida. Still unable to make a decision, the department of mechanical engineering chose him, and he hasn’t looked back since.

Finishing a Bachelor of Science degree in 1991, he decided he wanted to continue his education. While in mechanical engineering, he was still dabbling within the sciences of computers and electronics. To combine all three disciplines, he pursued the field of robotics with the mechanical engineering department. In 1993, he obtained a Master of Engineering degree with minors in computer science, and electrical engineering.

Still wanting to improve, he stayed at the University of Florida for a Ph.D. in mechanical engineering. Now greatly involved within the field of robotics, he again achieved dual minors in computer science and electrical engineering, finally finishing the dissertation in May of 2002. From here, he plans to continue into the industrial world of robotics.