INFORMATION TO USERS

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

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

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

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

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600
Building a map for robot path planning by fusing video images and laser rangefinder data

Reynolds, Steven Lamar, Ph.D.
Rice University, 1993
RICE UNIVERSITY

Building a Map for Robot Path Planning by Fusing Video Images and Laser Rangefinder Data

by

Steven Reynolds

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Dr. Don H. Johnson, Chairman
Professor
Electrical and Computer Engineering

Dr. Ian Walker
Assistant Professor
Electrical and Computer Engineering

Dr. John Cheatham
Professor
Mechanical Engineering and Material Science

Houston, Texas
April, 1993
Building a Map for Robot Path Planning by Fusing Video Images and Laser Rangefinder Data

Steven Reynolds

Abstract

This thesis describes algorithms that fuse the data from a single video camera and a laser rangefinder. By merging information from these sensors, the algorithms build a digital map of the local environment for a robot to use in navigation. The video image is segmented, and these segments are used to construct a vector scan for the laser rangefinder. The vector scan is a sequence of straight line segments that is created by a greedy algorithm; the range measurements along this vector scan are used to build the map. To build or augment the map, the range measurements are projected to the floor plane and then accumulated in a histogram. This histogram is then summed in an uncertainty region around each point, and points with probability mass greater than one half are marked as obstacle points. The algorithms are tested on an example scenario incorporating actual video images and simulated laser rangefinder data. The map produced by the algorithms shows a good representation of the obstacles without any false alarms.
Acknowledgments

My colleagues have been very helpful: Peter Weiland, Larry Ciscon, J. D. Wise, and Brad Denney. Peter and Brad answered my questions when I didn’t even know what a laser rangefinder was. Larry and J. D. have endured an endless stream of questions without complaint and with good nature. Hurray, the stream has stopped. J. D. also gracefully accepted my constant demands for more disk space. My advisor, Dr. Johnson, got me started, kept me from going astray, and made sure I had what I needed. Many thanks. He also suffered though many rough drafts, and immeasurably improved this document. Many many thanks. I also thank the rest of the committee, Dr. Walker and Dr. Cheatham, for their time and patience. My high school physics teacher, Dr. Evers, got me started on this long path — thanks. Gerda found a mistake in Figure 4.6, BAS drew Figure 4.4, and Dr. Wasz gave me an existence proof. Thanks.

I have been supported by a MITRE fellowship, which was invaluable. I also thank Jack Heberlig for all his efforts and his contagious enthusiasm.

I thank my loving wife, Barbara for tolerating all this stress. Without love, life is an empty cup. I also thank my family who’ve supported me in my every endeavor. They always said I could do anything; who’d have thought I would have ended up here? I finished this tome with the help of Prudence (my cat) and Tavi (the computer). Prudence woke me up in the mornings — feed me NOW — and Tavi checked my spelling. Prudence was always ready to act as an obstacle so that I could study obstacle avoidance. We had many research sessions.
I also thank Calvin and Hobbes; *laughter is the best medicine.*

Let freedom ring

— Dr. M. L. King Jr.
# Contents

Abstract ii
Acknowledgments iii
List of Illustrations vii
List of Tables x

1 Introduction 1
1.1 Sensor Fusion 1
1.2 Path Planning 4
1.3 System Overview 7

2 Background 12
2.1 Notation 12
2.2 Laser Rangefinders 15

3 A Model of Sensor Fusion Processing 22
3.1 Sensing the Environment 23
3.2 Forming the Refined Measurements 24
3.3 Fusing the Data 26
3.4 Implementation Issues 27
3.5 The Proposed Algorithms 27

4 Algorithms for Fusing the Sensor Data 31
4.1 Segmenting the Camera Images .......................... 33
4.2 Generating the Vector Scans .............................. 34
  4.2.1 Map Vector Scans .................................. 35
  4.2.2 Segment Vector Scans ................................ 37
  4.2.3 Timing the Laser Scans .............................. 40
4.3 Building the Map ........................................ 42
  4.3.1 Estimating the Obstacle Regions .................... 44
  4.3.2 The Uncertainty Region ............................. 50
  4.3.3 Spreading in the Estimate of the Obstacle Regions..... 53

5 Results .................................................. 56

6 Conclusions .............................................. 83

Bibliography ............................................... 88

A Characterizing the Camera .............................. 94
  A.1 Calibration of the Camera ............................. 94
  A.2 The Camera Orientation and Position ................ 97

B Distribution of the Range Errors ...................... 100

C The Laser Rangefinder Simulator ....................... 102

D Calculation of the Dwell Time .......................... 108
Illustrations

1.1 Sensor data flow with no fusion. ........................................... 8
1.2 Data flow for the software implemented in this work. ............... 9

2.1 Map coordinate system. ...................................................... 16
2.2 Camera coordinate system. ................................................. 17
2.3 Image coordinate system. ................................................. 17
2.4 The imaging process. ....................................................... 18
2.5 A laser rangefinder. ......................................................... 19
2.6 Diagram of a receiver for a laser rangefinder. ......................... 19

3.1 The transformations in sensor fusion processing. ....................... 22
3.2 Block diagram of sensor fusion processing. ............................ 25
3.3 Interactions between the processing stages. .......................... 28
3.4 Comparison of the proposed software and the model. ................ 30

4.1 Detailed software data flow. .............................................. 32
4.2 Example mini-image. ....................................................... 34
4.3 Segmentation of the image given in Figure 4.2. ....................... 34
4.4 Examples of a footprint and an AOI. ................................. 36
4.5 Indicator function for a set. ............................................. 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>Distance transform for the set in Figure 4.5.</td>
<td>39</td>
</tr>
<tr>
<td>4.7</td>
<td>The priority polygons.</td>
<td>41</td>
</tr>
<tr>
<td>4.8</td>
<td>Step response time of two galvanometers.</td>
<td>43</td>
</tr>
<tr>
<td>4.9</td>
<td>Projection of object sample points to the map cells.</td>
<td>45</td>
</tr>
<tr>
<td>4.10</td>
<td>Steps to build the map.</td>
<td>46</td>
</tr>
<tr>
<td>4.11</td>
<td>The map lattice.</td>
<td>47</td>
</tr>
<tr>
<td>4.12</td>
<td>Clipping of large range values by the labeling operator.</td>
<td>49</td>
</tr>
<tr>
<td>4.13</td>
<td>Effects of $x$ pixel quantization on the range measurements.</td>
<td>53</td>
</tr>
<tr>
<td>4.14</td>
<td>Number of object sample points verses $t_0/\sigma$.</td>
<td>54</td>
</tr>
<tr>
<td>5.1</td>
<td>Experimental setup.</td>
<td>58</td>
</tr>
<tr>
<td>5.2</td>
<td>Video images.</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>Segmented video images.</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Laser images and map vector scan from position one.</td>
<td>66</td>
</tr>
<tr>
<td>5.5</td>
<td>Range data from a raster scan at position one.</td>
<td>67</td>
</tr>
<tr>
<td>5.6</td>
<td>Range data from a raster scan at position one, excluding “floor” data.</td>
<td>67</td>
</tr>
<tr>
<td>5.7</td>
<td>Source of the artifacts in the simulator.</td>
<td>68</td>
</tr>
<tr>
<td>5.8</td>
<td>Segment vector scans.</td>
<td>69</td>
</tr>
<tr>
<td>5.9</td>
<td>Contour plot of the number of object sample points that project to</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>each map cell.</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td>Range data from three segment vector scans at position one.</td>
<td>72</td>
</tr>
<tr>
<td>5.11</td>
<td>Range data from three segment vector scans at position two.</td>
<td>73</td>
</tr>
<tr>
<td>5.12</td>
<td>Dilated histogram derived from the segment vector scans.</td>
<td>74</td>
</tr>
<tr>
<td>5.13</td>
<td>Contour plot of the dilated histogram from Figure 5.12.</td>
<td>75</td>
</tr>
<tr>
<td>5.14</td>
<td>Map derived from the histogram in Figure 5.12.</td>
<td>76</td>
</tr>
<tr>
<td>5.15</td>
<td>Free space calculated from the map in Figure 5.14.</td>
<td>77</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>5.16 Histogram of $y_a$ from equation 5.1.</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>5.17 Histogram of $y_d$ from equation 5.1.</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>5.18 AOIs calculated from position one.</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>5.19 Contour plot of the dilated histogram from the map vector scan.</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>5.20 Map from the map vector scan.</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>A.1 Measurement of the front focal point.</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>A.2 Barrel distortion.</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>A.3 Camera pitch.</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>A.4 Camera yaw.</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>C.1 Data flow for the laser simulator.</td>
<td>106</td>
<td></td>
</tr>
</tbody>
</table>
Tables

2.1 Pseudo-code for algorithm1. ........................................... 13
2.2 Pseudo-code for algorithm2. ........................................... 13
2.3 Examples of a vector in each of the coordinate systems used in this
thesis. ............................................................................ 14
2.4 Description of variables in equation (2.7). ....................... 20

4.1 Data files. ....................................................................... 33
4.2 Partial pseudo-code for analyze_map. ............................... 36
4.3 Pseudo-code for ma_extend. .............................................. 37
4.4 Pseudo-code for scan_map. .............................................. 37
4.5 Pseudo-code for scan_seg. .............................................. 41
4.6 Pseudo-code for update_map1. ....................................... 51
4.7 Pseudo-code for update_map2. ....................................... 51

5.1 Procedure to build a map using segment vector scans........ 57
5.2 Procedure to build a map using map vector scans ............ 59
5.3 Run times for the software modules. ................................ 60
5.4 The segments from position one. .................................... 64
5.5 Segment vector scan from position one. ......................... 70
5.6 Map vector scan from position one. ............................... 80
A.1 Calibration constants for equation (A.1) ......................... 97

C.1 Record from scene.asc specifying a segment region .............. 105
C.2 Record from scene.asc specifying a polygonal region .......... 107
C.3 Assumed reflectivity of the objects used in the simulation .... 107
C.4 Pseudo-code for sim_laser ........................................ 107
Chapter 1

Introduction

Sensing the environment is an important part of operating any robot whether it is an industrial arm, a teleoperated mobile robot, or an autonomous mobile robot. Any robot experiences unpredictable external forces and has inaccuracies in its world and robot models. Sensing allows a closed loop control strategy to be used that can compensate for these unmodeled factors, while an open loop control would fail utterly. Closed loop control allows good system performance even when perturbances occur.

The most basic competency for any mobile robot is obstacle avoidance: to sense the environment, to decide what is an obstacle, to locate the obstacles, and to maneuver around them. We focus here on acquisition and merging of information to create a map of obstacles in the robot's environment that can be used in path planning.

1.1 Sensor Fusion

Sensor fusion implies combining data from multiple sensors possibly using multiple types of sensors. Any single sensor has difficulty acquiring information to construct an obstacle map. Video data from a single camera clearly lacks depth information unless we resort to expensive analysis of multiple images (depth from motion [1]). Using a single laser rangefinder has proven slow [2]. By fusing information from multiple sensors, we can more easily and efficiently construct the obstacle map. Most
autonomous robots have many sensors: shaft encoders, touch sensors, ultrasonic rangefinders, laser rangefinders, and cameras. Sensor fusion is used for any of the following reasons:

1. Use similar sensors to increase confidence or accuracy.

2. Use sensors cooperatively to sense a parameter that they cannot measure individually.

3. Use dissimilar sensors to sense a single parameter because they have different failure modes or sensor errors.

4. Use dissimilar sensors to sense different aspects of the scene.

In the literature, there are four general approaches to sensor fusion: statistical, map-based, reactive, and abstract methods. The statistical methods use parameter estimation techniques. Durrant-Whyte [3] used Bayesian estimation for which one must assume knowledge of the prior distribution. A priori object position, for example, is modeled as a random variable with known distribution. In an indoor environment, most objects are stationary except when moved by humans. Human actions tend to be unpredictable and thus unsuited for a random variable description. Attempts to robustly compensate for the lack of knowledge about the prior distribution and the sensor noise have not worked well. For instance, McKendall and Mintz [4] derived a robust estimator that essentially biased the estimates toward the center of their possible range. Biasing the data because of a lack of knowledge seems ad hoc and unjustifiable. Luo and Lin [5] avoided prior distributions by using maximum likelihood methods. However, they made numerous mistakes in their presentation, the first of which was to show probability as the ordinate of a graph.
of a probability density function [6]. More importantly, the statistical methods ignore a crucial aspect of sensor fusion; they implicitly assume that the environment can be modeled by a few isolated points or by a function with a few parameters. For most robotic problems, objects have significant size and cannot be described by points. Additionally, we can not assume that the environment is shaped in a particular way because this assumption does not allow for unexpected objects. The real-world environment that we must sense is complex and cannot be described by a few parameters.

Map-based systems build an explicit description of the environment. Many such descriptions have been used. Moravec and Elfes [7] used a two-dimensional digital map. Allen and Bajesy [8] used a surface description. Herbert et. al [9] used a polygonal map in addition to a digital map. To some extent, the intended use drives the choice of representation. For instance, to find a grasp point on an object requires a surface or volume description while for path planning a two-dimensional digital map suffices. Moravec and Elfes' work is particularly interesting; it inspired our map building methods. They built a two-dimensional digital map using a method based on histograms, but their methods were heuristic rather than rigorous. Their use of the term "histogram" is rather loose; they did not construct a histogram estimate of any density function. They used data from an ultrasonic rangefinder to compute certainty factors for each element in the map. Each certainty factor expressed the "chance" that the corresponding region on the floor was occupied. To account for the finite beamwidth of the ultrasonic sensor, they updated map elements in a pie-wedge shaped region for each range measurement.

Reactive methods [10, 29] do not construct a map, rather the sensor measurements directly control the robot motors and may cause the robot to change states. Brooks [12] is well known for his behavior-based approach to robotics. He has im-
plemented some interesting systems, for example Genghis, which can walk on rough terrain. Another similar approach is artificial neural networks [13]. Again, the inputs to the network are the sensor measurements and the outputs are the motor signals. Reactive methods have a long history dating back to the 40's and 50's [14, 15]. These methods are related to the path planning methods of Lumelsky and Stepanov that are described in section 1.2.

Abstract methods [16, 17, 18] describe sensor fusion as a mapping from the sensor data to some representation of the environment, however they give little guidance on how these mappings should be constructed. This type of approach is derived from Marr [19]. His pioneering work developed a computational theory of vision. One aspect of his theories is the systems viewpoint, examining the inputs and outputs of the sensor/perception system. The abstract methods evolved from this type of viewpoint.

1.2 Path Planning

There have been many algorithms proposed for the path planning problem. Some of the basic choices in path planning are:

1. Use a two or three-dimensional map?

2. Use a digital or polygonal (polyhedral) map?

3. Consider robot rotations?

For an ordinary wheeled robot, a two-dimensional map suffices because the robot cannot navigate rough terrain. For cluttered environments, computing robot rotations in addition to translations can be important. None of the approaches using a digital map report consideration of robot rotations [25, 27, 29]. The configuration
space is the space of possible positions and orientations of the robot. For a planar robot that can rotate, the configuration space is $\mathbb{R}^2 \times S^1$ where $S^1$ is the unit circle. The free space is the subset of the configuration space that is not blocked by any obstacle.

One of the earliest path planners was Moravec's work on the Stanford cart [1]. In his work, the map was a list of all the two-dimensional points computed by a stereo vision system. Each point had an associated uncertainty circle and the path was constructed from tangents to the circles and arcs from the circles.

The two most powerful approaches to path planning are projection and retraction. The projection methods are typified by Schwartz and Sharir's work [20]. They solved the two-dimensional problem by partitioning the configuration space into regions where no rotation was necessary. The overall path was constructed by linking paths in each region. At the linking points, the robot could rotate. Lozano-Perez extended this approach to polyhedral obstacles and three-dimensional paths [21]. A problem with these algorithms is that they require the robot to slide along the faces of the obstacles. Any imprecision in the obstacle boundary or the robot position could result in a collision.

The second approach is retraction, a term from topology [22]. A subset of $X$, $B$, is a retract of $X$ if there exists a continuous function from $X$ to $B$. This function must have the elements of $B$ as fixed points ($f(b) = b$). A deformation retract is a retract such that $X$ can be deformed into $B$. By deformed we mean that $X$ can be stretched, shrunk and twisted but not cut or pasted. With a deformation retract, we are guaranteed that for any two points in the same connected component, a path connecting the points can be found by the following procedure: start at one point, follow the deformation to the retract, move in the retract, and finally go from the retract to the second point in the reverse direction of the deformation. Thus, we
have a decomposition of the original problem: finding a good deformation retract, computing the deformation (and its inverse), and path planning in the retract. The retract is chosen to be lower dimensional than the configuration space, so that path planning in the retract is an easier problem. An example of this method is the work done by Meng using a Voroni diagram [23]. The Voroni diagram is the set of points that are equidistant to two adjacent object boundaries. The Voroni diagram is a deformation retract with the additional nice property that it is far from the obstacles. Consequently, the paths computed by this method are close to the obstacles only at the endpoints or choke points. Canny extended the retraction method to use polyhedral obstacles and to consider robot rotations [24].

Another path planning approach that can be used for digital maps is the A* algorithm [25]. The A* algorithm finds the shortest path through a graph [26]; each map cell is considered a node in this graph. A* uses a best first search of the graph, where the best path is determined by summing the actual distance traveled from the start point and a lower bound of the remaining distance to the goal. When a path is found to the goal with distance shorter than the lower bound estimate distance of all the other candidate paths, the shortest path has been found. Another use of the A* algorithm was by Giralt who used a graph that described how the rooms in the local environment were connected [27].

Another approach is the gradient method. The obstacles generate a repelling potential and the goal point generates an attractive potential. The gradient of the potential function specifies the tangent to the path at any point. Finding the entire path is then a matter of integrating out a differential equation. Khatib [28] used a gradient method with polygonal obstacles and Connolly and Burns [29] used a gradient method with a digital map.
The last approach uses no map at all. Lumelsky and Stepanov described algorithms that assume that the robot has only a tactile sensor [30]. The robot is assumed to be in a two-dimensional space. The robot moves to the goal point until it hits an obstacle. Algorithms specify the subsequent behavior of the robot so that it will go to the goal point.

1.3 System Overview

In this thesis, we use a map-based approach for obstacle avoidance. By partitioning the avoidance algorithms into a map builder and a path planner, each part can be checked and understood separately. The map provides a way to give the robot initial information about the environment as well as serving as a good debugging tool. At any time, the robot's perception of the environment is expressed by the map.

The sensor fusion algorithms use a video camera and a laser rangefinder to build this map. The algorithms are intended for an indoor robot that also has an ultrasonic rangefinder. The ultrasonic rangefinder has a broad beam [31], making it valuable for preventing collisions, but unsuitable for building high resolution maps. Therefore, the goal of the algorithms is to build a map of near and far obstacles without overly fearing collisions. These obstacles are assumed to be stationary. We also assume that the robot has a path planner and that a path exists to a goal point. The path planner would use the map to compute a new path around any unexpected obstacles. In this study, we used an actual video camera, but simulated the laser rangefinder.

Before sensor fusion became popular, usually only one sensor at a time was considered. When multiple sensors were considered, a typical paradigm was to process the data from each sensor individually. Each sensor system would produce
a high level description and fusion, if any, would occur at this level. Figure 1.1 shows this type of data flow. The key aspects of this paradigm are that the sensors are treated individually, and that the high level representation is strictly an output. More generally, data from other sensors and from the high level description can be used to help process each sensor’s data. The recent research in sensor fusion has popularized cross flow between the sensors, and the work in active vision has popularized top-down flows. Figure 1.2 shows a simple diagram of the data flow for the algorithms proposed in this work. This diagram shows both cross flow and top-down flow. Briefly, v1abel segments the video image, scan_seg computes a vector scan from the segments, update_map1 and update_map2 build the digital map, and analyze_map and scan_map compute a vector scan from the polygonal map.

![Diagram](image)

Figure 1.1: Sensor data flow with no fusion. The ovals are software modules, and the squares are data files.

Most researchers use the laser rangefinder to scan the environment in a raster pattern. This approach has a serious drawback: the scan takes a long time (0.5-1 s). In our approach, the camera image is segmented to find potential objects and the laser is sequentially pointed at each object. We term this type of scanning a \textit{vector scan} due to the similarity to a vector graphics display device. This vector scan is
Figure 1.2: Data flow for the software implemented in this work.
composed of straight line segments. We also developed a method to rank the importance of the segments. This ranking allows the laser to scan the most important targets first. A second type of vector scan is computed from a stored polygonal map. This map contains the footprints of objects and their approximate height. It contains stored information about the robot’s environment that must be verified before the robot uses the information in path planning. The polygonal map could be built from the digital map, although in this work it is an operator input.

The digital map is two-dimensional in this work; elements in the map indicate whether the corresponding region in the floor is an obstacle or not. This map is built by a method that is equivalent to histogram density estimation. Multiple scans of the laser rangefinder can be used to achieve noise rejection. The range measurements that correspond to non-floor measurements are accumulated in a histogram. The histogram is then effectively integrated and the regions with probability mass greater than one half are labeled as obstacles. This method assumes no prior knowledge about the shapes or reflectivity of objects; nor does it rely on any prior knowledge about the environment.

Few researchers have reported using both a laser ranger and a camera. Aggarwal and Magee [32] used a camera and a laser rangefinder for object recognition and for determining the motion parameters of a moving object. The object recognition resulted from matching to a database of known objects. This system is similar to our work because they used camera images to guide a rangefinder that samples the range at a few key points. Trevedi et al. [17] used a camera and a rangefinder for pose estimation. They also matched sensor data to stored object models. However, they used the curvatures from a raster scanned range image as the features to be matched.
We found Ballard [33] influential. He stressed the use of vision systems that are actively aimed. In particular, the diagrams of eye movements in humans stimulated our ideas for vector scanning the laser. Carpenter comprehensively examined eye movements in his book [34]. Because the human eye has better resolution in the small central region of the retina, the fovea, we tend to move the eye to point the fovea at important targets. During eye movement, vision is seriously degraded, so the movements tend to be either very high velocity, or quite low velocity. The eye movements show different patterns depending on the cognitive task that the human is performing, even with the same sensory data. Thus, the eye movements are, to some extent, controlled by the cognitive task and the current interpretation of the sensor data. In a loosely analogous fashion, we use the polygonal map and the video image to compute the laser scans. The polygonal map corresponds to things the system expects to be present and the video image corresponds to things of which the system has no prior knowledge. The priority ranking of the video segments is loosely analogous to a humans interpretation of the importance of a target.

At Rice, some research has been done in the area of laser rangefinders and path planning. Wu [35] simulated a laser rangefinder and explored methods for building an elevation map and for three-dimensional path planning. He used optimization techniques on spline functions. Weiland [36, 37] investigated building a two dimensional map from the laser range data and path planning using an artificial neural net. Denney [39] built a prototype laser rangefinder. Dr. Cheatham's ME 407 class recently built a second laser rangefinder. Ciscon [38] developed a hierarchical path planner that operated in a distributed environment.
Chapter 2

Background

In this chapter, we introduce notation and describe the operation of the camera and the laser rangefinder.

2.1 Notation

All entities that exist on the computer such as files, programs, and data structures are shown in a fixed width font such as `seg-image.dat` or `scene.asc`. The file extension "dat" means that we are referring to a data file; the extension "asc" means that we are referring to an ascii file. When we refer to an array, we use the notation: `seg-image[][]`. Thus, the reader will encounter statements like "read `seg-image.dat` into `seg-image[][]`," which means that the data from the file `seg-image.dat` is to be read into the array `seg-image[][]`. The k,lth element of this array is shown by `seg-image[k][l]`. Algorithms are specified with pseudo-code, which is a mix of English statements, formula, and branching statements. The only branching statements that I use are `for`, which defines a loop, and `if else`, which defines the usual branching. The scope of these statements are indicated by indentation. Table 2.1 shows a simple algorithm that illustrates a `for` loop; it sums up the numbers from one to ten, and then prints the sum. Table 2.2 shows a simple algorithm that illustrates an `if else` construction; it prints "test is true, done".
algorithm1
1    sum = 0
2    for i = 1 up to 10
3        sum = sum + i
4    print sum

Table 2.1: Pseudo-code for algorithm1.

algorithm2
1    test = true
2    if test
3        print "test is true,"
4    else
5        print "test is false,"
6    print "done"

Table 2.2: Pseudo-code for algorithm2.

Next, we introduce some notation for set operations that are used in the map building techniques. The indicator function of a set is denoted

\[ I(B)(x, y) = \begin{cases} 
1 & \text{if } (x, y) \in B \\
0 & \text{otherwise.} 
\end{cases} \tag{2.1} \]

The notation, \( B \oplus D \), indicates the dilation of two sets, which is the set defined by

\[ B \oplus D = \{ b + d : b \in B, d \in D \}. \tag{2.2} \]

Note that we are using a definition of dilation consistent with Haralick, Sternberg and Zhuang [40]. Serra [41] used a definition that is slightly different; he used \( b - d \) in equation (2.2). The current literature at times uses either definition.

Next, we introduce the notation used for vectors and matrices. Scalars are usually denoted by lower case roman or greek letters whereas vectors are always
given by lower case bold roman letters. Subscripts indicate the components of the vector while primes indicate the reference. For example, a vector \( \mathbf{w} \) in the camera frame is

\[
\mathbf{w'} = [w'_x \ w'_y \ w'_z]^T.
\]

In the above equation, the T indicates transpose because the vectors are column vectors. Table 2.3 shows the notation for all the coordinate frames, which are defined shortly. A form like \( x' \) denotes the \( x \) variable in the camera frame. Parenthesis enclose any variable that is raised to a power. For example, \( w'_x^2 \) squared is denoted as \( (w'_x)^2 \). The only matrix is \( A \); \( A \mathbf{w'} \) indicates the usual matrix product. We also use some nonlinear operators; these are given in uppercase roman with an underbar such as \( \mathbf{L} \). Thus, \( \mathbf{L} \mathbf{w} \) does not mean matrix multiplication, but rather applying \( \mathbf{L} \) to \( \mathbf{w} \).

| \( \mathbf{w} \) | map frame |
| \( \mathbf{w'} \) | camera frame |
| \( \mathbf{w''} \) | image frame |

**Table 2.3:** Examples of a vector in each of the coordinate systems used in this thesis.

Next, we introduce the coordinate systems. To describe the range measurements, we need two coordinate systems: one fixed to the room and one fixed to the camera and laser rangefinder. Figure 2.1 shows the map coordinate system. This coordinate system was chosen to make measurements easier as described in Chapter 5. Figure 2.2 shows the camera coordinate system. This coordinate system is oriented and located in the conventional way. To simplify the notation, we shall also use the camera frame for the laser rangefinder. This simplification is equivalent to assuming that they are perfectly registered (pointing in the same direction). The map and
camera frames are related by

$$w' = Aw + t'$$

(2.3)

where $A$ is a rotation matrix and $t'$ is a translation vector. To describe features in the camera image, we need a coordinate system for the images. Figure 2.3 shows this image coordinate system.

The imaging equations are

$$x'' = \frac{f_x w'_x}{w'_z}$$
$$y'' = \frac{f_y w'_y}{w'_z}$$

(2.4)

where $w'$ gives the location of a point in the camera frame, $(x'', y'')$ gives the location of the imaged point, $f_x$ gives the focal length of the lens in $x$ pixel units, and $f_y$ gives the focal length of the lens in $y$ pixel units. These equations form the paraxial model in optics [64]. Figure 2.4 depicts the imaging process. Note that we are using a convenient abstraction: the real image is formed inside the camera, not in front of the lens. Section A.1 discusses the calibration of the camera and section A.2 discusses the computation of the rotation matrix $A$ and the translation vector $t$.

### 2.2 Laser Rangefinders

Figure 2.5 gives a block diagram of a laser rangefinder, which works as follows. The RF-source modulates the amplitude of the laser. Then, the optics aim the laser beam at the target. The reflected light is then focused on the photodetector by the receiving optics. Finally, the resulting RF signal is sent to a phase detector; the output of the phase detector is proportional to the range. The laser is typically a laser diode: it can be amplitude modulated by simply modulating the power supply to the diode. The range can only be measured unambiguously up to $1/2\lambda_m$, where
Figure 2.1: Map coordinate system. The room is Abercrombie C131C. The z axis is pointing up out of the page.
Figure 2.2: Camera coordinate system. The $x$ axis is pointing up out of the page. See section A.1 for the position of the origin.

Figure 2.3: Image coordinate system. The origin of the coordinate system is the center of the image.
Figure 2.4: The imaging process. The point \( p' = (x''/f_x, y''/f_y, 1) \) is the image of \( w' \) and is at the intersection of the plane \( z' = 1 \) and the straight line from the origin to \( w' \).

\( \lambda_m \) is the modulating wavelength. A reasonable frequency for the RF source is 30 MHz, which gives a maximum unambiguous range of 5 m. In this work, we ignore possible range ambiguities. These ambiguities can be mitigated by using multiple modulating frequencies. The optics include some sort of scanning device to aim the beam. For this work, we assume that mirrors mounted on galvanometers scan the laser. Raster scanned systems typically use a rotating polygonal mirror and a nodding mirror. We discuss timing the laser scans in section 4.2.3.

Figure 2.6 shows the receiver circuitry. This receiver structure is consistent with Nitzan et. al. [42], and the laser rangefinder that was recently built here at Rice. The phase estimate is

\[
\hat{\phi} = \tan^{-1} \frac{i_2}{i_1}
\]  

(2.5)

and the range estimate is

\[
\hat{r} = \frac{\lambda_m}{4\pi} \hat{\phi}.
\]  

(2.6)
Appendix C describes the simulation of the laser rangefinder.

![Diagram of a laser rangefinder]

**Figure 2.5:** A laser rangefinder. The transmitted and reflected beams are actually coaxial. They are drawn separated for clarity.

![Diagram of a receiver for a laser rangefinder]

**Figure 2.6:** Diagram of a receiver for a laser rangefinder.
Nitzan et. al. at SRI [42] provide an extensive analysis of the range errors of a laser rangefinder. They assumed that the dominant noise source was due to the quantum nature of light and that the environment surfaces were Lambertian. Under these assumptions, they calculated an approximate expression for the range errors

\[
\sigma_r = \frac{\lambda_m}{m} \sqrt{\frac{hc}{8\pi\alpha\eta\lambda A_R F_T T \rho_d \cos \theta}}.
\]

Table 2.4 describes the variables in this equation. They also performed experiments that support equation (2.7). They give most of the derivation for the distribution of range. Appendix B shows the remainder of the derivation and the distribution. Equation (2.7) gives the range error as a linear function of range. This linear relationship is supported by experiments done at AT&T by Miller and Wagner [43].

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_R )</td>
<td>receiver capture area</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>efficiency of optical system (lens's &amp; mirrors)</td>
</tr>
<tr>
<td>( c )</td>
<td>speed of light</td>
</tr>
<tr>
<td>( \eta )</td>
<td>quantum efficiency of photodetector</td>
</tr>
<tr>
<td>( F_T )</td>
<td>Average transmitted flux (watts)</td>
</tr>
<tr>
<td>( h )</td>
<td>Plank's constant</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>wavelength of light from the laser</td>
</tr>
<tr>
<td>( \lambda_m )</td>
<td>modulating wavelength</td>
</tr>
<tr>
<td>( m )</td>
<td>modulation index</td>
</tr>
<tr>
<td>( \rho )</td>
<td>reflectivity of the target</td>
</tr>
<tr>
<td>( T )</td>
<td>integration time</td>
</tr>
<tr>
<td>( \theta )</td>
<td>angle between surface normal and laser beam</td>
</tr>
</tbody>
</table>

**Table 2.4:** Description of variables in equation (2.7).

In contrast, the CMU group [9] [44] reported that range error is a quadratic function of range. They claimed that the standard deviation of range is proportional to the surface area illuminated by the laser beam [9], but they gave no explanation
for this claim. Some of their data is quite confusing; their Figure 6.6 shows measured range negatively correlated with true range [44]. Even if we believe the CMU group’s results, the simulation would not change much because their quadratic functions are reasonably close to linear.
Chapter 3

A Model of Sensor Fusion Processing

We are interested in properties that are common to all sensor fusion systems. These theoretical issues are explored by constructing a model of sensor fusion processing. The world is modeled by a surface, or collection of surfaces, in the world space, \( \mathcal{W} \). The sensors sample these surfaces and thereby map some portion of the world space into the measurement space, \( \mathcal{M} \). The system then performs simple processing on these data to form the prefusion measurement space, \( \tilde{\mathcal{M}} \). Next, the robotic system fuses the data into a high level description of the world and arrives at the description space, \( \mathcal{D} \). Figure 3.1 illustrates these transformations.

![Diagram](image)

**Figure 3.1**: The transformations in sensor fusion processing.
3.1 Sensing the Environment

The robot’s environment is described as a collection of dynamic surfaces. Each surface can change with time. Disjoint objects in the environment are represented as a collection of surfaces. A surface can be given by a parameterization \( w : D \rightarrow R^q \) \( (q \geq 3) \), where \( D \) is the three-dimensional domain of \( w \). One component in the domain represents time and the remaining two represent spatial location. The surface given by \( w \) has \( q \) dimensions: three components describe the location of points on the surface, and the remaining components describe the physical properties at each point. For example, the physical properties might be reflectivity, color, or roughness. We constrain the parameterizations to be continuous, but we do allow point objects, which are intended to describe point sources of energy. This addition is useful in far-field sonar problems, for example.

The parameterizations are not unique; many different parameterizations produce the same surface. Additionally, a complex surface can be decomposed into multiple simpler surfaces. Consider, for instance, a bookshelf full of books. Because the books all touch the bookcase, the number of surfaces used is somewhat arbitrary. The number of surfaces used to describe the world is denoted by \( J \).

We assume that the sensing system contains \( K \) sensors. Their outputs — the measurements — form the measurement space, \( M \). A measurement from the \( k \)th sensor is denoted \( m_k \), and is a multi-dimensional array whose dimension depends on the sensor. For a color camera with 512 \( \times \) 512 resolution, the measurement is a 3 \( \times \) 512 \( \times \) 512 array. For a sonar, the measurement is the amplitude of the sound wave and thus is a scalar. For the noncontact sensors, the sensing process is described by the equation for the underlying information transmission. The wave equation would be appropriate for any electromagnetic or acoustic sensor while the heat equation
would be appropriate for any diffusion process such a scent or smoke. The mapping
S also depends on the sensor state. For a video camera, the sensor state might be
position, orientation, focus setting, and gain.

The measurements are related to the world in complex ways. The mapping S can
be nonlinear. Consider two objects on a black background. If one object occludes
the other, the composite image is not the sum of the two objects imaged separately.
Each surface in the world space can contribute to the measurement of a sensor. For
instance, one surface could cast a shadow on a second nearby surface. Consequently,
it may not be possible to treat the surfaces individually; they must be considered
collectively. Furthermore, the mapping is not one-to-one. For example, a single
camera does not give depth nor does it give information about the obscured parts
of objects.

Each sensor is assumed to produce data in discrete time, either synchronously
or asynchronously. At times other than the sampling time, the measurement is
undefined. Because the sensors are not synchronized, one sensor might take a mea-
surement while the others are inactive. The entirety of these mappings from the
world space to measurements is denoted by the collective mapping S. Figure 3.2
details S. This figure also shows the remaining mappings that are described in the
following paragraphs.

3.2 Forming the Refined Measurements

The mapping P represents elementary signal processing operations performed inde-
dependently on each sensor’s measurements. The range of P is the prefusion measure-
ment space. The processing on the kth sensor stream is denoted P_k and the resulting
prefusion measurements are denoted \( \tilde{m}_k \). Examples of these operations are filtering,
edge detection, and coordinate transformation. The mapping $P$ is constrained to be causal. The data in the prefusion measurement space is still organized in terms of sensors as opposed to world features; no assimilation of the data has taken place yet.

The measurement space and the prefusion measurement space have similar structure with one very important difference. Pre-processing admits the creation of categorical data, a term given by statisticians to non-numeric data. In computer science, these are termed symbolic data. Categorical data take values from some finite set. The set has no algebraic structure, which means no addition or subtraction operation is defined. Also, there might be no ordering relationship. An example of categorical data is the output of an edge detector. Each pixel is labeled either "edge" or "not-edge". Attempting to add "edge" and "not-edge" is meaningless. Likewise we cannot say if "edge" > "not-edge". In the output from the edge detector, "edge" and "not-edge" are represented by numbers, usually zero and one. However, this representation is arbitrary, which is one distinguishing feature of categorical data.
3.3 Fusing the Data

The mapping $F$ takes the prefusion measurement space into the description space, $D$; the description itself is denoted $d$. The goal is to merge data from different sensors that pertain to the same world feature. Determining how to match these data is called the association problem. In stereo vision, matching image features to calculate disparity is an example of data association. Some of the data in the description space may be categorical. The description space can take two types of structures. In the first case, the system builds a map of the environment, without distinguishing objects. An example would be a road following system that builds a surface representation of the environment in front of the vehicle. In the second case, the system describes the world in terms of discrete objects. An example would be a sonar system that creates a list of ships in the vicinity. The number of objects in the world is not known a priori to the system. In this case, the dimension of $D$ is not known beforehand, and must be computed from the data.

Parameter estimation techniques are incomplete for two reasons: (1) we must decide which measurements apply to a particular world feature before fusion and (2) categorical data cannot be cast into the usual probabilistic formalism. The fusion process is a transformation from a sensor-based domain to a description of the environment. Because $S$ is not one-to-one, computation of $S^{-1}$ is an ill-posed problem. Thus, the fusion step is not only desirable, it is necessary.

The goal of the fusion process is to build a good description of the environment. Notice that the description frequently does not contain as much information as the world space. For instance, the description could be a two-dimensional map, even though the world space consists of $q$-dimensional surfaces. We can consider the goodness of the description by forming some ideal description based directly on the
surfaces (instead of the measurements)

\[ R : \{w_j\} \mapsto d_R. \]  

Thus, \( R \) takes the surface representation into some ideal description, \( d_R \). This could then be compared to the description, \( d \), from the sensor fusion system. Any computation of \( R \) must be done by some agent not subject to the limitations imposed by the original sensing system.

### 3.4 Implementation Issues

By considering just the data transformations, we have ignored some implementation issues. The system uses more than just the sensor data to generate the description. The fusion program might also use the objects that have been already identified or a database of known objects to help interpret the sensor data. The system might also use a map of the known environment.

The processing stages do more than transform data; they might control the previous programs/mappings as shown in Figure 3.3. Some examples should clarify this point. The prefusion processing stage or the fusion algorithm could modify the sensor state. During processing of an image, it may be determined that the image is too dark in an interesting region, and the camera should increase its sensitivity. The fusion program could also direct a sensor to examine a troublesome area or object, modify the prefusion processing, or change the data processing priorities or the data resolution. Apparently, the programs can interact considerably.

### 3.5 The Proposed Algorithms

We can compare the algorithms that we propose for map building with our model of sensor fusion processing. Clearly, the sensors are the camera and the laser
rangefinder. The description that the system builds, \( d \), is the digital map. This map does indeed contain less information than the surface description in the world space. The ideal description, \( d_R \), is just the digital map with the map cells correctly marked as containing an obstacle or not. We can also categorize the algorithms. The program \( vLabel \) is clearly prefusion processing. The first part of \( update\_map_1 \) is also part of the prefusion stage because it does coordinate transformations. The fusion stage is harder to identify; no one program takes the data streams from the two sensors and merges them. The data are merged in a very indirect fashion. 

*Indirect fusion* occurs when one sensor data stream influences the processing of a second sensor stream. Symmetrically, *direct fusion* occurs when two sensor data streams are combined into a single data stream. The difference is one of combining vs. influencing. Clearly, the video data and the rangefinder data are so different that directly combining them is difficult to conceptualize. Direct fusion occurs when the two sensor streams are similar, such as in stereo vision. The second part of \( update\_map_1 \) and all of \( update\_map_2 \) are part of the fusion stage because they build the description (the map). The programs \( analyze\_map \) and \( scan\_map \) are also part of the fusion stage because they send information back to the sensors. Figure 3.4 illustrates these classifications.
Another aspect of the model is the use of symbolic data. The proposed algorithms do use symbolic data; the segment numbers and the classification of map cells into "floor" or "not-floor" are symbolic. This use of symbolic data is unavoidable — the very output of the system, the map, contains symbolic data. Lastly, the algorithms avoid the data association problem because they do not build a description in terms of objects. The laser rangefinder data that belong to the same world feature are combined based on proximity of the measurements, given that the measurements have full three-dimensional information.
Figure 3.4: Comparison of the proposed software and the model.
Chapter 4

Algorithms for Fusing the Sensor Data

Recall that we do not directly fuse the video images and the laser rangefinder data. Rather we use the video data or a stored polygonal map to direct the laser rangefinder. To avoid overly complex and lengthy phrases, we term a scan path computed from a segmented image a \textit{segment vector scan}. Similarly, we term a scan path computed from the polygonal map a \textit{map vector scan}. This does, however, lead to the rather curious phrase: "build a map from a map vector scan." The reader should understand that the system is building a digital map, but that the scan is made from the polygonal map — two entirely different entities. To compute the segment vector scans, we segment the image, pick the central pixels of each segment, fit them with a scan line, and link the scan lines with a greedy algorithm. To compute the map vector scans, we image the footprint vertices in the map, choose two from each footprint to make a scan line, and then link the scan lines with a greedy algorithm. The map building techniques are equivalent to two-dimensional histogram density estimation, followed by integration in an uncertainty region around each point and thresholding. This chapter begins with a brief description of the segmentation program, explains the vector scans, and finally describes the map building techniques. Figure 4.1 shows a detailed diagram of the data flow in the system. This figure shows all the programs and data files, thus detailing the inputs and outputs for the programs. Table 4.1 gives a brief description of the data files. The following sections discuss the files (and the programs) more fully.
Figure 4.1: Detailed software data flow.
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aoi.asc</td>
<td>AOI for each footprint</td>
</tr>
<tr>
<td>map.asc</td>
<td>polygonal obstacle footprints and heights</td>
</tr>
<tr>
<td>map.dat</td>
<td>histogram of the dilated measurements</td>
</tr>
<tr>
<td>map-decide.dat</td>
<td>binary map with object regions</td>
</tr>
<tr>
<td>map-free.dat</td>
<td>binary map of free space</td>
</tr>
<tr>
<td>position.asc</td>
<td>rotation matrix $A$ and translation vector $t'$</td>
</tr>
<tr>
<td>scan.asc</td>
<td>scan path for the laser</td>
</tr>
<tr>
<td>scene.asc</td>
<td>surfaces in the environment</td>
</tr>
</tbody>
</table>

Table 4.1: Data files. The file extension "asc" means the file is ascii and the file extension "dat" means the file is binary.

Figure 4.1 shows two types of maps: digital (map.dat) and polygonal (map.asc). In the file map.dat, map cells indicate whether or not the corresponding floor area is clear or occupied by an obstacle. In map.asc, polygons are used to represent the footprints of obstacles. Additionally, map.asc contains an estimate of the maximum height of each obstacle. These two maps are connected in Figure 4.1 with a dashed line, which indicates that the two maps are related. Transforming between the two representations can be done although we have not done so in this work. A digital representation can be generated from the polygonal one by using a polygonal scan line algorithm [45, p. 98]. We have written a routine, fill_polygon, that implements this algorithm. Generating a polygonal representation from a digital one has been discussed extensively in the pattern recognition literature [46, 47, 48].

4.1 Segmenting the Camera Images

The segmentation program vlabel was taken from the Khoros image processing package from the University of New Mexico [49]. Compared to other segmentation algorithms, vlabel is straightforward. The first segment is begun by choosing a seed point. All pixels neighboring the seed point are examined. If their grey value
lies in a certain window, they are included in the segment. The window is defined as a symmetric interval around the grey value of the seed pixel, and is fixed once a seed pixel is chosen. Pixels neighboring the segment are examined until none are in the grey value window. The next segment is grown by picking a new seed pixel and recursively checking neighbors. This operation continues until all pixels belong to a segment. Segments that are too small are merged into a nearby segment. Figure 4.2 shows the grey levels of a (very) small image. Figure 4.3 shows how this image would be segmented assuming that the upper left and right pixels are chosen as the seed pixels and that the grey level window is ±3.

\[
\begin{array}{cccc}
10 & 9 & 4 & 2 \\
12 & 7 & 1 & 2 \\
13 & 3 & 2 & 3 \\
\end{array}
\]

**Figure 4.2:** Example mini-image.

\[
\begin{array}{cccc}
2 & 2 & 1 & 1 \\
2 & 2 & 1 & 1 \\
2 & 1 & 1 & 1 \\
\end{array}
\]

**Figure 4.3:** Segmentation of the image given in Figure 4.2. The segment numbers are arbitrary.

### 4.2 Generating the Vector Scans

To maximize our chances of finding new objects in the environment, we would like the laser beam to sweep out new regions as quickly as possible. Thus, we would like
to maximize the solid angle that the scan path subtends. This solid angle is equal to the area of the set given by the dilation

$$v \oplus b_s$$

(4.1)

where $b_s$ is a ball that is the same size as the laser beam ($s$ is the spot radius) and $v$ is the scan path. Because we are interested in solid angles, this path must be given on the surface of a unit sphere. In the mathematical literature, this type of set is known as a tube. From Hotelling [52], the area of a tube on a hypersphere depends only on the path length and the ball radius, as long as the path curvature is less than $1/s$, and the path does not loop back over itself. If the path curvature is too high, the laser beam overlaps already scanned areas. Because we want a path that is not often highly curved, we make the scan path from a sequence of straight line segments (scan lines).

4.2.1 Map Vector Scans

The programs analyze_map and scan_map compute the map vector scans. For each footprint in the polygonal map, analyze_map computes an area of interest (AOI). The AOI is a camera view of the volume that the map object occupies, as constrained by the footprint and the height. Figure 4.4 shows an example of a footprint and an AOI. The scan path for an AOI is found by choosing the two vertices with largest and smallest $y''$. This choice guarantees that we get maximum $y''$ movement in the scan line. For any choice that we make, the scan could miss the object because we do not know its exact shape. The system can detect a miss because no range data projects to the current footprint. In case of a miss, a more complex scan pattern would be needed. For the experiments that we performed, this rescanning did not prove necessary. Determining the order to visit the line scans is equivalent to a
traveling salesman problem and is solved with a greedy algorithm. Tables 4.2 and 4.4 show the pseudo-code for `analyze_map` and `scan_map`. Steps 2–4 of `analyze_map` are identical to drawing the image of a three-dimensional polygon, which is discussed in any good computer graphics book [45, sections 3.14 & 6.2].

![Footprint and AOI](image)

**Figure 4.4**: Examples of a footprint and an AOI. The vertical lines that are dashed in the AOI show the connection between the upper and lower pieces as described in Table 4.3.

```plaintext
analyze_map
1  for each footprint in map.asc
2       clip the footprint to the front of the camera
3       image the clipped footprint (equation (2.4))
4       clip the resulting polygon to the viewport
5       ma_extend /* incorporate height information */
6       write the AOIs to aoi.asc
```

**Table 4.2**: Partial pseudo-code for `analyze_map`. 
ma_extend
1 find the two vertices with the smallest and largest x values
2 keep the lower polygonal piece between these vertices
3 for each vertex in the upper piece
4 add (0,0,height) to the vertex (in the map frame)
5 image this new point and keep as a vertex of the AOI

Table 4.3: Pseudo-code for ma_extend.

scan_map
1 for each aoi in aoi.asc
2 set the scan line to be the highest and lowest vertices
3 link the scan lines with a greedy algorithm
4 write the scan path to scan.asc

Table 4.4: Pseudo-code for scan_map.

4.2.2 Segment Vector Scans

The program scan_seg computes the segment vector scans. The first step is to compute a scan line for each segment, which is similar to path planning once we pick entry and exit points. By definition, each segment is connected so we are guaranteed that a path through the segment does exist. However, path planning finds a short path between two given points; we want to find a long straight line inside the segment. Therefore, we fit a straight line through the central pixels in the segment. The intersection of this scan line with the segment boundary gives the entry and exit points. We use only the central pixels so that we can ignore small peninsulas off the main body of the segment, and have a better chance of fitting a scan line. However, this method can fail for sufficiently nonconvex segments. Experience has shown that this method almost always works for the type of images generated in the lab. To find these central pixels, we use a distance transform [41]
of the segment. The distance transform for a pixel in the segment gives the shortest $l_1$ or city-block distance to the segment boundary. Figure 4.5 gives an example set and Figure 4.6 gives the associated distance transform.

We set the central pixels to be those with distances in the interval $[d_{\text{min}}, d_{\text{max}}]$, where $d_{\text{max}}$ is the maximum distance in the segment and

$$d_{\text{min}} = \max(2, d_{\text{max}} - 3).$$ (4.2)

Thus, we are ignoring any small peninsulas that are attached to the main body of the segment. In fact, we are using the middle of the fattest part of the segment. We fit a line through these central pixels using ordinary least squares [53, p. 590], which means that we must choose either $x''$ or $y''$ to be the error free variable. A poor choice can lead to the fitted line being horizontal while the central pixels are nearly on a vertical line. We considered using total least squares because it is invariant under the choice of coordinate axis, but it is considerably more expensive computationally and thus was not used. Coordinate choice is made by examining the bounding rectangle for the segment and picking the larger side as the error-free variable.

```
1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1
1 1 1 1
```

Figure 4.5: Indicator function for a set.

Rosenfeld [50] presents the algorithm for computing the distance transform is from Rosenfeld This implementation computes the transform in two passes over the pixels and is particularly efficient [51]. In order to compute the transform of a set,
the pixels in the set must have their grey value set to one, and all other pixels must be set to zero. We use \( a_{i,j} \) (1 \( \leq i \leq m \) and 1 \( \leq j \leq n \)) to denote the current computation of the distance at the \( i,j \)th pixel. The distance transform is computed by the application of \( f_1 \) in the forward raster direction and then \( f_2 \) in the reverse raster direction. These operators are given by

\[
\begin{align*}
    f_1(a_{i,j}) &= \begin{cases} 
    0 & \text{if } a_{i,j} = 0 \\
    \min(a_{i-1,j} + 1, a_{i,j-1} + 1) & \text{if } (i,j) \neq (1,1) \text{ and } a_{i,j} = 1 \\
    m + n & \text{if } (i,j) = (1,1) \text{ and } a_{1,1} = 1
    \end{cases} \\
    f_2(a_{i,j}) &= \min(a_{i,j}, a_{i+1,j} + 1, a_{i,j+1} + 1).
\end{align*}
\] (4.3) (4.4)

We want to be able to check the scan line that we compute. For non-convex shapes, the scan line could be quite short and thus not be a reasonable sampling of the segment. Among the convex regions with a particular area, a circle has the shortest best scan line. By best scan line, we mean the longest line segment that is contained in the region. For a circle, the best scan line is length

\[
l = 2\sqrt{\frac{A}{\pi}}
\] (4.5)

where \( A \) is the area. Notice that the area is proportional to the number of pixels, which we know for each segment. Thus, we keep only reasonable length scan lines with

\[
l > \frac{\sqrt{n_p}}{2.5}
\] (4.6)
with \( n_p \) denoting the number of pixels in the segment.

Now we must decide the order to visit the scan lines; we would like to visit the most important segments first. To accomplish this goal, each segment is assigned a priority based on the following guidelines: segments in the lower part of the image are likely to be closer to the current position, and segments that are near the center of the image are likely to be near the robot path. In either case, the segment is important, and thus should have a higher priority. Figure 4.7 shows the priority polygons. The polygons were chosen based on the camera orientation for the experiments, but could be computed dynamically. Segments may be in more than one priority polygon. The segment is assigned the highest priority of these polygons. To compute the visiting order, the segments in each priority class are essentially treated separately. The classes are separated, without modifying the greedy algorithm, by using an effective distance which is computed by

\[
\hat{d} = a_r 10^{p_{\text{max}} - p}
\]  

(4.7)

where \( a_r \) is the \( l_\infty \) angular distance given by equation 4.10, \( p_{\text{max}} \) is the maximum priority, and \( p \) is the priority of the destination. Table 4.5 gives the pseudo-code for \texttt{scan\_seg}. Step 2 is done with a call to \texttt{fill\_polygon}, which uses a polygon scan line fill algorithm [45, p. 98].

### 4.2.3 Timing the Laser Scans

Timing the laser scans is a matter of two factors: how quickly the mirrors move and how long the laser must dwell on each pixel. Note that although we use the term dwell, the mirrors do not stop at each pixel. They move smoothly from the start to the stop pixel. Information about galvanometers can be found in [55] [54]. Montagu [54, Table 9] gives the step response times for two galvanometers made by General
Figure 4.7: The priority polygons. Darker regions are more important.

```plaintext
scan_seg
1    for each priority polygon
2        for each pixel in the polygon
3            determine the segment number for the current pixel from seg.dat
4            update the priority for the current segment
5            update the bounding box for the current segment
6        for each segment
7            compute the distance transform (equation (4.4))
8            compute the scan line using least squares
9            if the scan line is long enough (equation (4.6))
10               save the scan line
11        link the saved scan lines with a greedy algorithm
12    write the scan path to scan.asc
```

Table 4.5: Pseudo-code for `scan_seg`. 
Scanning, the G120D and the G325D. Figure 4.8 summarizes these data. We use the data from the G325D, the one having the larger mirror. The data are neatly interpolated by the straight line

\[ t = 0.1052a_m + 2.75 \]  \hspace{1cm} (4.8)

where \( t \) is the step time in milliseconds and \( a_m \) is the mechanical step angle in degrees. If the galvanometer moves one mechanical degree, the laser beam scans two degrees. Converting to scanned angle in radians, we have

\[ t = 3.014a_r + 2.75. \]  \hspace{1cm} (4.9)

If we move from \((x_0^\prime, y_0^\prime)\) to \((x^\prime\prime, y^\prime\prime)\), the \( l_\infty \) angular distance is given by

\[ a_r = \max \left( \left| \tan^{-1} \frac{x^\prime\prime}{f_x} - \tan^{-1} \frac{x_0^\prime}{f_x} \right|, \left| \tan^{-1} \frac{y^\prime\prime}{f_y} - \tan^{-1} \frac{y_0^\prime}{f_y} \right| \right). \]  \hspace{1cm} (4.10)

The \( l_\infty \) or max distance is used because the scan time is determined by the largest of the distances that the two mirrors must move (the mirrors move independently). Additionally, a dwell time of 0.025 ms/pixel is added to ensure that our equation for the step response time does not allow the beam to be scanned too quickly, which would increase the range errors (see equation (2.7)). This dwell time is consistent with the scan speeds of commercial rangefinders, and is discussed more fully in Appendix D.

4.3 Building the Map

Our goal is to estimate the set

\[ o = \{ (k, l) : \text{an obstacle is in or above the} \ (k, l) \ \text{map cell} \}. \]  \hspace{1cm} (4.11)

This set is the points that the robot must avoid. We call a point on some surface that the laser ranger samples a sample point. Similarly, an object sample point is a
Figure 4.8: Step response time of two galvanometers. The step angle is given in mechanical degrees. The step time includes settling time for an accuracy of 0.21°. The asterisks are for the G120D and the circles are for the G325D. The G120D is connected to a 7 x 7 mm mirror, and the G325D is connected to a 26 x 26 mm mirror.

sample point that is not on the floor. We cannot estimate the position of each object sample point; they are too close together to separate. Instead, we estimate the set 0 using the steps shown in Figure 4.10. These steps are described in considerable detail in the following sections. Briefly, the measurements are converted to Cartesian, the “floor” measurements are discarded, and the remaining ones are accumulated in the dilated histogram. This dilated histogram is thresholded to provide the estimate of the obstacle regions.

The \( (x, y) \) pairs for a particular object sample point are distributed according to some distribution. Recall from section 2.2, the only random variable is \( r \). An actual rangefinder would also have inaccuracies pointing the laser. Let the distribution for the \( i \)th object sample point be denoted \( f_i(x, y) \). The \( (x, y) \) pairs for all the object
sample points are distributed by the sum of these distributions, denoted

\[ f(x, y) = \sum_i f_i(x, y). \]  \hspace{1cm} (4.12)

Note that \( f \) is not a density because it does not integrate to one; we call it a pseudo-density. A key step in estimating the obstacle regions is to construct a histogram estimate of this pseudo-density. By using multiple scans this way, we are able to achieve some noise rejection. Even though \( f \) is a sum of densities, we can still estimate it with a histogram, because the histogram is constructed by a summing process. Thus we are just exchanging independent sums

\[ \text{histogram}(f) = \text{histogram} \left( \sum_i f_i \right) = \sum_i \text{histogram}(f_i). \]  \hspace{1cm} (4.13)

The histogram estimate of \( f \) approximates \( f \) [56] in the sense that

\[ E\{\text{histogram}(f)(k, l)\} = \int_{cell,k,l} f(x, y). \]  \hspace{1cm} (4.14)

For each map cell, some number of object sample points project to that cell. This number can vary widely. Some cells that are far from the scan path can have no object sample points that project to them. Other map cells that are near the vertical face of an object can have several object sample points that project to them, particularly if the scan path vertically climbs the face. Figure 4.9 illustrates an example situation.

4.3.1 Estimating the Obstacle Regions

First, we must express the rangefinder measurements in the map frame. The measurements are given in range-pixel coordinates and are transformed to the camera frame by

\[ w'_z = \frac{r}{\sqrt{(\frac{x''}{f_x})^2 + (\frac{y''}{f_y})^2 + 1}} \]
Figure 4.9: Projection of object sample points to the map cells. When the laser scans vertically, more object sample points project to the corresponding map cell.

\[
\begin{align*}
w'_x &= \frac{x''}{f_x} w'_z \\
w'_y &= \frac{y''}{f_y} w'_z.
\end{align*}
\]  
(4.15)

Transforming to the map frame, we have

\[
w = A^T(w' - t').
\]  
(4.16)
Figure 4.10: Steps to build the map. Each step is described in the text.
The digital map is defined on a rectangular lattice of points in the map $xy$ plane. Each lattice point defines the center of a map cell. This lattice is shown in Figure 4.11. The origin of the lattice is given by $(v_x, v_y)$, and the lattice spacing is $\zeta$. A point in the map $xy$ plane is quantized to a point in the lattice with

$$(w_x, w_y) \mapsto \left(\left\lfloor \frac{w_x - v_x}{\zeta} + .5 \right\rfloor, \left\lfloor \frac{w_y - v_y}{\zeta} + .5 \right\rfloor\right)$$

(4.17)

where $[x]$ is the largest integer that is less than $x$. The lattice points are referenced by integers like $(k, l)$, which refers to the lattice point at the map coordinates given by

$$(k, l) \mapsto (k\zeta + v_x, l\zeta + v_y).$$

(4.18)

![Figure 4.11: The map lattice. The lattice points are shown at the center of the map cells.](image)

In the sequel, $j$ indexes the rangefinder measurements, so that $w(j)$ denotes the $j$th measurement expressed in the map frame. We need to decide if each measurement is from an obstacle or from the floor. To choose, we use a threshold test on
the $z$ component of the measurement in the map frame. We refer to this process as labeling the measurement as obstacle or floor. We denote this labeling process by $L$, defined as

$$L_w = \begin{cases} \emptyset & \text{if } w_z < 3\sigma_z \\ \{(\bar{w}_x, \bar{w}_y)\} & \text{otherwise} \end{cases} \quad (4.19)$$

where $\sigma_z$ is an estimate of the standard deviation for $w_z$ and $(\bar{w}_x, \bar{w}_y)$ is the map lattice point corresponding to $(w_x, w_y)$ from equation (4.17). We notice that

$$\frac{1}{n_{\text{scan}} a_{\text{cell}}} \sum_{j=1}^{n_{\text{sum}}} I(L_w(j))(k, l) \approx \text{histogram}(f)(k, l) \quad (4.20)$$

where $a_{\text{cell}}$ is the area of a map cell, $n_{\text{scan}}$ is the number of vector scans that the laser makes, and $n_{\text{sum}}$ is the number of measurements in these vector scans. The factor $1/n_{\text{scan}}$ appears because each object sample point is sampled $n_{\text{scan}}$ times and thus each density, $f_i$, would have $n_{\text{scan}}$ points in its histogram. Thus, the above equation approximates the histogram estimate of $f$, with two differences. First, the end of the distribution is clipped off when it goes below the $L$ threshold. Figure 4.12 shows this clipping. Secondly, we also allow some extra points: floor cells where $w_z$ randomly exceeds $3\sigma_z$.

Unfortunately, each of the densities in $f$ have varying mean and standard deviation. The means differ because the object sample points are in different places. The standard deviations differ because the object sample points are at different ranges and their surface patches are at different orientations. Therefore, the densities have different heights and widths. These differences are crucial because the final stage of estimating the obstacle regions is a thresholding stage. If we threshold the histogram of $f$ directly, we penalize unnecessarily the object sample points with larger standard deviation and correspondingly lower height densities. Integrating $f$ compensates for these height differences, as we shall see. If we dilate the measurements
by the uncertainty region $u(j)$, we have
\[
\frac{1}{n_{\text{scan}}} \sum_j I(Lw(j) \oplus u(j))(k, l) \approx \sum_i \int_{u(i)} f_i(x, y).
\]  
(4.21)

The uncertainty region is defined in the next section; it is basically a rectangle with dimensions $2\sigma_r \times$ the beam width. The above equation assumes that the uncertainty region for the $j$th sample point, $u(j)$, is essentially the same for each scan. Thus, the dilation corresponds to integrating the pseudo-density in the uncertainty region around each point. This integration equalizes the densities in $f$ because the low broad densities are integrated over larger regions and the narrow tall densities are integrated over smaller regions.

We need one final operator, $D$. It compares the argument function to a threshold and is defined as
\[
Dg(x, y) = \begin{cases} 
1 & \text{if } g(x, y) \geq \kappa n_{\text{scan}} \\
0 & \text{otherwise}
\end{cases}.
\]  
(4.22)
In our usage, $D$ compares the count in a map cell to $\kappa n_{\text{scan}}$. The threshold increases as we make more scans, which compensates for the spreading of the measurements. We compute the obstacle regions with

$$I(o)(k, l) \approx D \sum_{j=1}^{n_{\text{sum}}} I(L w_j \oplus u(j))(k, l).$$

The sum gives some immunity to noise. However, if we encounter an unscanned map cell after some number of scans, we must make an equal number of additional scans before the new cell can be labeled as an obstacle. We thus have a trade off between responsiveness and noise immunity.

The file `map.dat` contains the resulting computation from

$$\sum_{j=1}^{n_{\text{sum}}} I(L w_j \oplus u(j))(k, l).$$

The file `map-decide.dat` contains the result from equation (4.23). The file `map-free.dat` contains a map showing the free space, which is the region that the robot is free to maneuver in. Tables 4.6 and 4.7 give the pseudo-code for `update_map1` and `update_map2`. In step 8 of `update_map2`, the arrays are treated as indicator functions for sets, with the dilation being performed on the sets. Robot-hexagon refers to an hexagonal approximation to the robot that is the same size as the robot. This dilation is done directly, although a more efficient implementation would be to take the distance transform of the complement of `map-decide[][]`. Free space would be those cells with distance greater than the robot radius.

### 4.3.2 The Uncertainty Region

The uncertainty region is computed as a rectangle in the map $xy$ plane. The estimate of $\sigma_r$ is given by equation (C.6)

$$\hat{\sigma}_r = \frac{k_\sigma}{\sqrt{a}}$$

(4.25)
update_map1
1   for each range measurement
2       convert from range-pixel to map coordinates (equations (4.15) & (4.16))
3       label the measurement as floor or obstacle (equation (4.19))
4       if obstacle
5           increment map cells in the uncertainty region
6       write map.dat

Table 4.6: Pseudo-code for update_map1.

update_map2
1   read map.dat into map[][]
2   for all k,l in the map
3       if map[k][l] < \kappa_{\text{scan}}
4           map-decide[k][l] = not-floor
5       else
6           map-decide[k][l] = floor
7   write map-decide[][] to map-decide.dat
8   map-free[][] = map-decide[][] @robot-octagon
9   write map-free[][] to map-free.dat

Table 4.7: Pseudo-code for update_map2.
When we run the software on the simulated laser rangefinder data, this estimate is exact because no noise is added to the amplitude. The uncertainty rectangle has sides $2\Delta x$ by $2\Delta y$. They are given by

$$\begin{align*}
\Delta x &= |\hat{\sigma}_r a_{31}| + \left| \frac{z'}{2f_x} a_{11} \right| + \left| \frac{z'}{2f_y} a_{21} \right| \\
\Delta y &= |\hat{\sigma}_r a_{32}| + \left| \frac{z'}{2f_x} a_{12} \right| + \left| \frac{z'}{2f_y} a_{22} \right|.
\end{align*}$$

(4.26)

The first terms in these equations represent range noise, and the second and third terms represent pixel quantization effects from pointing the laser. To derive the range error terms, we first note that the laser approximately points in the same direction as the camera $z$ axis. The camera $z$ axis expressed in the map frame is

$$A^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{31} \\ a_{32} \\ a_{33} \end{bmatrix}.$$ 

(4.27)

Thus, $a_{31}$ and $a_{32}$ are the map $x$ and $y$ components of the camera $z$ axis. Thus we have the first terms in equations (4.26). The second terms represent the error from quantization in the $x$ pixel, which is shown in Figure 4.13. Assuming that

$$w' = \begin{bmatrix} \frac{x''}{f_x} \\ \frac{y''}{f_y} \\ z' \end{bmatrix},$$

(4.28)

we have that

$$g' = \begin{bmatrix} \frac{(x''+1)z'}{f_x} \\ \frac{y''z'}{f_y} \\ z' \end{bmatrix}.$$ 

(4.29)
Thus,

\[ g' - w' = \begin{bmatrix} \frac{x'}{f_x} \\ 0 \\ 0 \end{bmatrix}. \tag{4.30} \]

The map \( x \) and \( y \) components of the camera \( x \) axis are \( a_{11} \) and \( a_{12} \). Thus we have the second terms in equations (4.26). The third terms represent the error from quantization in the \( y \) pixel, and are derived similarly.

![Diagram](image_url)

**Figure 4.13:** Effects of \( x \) pixel quantization on the range measurements.

### 4.3.3 Spreading in the Estimate of the Obstacle Regions

We can quantify the spreading of our estimate \( I(o) \) if we assume that the noise is Gaussian. The problem is essentially one-dimensional because the only real noise is in the range (not pointing angle). Let the dilation interval be \((-\bar{u}, \bar{u})\) and the
resulting estimate of the object be \((-t_0, t_0)\) (setting the origin to coincide with the object for ease of notation). We have

\[
\kappa = \int_{t_0 - \tilde{u}}^{t_0 + \tilde{u}} f(t) \, dt,
\]

\[
\kappa = \frac{n}{\sqrt{2\pi}\sigma} \int_{t_0 - \tilde{u}}^{t_0 + \tilde{u}} \exp\left\{\frac{-t^2}{2\sigma^2}\right\} \, dt,
\]

\[
\kappa = \frac{n}{2} \left\{ \Phi\left( \frac{1}{\sqrt{2}} \left[ \frac{t_0}{\sigma} + \frac{\tilde{u}}{\sigma} \right] \right) - \Phi\left( \frac{1}{\sqrt{2}} \left[ \frac{t_0}{\sigma} - \frac{\tilde{u}}{\sigma} \right] \right) \right\}
\]

where

\[
\Phi(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} \exp\left\{-t^2\right\} \, dt.
\]

A plot of \(n\) versus \(t_0\) is shown in figure 4.14. This calculation is an approximation.

![Figure 4.14: Number of object sample points versus \(t_0/\sigma\).](image)

We used the expression for \(f\) in the calculation, whereas the actual map uses the histogram estimate of \(f\). Equation 4.14 showed that this substitution is reasonable. The expected value of the area under the two curves is the same, except for quantization effects due to the map cells.
In this chapter, we briefly described the program that segments the video images, \texttt{vlabel}. This algorithm builds segments out of neighboring pixels that fit in a grey value window. We also gave the algorithms for the modules that compute the scan paths for the laser: \texttt{analyze\_map} and \texttt{scan\_map} compute the map vector scans, and \texttt{scan\_seg} computes the segment vector scans. These algorithms are given in Tables 4.2, 4.4 and 4.5. Lastly, we gave the algorithms for the modules that build the map: \texttt{update\_map1} and \texttt{update\_map2}. They compute equations (4.24) and (4.23) and are given in Tables 4.6 and 4.7.
Chapter 5

Results

To test the algorithms, we set up an example scenario with the Rice-obot. The Rice-obot is a mobile robot designed and built in Rice’s Electrical and Computer Engineering Department. The Rice-obot was positioned in a room along with some obstacles. We chose four positions for the robot that were along a plausible path for it. The images from these places were digitized and served as input to the algorithms.

The images were taken with a black and white Sony SSC-D5 CCD camera, using a D. O. Industries Navtron 8 mm lens. For these experiments, the images were digitized by a Truevision TARGA+ card in an IBM AT clone. The images were digitized at 512 × 400 pixels.

The experiment was performed in Abercrombie C131C; the floor tiles there form a natural coordinate system. The tiles are one foot square and are aligned with the walls of the room. We aligned the map coordinate axes with the tiles. We picked the origin of the map frame to be the projection to the floor of the first camera position.

The position of the boxes was measured relative to the floor tiles with resolution of about 1–2 mm. The position of the camera was measured by hanging a plumb bob from the camera and reading from a ruler on the floor. The measurement errors for the camera position were about 5 mm. The orientation of the camera was measured indirectly from the images by the program orient as described in Appendix A.2.
Figure 5.1 shows the positions of the Rice-obot and the three boxes that were used as obstacles. This figure also shows the direction that the camera pointed at each position. Recall that we assumed that the camera and the laser rangefinder were perfectly registered (pointing in the same direction). Table 5.1 shows the steps used to build a map using segment vector scans. At each of the four positions, the same scan pattern was repeated three times for a total of four scans. Repeating the scan pattern helps reduce the effect of the unequal scan problem, and saves computation time. The program `update_map1` has the capability to use the data that it computed and stored previously in the dilated histogram (`map.dat`). This capability allows it to accumulate data from different positions in the same histogram. Thus, we repeated steps one through eight for each position. The digital map was formed by running `update_map2` only once (step nine). No prior information was used to build this map. Table 5.2 shows the steps used to build a map using map vector scans. We used three scans from position one. The polygonal map that we used contained the footprints of the three boxes but did not include any of the walls of the room.

The box heights were given as 0.6 m, which is their true height.

1. Position the robot, measure its position, and capture the image.
2. Segment the image with `vlabel`.
3. Compute camera orientation with `orient`. The file `position.asc` is updated automatically.
4. Update `scene.asc`.
5. Run `sim_laser`.
7. Run `scan_seg`.
8. Run `update_map1`.

Table 5.1: Procedure to build a map using segment vector scans.
Figure 5.1: Experimental setup. The room is Abercrombie C131C. The three squares show the position of the three boxes used as obstacles. The arrows show the position and orientation of the camera at the four positions.
1. Update map.asc.
2. Ensure that position.asc is correct.
3. Run analyze_map. Select “save aoi” option.
4. Ensure laser_a.dat and laser_r.dat are correct.
5. Run scan_map.
7. Run update_map2.

**Table 5.2**: Procedure to build a map using map vector scans.

Could the Rice-obot do the scanning and process the data in real-time? The positions are 0.6–0.8 m apart, and the Rice-obot’s maximum velocity is about 3.5 m/s. Traveling at about 1/3 speed, the robot would take about 0.6 s between positions. This time is about the same that the three scans take; a single scan at the four positions took 0.16, 0.18, 0.13, and 0.14 s respectively. Table 5.3 shows the processing time required for the software modules when run on a Sun 3/60 and a Sun sparc 1+, neither of which is actually on the Rice-obot. For purposes of comparing to real-time, the run time of sim_laser does not matter. However, the other times show that the modules do not run in real-time. The most troublesome are vlabel at 26 s and scan_seg at 16 s. The modules could be speeded up by reducing the data size; both scan_seg and vlabel have run times that are proportional to the number of pixels in the image. Subsampling by two in both directions would achieve a speed up of a factor of four. Also, the software could be tightened. When we wrote it, we emphasized clarity and correctness over speed. No doubt, savings could be found. To preserve modularity, vlabel and scan_seg duplicate effort. The program vlabel segments all the pixels, but then does not give scan_seg any information about the segments other than the segmented image. The first thing that scan_seg must do is to find all the segments. Some savings could be realized by tightly coupling the two modules. Additionally, update_map2 could be speeded up.
by using a distance transform to compute the free space instead of using a dilation, as mentioned in section 4.3. Generally speaking, real-time image processing is so data intensive that it requires very fast hardware. Part of scan_segment would be amenable to parallel processing. The most compute intensive part is finding the scan line for each segment. Each segment could be assigned to a different processing element. Computer architecture for real-time image processing is an active research topic [57] [58].

<table>
<thead>
<tr>
<th>module</th>
<th>Sun 3/60</th>
<th>Sun sparc 1+</th>
</tr>
</thead>
<tbody>
<tr>
<td>vlabel</td>
<td>212.68</td>
<td>26.18</td>
</tr>
<tr>
<td>sim_laser</td>
<td>281.00</td>
<td>28.22</td>
</tr>
<tr>
<td>scan_segment</td>
<td>77.96</td>
<td>15.94</td>
</tr>
<tr>
<td>analyze_map</td>
<td>0.70</td>
<td>0.4</td>
</tr>
<tr>
<td>scan_map</td>
<td>1.18</td>
<td>-</td>
</tr>
<tr>
<td>update_map1</td>
<td>4.90</td>
<td>1.14</td>
</tr>
<tr>
<td>update_map2</td>
<td>26.92</td>
<td>9.50</td>
</tr>
</tbody>
</table>

Table 5.3: Run times for the software modules. The times are given in seconds, and were obtained by averaging five measurements from the UNIX time command. The times reported are the sum of the user and system times, and reflect the total CPU time used. The run times for scan_map on the sparc 1+ were too short to measure with this technique.

Figure 5.2 shows the video images at each of the four positions. The back wall shows considerable detail. The walls have a power strip running horizontally about one foot above the floor and unistrut grooves running vertically every four feet. These grooves are just less than one inch wide. Also on the back wall is a one by two board with two infrared emitter/detectors mounted on it. This board is just over four feet long and is mounted horizontally just above the power strip. The wiring going to the infrared sensor is also visible (the diagonal line going down to the right sensor from the unistrut groove). The images show only moderate
complexity for indoor images. The greater the image complexity, the more difficulty that the segmentation program has, and the longer that the resulting scans take. Outdoor scenes frequently contain extraordinary complexity, which is the reason for concentrating the algorithms for an indoor robot.

Figure 5.3 shows the segmentation of the images. They show the usual problems encountered in segmentation. Table 5.4 describes the 24 segments from the image at the first position. The segments were ranked into four high, four medium, and 18 low priority segments by scanseg. The left face of the right box and the front face of the middle box are grouped into one segment, segment 19. This merging happens because the two faces are almost the same grey level and they are contiguous in the image. Some objects are broken into multiple segments. For instance, the back wall is split into five segments: segments 1, 2, 3, 4, and 5. Additionally, vlabel produces an unusual segment that is exactly two pixels wide around the edge of the image: segment 0. This segment was produced because vlabel does not attempt to assign the border pixels into a real segment. For the back wall near the infrared sensors, vlabel computes many different segments because the shadowing effects there are quite complex (segments 7, 11, 13, 16, and 21). Of course, the range data shows this region to be indistinguishable from the back wall.

Recall that we are using a simulator to produce the laser rangefinder data because no suitable laser rangefinders were available. Figure 5.4 shows the simulated range and amplitude images from position one. The simulation has some artifacts because it has difficulty matching up the intersections of surfaces. For example, in Figure 5.5 the floor appears to extend somewhat beyond the back wall. Figure 5.6 shows some more examples. The artifacts are caused by the approximate imaging model and consequent inaccuracies registering the camera image and the environment model. These errors cause some pixels to be misclassified. The pixels in the
Figure 5.2: Video images. The images are from positions one through four going clockwise beginning at the upper left.
Figure 5.3: Segmented video images. The images are from positions one through four going clockwise beginning at the upper left.
<table>
<thead>
<tr>
<th>segment</th>
<th>p</th>
<th>$n_p$</th>
<th>bounding box</th>
<th>example point</th>
<th>grey value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4551</td>
<td>(0,0) (511,399)</td>
<td>(120,399)</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>106663</td>
<td>(2,128) (509,397)</td>
<td>(120,397)</td>
<td>183</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>9821</td>
<td>(2,2) (509,251)</td>
<td>(194,194)</td>
<td>214</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>8191</td>
<td>(114,100) (220,193)</td>
<td>(195,193)</td>
<td>91</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>244</td>
<td>(62,383) (93,397)</td>
<td>(86,383)</td>
<td>254</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1633</td>
<td>(356,120) (376,241)</td>
<td>(357,213)</td>
<td>173</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>19751</td>
<td>(367,132) (509,344)</td>
<td>(371,223)</td>
<td>193</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>481</td>
<td>(423,243) (509,346)</td>
<td>(439,270)</td>
<td>244</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>10180</td>
<td>(2,2) (109,97)</td>
<td>(2,2)</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>10303</td>
<td>(115,2) (233,100)</td>
<td>(115,2)</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>9367</td>
<td>(240,2) (361,89)</td>
<td>(241,2)</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>15650</td>
<td>(364,2) (509,138)</td>
<td>(369,2)</td>
<td>40</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>767</td>
<td>(476,2) (496,118)</td>
<td>(490,2)</td>
<td>224</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>469</td>
<td>(339,43) (359,91)</td>
<td>(359,43)</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>818</td>
<td>(133,88) (221,98)</td>
<td>(146,88)</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>214</td>
<td>(257,98) (310,102)</td>
<td>(257,98)</td>
<td>71</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>711</td>
<td>(2,100) (124,107)</td>
<td>(30,100)</td>
<td>81</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>206</td>
<td>(226,104) (260,109)</td>
<td>(226,104)</td>
<td>101</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>692</td>
<td>(257,105) (384,113)</td>
<td>(271,105)</td>
<td>112</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>602</td>
<td>(2,107) (126,112)</td>
<td>(6,107)</td>
<td>122</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>925</td>
<td>(2,111) (130,191)</td>
<td>(2,111)</td>
<td>234</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>339</td>
<td>(223,113) (328,117)</td>
<td>(224,113)</td>
<td>132</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1624</td>
<td>(366,114) (485,133)</td>
<td>(366,114)</td>
<td>142</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1276</td>
<td>(2,115) (124,126)</td>
<td>(2,115)</td>
<td>152</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1013</td>
<td>(225,119) (331,132)</td>
<td>(225,119)</td>
<td>163</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>861</td>
<td>(459,146) (509,179)</td>
<td>(459,146)</td>
<td>203</td>
</tr>
</tbody>
</table>

Table 5.4: The segments from position one. The segment numbers are those assigned by v1label. Recall that $p$ is the priority and $n_p$ is the number of pixels in the segment. The bounding box is given by its upper left and lower right corners. All points are given in image pixels.
laser range image are essentially classified when the simulator decides that they belong to a particular segment or polygon. Figure 5.7 shows a simplified view of how the artifacts occur. In actual practice, this misclassifying occurs over some 5-10 pixels. These artifacts are impossible to eliminate because the file scene.asc does not specify the intersections of surfaces. To eliminate the artifacts, we could use a ray tracing algorithm with a more sophisticated surface model that specifies all the surface intersections. Foley et. al. describe this algorithm [45, p. 701] and a suitable polyhedral model [45, p. 545]. These modifications would solve the problems because the polyhedral representation explicitly contains the surface intersections and because we would not rely on approximate registration between the camera image and the environment model. Wu used this approach for his laser rangefinder simulation [35].

Figure 5.8 shows the vector scans that were computed from the segmented images. We extensively describe the scan computed from the first position; the other scans had similar properties. Table 5.5 shows the details for this scan. The scan time was estimated to be 0.16 s. Because the left face of the right box and the front face of the middle box were grouped together into a single segment (segment 19), one scan line was computed for both faces. Fortunately, the scan line adequately sampled both faces. Three segments were rejected for having too short scan lines. One was the narrow segment that is around the outside edge of the image (segment 0). Another was from the bottom part of the power strip and the left edge of the left box (segment 23). This segment had 925 pixels but the scan line had only three. The last rejected segment (segment 21) came from the remaining edges of the left box, the edges of the middle box, the top edges of the right box, the board, and the left three unistrut grooves. This segment had 9821 pixels but the scan line had only 29. This segment had a rather unusual shape.
Figure 5.4: Laser images and map vector scan from position one. The upper left image is the laser range image and the upper right one is the laser amplitude image. For the range image, the light pixels are farther away. The lower image shows the map vector scan.
Figure 5.5: Range data from a raster scan at position one. This figure includes data labeled as "floor". Notice how some floor points extend beyond the back wall.

Figure 5.6: Range data from a raster scan at position one, excluding "floor" data. Data from different surfaces do not meet at the edges. The left box clearly illustrates this overlapping. The three dotted horizontal lines are from the top, the single solid horizontal line is from the front, and the solid vertical line broken in two is from the right surface.
Figure 5.7: Source of the artifacts in the simulator. The figure shows the range as it is computed at three pixels. The lower two pixels are classified as floor pixels and the top pixel is classified as a wall pixel. The middle pixel is misclassified and thus the range is erroneous.
Figure 5.8: Segment vector scans. The scans are from positions one through four going clockwise beginning at the upper left. The red lines indicate the lines that scan a segment and the blue lines are the linking lines.
<table>
<thead>
<tr>
<th>segment</th>
<th>p</th>
<th>entry point</th>
<th>entry time</th>
<th>exit point</th>
<th>exit time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>(220,164)</td>
<td>2.9</td>
<td>(132,128)</td>
<td>6.7</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>(4,272)</td>
<td>10.1</td>
<td>(436,272)</td>
<td>17.5</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>(456,304)</td>
<td>20.3</td>
<td>(464,316)</td>
<td>23.2</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>(468,316)</td>
<td>26.0</td>
<td>(416,136)</td>
<td>30.7</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>(368,128)</td>
<td>33.6</td>
<td>(364,228)</td>
<td>37.4</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>(88,388)</td>
<td>41.4</td>
<td>(80,392)</td>
<td>44.3</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>(228,124)</td>
<td>48.1</td>
<td>(324,124)</td>
<td>51.9</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>(308,100)</td>
<td>54.8</td>
<td>(284,100)</td>
<td>57.8</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>(272,108)</td>
<td>60.6</td>
<td>(360,108)</td>
<td>64.2</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>(376,120)</td>
<td>67.0</td>
<td>(464,124)</td>
<td>70.6</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>(480,116)</td>
<td>73.4</td>
<td>(492,4)</td>
<td>77.4</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>(480,56)</td>
<td>80.4</td>
<td>(368,56)</td>
<td>84.3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>(356,56)</td>
<td>87.0</td>
<td>(348,88)</td>
<td>90.2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>(352,44)</td>
<td>93.1</td>
<td>(244,44)</td>
<td>97.0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>(232,44)</td>
<td>99.8</td>
<td>(120,44)</td>
<td>103.8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(108,48)</td>
<td>106.6</td>
<td>(4,48)</td>
<td>110.5</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>(32,104)</td>
<td>113.6</td>
<td>(120,104)</td>
<td>117.3</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>(112,112)</td>
<td>120.1</td>
<td>(4,108)</td>
<td>124.1</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>(4,120)</td>
<td>126.9</td>
<td>(124,120)</td>
<td>131.0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>(144,92)</td>
<td>133.9</td>
<td>(216,92)</td>
<td>137.5</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>(228,108)</td>
<td>140.3</td>
<td>(260,108)</td>
<td>143.4</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>(272,116)</td>
<td>146.2</td>
<td>(224,116)</td>
<td>149.4</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>(476,156)</td>
<td>153.2</td>
<td>(508,160)</td>
<td>156.2</td>
</tr>
</tbody>
</table>

**Table 5.5:** Segment vector scan from position one. The times are given in ms and the points are in image pixels.
A surprise was the cross shaped segment in the lower left of the image (segment 25) that corresponded to tape on the floor used as a marker in other experiments. This sixth segment was scanned at about 0.04 s. Incidentally, this segment was the last of the medium priority segments to be scanned. This sequencing illustrates the capability of the system to find potentially important objects and scan them quickly. Because the tape was on the floor, all the range measurements were successfully rejected by the labeling process.

Figure 5.9 shows a contour plot of the number of sample points that project to each map cell. An interesting feature of this plot is that it has many spikes. When the laser scans vertically, the number of sample points is quite large. At other times, the number of sample points is small. The maximum number of sample points that project to a single map cell is 15.

Figures 5.10 and 5.11 show the range data from the first two positions. Some mislabeled points are visible near the boxes. An interesting feature of the data is the way that the right box seems stretched towards the middle box. This apparent stretching is an artifact in the laser rangefinder simulation, as we discuss above.

Figure 5.12 shows the dilated histogram derived from the range measurements at the four positions. Figure 5.13 shows a contour plot of the same histogram. The mislabeled points are still visible. The histogram values range up to 255. Figure 5.14 shows the map derived from these data. The mislabeled points have been successfully rejected. The far right part of the wall is also interesting. The obstacle region that was computed there is much thinner than that computed for the rest of the wall. Only the third position viewed this part of the wall. Thus, this region has about one fourth of the data that the rest of the wall has. Some typical values for the dilated histogram in this region are \{4, 5, 6, 8, 24\}. Many of these map cells did not qualify as obstacle regions even though they would have qualified using three
Figure 5.9: Contour plot of the number of object sample points that project to each map cell. Taken from a single scan at position one. The scan was as shown in Figure 5.8.

Figure 5.10: Range data from three segment vector scans at position one. Data that were labeled as "floor" were not included.
Figure 5.11: Range data from three segment vector scans at position two. Data that were labeled as "floor" were not included.

scans at the third position. We call this the unequal scan problem. This problem can occur any time we merge data from different positions that have significantly different views. Translation of the sensors is not as much of a problem as rotation. Figure 5.15 shows the free space for maneuvering during a path execution. The figure shows that the region between the left and middle boxes is too small for the Rice-obot to fit. However, the area behind the boxes is traversable. The maps show that we were able to successfully derive the gross structure of the obstacles in the environment in spite of segmentation errors and the range noise that causes mislabeling.

To further quantify the performance of the map building algorithm, we statistically analyzed the errors in this map (the digital map from the segment vector scans). We analyzed the left main part of the back wall because it is straight, it had no simulation artifacts, and it did not have the unequal scan problem. We generated
Figure 5.12: Dilated histogram derived from the segment vector scans. This figure is an image of map.dat. We used three scans at each of the four positions. The darker pixels show regions with larger histogram value. The green lines indicate the true positions of the boxes and the back wall. These lines were overlayed manually to assist interpretation of the figure.
Figure 5.13: Contour plot of the dilated histogram from Figure 5.12.
Figure 5.14: Map derived from the histogram in Figure 5.12. This figure is an image of map_decide.dat. The lines were overlayed manually to assist interpretation of the figure.
Figure 5.15: Free space calculated from the map in Figure 5.14.
15 random $x$ values that were in the range of this part of the wall. For each $x$ value, we measured the minimum and maximum $y$ of the map representation of the wall along the line given by the particular $x$ value. We then computed

$$y_a = \frac{y_{\text{max}} + y_{\text{min}}}{2} - 5572$$

$$y_d = y_{\text{max}} - y_{\text{min}}$$

(5.1)

where 5572 is the $y$ position of the back wall. Figures 5.16 and 5.17 show histograms of $y_a$ and $y_d$. The mean and standard deviation of $y_a$ was 5.2 and 6.4 mm. The mean and standard deviation of $y_d$ was 114 and 28 mm. Recall that the map resolution size was 15 mm. The standard deviation of the range measurements from the back wall was 27.9 to 32.9 mm at position one and 18.8 to 28.9 mm at position four. Using all 12 scans, the maximum number of sample points that projected to a map cell was 146. Using this value for $n$ in equation 4.33, the expected value for the spreading is $t_0 = 3.7\sigma$. Taking the values for $\sigma_r$ above, implies that we expect $y_d$ in the range $[139, 242]$ mm, which corresponds reasonably to the average $y_d$ of 114 mm given that the dilated histogram was constructed with only 12 scans.

Figure 5.18 shows the AOIs from the first position. Recall that the AOIs are computed from the stored polygonal map. Figure 5.4 shows the vector scan computed from this AOI. Table 5.6 describes this scan, which takes a rather quick 0.02 s. The speed is unsurprising because we told the robot where all the obstacles are. Figure 5.19 shows the dilated histogram from three scans of this type. There were not any mislabeled points in the histogram. Figure 5.20 shows the map derived from this histogram. The map clearly shows all the boxes, and as well the central part of the back wall. The back wall is a bonus that we found because the laser rangefinder continues to collect data even between the AOIs.
Figure 5.16: Histogram of $y_a$ from equation 5.1.

Figure 5.17: Histogram of $y_d$ from equation 5.1.
Figure 5.18: AOIs calculated from position one.

<table>
<thead>
<tr>
<th>map element</th>
<th>entry point</th>
<th>entry time</th>
<th>exit point</th>
<th>exit time</th>
</tr>
</thead>
<tbody>
<tr>
<td>left box</td>
<td>(216,192)</td>
<td>2.9</td>
<td>(140,84)</td>
<td>7.0</td>
</tr>
<tr>
<td>middle box</td>
<td>(360,104)</td>
<td>10.7</td>
<td>(408,248)</td>
<td>15.3</td>
</tr>
<tr>
<td>right box</td>
<td>(508,348)</td>
<td>18.4</td>
<td>(508,136)</td>
<td>23.4</td>
</tr>
</tbody>
</table>

Table 5.6: Map vector scan from position one. The times are given in ms and the points are in image pixels.
Figure 5.19: Contour plot of the dilated histogram from the map vector scan. We used three scans at only position one.
Figure 5.20: Map from the map vector scan. This figure is an image of map_decide.dat. The green lines indicate the true positions of the boxes and the back wall. These lines were overlayed manually to assist interpretation of the figure.
Chapter 6

Conclusions

Our overall concept was to vector scan the laser rangefinder to build a map for path planning. The overall system seemed to work quite well; in the example scenario, the system was able to identify and build a reasonable representation of all obstacles. The system worked in spite of range noise and the inevitable segmentation errors. However, due to the nature of vector scanning, we cannot guarantee that the system will find all obstacles, which is why an ultrasonic rangefinder is needed in any navigation application. Also, the system did not deal with moving objects; we discuss ideas to allow them subsequently. The use of both polygonal and digital maps in the system was not accidental. The polygonal map allows remarkable expressiveness in a very compact file, and is therefore well suited to represent very large areas in a stored map. The polygonal map was about 140 bytes whereas the digital map was 160,000 bytes. On the other hand, a digital map is well suited to accumulating the sensor data in the immediate vicinity of the robot. Both types of maps are useful. An examination of the timing of the system showed that the software does not run in real-time on a Sun sparc 1+. Chapter 5 discusses some possible changes to allow real-time operation. On the other hand, the scans in the example scenario were estimated to take less than real time.

The laser simulation worked reasonably well, although it had trouble matching up surfaces at their intersections. The simulation also was tedious to set up; each new video image required a new set up (scene.asc). In Chapter 5, we discuss the
solution to the above problems — a ray tracing algorithm that uses a polyhedral representation. This representation would not depend on the video images and therefore would only need to be changed if the scene changed.

Vector scanning allowed the system to acquire important data quickly. It also allows flexibility in acquiring data; the robot can thereby react to its current state and perception of the world. For example, the robot might be having difficulty characterizing some region. The laser could be directed back to that area instead of requiring a decision based on conflicting or incomplete data. A limitation of vector scanning is that the range of some neighbors of any range measurement are missing, whereas with raster scanning, all the neighbors are present. Thus, the system could not do any processing that depends on the neighbors such as computing curvature, two-dimensional filtering, or interpolation. Except for a few cases, the algorithm to generate the segment vector scans worked quite well. A more complex algorithm could be used to compute the scan paths for these difficult segments. Section 4.2.2 discusses the use of path planning algorithms in this regard. Ignoring these awkward segments proved adequate for the example scenario, and we conjecture that this would often be the case because the difficult segments seem to be often formed from odd bits and pieces from different objects. The algorithm to compute map vector scans worked well; it produced simple and quick scans.

The algorithm for map building was effective. It does not require any prior knowledge, and showed the ability to deal with a complex environment caused by many sample points with varying error distributions. We can change the size of the map cells without needing any change in the algorithm. The resolution of the map produced by the algorithm is limited by the spreading mechanism; section 4.3.3 discusses this spreading mechanism. For the example scenario, this resolution was \pm 1\% at 5.6 m (the range errors there were about 0.5\%). The map building algorithm
has several parameters that we can adjust. Firstly, we can adjust the labeling threshold. Increasing this threshold would make short obstacles harder to find, but would decrease the number of mislabeled points due to range noise. Secondly, the number of scans to a map, $n_{\text{scan}}$, can be adjusted. Increasing $n_{\text{scan}}$ would improve our ability to recognize and reject mislabeled points from the map (false alarms), but it would also delay the production of the map and exacerbate the unequal scan problem. Thirdly, we can adjust the threshold on the dilated histogram, $\kappa$. Decreasing $\kappa$ would ease the unequal scan problem but an overly large decrease would cause false alarms. The algorithm was generally robust to the parameter choices; it worked reasonably well for almost every choice that we made.

The map building algorithm did have a problem we termed the unequal scan problem that occurred when we merged across scans that did not view the same areas. To control this problem, we could analyze the viewpoints, identifying areas that were potentially scanned less than $n_{\text{scan}}$. In these areas, the value used for $n_{\text{scan}}$ would be reduced appropriately. A simpler idea would be to monitor the rotation of the laser rangefinder and not merge across scans when it rotates significantly.

One issue we did not address was how to combine maps, which is necessary because $n_{\text{scan}}$ must be limited as we discuss above. A simple technique would be to pixelwise OR the maps together. However, this or-ing would not provide any mechanism for erasing a false alarm; false alarms would persist forever. A way to erase false alarms would be to keep a “last seen time” associated with each obstacle in the polygonal map. Then, any obstacle in the field of view that could not be periodically verified would be deleted. We have shown the basics for this behavior, which is scanning from the polygonal map. We discuss ideas for constructing the polygonal map from a digital map in section 3.5.
To account for moving objects, the system needs two major changes. First, instead of segmentation, the system should directly detect the moving objects, which could be done by differencing the images. However, if the camera is moving, differencing would produce many spurious motion detections. A better method is that proposed by Thompson [59], which allows general camera motion. He computed the optical flow, and then checked to see if the flow was consistent with stationary objects. The second change is to incorporate tracking into the map building process. The system needs to match measurements across time. The old measurements correspond to an old position of the object that is now unoccupied and can be traversed by the robot. This matching across time, or tracking, is quite similar to data association, which we discuss in Chapter 3. Data association is a matching across sensor data streams. This need for tracking may necessitate a shift from the digital map to a more abstract representation (such as an augmented polygonal one).

We also investigated the general properties of sensor fusion by developing a general model for sensor fusion processing. Any general framework must encompass all sensor fusion methods; as an example we fitted the algorithms that we propose for map building into the general framework. We classified our algorithms as indirect fusion because the data streams from the video camera and the laser rangefinder are never directly merged. Instead, the video data is used to direct the laser rangefinder.

The framework pointed out the need for symbolic data. In fact, we used symbolic data at several stages in our processing: the segment numbers, and the classification of the map cells into “floor” and “not-floor”. Because the output (the map) contains symbolic data, we were forced to use symbolic data. For the map building algorithms, the symbolic data were produced from numerical data by windowing (vlabel) and thresholding (update_map2).
We also described the data association problem. We tried to avoid this problem with our histogram based method for map building, but data association reoccurred in our discussion of tracking moving objects. Thus, sensor fusion systems do at least three types of matching: data association, tracking, and object recognition (matching data to object models). Any general sensor fusion framework must include these matching operations and allow symbolic data. Consequently, any purely numerical procedure for sensor fusion, such as Bayesian estimation, is incomplete.
Bibliography


Appendix A

Characterizing the Camera

In this chapter, we describe the camera calibration and the calculation of the camera orientation.

A.1 Calibration of the Camera

To start the calibration process, we measured the front focal length (f\(_f\)) [64]. This is the distance from the front focal point to the front lens surface. The front focal point is important, because it gives the location of the camera coordinate system. For the Navtron lens, the front focal point is behind the front lens surface. To measure the f\(_f\), we shined a laser through the back of the lens as shown in Figure A.1. We then measured the width of the emerging light cone at several positions. Computing the divergence point gave the location of the front focal point. We computed the f\(_f\) as \(-23 \pm 4\) mm. The negative sign indicates that the front focal point is behind the front lens surface.

Due to focus magnification [63], the camera must be used and calibrated at a particular focus setting. The lens was focused at infinity because this setting is repeatable and gives good depth of field.

The images show quite a bit of barrel distortion [60, 64]. With barrel distortion, straight lines are imaged as bowing out. Figure A.2 shows an example of how some straight lines look under barrel distortion. This distortion is indicative of poor lens design or wide viewing angles.
Figure A.1: Measurement of the front focal point.

Figure A.2: Barrel distortion.
To calibrate the paraxial model, we must estimate the two parameters $f_x$ and $f_y$. This is easily done by imaging an object of known dimensions at a known distance. We computed $f_x$ and $f_y$ to be 573 and 625 pixels respectively. Given the barrel distortion, these parameters varied by about 10 pixels depending on the part of the optical field used for the measurements. For my computations, we used most of the width of the image running along the center. For example, to compute $f_y$, we used an image feature from $(0, -109)$ to $(0, 129)$. Chapter 5 discusses the techniques used to measure positions of objects in the lab.

To correct for the barrel distortion, we also computed the parameters for a model that is a slight generalization of the model given by Tsai [61, 62]. The location of a point under the distortion is given by

$$
\begin{align*}
\bar{x}'' &= \frac{x''}{1 + c_1(x'')^2 + c_2(y'')^2} \\
\bar{y}'' &= \frac{y''}{1 + c_3(x'')^2 + c_4(y'')^2}.
\end{align*}
\tag{A.1}
$$

We computed values for these constants using Newton's method. Table A.1 summarizes the results. We also found that we needed to rotate counterclockwise by 0.0099 rad. Notice that the images used for these computations were digitized by different hardware (DataCube DigiMax card), and are for information only. Before using equation (A.1), the errors were 10-12 pixels; after, they were 2-4 pixels ($l_2$ sense). The errors were measured near the edge of the optical field, $(\pm 195, \pm 185)$, where the modeling errors were greatest. Incorporating these corrections would increase the complexity of the later equations greatly without adding anything of substance. For our purposes, the calibration need not be very exact. Consequently, we used the paraxial model in equation (2.4). The calibration errors do lead to some artifacts in the laser simulation that are described Chapter 5.
Table A.1: Calibration constants for equation (A.1).

\[
\begin{array}{c|c}
\hline
 c_1 & 1.0482 \times 10^{-6} \\
c_2 & 3.071 \times 10^{-7} \\
c_3 & 4.586 \times 10^{-7} \\
c_4 & 2.670 \times 10^{-7} \\
\hline
\end{array}
\]

A.2 The Camera Orientation and Position

For simplicity, we assumed that the camera had zero roll angle. Thus, the camera orientation is given by two angles, pitch (\(\gamma\)) and yaw (\(\beta\)). Figures A.3 and A.4 show the pitch and yaw. The rotation matrix \(A\) has a simple form [65]

\[
A = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \gamma & -\sin \gamma \\
0 & \sin \gamma & \cos \gamma
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & -1 \\
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
\cos \beta & -\sin \beta & 0 \\
\sin \beta & \cos \beta & 0 \\
0 & 0 & 1
\end{bmatrix}.
\] (A.2)

Performing the multiplication,

\[
A = \begin{bmatrix}
\cos \beta & -\sin \beta & 0 \\
-\sin \beta \sin \gamma & -\cos \beta \sin \gamma & -\cos \gamma \\
\sin \beta \cos \gamma & \cos \beta \cos \gamma & -\sin \gamma
\end{bmatrix}.
\] (A.3)

Thus, in order to compute \(A\), all we need is \(\gamma\) and \(\beta\). If we are given the position of a point in the map frame \(w\), and its image \((x'', y'')\), we can compute \(\gamma\) and \(\beta\) with

\[
\gamma = \tan^{-1} \frac{-w_x + t_x}{\sqrt{(w_y - t_y)^2 + (w_x - t_x)^2}} - \tan^{-1} \frac{y''}{f_y}
\] (A.4)

and

\[
\beta = \tan^{-1} \frac{w_x - t_x}{w_y - t_y} - \tan^{-1} \frac{x''}{f_x}
\] (A.5)

where \(t\) gives the origin of the camera frame in the map frame. Note that due to barrel distortion, it is best to pick a point that is imaged near the center. Finally,
Figure A.3: Camera pitch. The page is in the camera $y'z'$ plane. The horizontal line is parallel to the map $xy$ plane (the floor).

Figure A.4: Camera yaw. The page is parallel to the map $xy$ plane (the floor). Note that because the camera roll is zero, the camera $x'$ axis is in the page. The reference line is parallel to both the $yz$ and $xy$ map planes.
\( t' \) is given by

\[
t' = -At. \tag{A.6}
\]

The program \texttt{orient} computes \( A \) and \( t' \) given \( t, \gamma, \) and \( \beta \).
Appendix B

Distribution of the Range Errors

Here we complete the calculation of the probability density function (pdf) for the range in a laser rangefinder under the assumptions used by Nitzan et. al [42, Appendix A]. They give that \( i_1 \) and \( i_2 \), from Figure 2.6, are independent and approximately Gaussian. Then we can use the fact that \((\tilde{\rho}, \tilde{\phi})\) are the polar coordinates of \((i_1, i_2)\). Integrating out \(\tilde{\rho}\) we have

\[
p_{\phi}(\tilde{\phi}) = \frac{1}{2\pi \sigma^2} \int_{\tilde{\rho}=0}^{\infty} \tilde{\rho} \exp \left\{ \frac{(\tilde{\rho} \cos \phi - \mu_1)^2}{2\sigma^2} \right\} \exp \left\{ \frac{(\tilde{\rho} \sin \phi - \mu_2)^2}{2\sigma^2} \right\} d\tilde{\rho}. \tag{B.1}
\]

This integral can be evaluated with the help of a table of integrals [66, section 3.462 number 5]. The distribution of the phase estimate is

\[
p_{\phi}(\phi) = \frac{\exp\{-c\}}{2\pi \sigma^2} \left[ \frac{1}{2a} + \frac{b}{4a} \sqrt{a} \exp \left\{ \frac{b^2}{4a} \right\} \left( 1 - \Phi \left( \frac{-b}{2\sqrt{a}} \right) \right) \right]. \tag{B.2}
\]

where

\[
a = \frac{1}{2\sigma^2} \\
b = \frac{\mu_1 \cos \phi + \mu_2 \sin \phi}{\sigma^2} \\
c = \frac{\mu_1^2 + \mu_2^2}{2\sigma^2} \\
\mu_1 = \frac{1}{2} e\bar{m} \cos \phi \\
\mu_2 = \frac{1}{2} e\bar{m} \sin \phi \\
\sigma^2 = \frac{\bar{a}^2}{2T} \\
\bar{a} = \frac{\alpha \eta \lambda A_R \bar{F}_T \rho \cos \theta}{\pi h c \frac{r^2}{T}}
\]
\[
\phi = \frac{4\pi}{\lambda_m} r
\]

\[
\Phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp\left\{-t^2\right\} dt
\]

The range estimate is obtained from the phase estimate by scaling

\[
\hat{r} = \frac{\lambda_m}{4\pi}\hat{\phi}.
\] (B.3)

Thus, the pdf of the range estimate is a scaled version of equation (B.2)

\[
p_{\hat{r}}(\hat{r}) = \frac{4\pi}{\lambda_m} p_{\hat{\phi}}\left(\frac{4\pi}{\lambda_m}\hat{\phi}\right).
\] (B.4)

Using this pdf in the simulation would be computationally expensive and require knowledge of a host of parameters. However, using a Gaussian approximation only requires knowledge of the standard deviation (\(\hat{\phi}\) is unbiased).
Appendix C

The Laser Rangefinder Simulator

We assume that the ranger makes its measurements in range-pixel coordinates. This is the method that Nitzan et al. [42] used on their laser rangefinder. Alternatively, the ranger could make measurements in spherical coordinates. There is not so much difference between the two methods, but the range-pixel coordinate system is easier for our method of simulation. Further, we assume that the laser spatial resolution is 128 by 100 pixels over the same field of view as the camera. This is comparable to commercial laser rangers. Thus, laser pixels differ from camera pixels by a factor of four. For all equations, terms like $x''$ and $f_x$ must be interpreted in the same pixel type, either camera or laser. We do not distinguish between the two types of pixels to save notational complexity.

We assume, for simplicity, that the laser and the camera are collocated so that we only need one coordinate system to describe them both. We are modeling the physical environment by giving the surfaces of all the obstacles [16]. We further assume that these surfaces are planar. We can derive the range to the surfaces as a function of pixel location in the following way. The equation of a plane is

$$a_1 w_x' + a_2 w_y' + a_3 w_z' + a_4 = 0. \quad (C.1)$$

Substituting equations (2.4), and solving for $w_z'$

$$w_z' = \frac{-a_4}{a_1 w_x'' + a_2 w_y'' + a_3}. \quad (C.2)$$
We also impose the condition that $w'_z > 0$, because we are only interested in intersections that occur in front of the range finder. We also have that

$$r = \sqrt{(w'_z)^2 + (w'_y)^2 + (w'_x)^2}.$$  \hfill (C.3)

Using the above equations, we can now compute the range at any given $(x'', y'')$

$$r = \frac{-a_4}{a_1 x'' + a_2 y'' + a_3 \sqrt{\left(\frac{x''}{f_x}\right)^2 + \left(\frac{y''}{f_y}\right)^2 + 1}}.$$  \hfill (C.4)

The amplitude is given by [42]

$$a = \frac{k_a \rho \cos \theta}{r^2}$$  \hfill (C.5)

where $k_a$ is some constant. We choose $k_a$ so that the amplitude measurements were in the range $[0, 255]$. The value that achieved this was $k_a = 1 \times 10^8$. The standard deviation for the range is given by

$$\sigma_r = \frac{k_a}{\sqrt{a}}$$  \hfill (C.6)

where $k_a = 158$. This value corresponds to $\sigma_r = 3$ cm at a range of 10 m (with $\rho = 1$ and $\theta = 0$). This is conservatively comparable to the errors in commercial rangefinders [2, 9, 43]. During simulation of the laser rangefinder, two images are produced, a range image using equation (C.4), and an amplitude image using equation (C.5). When a range is used, the corresponding amplitude is used in equation (C.6) to generate an appropriate Gaussian random variable to add to the range.

In several cases, our model for the measurements can be sensitive to small changes in $(x'', y'')$. The first case occurs when the laser is scanning a far object and suddenly encounters a near object. The second case happens when the normal to the object surface is nearly perpendicular to the laser beam. This happens, for instance, when the laser scans a table top that is just shorter than the laser
height. Both cases are affected by the finite beamwidth of the laser. At range discontinuities, the laser beam simultaneously illuminates the targets at both ranges. The resulting measurement is a weighted sum of the two ranges, with the weighting inversely proportional to $r^2$. Thus, the measurements tend to vary smoothly even when crossing a range discontinuity. Correspondingly, when the laser scans a surface that is oblique, the finite beamwidth laser illuminates points on the surface at a variety of ranges. The measurement is again a weighted sum of these ranges. This weighted sum tends to induce a bias in the measurements that depends on the obliqueness of the surface. The bias is difficult to compute and we have not done so.

The program that computes the simulated range images is sim_laser. Figure C.1 shows the data flow in the simulation. The file scene.asc describes the environment surfaces. Each planar surface is given by a point in the plane $q$ and the normal to the plane $n$, expressed in the map frame. Transforming to the camera frame

$$n' = An$$

and

$$q' = Aq + t'.$$

The equation for the plane in the camera frame is

$$(w' - q')n'^T = 0.$$  \hspace{1cm} (C.9)

Multiplying out we get

$$a_1 = n'_x$$

$$a_2 = n'_y$$

$$a_3 = n'_z$$

$$a_4 = -q'_x n'_x - q'_y n'_y - q'_z n'_z.$$  \hspace{1cm} (C.10)
These coefficients are then used in equation (C.4) to compute ranges.

Table C.4 shows the pseudo-code for sim_laser. This program computes the range and amplitude values for two types of regions, segments and polygons. A seed pixel specifies the segment regions while a list of the vertices of the polygon specifies the polygonal regions. Where segmentation errors have occurred, the polygons can be used to correct the simulation results. Each record in scene.asc specifies the type of region and the information needed to process the region. Tables C.2 and C.1 show examples of these records. Notice that the records in scene.asc must be given in the order of the surfaces from farthest to closest. The reflectivity of the surfaces in the lab was set as in Table C.3. The routine ls_flood uses a flood fill algorithm [45, p. 981]. Instead of setting pixels to some color, ls_flood sets the range and amplitude values for pixels in laser_r.dat and laser_a.dat. The range is computed with equation (C.4) and the amplitude is computed with equation (C.5). The routine ls_flood computes range and amplitude values for all the pixels in the segment specified. The routine fill_polygon uses a polygon scan line fill algorithm [45, p. 98]. This routine computes the range and amplitude values for all the pixels in the specified polygon.

\[ \text{sbw 57 51 0 5572 0 0 -1 0 1.} \]

**Table C.1:** Record from scene.asc specifying a segment region. The record contains the following: label seed-pixel q n r. The label begins with an ‘s’ to indicate that the record describes a segment region. Each item is given in the order x y z. Thus, n = (0, -1, 0). The seed-pixel is given in camera pixels.
Figure C.1: Data flow for the laser simulator.
Table C.2: Record from scene.asc specifying a polygonal region. The record contains the following: label polygon q n ρ. The label begins with a 'i' to indicate that the record describes a polygonal region. The polygon is given by the list of its vertices followed by '1 -1'. Thus, the polygon vertices are \{(503, 140), (511, 140), (511, 0), (503, 0)\}. The vertices are given in camera pixels.

| floor | 0.2 |
| walls | 1.0 |
| boxes | 0.8 |

Table C.3: Assumed reflectivity of the objects used in the simulation.

```python
sim_laser
for each record in scene.asc
q' = Aq + t' (equation (C.8))
n' = An (equation (C.7))
compute a_1, a_2, a_3, a_4 from equation (C.10)
if (record-type == 's')
    ls_flood(seed-pixel)
else if (record-type == 'i')
    fill_polygon(polygon)
```

Table C.4: Pseudo-code for sim_laser.
Appendix D

Calculation of the Dwell Time

The laser must dwell some time $T$ at each pixel; dwelling less than this time would cause an increase in the range noise. Everett [2] gave the scan times and viewing angles for several commercially available laser rangefinders. Of these, the ERIM ASV rangefinder had about medium scanning velocity. In 0.5 s, it scanned 80° horizontally and 60° vertically ($1.396 \times 1.047$ rad). The solid angle, in steradians, viewed by the entire scan of the ASV rangefinder is

$$\Omega_{\text{ASV}} = \int_{\phi=0}^{1.396} \int_{\theta=-0.5235}^{0.5235} \sin \phi \, d\phi \, d\theta = 1.4 \text{ sr} \quad (D.1)$$

where $\theta$ and $\phi$ are the usual spherical coordinates. Thus, the ASV scans at the rate of 2.8 sr/s. To use this figure, we must know the solid angle viewed by a pixel of the rangefinder that we are simulating. Recall that we are assuming that the laser rangefinder and the camera view the same region, and that the laser resolution is $128 \times 100$ pixels. From the camera specification sheet [67], the CCD sensing area is $8.8 \times 6.6$ mm. The camera lens focal length is 8 mm. A pixel in the center of the laser rangefinder image would view the largest solid angle, which is given by

$$\Omega_{\text{pixel}} \leq \frac{A}{r^2} = \frac{0.00454}{64} = 7.1 \times 10^{-5} \text{ sr}. \quad (D.2)$$

Thus we can lower bound $T$ with

$$T \geq \frac{1 \text{ s}}{2.8 \text{ sr}} \frac{7.1 \times 10^{-5} \text{ sr}}{1 \text{ pixel}} = .025 \text{ ms/pixel}. \quad (D.3)$$
Notice that this lower bound on $T$ was imposed by range noise considerations. Section 4.2.3 discusses the timing requirements imposed by the scan speeds of the galvanometers.