3-D Vision Techniques for Autonomous Vehicles

Martial Hebert, Takeo Kanade, Inso Kweon

CMU-RI-TR-88-12

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 1988

This research was sponsored in part by the Defense Advanced Research Projects Agency, DoD, through ARPA Order 5351, monitored by the U.S. Army Engineer Topographic Laboratories under contract DACA76-85-C-0003, by the National Science Foundation contract DCR-8604199, by the Digital Equipment Corporation External Research Program, and by NASA grant NAGW-1175. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

Contents

1 Introduction 1

2 Active range and reflectance sensing 2
- 2.1 From range pixels to points in space . . . 4
- 2.2 Reflectance images . . . 5
- 2.3 Resolution and noise . . . 7

3 Terrain representations 9
- 3.1 The elevation map as the data structure for terrain representation . . . 9
- 3.2 Terrain representations and path planners . . . 10
- 3.3 Low resolution: Obstacle map . . . 12
- 3.4 Medium resolution: Polygonal terrain map . . . 14
- 3.5 High resolution: Elevation maps for rough terrain . . . 16
  - 3.5.1 The locus algorithm for the optimal interpolation of terrain maps . . . 17
  - 3.5.2 Generalizing the locus algorithm . . . 18
  - 3.5.3 Evaluating the locus algorithm . . . 25
  - 3.5.4 Representing the uncertainty . . . 25
  - 3.5.5 Detecting the range shadows . . . 27
  - 3.5.6 An application: footfall selection for a legged vehicle . . . 28
  - 3.5.7 Extracting local features from an elevation map . . . 30

4 Combining multiple terrain maps 31
- 4.1 The terrain matching problem: iconic *vs.* feature-based . . . 33
- 4.2 Feature-based matching . . . 35
  - 4.2.1 Example: Matching polygonal representations . . . 36
  - 4.2.2 Example: Matching local features from high resolution maps . . . 37
- 4.3 Iconic matching from elevation maps . . . 39

5 Combining range and intensity data 46
- 5.1 The geometry of video cameras . . . 46
- 5.2 The registration problem . . . 47
  - 5.2.1 The calibration problem as a minimization problem . . . 48
  - 5.2.2 Initial estimation of camera position . . . 48
  - 5.2.3 Optimal estimation of the calibration parameters . . . 49
  - 5.2.4 Implementation and performance . . . 50
- 5.3 Application to outdoor scene analysis . . . 50
  - 5.3.1 Feature extraction from a colored-range image . . . 50
  - 5.3.2 Scene interpretation from the colored-range image . . . 53

6 Conclusion 54

References 59

List of Figures

1 The Terregator . . . 2
2 The Navlab . . . 3
3 The Ambler . . . 3
4 Geometry of the range sensor . . . 5
5 Range and reflectance images . . . 5
6 Overhead view . . . 6
7 Corrected reflectance image . . . 7
8 Sources of noise in range data . . . 8
9 Noise in range data . . . 8
10 Structure of an elevation map . . . 10
11 The sampling problem . . . 10
12 An example of a range shadow . . . 11
13 Representing uncertainty . . . 11
14 Levels of terrain representation . . . 12
15 Building the obstacle map . . . 14
16 Obstacle detection on a sequence of images . . . 15
17 Range image and elevation map . . . 16
18 Polygonal boundaries of terrain regions . . . 17
19 The locus algorithm for elevation maps . . . 19
20 Image interpolation around the intersection point . . . 20
21 The locus algorithm on range images . . . 21
21 The locus algorithm on range images (Continued) . . . 22
21 The locus algorithm on range images (Continued) . . . 23
21 The locus algorithm on range images (Continued) . . . 24
22 Evaluation of the locus algorithm on synthesized images . . . 25
23 Computing the uncertainty from the locus algorithm . . . 27
24 Detecting range shadows . . . 28
25 Shadow regions in an elevation map . . . 29
26 Footfall support area . . . 29
27 Support area versus elevation . . . 30
28 The high curvature points of an elevation map . . . 31
29 Local features from a high resolution elevation map . . . 32
30 Reducing the range shadow . . . 33
31 Matching maps for position estimation . . . 34
32 A sequence of range and reflectance images . . . 37
33 Individual maps . . . 38
34 Perspective view of the combined map . . . 39
35 Matching maps using local features . . . 40
36 Matching maps using local features (large rotation component) . . . 40
37 Principle of the iconic matching algorithm . . . 41

38 Combining four maps by the iconic matching algorithm 44
39 Convergence rate of the matching algorithm 45
40 Geometry of the video camera 47
41 Geometry of the calibration problem 48
42 Geometry of the "video → range" transformation 51
43 Colored-range image of stairs 51
44 Perspective view of registered range and color images 51
45 Color and range images of an outdoor scene 55
46 A view of the corresponding colored-range image 56
47 Edge features from the colored-range image 56
48 Final scene interpretation 57

Abstract

A mobile robot needs an internal representation of its environment in order to accomplish its mission. Building such a representation involves transforming raw data from sensors into a meaningful geometric representation. In this paper, we introduce techniques for building terrain representations from range data for an outdoor mobile robot. We introduce three levels of representations that correspond to levels of planning: obstacle maps, terrain patches, and high resolution elevation maps. Since terrain representations from individual locations are not sufficient for many navigation tasks, we also introduce techniques for combining multiple maps. Combining maps may be achieved either by using features or the raw elevation data. Finally, we introduce algorithms for combining 3-D descriptions with descriptions from other sensors, such as color cameras. We examine the need for this type of sensor fusion when some semantic information has to be extracted from an observed scene and provide an example application of outdoor scene analysis. Many of the techniques presented in this paper have been tested in the field on three mobile robot systems developed at CMU.

1 Introduction

A mobile robot is a vehicle that navigates autonomously through an unknown or partially known environment. Research in the field of mobile robots has received considerable attention in the past decade due to its wide range of potential applications, from surveillance to planetary exploration, and the research opportunities it provides, including virtually the whole spectrum of robotics research from vehicle control to symbolic planning (see for example [18] for an analysis of the research issues in mobile robots). In this paper we present our investigation of some the issues in one of the components of mobile robots: perception. The role of perception in mobile robots is to transform data from sensors into representations that can be used by the decision-making components of the system. The simplest example is the detection of potentially dangerous regions in the environment (*i.e.* obstacles) that can be used by a path planner whose role is to generate safe trajectories for the vehicle. An example of a more complex situation is a mission that requires the recognition of specific landmarks, in which case the perception components must produce complex descriptions of the sensed environment and relate them to stored models of the landmarks.

There are many sensing strategies for perception for mobile robots, including single camera systems, sonars, passive stereo, and laser range finders. In this report, we focus on perception algorithms for range sensors that provide 3-D data directly by active sensing. Using such sensors has the advantage of eliminating the calibration problems and computational costs inherent in passive techniques such as stereo. We describe the range sensor that we used in this work in Section 2. Even though we tested our algorithm on one specific range sensor, we believe that the sensor characteristics of Section 2 are fairly typical of a wide range of sensors [4].

Research in perception for mobile robots is not only sensor-dependent but it is also dependent on the environment. A considerable part of the global research effort has concentrated on the problem of perception for mobile robot navigation in indoor environments, and our work in natural outdoor environments through the Autonomous Land Vehicle and Planetary Exploration projects is an important development. This report describes some of the techniques we have developed in this area of research. The aim of our work is to produce models of the environment, which we call the *terrain*, for path planning and object recognition.

The algorithms for building a terrain representation from a single sensor frame are discussed in Section 3 in which we introduce the concept of dividing the terrain representation algorithms into three levels depending on the sophistication of the path planner that would use the representation, and on the anticipated difficulty of the terrain. Since a mobile robot is by definition a dynamic system, it must process not one, but many observations along the course of its trajectory. The 3-D vision algorithms must therefore be able to reason about representations that are built from sensory data taken from different locations. We investigate this type of algorithms in Section 4 in which we propose algorithms for matching and merging multiple terrain representations. Finally, the 3-D vision algorithms that we propose are not meant to be used in isolation, they have to be eventually integrated in a system that include other sensors. A typical example is the case of road following in which color cameras can track the road, while a range sensor can detect unexpected obstacles. Another example is a mission in which a scene must be interpreted in order to identify specific objects, in which case all the available sensors must contribute to the final scene analysis. We propose some algorithms for fusing 3-D representations with representations obtained

from a color camera in Section 5. We also describe the application of this sensor fusion to a simple natural scene analysis program. Perception techniques for mobile robots have to be eventually validated by using real robots in real environments. We have implemented the 3-D vision techniques presented in this report on three mobile robots developed by the Field Robotics Center: the Terregator, the Navlab, and the Ambler. The Terregator (Figure 1) is a six-wheeled vehicle designed for rugged terrain. It does not have any onboard computing units except for the low-level control of the actuators. All the processing was done on Sun workstations through a radio connection. We used this machine in early experiments with range data, most notably the sensor fusion experiments of Section 5. The Navlab [36] (Figure 2) is a converted Chevy van designed for navigation on roads or on mild terrains. The Navlab is a self-contained robot in that all the computing equipment is on board. The results presented in Sections 3.3 and 3.4 come from the 3-D vision module that we integrated in the Navlab system [42]. The Ambler [2] is an hexapod designed for the exploration of Mars (Figure 3). This vehicle is designed for navigation on very rugged terrain including high slopes, rocks, and wide gullies. This entirely new design prompted us to investigate alternative 3-D vision algorithms that are reported in Section 3.5. Even though the hardware for the Ambler does not exist at this time, we have evaluated the algorithms through simulation and careful analysis of the planetary exploration missions.

Figure 1: The Terregator

2 Active range and reflectance sensing

The basic principle of active sensing techniques is to observe the reflection of a reference signal (sonar, laser, radar..etc.) produced by an object in the environment in order to compute the distance between the sensor and that object. In addition to the distance, the sensor may report the intensity of the reflected

Figure 2: The Navlab

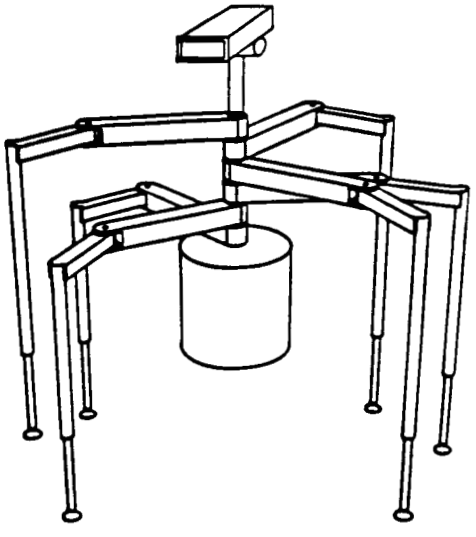

Figure 3: The Ambler

signal which is related to physical surface properties of the object. In accordance with tradition, we will refer to this type of intensity data as "reflectance" data even though the quantity measured is not the actual reflectance coefficient of the surface.

Active sensors are attractive to mobile robots researchers for two main reasons: first, they provide range data without the computation overhead associated with conventional passive techniques such as stereo vision, which is important in time critical applications such as obstacle detection. Second, it is largely insensitive to outside illumination conditions, simplifying considerably the image analysis problem. This is especially important for images of outdoor scenes in which illumination cannot be controlled or predicted. For example, the active reflectance images of outside scenes do not contain any shadows from the sun. In addition, active range finding technology has developed to the extent that makes it realistic to consider it as part of practical mobile robot implementations in the short term [4].

The range sensor we used is a time-of-flight laser range finder developed by the Environmental Research Institute of Michigan (ERIM). The basic principle of the sensor is to measure the difference of phase between a laser beam and its reflection from the scene [46]. A two-mirror scanning system allows the beam to be directed anywhere within a $30° \times 80°$ field of view. The data produced by the ERIM sensor is a 64×256 range image, the range is coded on eight bits from zero to 64 feet, which corresponds to a range resolution of three inches. All measurements are all relative since the sensor measures differences of phase. That is, a range value is known *modulo* 64 feet. We have adjusted the sensor so that the range value 0 corresponds to the mirrors for all the images presented in this report. In addition to range images, the sensor also produces active reflectance images of the same format ($64 \times 256 \times 8$ bits), the reflectance at each pixel encodes the energy of the reflected laser beam at each point. Figure 5 shows a pair of range and reflectance images of an outdoor scene. The next two Sections describe the range and reflectance data in more details.

2.1 From range pixels to points in space

The position of a point in a given coordinate system can be derived from the measured range and the direction of the beam at that point. We usually use the Cartesian coordinate system shown in Figure 4, in which case the coordinates of a point measured by the range sensor are given by the equations[1]:

$$\begin{aligned} x &= D\sin\theta \\ y &= D\cos\phi\cos\theta \\ z &= D\sin\phi\cos\theta \end{aligned} \tag{1}$$

where ϕ and θ are the vertical and horizontal scanning angles of the beam direction. The two angles are derived from the row and column position in the range image, (r, c), by the equations:

$$\begin{aligned} \theta &= \theta_0 + c \times \Delta\theta \\ \phi &= \phi_0 + r \times \Delta\phi \end{aligned} \tag{2}$$

[1]Note that the reference coordinate system is not the same as in [20] for consistency reasons

where θ_0 (respectively ϕ_0) is the starting horizontal (respectively vertical) scanning angle, and $\Delta\theta$ (respectively $\Delta\phi$) is the angular step between two consecutive columns (respectively rows). Figure 6 shows an overhead view of the scene of Figure 5, the coordinates of the points are computed using Eq. (3).

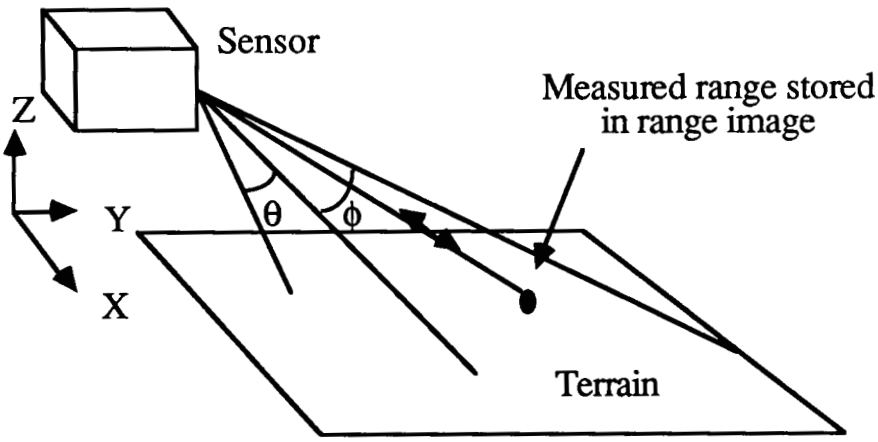

Figure 4: Geometry of the range sensor

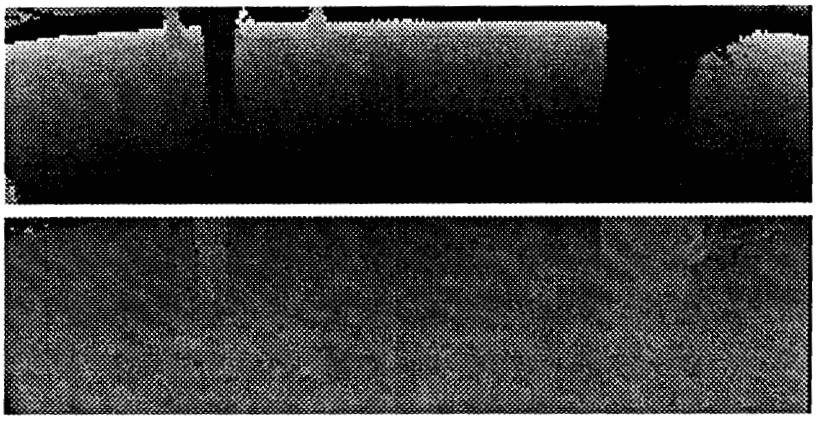

Figure 5: Range and reflectance images

2.2 Reflectance images

A reflectance image from the ERIM sensor is an image of the energy reflected by a laser beam. Unlike conventional intensity images, this data provides us with information which is to a large extent independent

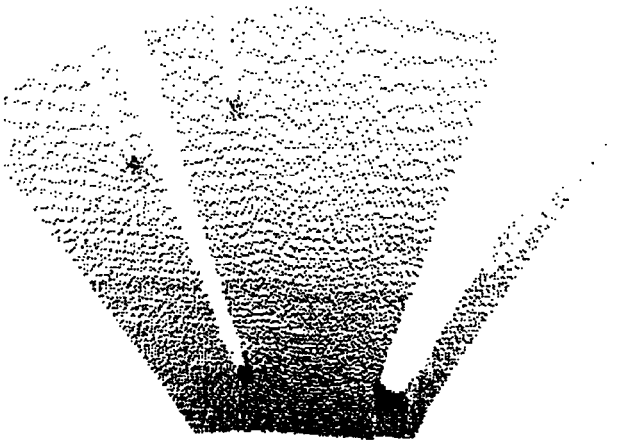

Figure 6: Overhead view

of the environmental illumination. In particular, the reflectance images contain no shadows from outside illumination. The measured energy does depend, however, on the shape of the surface and its distance to the sensor. We correct the image so that the pixel values are functions only of the material reflectance. The measured energy, P_{return}, depends on the specific material reflectance, ρ, the range, D, and the angle of incidence, γ:

$$P_{return} = \frac{K\rho\cos\gamma}{D^2} \tag{3}$$

Due to the wide range of P_{return}, the value actually reported in the reflectance image is compressed by using a log transform. That is, the digitized value, P_{image} is of the form [44]:

$$P_{image} = A\log(\rho\cos\gamma) + B\log D \tag{4}$$

where A and B are constants that depend only on the characteristics of the laser, the circuitry used for the digitization, and the physical properties of the ambiant atmosphere. Since A and B cannot be computed directly, we use a calibration procedure in which a homogeneous flat region is selected in a training image; we then use the pixels in this region to estimate A and B by least-squares fitting Eq. (4) to the actual reflectance/range data. Given A and B, we correct subsequent images by:

$$P_{new-image} = (P_{image} - B\log D)/A \tag{5}$$

The value $P_{new-image}$ depends only on the material reflectance and the angle of incidence. This is a sufficient approximation for our purposes since for smooth surfaces such as smooth terrain, the $cos\gamma$ factor does not vary widely. For efficiency purposes, the right-hand side of (5) is precomputed for all possible combinations (P_{image}, D) and stored in a lookup table. Figure 5 shows an example of an ERIM image, and Figure 7 shows the resulting corrected image.

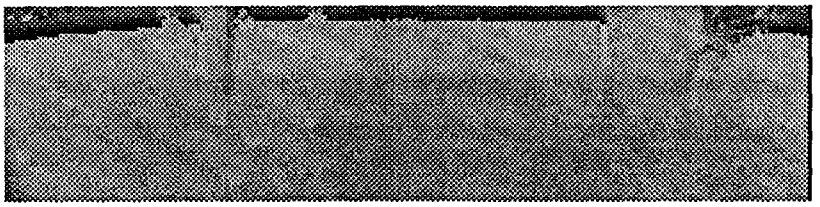

Figure 7: Corrected reflectance image

2.3 Resolution and noise

As is the case with any sensor, the range sensor returns values that are measured with a limited resolution which are corrupted by measurement noise. In the case of the ERIM sensor, the main source of noise is due to the fact that the laser beam is not a line in space but rather a cone whose opening is a 0.5° solid angle (the instantaneous field of view). The value returned at each pixel is actually the average of the range of values over a 2-D area, the *footprint*, which is the intersection of the cone with the target surface (Figure 8). Simple geometry shows that the area of the footprint is proportional to the square of the range at its center. The size of the footprint also depends on the angle θ between the surface normal and the beam as shown in Figure 8. The size of the footprint is roughly inversely proportional to $\cos\theta$ if we assume that the footprint is small enough and that θ is almost constant. Therefore, a first order approximation of the standard deviation of the range noise, σ is given by:

$$\sigma \propto \frac{D^2}{\cos\theta} \tag{6}$$

The proportionality factor in this equation depends on the characteristics of the laser transmitter, the outside illumination, and the reflectance ρ of the surface which is assumed constant across the footprint in this first order approximation. We validated the model of Equation 6 by estimating the RMS error of the range values on a sequence of images. Figure 9 shows the standard deviation with respect to the measured range. The Figure shows that σ follows roughly the D^2 behavior predicted by the first order model. The footprint affects all pixels in the image.

There are other effects that produce distortions only at specific locations in the image. The main effect is known as the "mixed point" problem and is illustrated in Figure 8 in which the laser footprint crosses the edge between two objects that are far from each other. In that case, the returned range value is some combination of the range of the two objects but does not have any physical meaning. This problem makes the accurate detection of occluding edges more difficult. Another effect is due to the reflectance properties of the observed surface; if the surface is highly specular then no laser reflection can be observed. In that case the ERIM sensor returns a value of 255. This effect is most noticeable on man-made objects that contain a lot of polished metallic surfaces. It should be mentioned, however, that the noise characteristics of the ERIM sensor are fairly typical of the behavior of active range sensors [5].

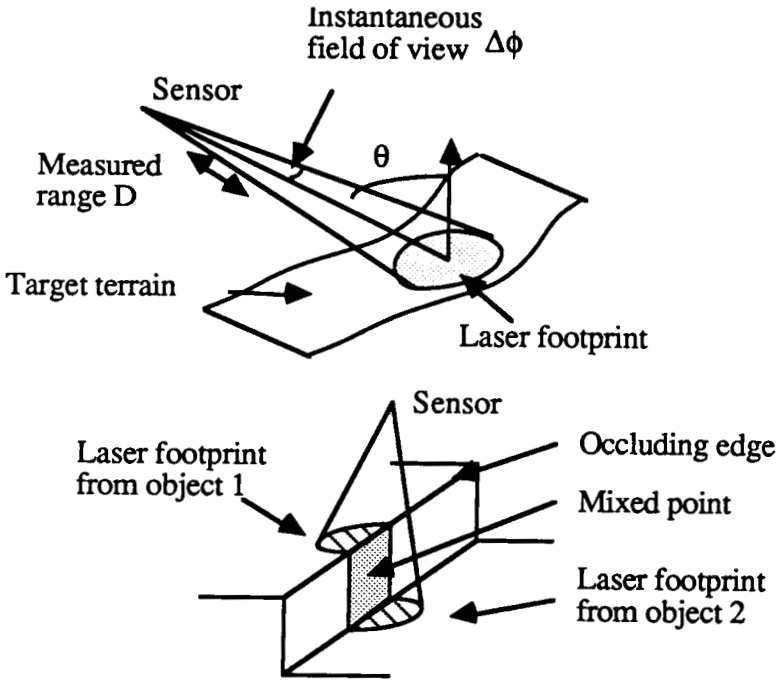

Figure 8: Sources of noise in range data

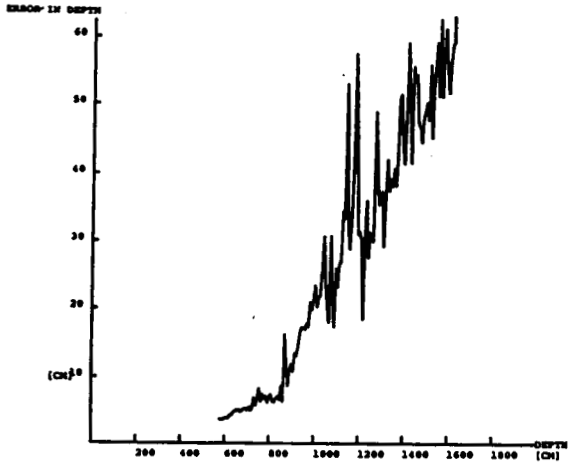

Figure 9: Noise in range data

3 Terrain representations

The main task of 3-D vision in a mobile robot system is to provide sufficient information to the path planner so that the vehicle can be safely steered through its environment. In the case of outdoor navigation, the task is to convert a range image into a representation of the terrain. We use the word "terrain" in a very loose sense in that we mean both the ground surface and the objects that may appear in natural environments (*e.g.* rocks or trees). In this Section we discuss the techniques that we have implemented for the Navlab and Mars Rover systems. We first introduce the concept of the elevation map as a basis for terrain representations and its relationship with different path planning techniques. The last four Sections spell out the technical details of the terrain representation algorithms.

3.1 The elevation map as the data structure for terrain representation

Even though the format of the range data is an image, this may not be the most suitable structuring of the data for extracting information. For example , a standard representation in 3-D vision for manipulation is to view a range image as a set of data points measured on a surface of the equation $z = f(x, y)$ where the $x-$ and $y-$axes are parallel to the axis of the image and z is the measured depth. This choice of axis is natural since the image plane is usually parallel to the plane of the scene. In our case, however, the "natural" reference plane is not the image plane but is the ground plane. In this context, "ground plane" refers to a plane that is horizontal with respect to the vehicle or to the gravity vector. The representation $z = f(x, y)$ is then the usual concept of an elevation map. To transform the data points into an elevation map is useful only if one has a way to access them. The most common approach is to discretize the (x, y) plane into a grid. Each grid cell (x_i, y_i) is the trace of a vertical column in space, its *field* (Figure 10). All the data that falls within a cell's field is stored in that cell. The description shown in Figure 10 does not necessarily reflect the actual implementation of an elevation map but is more of a framework in which we develop the terrain representation algorithms. As we shall see later, the actual implementation depends on the level of detail that needs to be included in the terrain description.

Although the elevation map is a natural concept for terrain representations, it exhibits a number of problems due to the conversion of a regularly sampled image to a different reference plane [25]. Although we propose solutions to these problems in Section 3.5, it is important to keep them in mind while we investigate other terrain representations. The first problem is the sampling problem illustrated in Figure 11. Since we perform some kind of image warping, the distribution of data points in the elevation map is not uniform, and as a result conventional image processing algorithms cannot be applied directly to the map. There are two ways to get around the sampling problem: We can either use a base structure that is not a regularly spaced grid, such as a Delaunay triangulation of the data points [33], or we can interpolate between data points to build a dense elevation map. The former solution is not very practical because of the complex algorithms required to access data points and their neighborhoods. We describe an implementation of the latter approach in Section 3.5. A second problem with elevation maps is the representation of the range shadows created by some objects (Figure 12). Since no information is available within the shadowed regions of the map, we must represent them separately so that no interpolation takes place across them and no "phantom" features are reported to the path planner. Finally, we have to convert the noise on the original measurements into a measure of uncertainty on the z value at each grid point

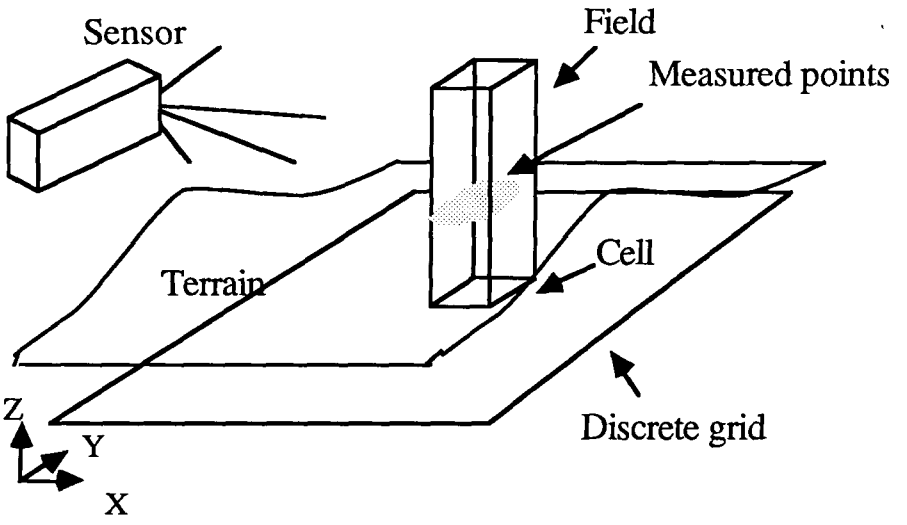

Figure 10: Structure of an elevation map

(x, y). This conversion is difficult due to the fact that the sensor's uncertainty is most naturally represented with respect to the direction of measurement (Figure 13) and therefore spreads across a whole region in the elevation map.

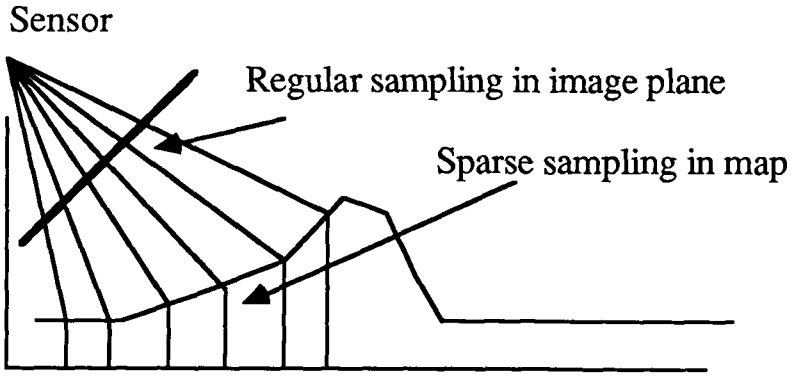

Figure 11: The sampling problem

3.2 Terrain representations and path planners

The choice of a terrain representation depends on the path planner used for actually driving the vehicle. For example, the family of planners derived from the Lozano-Perez's A^* approach [28] uses discrete obstacles represented by 2-D polygons. By contrast, planners that compare a vehicle model with the local terrain [9,38] use some intermediate representation of the raw elevation map. Furthermore, the choice of a terrain representation and a path planner in turn depend on the environment in which the vehicle has to

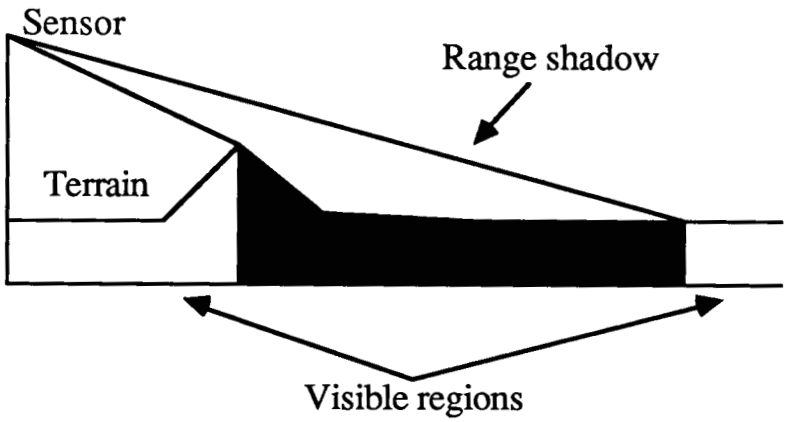

Figure 12: An example of a range shadow

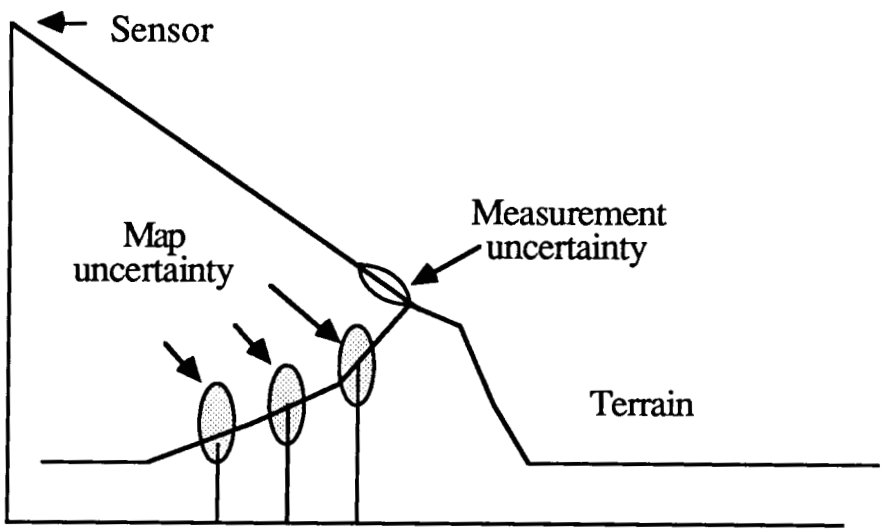

Figure 13: Representing uncertainty

navigate. For example, representing only a small number of discrete upright objects may be appropriate if it is known in advance that the terrain is mostly flat, (*e.g.* a road) with a few obstacles (*e.g.* trees) while cross-country navigation requires a more detailed description of the elevation map. Generating the most detailed description and then extracting the relevant information is not an acceptable solution since it would significantly degrade the performance of the system in simple environments. Therefore, we need several levels of terrain representation corresponding to different resolutions at which the terrain is described (Figure 14). At the low resolution level we describe only discrete obstacles without explicitly describing the local shape of the terrain. At the medium level, we include a description of the terrain through surface patches that correspond to significant terrain features. At that level, the resolution is the resolution of the operator used to detect these features. Finally, the description with the highest resolution is a dense elevation map whose resolution is limited only by the sensor. In order to keep the computations involved under control, the resolution is typically related to the size of the vehicle's parts that enter in contact with the terrain. For example, the size of one foot is used to compute the terrain resolution in the case of a legged vehicle.

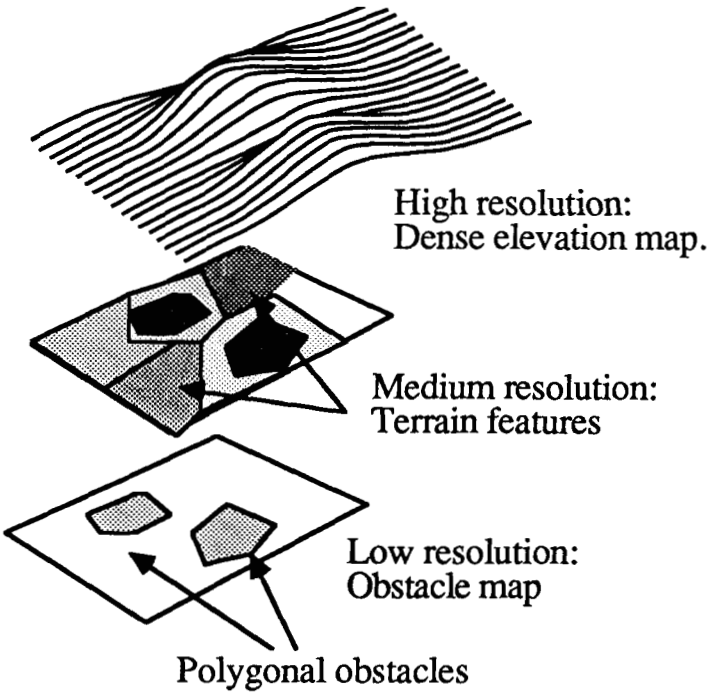

Figure 14: Levels of terrain representation

3.3 Low resolution: Obstacle map

The lowest resolution terrain representation is an obstacle map which contains a small number of obstacles represented by their trace on the ground plane. Several techniques have been proposed for obstacle detection. The Martin-Marietta ALV [10,11,43] detects obstacles by computing the difference between

the observed range image and pre-computed images of ideal ground at several different slope angles. Points that are far from the ideal ground planes are grouped into regions that are reported as obstacles to a path planner. A very fast implementation of this technique is possible since it requires only image differences and region grouping. It makes, however, very strong assumptions on the shape of the terrain. It also takes into account only the positions of the potential obstacle point, and as a result a very high slope ridge that is not deep enough would not be detected.

Another approach proposed by Hughes AI group [8] is to detect the obstacles by thresholding the normalized range gradient, $\Delta D/D$, and by thresholding the radial slope, $D\Delta\phi/\Delta D$. The first test detects the discontinuities in range, while the second test detects the portion of the terrain with high slope. This approach has the advantage of taking a vehicle model into account when deciding whether a point is part of an obstacle. We used the terrain map paradigm to detect obstacles for the Navlab. Each cell of the terrain contains the set of data points that fall within its field (Figure 10). We can then estimate surface normal and curvatures at each elevation map cell by fitting a reference surface to the corresponding set of data points. Cells that have a high curvature or a surface normal far from the vehicle's idea of the vertical direction are reported as part of the projection of an obstacle. Obstacle cells are then grouped into regions corresponding to individual obstacles. The final product of the obstacle detection algorithm is a set of 2-D polygonal approximations of the boundaries of the detected obstacles that is sent to an A^*-type path planner (Figure 15). In addition, we can roughly classify the obstacles into holes or bumps according to the shape of the surfaces inside the polygons.

Figure 16 shows the result of applying the obstacle detection algorithm to a sequence of ERIM images. The Figure shows the original range images (top), the range pixels projected in the elevation map (left), and the resulting polygonal obstacle map (right). The large enclosing polygon in the obstacle map is the limit of the visible portion of the world. The obstacle detection algorithm does not make assumptions on the position of the ground plane in that it only assumes that the plane is roughly horizontal with respect to the vehicle. Computing the slopes within each cell has a smoothing effect that may cause real obstacles to be undetected. Therefore, the resolution of the elevation map must be chosen so that each cell is significantly larger than the typical expected obstacles. In the case of Figure 16, the resolution is twenty centimeters. The size of the detectable obstacle also varies with the distance from the vehicle due to the sampling problem (Section 3.1).

One major drawback of our obstacle detection algorithm is that the computation of the slopes and curvatures at each cell of the elevation map is an expensive operation. Furthermore, since low-resolution obstacle maps are most useful for fast navigation through simple environments, it is important to have a fast implementation of the obstacle detection algorithm. A natural optimization is to parallelize the algorithm by dividing the elevation map into blocks that are processed simultaneously. We have implemented such a parallel version of the algorithm on a ten-processor Warp computer [45,21]. The parallel implementation reduced the cycle time to under two seconds, thus making it possible to use the obstacle detection algorithm for fast navigation of the Navlab. In that particular implementation, the vehicle was moving at a continuous speed of one meter per second, taking range images, detecting obstacles, and planning a path every four meters.

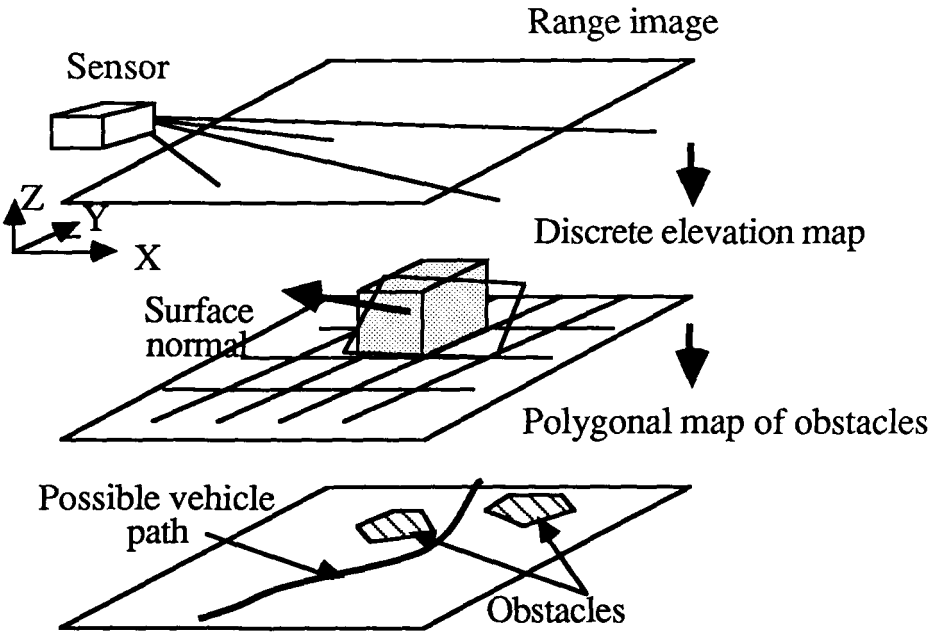

Figure 15: Building the obstacle map

3.4 Medium resolution: Polygonal terrain map

Obstacle detection is sufficient for navigation in flat terrain with discrete obstacles, such as following a road bordered by trees. We need a more detailed description when the terrain is uneven as in the case of cross-country navigation. For that purpose, an elevation map could be used directly [9] by a path planner. This approach is costly because of the amount of data to be handled by the planner which does not need such a high resolution description to do the job in many cases (although we will investigate some applications in which a high resolution representation is required in Section 3.5). An alternative is to group smooth portions of the terrain into regions and edges that are the basic units manipulated by the planner. This set of features provides a compact representation of the terrain thus allowing for more efficient planning [38].

The features used are of two types: smooth regions, and sharp terrain discontinuities. The terrain discontinuities are either discontinuities of the elevation of the terrain, as in the case of a hole, or discontinuities of the surface normals, as in the case of the shoulder of a road [3]. We detect both types of discontinuities by using an edge detector over the elevation map and the surface normals map. The edges correspond to small regions on the terrain surface. Once we have detected the discontinuities, we segment the terrain into smooth regions. The segmentation uses a region growing algorithm that first identifies the smoothest locations in the terrain based on the behavior of the surface normals, and then grows regions around those locations. The result of the processing is a covering of the terrain by regions corresponding either to smooth portions or to edges.

The final representation depends on the planner that uses it. In our case, the terrain representation is embedded in the Navlab system using the path planner described in [38]. The basic geometric object used

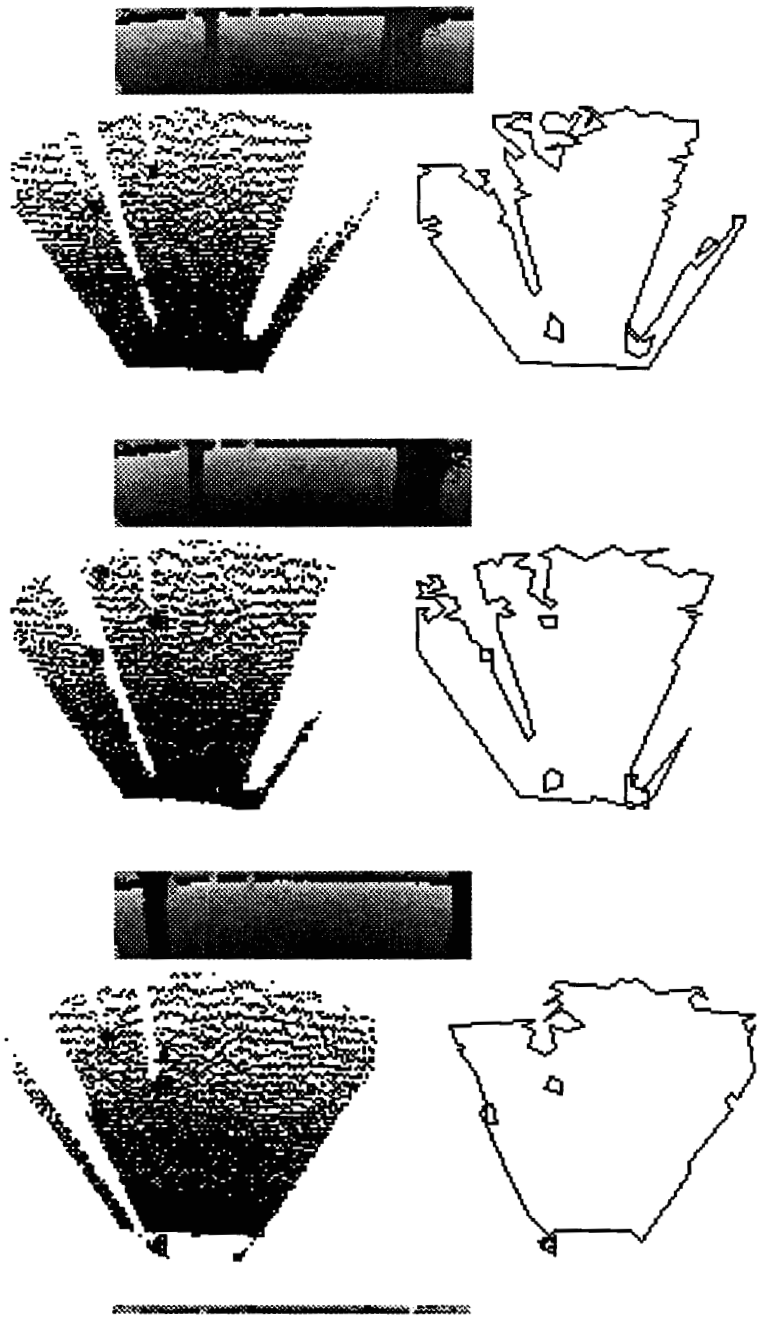

Figure 16: Obstacle detection on a sequence of images

by the system is the three-dimensional polygon. We therefore approximate the boundary of each region by a polygon. The approximation is done in a way that ensures consistency between regions in that the polygonal boundaries of neighboring regions share common edges and vertices. This guarantees that no "gaps" exist in the resulting polygonal mesh. This is important from the point of view of the path planner since such gaps would be interpreted as unknown portions of the terrain. Each region is approximated by a planar surface that is used by the planner to determine the traversability of the regions. Since the regions are not planar in reality, the standard deviation of the parameters of the plane is associated with each region.

Figure 18 shows the polygonal boundaries of the regions extracted from the image of Figure 17. In this implementation, the resolution of the elevation map is twenty centimeters. Since we need a dense map in order to extract edges, we interpolated linearly between the sparse points of the elevation map. Figure 17 shows the interpolated elevation map. This implementation of a medium resolution terrain representation is integrated in the Navlab system and will be part of the standard core system for our future mobile robot systems.

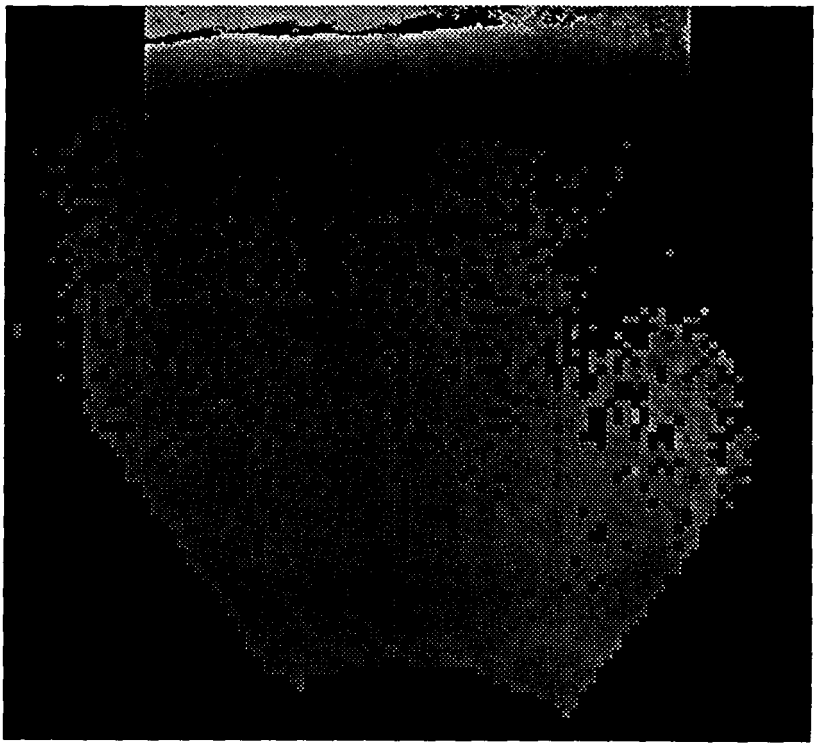

Figure 17: Range image and elevation map

3.5 High resolution: Elevation maps for rough terrain

The elevation map derived directly from the sensor is sparse and noisy, especially at greater distances from the sensor. Many applications, however, need a dense and accurate high resolution map. One way

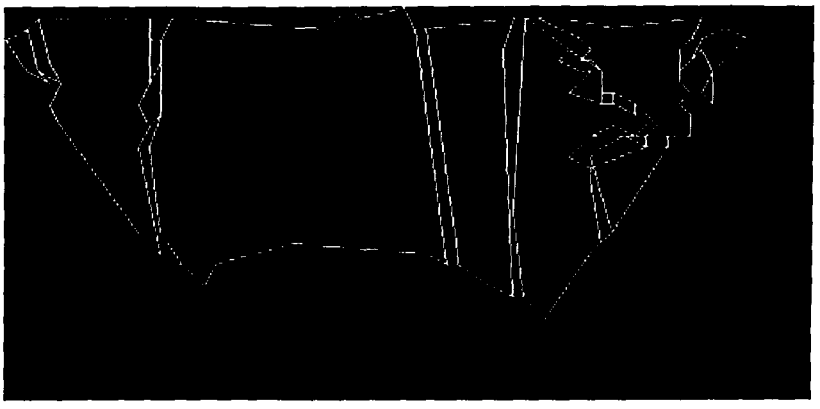

Figure 18: Polygonal boundaries of terrain regions

to derive such a map is to interpolate between the data points using some mathematical approximation of the surface between data points. The models that can be used include linear, quadratic, or bicubic surfaces [33]. Another approach is to fit a surface globally under some smoothness assumptions. This approach includes the family of regularization algorithms [6] in which a criterion of the form:

$$\int \|h_{data} - h_{interpolation}\|^2 + \lambda \int f(h_{interpolation}) \tag{7}$$

is minimized, where f is a regularization function that reflects the smoothness model (*e.g.* thin plate). Two problems arise with both interpolation approaches: They make apriori assumptions on the local shape of the terrain which may not be valid (*e.g.* in the case of very rough terrain), and they do not take into account the image formation process since they are generic techniques independent of the origin of the data. In addition, the interpolation approaches depend heavily on the resolution and position of the reference grid. For example, they cannot compute an estimate of the elevation at an (x, y) position that is not a grid point without resampling the grid. We propose an alternative, the *locus* algorithm [25], that uses a model of the sensor and provides interpolation at arbitrary resolution without making any assumptions on the terrain shape other than the continuity of the surface.

3.5.1 The locus algorithm for the optimal interpolation of terrain maps

The problem of finding the elevation z of a point (x, y) is trivially equivalent to computing the intersection of the surface observed by the sensor and the vertical line passing through (x, y). The basic idea of the locus algorithm is to convert the latter formulation into a problem in image space (Figure 19). A vertical line is a curve in image space, the *locus*, whose equation as a function of ϕ is:

$$D = D_l(\phi) = \sqrt{\frac{y^2}{\cos^2\phi} + x^2} \tag{8}$$

$$\theta = \theta_l(\phi) = \arctan\frac{x\cos\phi}{y} \tag{9}$$

where ϕ, θ, and D are defined as in Section 2. Equation (9) was derived by inverting Equation (2), and assuming x and y constant. Similarly, the range image can be viewed as a surface $D = I(\phi,\theta)$ in ϕ,θ, D space. The problem is then to find the intersection, if it exists, between a curve parametrized by ϕ and a discrete surface. Since the surface is known only from a sample of data, the intersection cannot be computed analytically. Instead, we have to search along the curve for the intersection point. The search proceeds in two stages: We first locate the two scanlines of the range image, ϕ_1 and ϕ_2, between which the intersection must be located, that is the two consecutive scanlines such that, $Diff(\phi_1) = D_l(\phi_1) - I(\phi_1, \hat{\theta}_l(\phi_1))$ and $Diff(\phi_2) = D_l(\phi_1) - I(\phi_2, \hat{\theta}_l(\phi_2))$ have opposite signs, where $\hat{\theta}_l(\phi)$ is the image column that is the closest to $\theta_l(\phi)$. We then apply a binary search between ϕ_1 and ϕ_2. The search stops when the difference between the two angles ϕ_n and ϕ_{n+1}, where $Diff(\phi_n)$ and $Diff(\phi_{n+1})$ have opposite signs, is lower than a threshold ϵ. Since there are no pixels between ϕ_1 and ϕ_2, we have to perform a local quadratic interpolation of the image in order to compute $\theta_l(\phi)$ and $D_l(\phi)$ for $\phi_1 < \phi < \phi_2$. The control points for the interpolation are the four pixels that surround the intersection point (Figure 20). The final result is a value ϕ that is converted to an elevation value by applying Equation (2) to $\phi, \theta_l(\phi), D_l(\phi)$. The resolution of the elevation is controlled by the choice of the parameter ϵ.

The locus algorithm enables us to evaluate the elevation at any point since we do not assume the existence of a grid. Figure 21 shows the result of applying the locus algorithm on range images of uneven terrain, in this case a construction site. The Figure shows the original range images and the map displayed as an isoplot surface. The centers of the grid cells are ten centimeters apart in the (x, y) plane.

3.5.2 Generalizing the locus algorithm

We can generalize the locus algorithm from the case of a vertical line to the case of a general line in space. This generalization allows us to build maps using any reference plane instead of being restricted to the (x, y) plane. This is important when, for example, the sensor's (x, y) plane is not orthogonal to the gravity vector. A line in space is defined by a point $u = [u_x, u_y, u_z]^t$, and a unit vector $v = [v_x, v_y, v_z]^t$. Such a line is parametrized in λ by the relation $p = u + \lambda v$ if p is a point on the line. A general line is still a curve in image space that can be parametrized in ϕ if we assume that the line is not parallel to the (x, y) plane. The equation of the curve becomes:

$$\begin{aligned} D_l(\phi) &= \sqrt{(v_x\lambda(\phi) + u_x)^2 + (v_y\lambda(\phi) + u_y)^2 + (v_z\lambda(\phi) + u_z)^2} \\ \theta_l(\phi) &= \arcsin \frac{v_x\lambda(\phi) + u_x}{D} \\ \lambda(\phi) &= \frac{u_y \tan\phi - u_z}{v_z - v_y \tan\phi} \end{aligned} \tag{10}$$

We can then compute the intersection between the curve and the image surface by using the same algorithm as before except that we have to use Equation (10) instead of Equation (9).

The representation of the line by the pair (u, v) is not optimal since it uses six parameters while only four parameters are needed to represent a line in space. For example, this can be troublesome if we want to compute the Jacobian of the intersection point with respect to the parameters of the line. A better

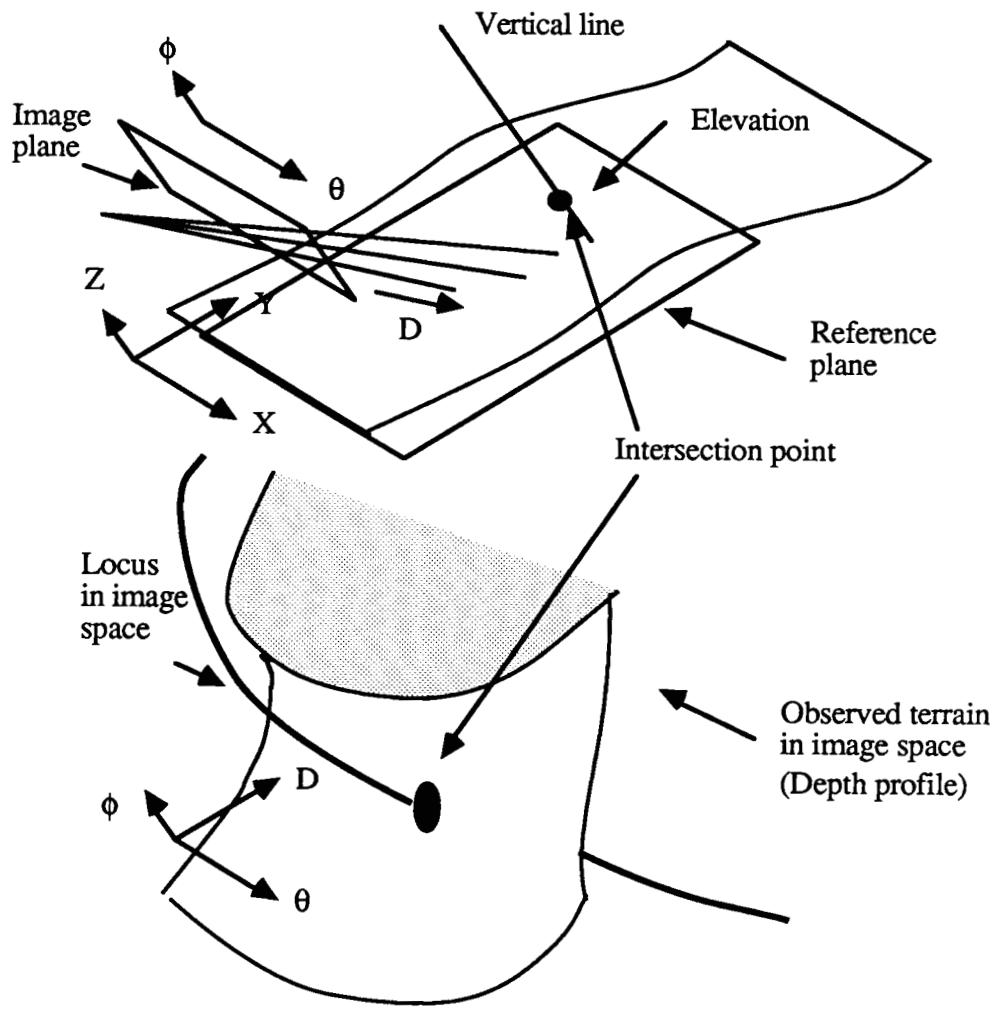

Figure 19: The locus algorithm for elevation maps

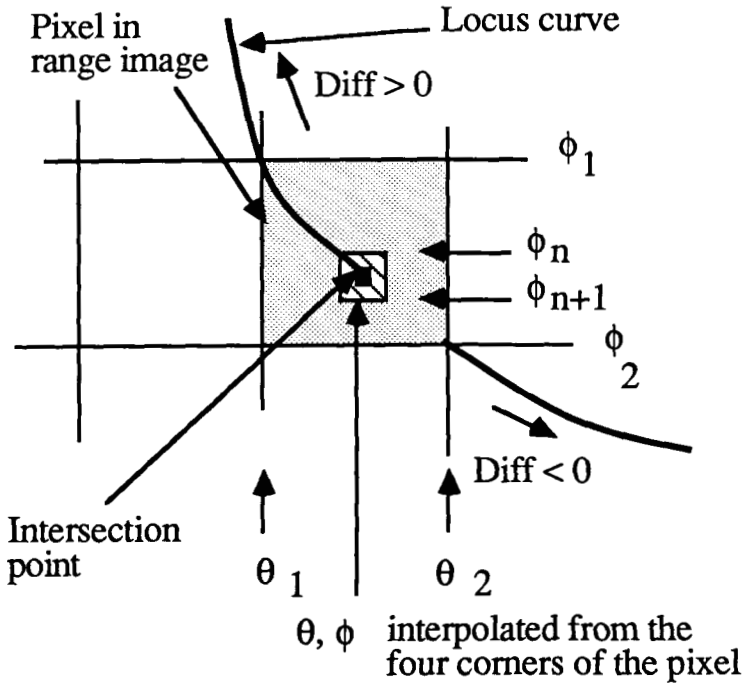

Figure 20: Image interpolation around the intersection point

alternative [22] is to represent the line by its slopes in x and y and by its intersection with the plane $z = 0$ (See [35] for a complete survey of 3-D line representations). The equation of the line then becomes:

$$\begin{aligned} x &= az + p \\ y &= bz + q \end{aligned} \tag{11}$$

We can still use Equation (10) to compute the locus because we can switch between the (a, b, p, q) and (u, v) representations by using the Equations:

$$\begin{aligned} v &= \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}, u = \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} \\ a &= \frac{u_x}{u_z}, p = -\frac{v_x}{u_z} v_z \\ b &= \frac{u_y}{u_z}, q = -\frac{v_y}{u_z} v_z \end{aligned} \tag{12}$$

In the subsequent Sections, we will denote by $h(a, b, p, q)$ the function from R^4 to R^3 that maps a line in space to the intersection point with the range image.

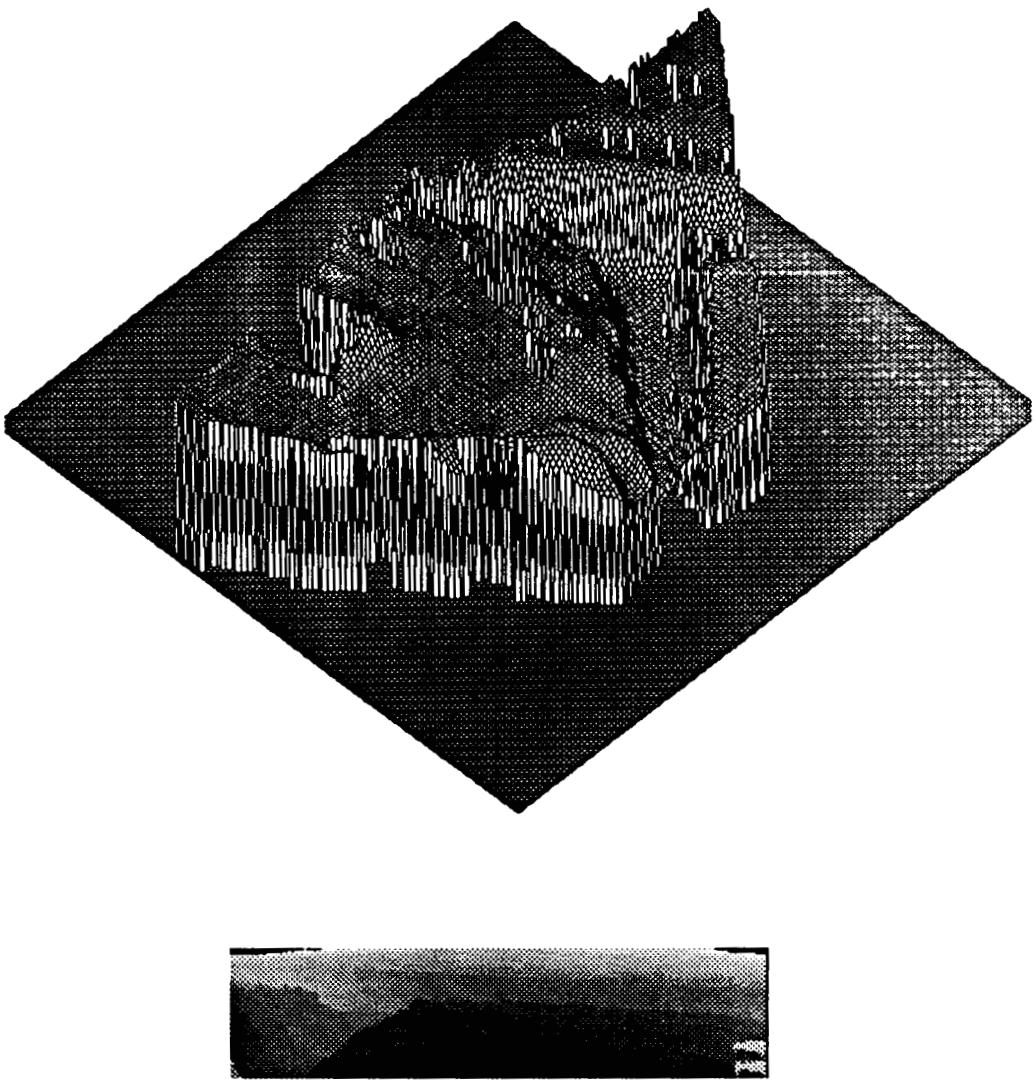

Figure 21: The locus algorithm on range images

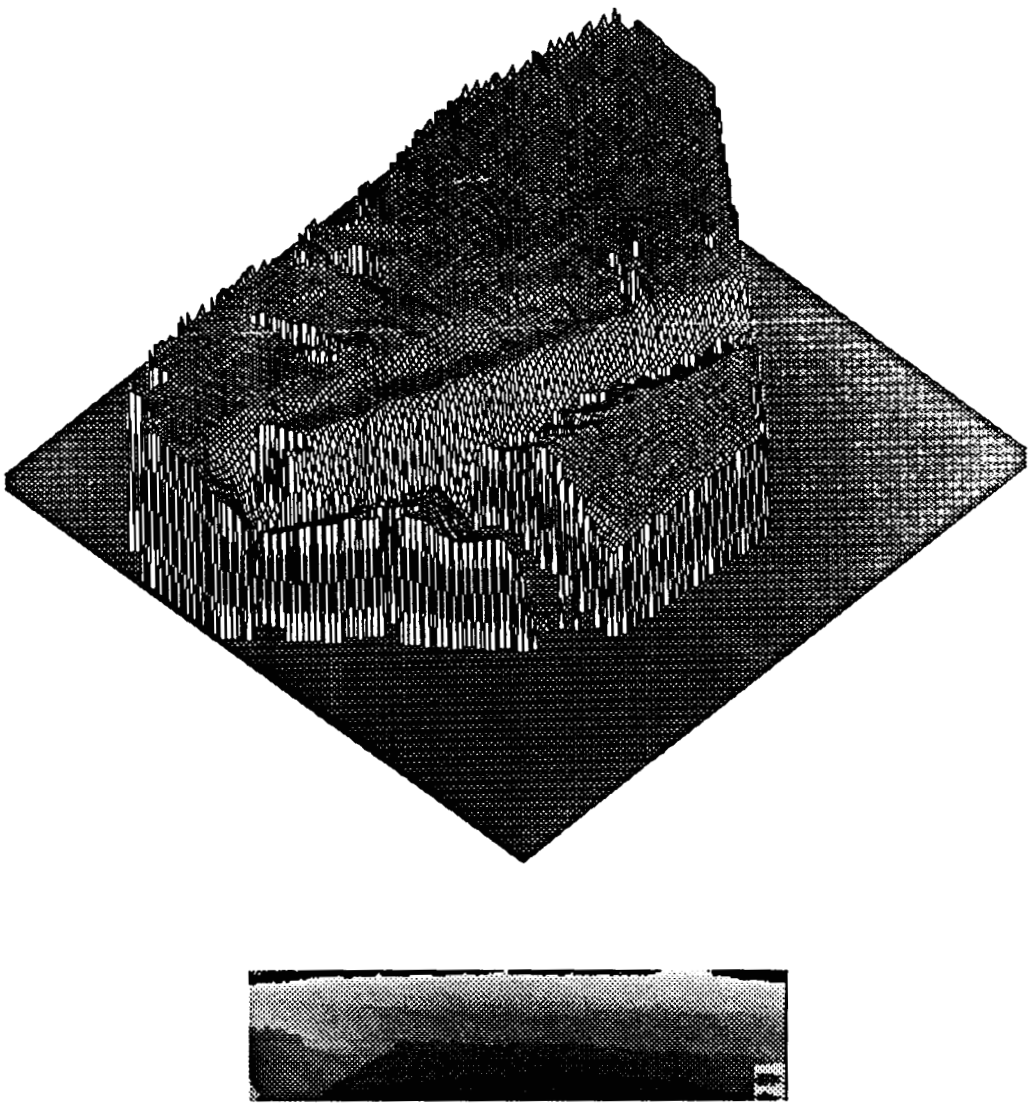

Figure 21: The locus algorithm on range images (Continued)

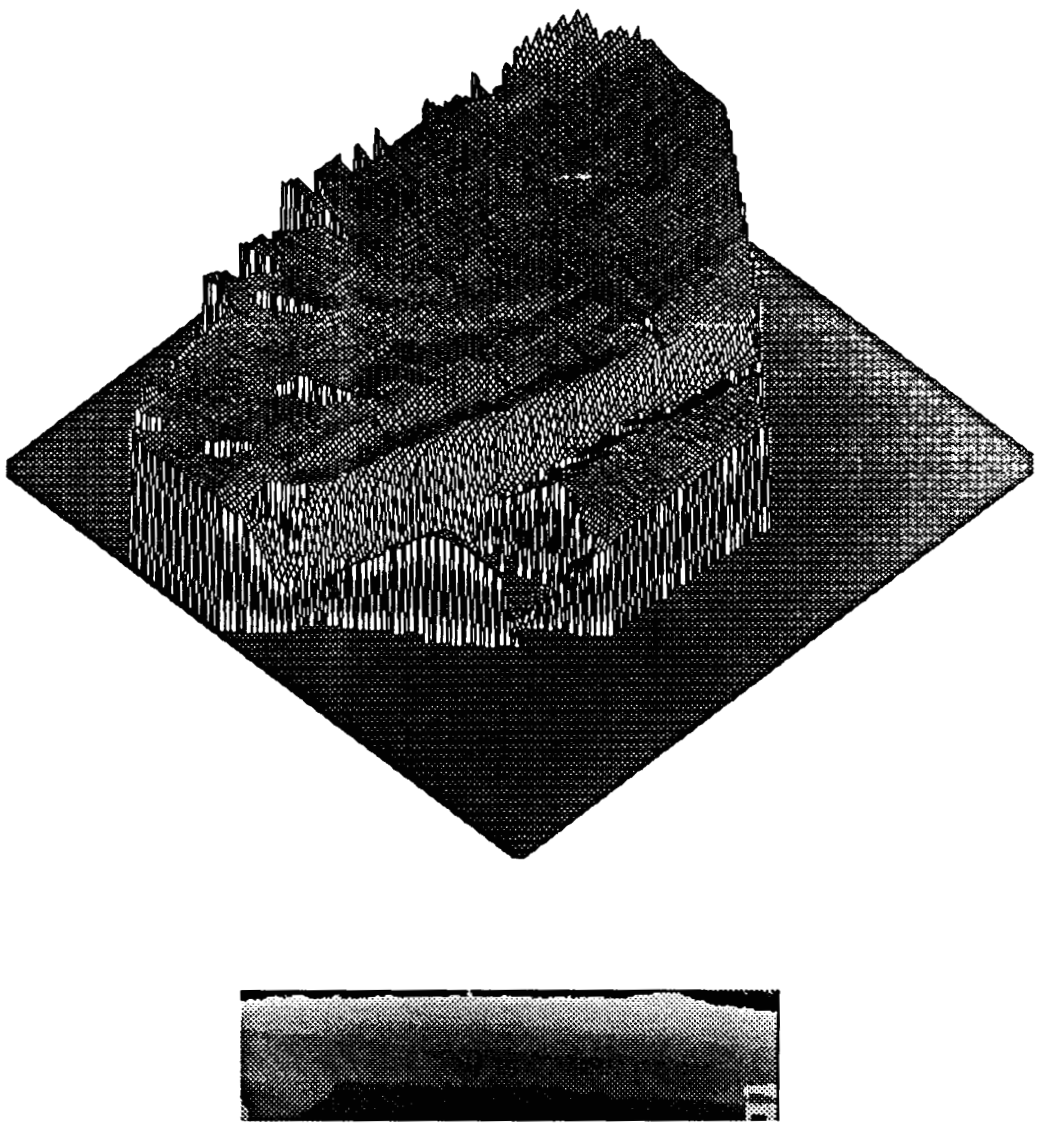

Figure 21: The locus algorithm on range images (Continued)

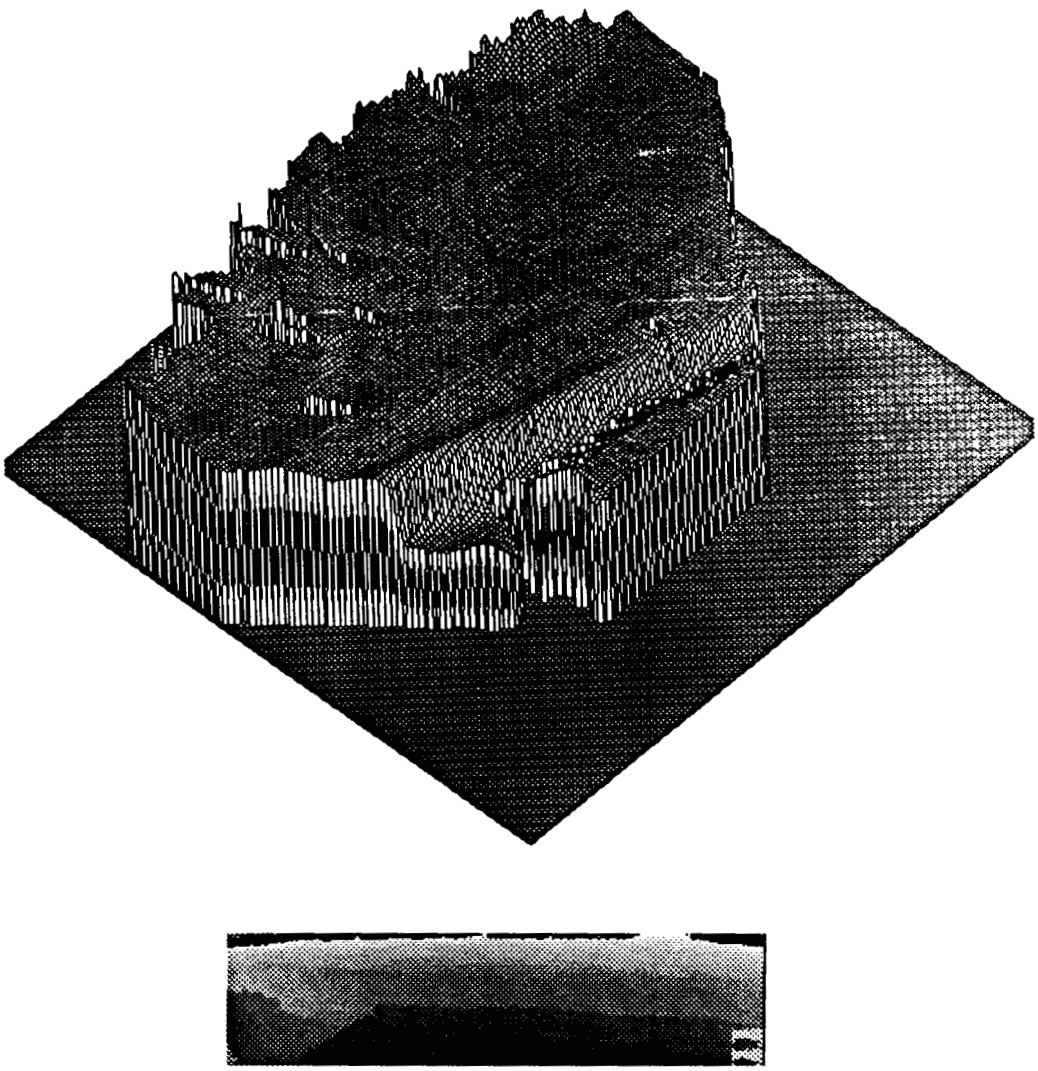

Figure 21: The locus algorithm on range images (Continued)

3.5.3 Evaluating the locus algorithm

We evaluate the locus algorithm by comparing its performance with the other "naive" interpolation algorithms on a set of synthesized range images of simple scenes. The simplest scenes are planes at various orientations. Furthermore, we add some range noise using the model of Section 2.3 in order to evaluate the robustness of the approach in the presence of noise. The performances of the algorithms are evaluated by using the mean square error:

$$E = \frac{\sum_{i=1}^{N}(h_i - \tilde{h}_i)^2}{N} \tag{13}$$

where h_i is the true elevation value and $\tilde{h}_i$ is the estimated elevation. Figure 22 plots E for the locus algorithm and the naive interpolation as a function of the slope of the observed plane and the noise level. This result shows that the locus algorithm is more stable with respect to surface orientation and noise level than the other algorithm. This is due to the fact that we perform the interpolation in image space instead of first converting the data points into the elevation map.

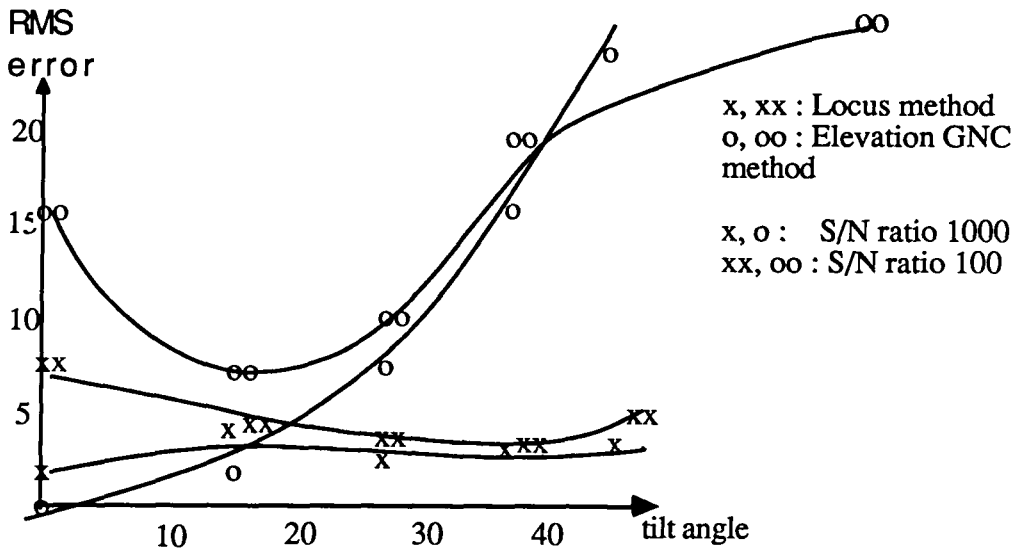

Figure 22: Evaluation of the locus algorithm on synthesized images

3.5.4 Representing the uncertainty

We have presented in Section 2.3 a model of the sensor noise that is a Gaussian distribution along the direction of measurement. We need to transform this model into a model of the noise, or uncertainty, on the elevation values returned by the locus algorithm. The difficulty here is that the uncertainty in a given range value spreads to many points in the elevation map, no matter how the map is oriented with respect to the image plane (Figure 13). We cannot therefore assume that the standard deviation of an elevation is the same as the one of the corresponding pixel in the range image. Instead, we propose to

use the nature of the locus algorithm itself to derive a meaningful value for the elevation uncertainty. To facilitate the explanation, we consider only the case of the basic locus algorithm of Section 3.5.1 in which we compute an elevation z from the intersection of the locus of a vertical line with a depth profile from a range image. Figure 23 shows the principle of the uncertainty computation by considering a locus curve that corresponds to a line in space and the depth profile from the range image in the neighborhood of the intersection point, each point on the depth profile has an uncertainty whose density can be represented by a Gaussian distribution as computed in Section 2.3. The problem is to define a distribution of uncertainty along the line. The value of the uncertainty reflects how likely is the given point to be on the actual surface given the measurements.

Let us consider an elevation h along the vertical line. This elevation corresponds to a measurement direction $\phi(h)$ and a measured range $d'(h)$. If $d(h)$ is the distance between the origin and the elevation h, we assign to h the confidence [39]:

$$l(h) = \frac{1}{\sqrt{2\pi}\sigma(d'(h))} e^{-\frac{(d(h)-d'(h))^2}{2\sigma(d'(h))^2}} \tag{14}$$

where $\sigma(d'(h)$ is the variance of the measurement at the range $d'(h)$. Equation 14 does not tell anything about the shape of the uncertainty distribution $l(h)$ along the h axis except that it is maximum at the elevation h_o at which $d(h) = d'(h)$, that is the elevation returned by the locus algorithm. In order to determine the shape of $l(h)$, we approximate $l(h)$ around h_o by replacing the surface by its tangent plane at h_o. If α is the slope of the plane, and H is the elevation of the intersection of the plane with the z axis, we have:

$$\sigma(d'(h)) = K\frac{H^2(a^2+h^2)}{(a\tan\alpha + h)^2} \tag{15}$$

$$\frac{(d(h)-d'(h))^2}{2\sigma(d'(h))^2} = \frac{(h-h_o)^2(a\tan\alpha+h)^2}{K^2H^4(a^2+h^2)} \tag{16}$$

where a is the distance between the line and the origin in the $x-y$ plane and K is defined in Section 2.3 by $\sigma(d) \approx Kd^2$. By assuming that h is close to h_o, that is $h = h_o + \epsilon$ with $\epsilon \ll h_o$, and by using the fact that $H = h_o + a\tan\alpha$, we have the approximations:

$$\sigma(d'(h)) \approx K(a^2+h_o^2) \tag{17}$$

$$\frac{(d(h)-d'(h))^2}{2\sigma(d'(h))^2} \approx \frac{(h-h_o)^2}{2K^2H^2(a^2+h_o^2)} \tag{18}$$

In the neighborhood of h_o, Equation 18 shows that $(d(h) - d'(h))^2/2\sigma(d'(h))^2$ is quadratic in $h - h_o$, and that $\sigma(d'(h))$ is constant. Therefore, $l(h)$ can be approximated by a Gaussian distribution of variance:

$$\sigma_h^2 = K^2H^2(a^2+h_o^2) = K^2H^2d_o^2 \tag{19}$$

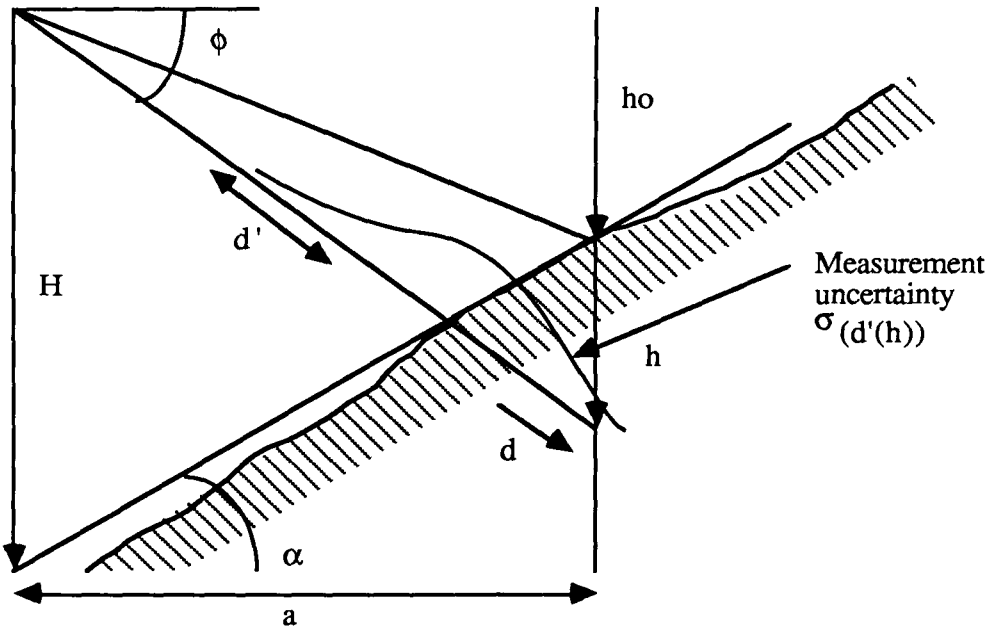

Figure 23: Computing the uncertainty from the locus algorithm

Equation 19 provides us with a first order model of the uncertainty of h derived by the locus algorithm. In practice, the distance $D(h) = (d(h) - d'(h))^2/2\sigma(d'(h))^2$ is computed for several values of h close to h_o, the variance σ_h is computed by fitting the function $(h - h_o)^2/2\sigma_h^2$ to the values of $D(h)$. This is a first order model of the uncertainty in the sense that it takes into account the uncertainty on the sensor measurements, but it does not include the uncertainty due to the locus algorithm itself, in particular the errors introduced by the interpolation.

3.5.5 Detecting the range shadows

As we pointed out in Section 3.1, the terrain may exhibit range shadows in the elevation map. It is important to identify the shadow regions because the terrain may have any shape within the boundaries of the shadows, whereas the surface would be smoothly interpolated if we applied the locus algorithm directly in those areas. This may result in dangerous situations for the robot if a path crosses one of the range shadows. A simple idea would be to detect empty regions in the raw elevation map, which are the projection of images in the map without any interpolation. This approach does not work because the size of the shadow regions may be on the order of the average distance between data points. This is especially true for shadows that are at some distance from the sensor in which case the distribution of data points is very sparse. It is possible to modify the standard locus algorithm so that it takes into account the shadow areas. The basic idea is that a range shadow corresponds to a strong occluding edge in the image (Figure 12). An (x,y) location in the map is in a shadow area if its locus intersects the image at a pixel

that lies on such an edge (Figure 24).

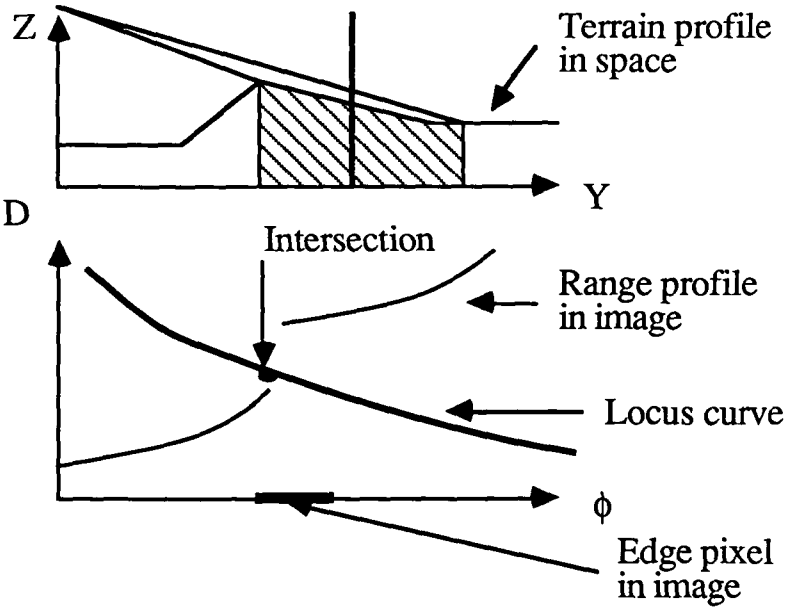

Figure 24: Detecting range shadows

We implement this algorithm by first detecting the edges in the range image by using a standard technique, the GNC algorithm [6]. We chose this algorithm because it allows us to vary the sensitivity of the edge detector across the image, and because it performs some smoothing of the image as a side effect. When we apply the locus algorithm we can then record the fact that the locus of a given location intersects the image at an edge pixel. Such map locations are grouped into regions that are the reported range shadows. Figure 25 shows an overhead view of an elevation map computed by the locus algorithm, the white points are the shadow points, the gray level of the other points is proportional to their uncertainty as computed in the previous Section.

3.5.6 An application: footfall selection for a legged vehicle

The purpose of using the locus algorithm for building terrain is to provide high resolution elevation data. As an example of an application in which such a resolution is needed, we briefly describe in this Section the problem of perception for a legged vehicle [24]. One of the main responsibilities of perception for a legged vehicle is to provide a terrain description that enables the system to determine whether a given foot placement, or *footfall*, is safe. In addition, we consider the case of locomotion on very rugged terrain such as the surface of Mars.

A foot is modeled by a flat disk of diameter 30 cms. The basic criterion for footfall selection is to select a footfall area with the maximum support area which is defined as the contact area between the foot and the terrain as shown in Figure 26. Another constraint for footfall selection is that the amount of energy necessary to penetrate the ground in order to achieve sufficient support area must be minimized. The

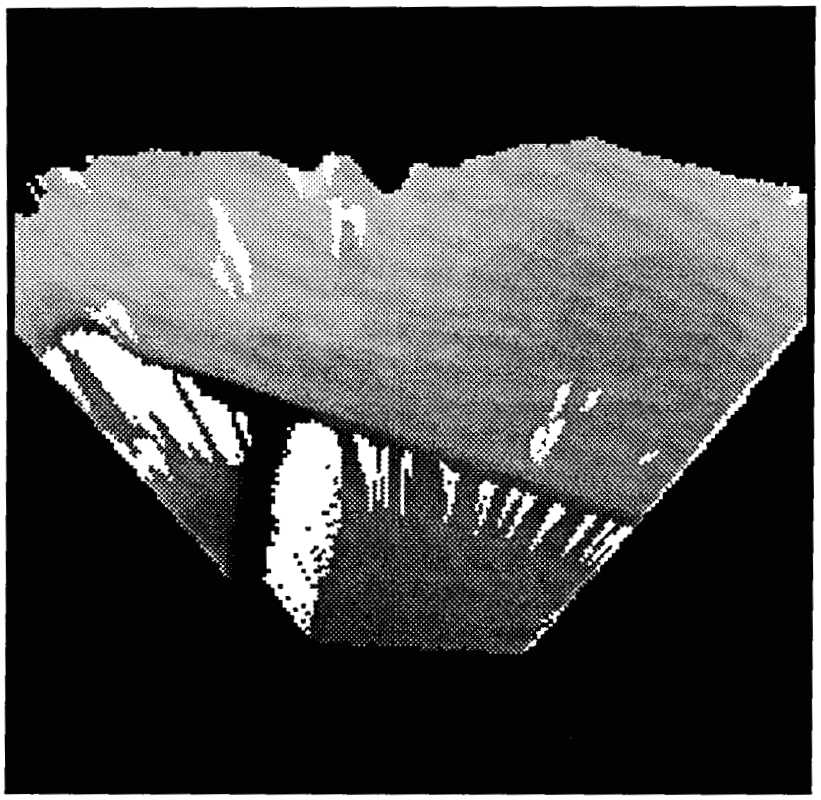

Figure 25: Shadow regions in an elevation map

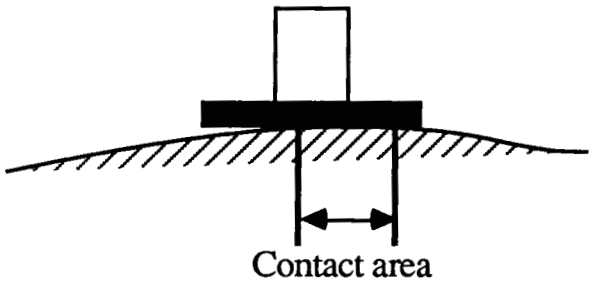

Figure 26: Footfall support area

energy is proportional to the depth of the foot in the ground. The support area is estimated by counting the number of map points within the circumference of the disk that are above the plane of the foot. This is where the resolution requirement originates because the computation of the support area makes sense only if the resolution of the map is significantly smaller than the diameter of the foot. Given a minimum allowed support area, S_{min}, and the high resolution terrain map, we can find the optimal footfall position within a given terrain area: First, we want to find possible flat areas by computing surface normals for each footfall area in a specified footfall selection area. Footfalls with a high surface normal are eliminated. The surface normal analysis, however, will not be sufficient for optimal footfall selection. Second, the support area is computed for the remaining positions. The optimal footfall position is the one for which the maximum elevation, h_{opt} that realizes the minimum support area S_{min} is the maximum across the set of possible footfall positions. Figure 27 shows a plot of the surface area with respect to the elevation from which h_{min} can be computed.

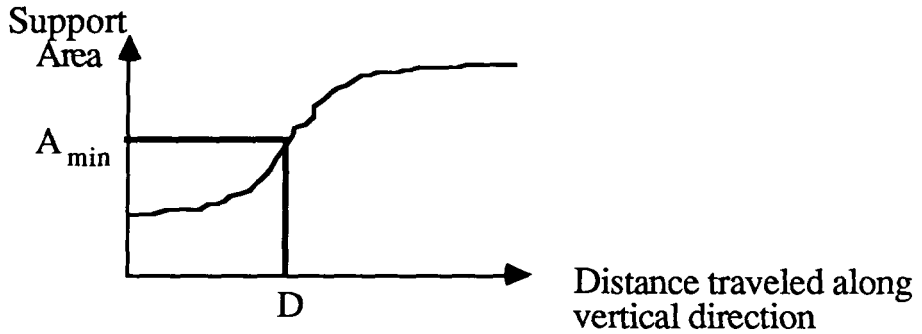

Figure 27: Support area versus elevation

3.5.7 Extracting local features from an elevation map

The high resolution map enables us to extract very local features, such as points of high surface curvature, as opposed to the larger terrain patches of Section 3.4. The local features that we extract are based on the magnitude of the two principal curvatures of the terrain surface. The curvatures are computed as in [34] by first smoothing the map, and then computing the derivatives of the surface for solving the first fundamental form. Figure 28 shows the curvature images computed from an elevation map using the locus algorithm. The resolution of the map is ten centimeters. Points of high curvature correspond to edges of the terrain, such as the edges of a valley, or to sharp terrain features such as hills, or holes. In any case, the high curvature points are viewpoint-independent features that can be used for matching. We extract the high curvature points from both images of principal curvature. We group the extracted points into regions, then classify each region as point feature, line, or region, according to its size, elongation, and curvature distribution. Figure 28 shows the high curvature points extracted from an elevation map.

The two images correspond to the two principal curvatures. Figure 29 shows the three types of local features detected on the map of Figure 28 superimposed in black over the original elevation map. The Figure shows that while some features correspond merely to local extrema of the surface, some such as the edges of the deep gully are characteristic features of the scene. This type of feature extraction plays an important role in Section 4 for combining multiple maps computed by the locus algorithm.

Figure 28: The high curvature points of an elevation map

4 Combining multiple terrain maps

We have so far addressed the problem of building a representation of the environment from sensor data collected at one fixed location. In the case of mobile robots, however, we have to deal with a stream of images taken along the vehicle's path. We could ignore this fact and process data from each viewpoint as if it were an entirely new view of the world, thus forgetting whatever information we may have extracted at past locations. It has been observed that this approach is not appropriate for mobile robot navigation, and that there is a need for combining the representations computed from different vantage points into a coherent map. Although this has been observed first in the context of indoor mobile robots [13,15], the reasoning behind it holds true in our case. First of all, merging representations from successive viewpoints will produce a map with more information and better resolution than any of the individual maps. For example, a tall object observed by a range sensor creates an unknown area behind it, the range shadow, where no useful information can be extracted (Section 3.1). The shape and position of the range shadow changes as we move to another location; merging images from several locations will therefore reduce the size of the shadow, thus providing a more complete description to the path planner (Figure 30). Another reason why merging maps increases the resolution of the resulting representation concerns the fact that the resolution of an elevation map is significantly better at close range. By merging maps, we can increase the resolution of the parts of the elevation map that were originally measured at a distance from the vehicle.

The second motivation for merging maps is that the position of the vehicle at any given time is uncertain. Even when using expensive positioning systems, we have to assume that the robot's idea of its position in the world will degrade in the course of a long mission. One way to solve this problem

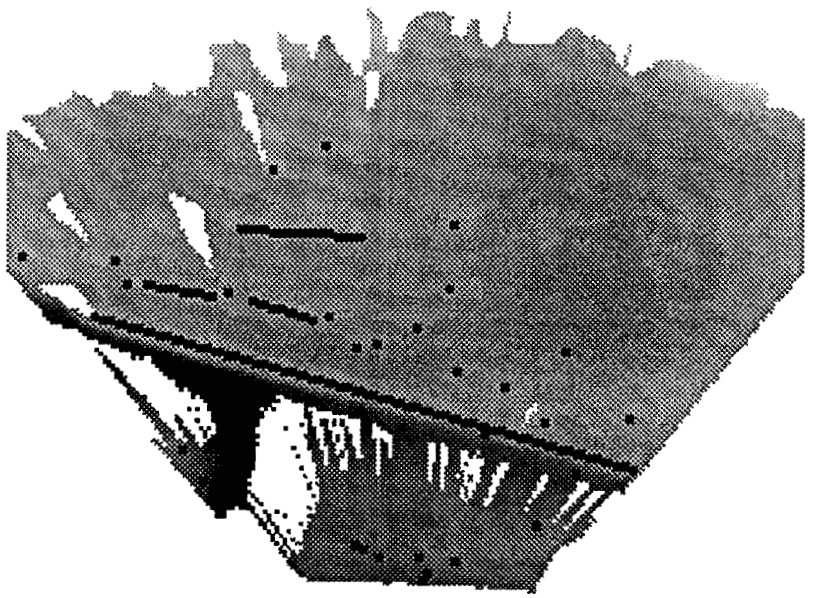

Figure 29: Local features from a high resolution elevation map

is to compute the position with respect to features observed in the world instead of a fixed coordinate system [37,30]. That requires the identification and fusion of common features between successive observations in order to estimate the displacement of the vehicle (Figure 31). Finally, combining maps is a mission requirement in the case of an exploration mission in which the robot is sent into an unknown territory to compile a map of the observed terrain.

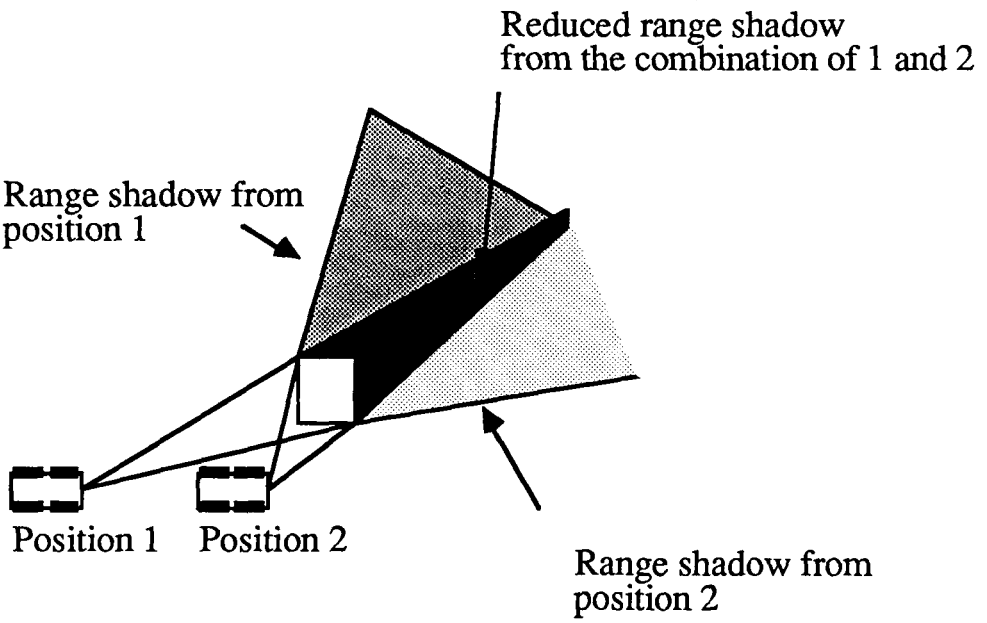

Figure 30: Reducing the range shadow

Many new problems arise when combining maps: representation of uncertainty, data structures for combined maps, predictions from one observation to the next etc. We shall focus on the terrain matching problem, that is the problem of finding common features or common parts between terrain maps so that we can compute the displacement of the vehicle between the two corresponding locations and then merge the corresponding portions of the terrain maps. We always make the reasonable assumption that a rough estimate of the displacement is available since an estimate can always be computed either from dead reckoning or from past terrain matchings.

4.1 The terrain matching problem: iconic *vs.* feature-based

In the terrain matching problem, as in any problem in which correspondences between two sets of data must be found, we can choose one of two approaches: feature-based or iconic matching. In feature-based matching, we first have to extract two sets of features (F_i^1) and (F_j^2) from the two views to be matched, and to find correspondences between features, $(F_{i_k}^1, F_{j_k}^2)$ that are globally consistent. We can then compute the displacement between the two views from the parameters of the features, and finally merge them into one common map. Although this is the standard approach to object recognition problems [5], it has also been

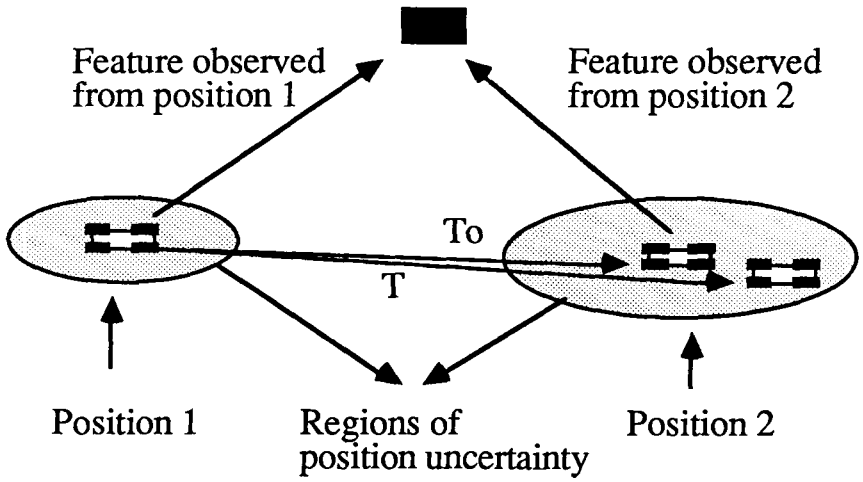

Figure 31: Matching maps for position estimation

widely used for map matching for mobile robots [13,23,30,7,1,41]. In contrast, iconic approaches work directly on the two sets of data points, P^1 and P^2 by minimizing a cost function of the form $F(T(P^2), P^1)$ where $T(P^2)$ is the set of points from view 2 transformed by a displacement T. The cost is designed so that its minimum corresponds to a "best" estimate of T in some sense. The minimization of F leads to an iterative gradient-like algorithm. Although less popular, iconic techniques have been successfully applied to incremental depth estimation [30,29] and map matching [40,12].

The proponents of each approach have valid arguments. The feature-based approach requires a search in the space of possible matches which may lead to a combinatorial explosion of the matching program. On the other hand, iconic approaches are entirely predictable in terms of computational requirements but are usually quite expensive since the size of the points sets P^i is typically on the order of several thousands. As for the accuracy of the resulting displacement T, the accuracy of iconic techniques can be better than the resolution of the sensors if we iterate the minimization of F long enough, while any feature extraction algorithm loses some of the original sensor accuracy. Furthermore, feature matching could in theory be used even if no a-priori knowledge of T, T_0, is available while iconic approaches require T_0 to be close to the actual displacement because of the iterative nature of the minimization of F.

Keeping these tenets in mind, we propose to combine both approaches into one terrain matching algorithm. The basic idea is to use the feature matching to compute a first estimate $\hat{T}$ given a rough initial value T_0, and then to use an iconic technique to compute an accurate estimate $\hat{\hat{T}}$. This has the advantage of retaining the level of accuracy of iconic techniques while keeping the computation time of the iconic stage under control because the feature matching provides an estimate close enough to the true value. We describe in detail the feature-based and iconic stages in the next three sections.

4.2 Feature-based matching

Let F_i^1 and F_j^2 be two sets of features extracted from two images of an outdoor scene, I_1 and I_2. We want to find a transformation $\hat{T}$ and a set of pairs $C_k = (F_{i_k}^1, F_{j_k}^2)$ such that $F_{j_k}^2 \approx \hat{T}(F_{i_k}^1)$, where $T(F)$ denotes the transformed by T of a feature F. The features can be any of those discussed in the previous Sections: points or lines from the local feature extractor, obstacles represented by a ground polygon, or terrain patches represented by their surface equation and their polygonal boundaries. We first investigate the feature matching algorithm independently of any particular feature type so that we can then apply it to any level of terrain representation.

For each feature F_i^1, we can first compute the set of features F_{ij}^2 that could correspond to F_i^1 given an initial estimate T_0 of the displacement. The F_{ij}^2's should lie in a prediction region centered at $T_0(F_i^1)$. The size of the prediction region depends on the confidence we have in T_0 and in the feature extractors. For example, the centers of the polygonal obstacles of Section 3.4 are not known accurately, while the curvature points from Section 3.5.7. can be accurately located. The confidence on the displacement T is represented by the maximum distance δ between a point in image 1 and the transformed of its homologue in image 2, $\|Tp^2 - p^1\|$, and by the maximum angle ϵ, between a vector in image 2 and the transformed of its homologue in image 1 by the rotation part of T. The prediction is then defined as the set of features that are at a Cartesian distance lower than δ, and at an angular distance lower than ϵ from $T_0(F_i^2)$. The parameters used to determine if a feature belongs to a prediction region depend on the type of that feature. For example, we use the direction of a line for the test on the angular distance, while the center of an obstacle is used for the test on the Cartesian distance. Some features may be tested only for orientation, such as lines, or only for position, such as point features. The features in each prediction region are sorted according to some feature distance $d(F_i^1, T_0(F_{ij}^2))$ that reflects how well the features are matched. The feature distance depends also on the type of the feature: for points we use the usual distance, for lines we use the angles between the directions, and for polygonal patches (obstacles or terrain patches) we use a linear combination of the distance between the centers, the difference between the areas, the angle between the surface orientations, and the number of neighboring patches. The features in image 1 are also sorted according to an "importance" measure that reflects how important the features are for the matching. Such importance measures include the length of the lines, the strength of the point features (*i.e.* the curvature value) , and the size of the patches. The importance measure also includes the type of the features because some features such as obstacles are more reliably detected than others, such as point features.

Once we have built the prediction regions, we can search for matches between the two images. The search proceeds by matching the features F_i^1 to the features F_{ij}^2 that are in their prediction region starting at the most important feature. We have to control the search in order to avoid a combinatorial explosion by taking advantage of the fact that each time a new match is added both the displacement and the future matches are further constrained. The displacement is constrained by combining the current estimate T with the displacement computed from a new match (F_i^1, F_{ij}^2). Even though the displacement is described by six components, the number of components of the displacement that can be computed from one single match depends on the type of features involved: point matches provide only three components, line matches provide four components (two rotations and two translations), and region matches provide three components. We therefore combine the components of T with those components of the new match that can be computed. A given match prunes the search by constraining the future potential matches in two

ways: if connectivity relations between features are available, as in the case of terrain patches, then a match (F_i^1, F_{ij}^2) constrains the possible matches for the neighbors of F_i^1) in that they have to be adjacent to F_{ij}^2. In the case of points or patches, an additional constraint is induced by the relative placement of the features in the scene: two matches, (F_i^1, F_{ij}^2) and $(F_{i'}^1, F_{i'j'}^2)$, are compatible only if the angle between the vectors $w^1 = \overrightarrow{F_{i'}^1 F_i^1}$ and $w^2 = \overrightarrow{F_{i'j'}^2 F_{ij}^2}$ is lower than π, provided the rotation part of T is no greater than π which is the case in realistic situations. This constraint means that the relative placement of the features remains the same from image to image which is similar to the classical ordering constraint used in stereo matching.

The result of the search is a set of possible matchings, each of which is a set of pairs $S = (F_{i_k}^1, F_{j_k}^2)_k$ between the two sets of features. Since we evaluated T simply by combining components in the course of the search, we have to evaluate T for each S in order to get an accurate estimate. T is estimated by minimizing an error function of the form:

$$E = \sum_k d(F_{i_k}^1 - T(F_{i_k}^2)) \tag{20}$$

The distance $d(.)$ used in Equation (20) depends on the type of the features involved: For point features, it is the usual distance between two points; for lines it is the weighted sum of the angle between the two lines and the distance between the distance vectors of the two lines; for regions it is the weighted sum of the distance between the unit direction vectors and the distance between the two direction vectors. All the components of T can be estimated in general by minimizing E. We have to carefully identify, however, the cases in which insufficient features are present in the scene to fully constrain the transformation. The matching S that realizes the minimum E is reported as the final match between the two maps while the corresponding displacement $\hat{T}$ is reported as the best estimate of the displacement between the two maps. The error $E(\hat{T})$ can then be used to represent the uncertainty in T.

This approach to feature based matching is quite general so that we can apply it to many different types of features, provided that we can define the distance $d(.)$ in Equation (20), the importance measure, and the feature measure. The approach is also fairly efficient as long as δ and ϵ do not become too large, in which case the search space becomes itself large. We describe two implementations of the feature matching algorithm in the next two Sections.

4.2.1 Example: Matching polygonal representations

We have implemented the feature-based matching algorithm on the polygonal descriptions of Section 3.4 and 3.3. The features are in this case:

- The polygons describing the terrain parametrized by their areas, the equation of the underlying surface, and the center of the region
- The polygons describing the trace of the major obstacles detected (if any).
- The road edges found in the reflectance images if the road detection is reliable enough. The reliability is measured by how much a pair of road edges deviates from the pair found in the previous image.

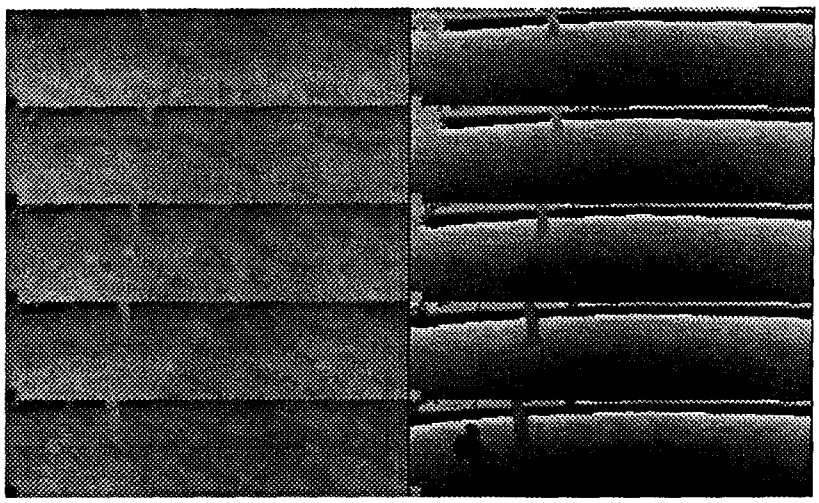

Figure 32: A sequence of range and reflectance images

The obstacle polygons have a higher weight in the search itself because their detection is more reliable than the terrain segmentation, while the terrain regions and the road edges contribute more to the final estimate of the displacement since their localization is better. Once a set of matches and a displacement T are computed, the obstacles and terrain patches that are common between the current map and a new image are combined into new polygons, the new features are added to the map while updating the connectivity between features.

This application of the feature matching has been integrated with the rest of the Navlab system. In the actual system, the estimates of the displacement T_0 are taken from the central database that keeps track of the vehicle's position. The size of prediction region is fixed with δ = one meter, and $\epsilon = 20°$. This implementation of the feature matching has performed successfully over the course of runs of several hundred meters. The final product of the matching is a map that combines all the observations made during the run, and a list of updated obstacle descriptions that are sent to a map module at regular intervals. Since errors in determining position tend to accumulate during such long runs, we always keep the map centered around the current vehicle position. As a result, the map representation is always accurate close to the current vehicle position. As an example, Figure 34 shows the result of the matching on five consecutive images separated by about one meter. The scene in this case is a road bordered by a few trees. Figure 32 shows the original sequence of raw range and reflectance images, Figure 33 shows perspective views of the corresponding individual maps, and Figure 34 is a rendition of the combined maps using the displacement and matches computed from the feature matching algorithm. This last display is a view of the map rotated by 45° about the x axis and shaded by the values from the reflectance image.

4.2.2 Example: Matching local features from high resolution maps

Matching local features from high resolution maps provides the displacement estimate for the iconic matching of high resolution maps. The primitives used for the matching are the high curvature points and

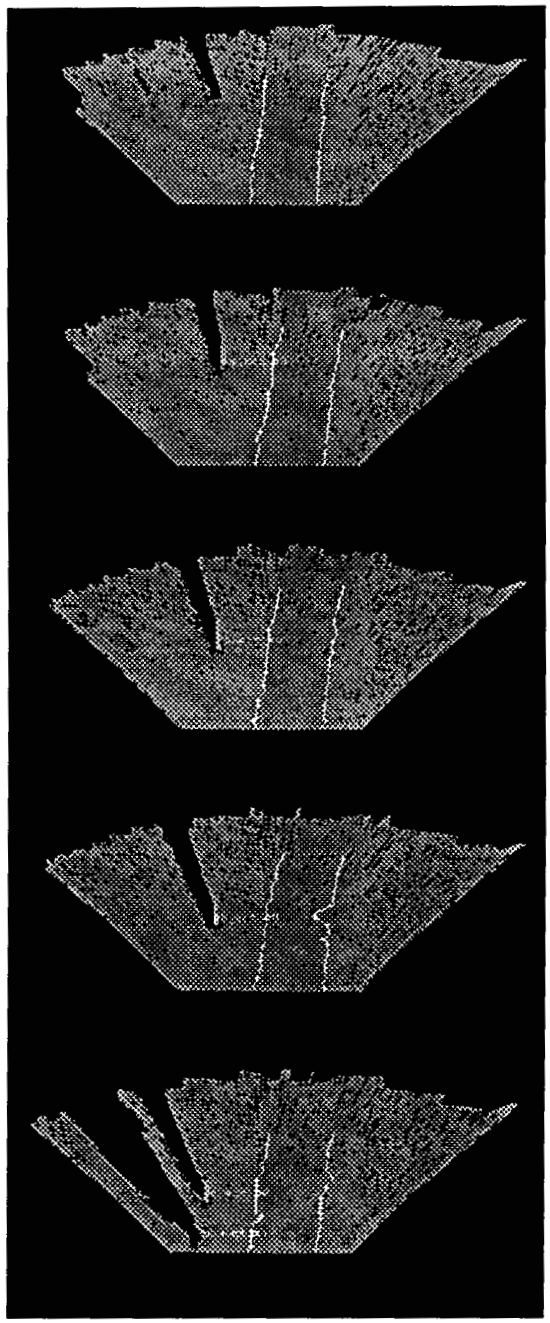

Figure 33: Individual maps

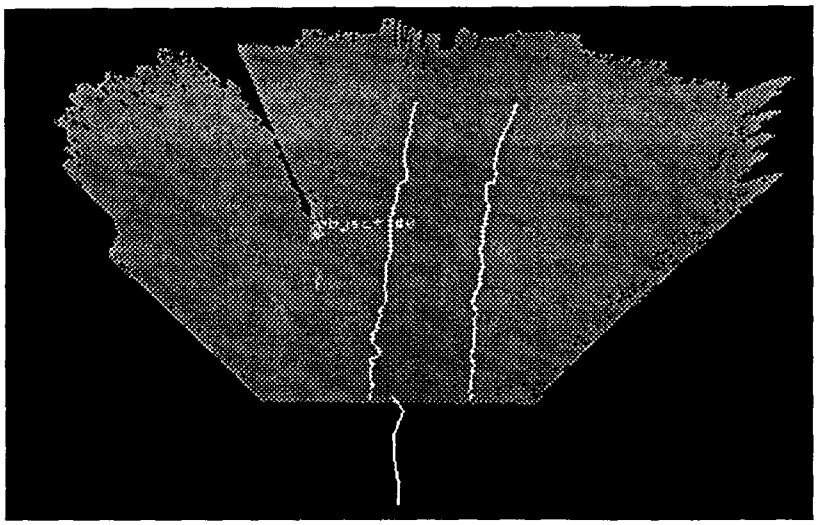

Figure 34: Perspective view of the combined map

lines described in Section 3.5.7. The initial matches are based on the similarity of the length of the lines and the similarity of the curvature strength of the points. The search among candidate matches proceeds as described in Section 4.2. Since we have dense elevation at our disposal in this case, we can evaluate a candidate displacement over the entire map by summing up the squared differences between points in one map and points in the transformed map. Figure 35 shows the result of the feature matching on a pair of maps. The top image shows the superimposition of the contours and features of the two maps using the estimated displacement (about one meter translation and 4° rotation), while the bottom image shows the correspondences between the point and line features in the two maps. The lower map is transformed by T with respect to the lower right map. Figure 36 shows the result of the feature matching in a case in which the maps are separated by a very large displacement. The lower left display shows the area that is common between the two maps after the displacement. Even though the resulting displacement is not accurate enough to reliably merge the maps, it is close enough to the optimum to be used as the starting point of a minimization algorithm.

4.3 Iconic matching from elevation maps

The general idea of the iconic matching algorithm is to find the displacement T between two elevation maps from two different range images that minimizes an error function computed over the entire combined elevation map. The error function E measures how well the first map and the transformed of the second map by T do agree. The easiest formulation for E is the sum of the squared differences between the elevation at a location in the first map and the elevation at the same location computed from the second map using T. To be consistent with the earlier formulation of the locus algorithm, the elevation at any point of the first map is actually the intersection of a line containing this point with the range image. We need some additional notations to formally define E: R and t denote the rotation and translation parts of T respectively, $f_i(u, v)$ is the function that maps a line in space described by a point and a unit vector to

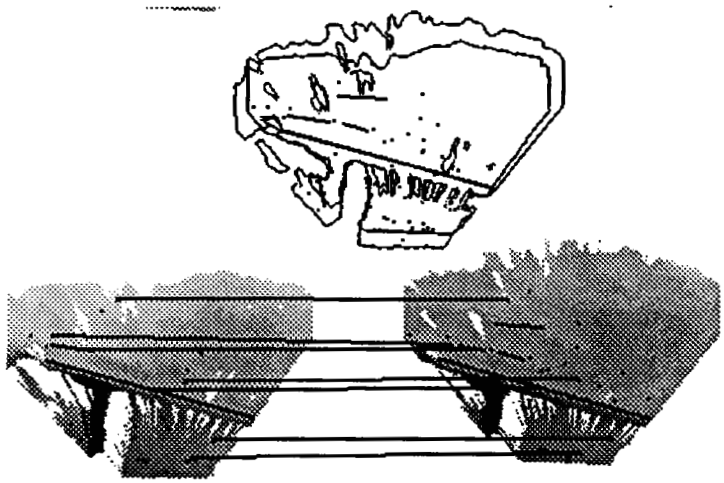

Figure 35: Matching maps using local features

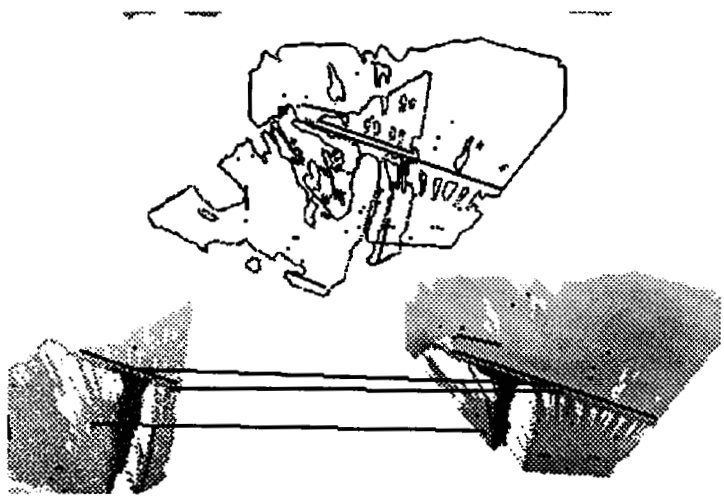

Figure 36: Matching maps using local features (large rotation component)

a point in by the generalized locus algorithm of Section 3.5.2 applied to image i. We have then:

$$E = \sum \|f_1(u,v) - g(u,v,T)\|^2 \tag{21}$$

where $g(u,v,T)$ is the intersection of the transformed of the line (u,v) by T with image 2 expressed in the coordinate system of image 1 (Figure 37). The summation in Equation (21) is taken over all the locations (u,v) in the first map where both $f_1(u,v)$ and $g(u,v,T)$ are defined. The lines (u,v) in the first map are parallel to the z-axis. In other words:

$$g(u,v,T) = T^{-1}(f_2(u',v')) = R'f_2(u',v') + t' \tag{22}$$

where $T^{-1} = (R',t') = (R^{-1}, -R^{-1}t)$ is the inverse transformation of T, and $(u',v') = (Ru + t, Rv)$ is the transformed of the line (u,v). This Equation demonstrates one of the reasons why the locus algorithm is powerful: in order to compute $f_2(Ru + t, Rv)$ we can apply directly the locus algorithm, whereas we would have to do some interpolation or resampling if we were using conventional grid-based techniques. We can also at this point fully justify the formulation of the generalized locus algorithm in Section 3.5.2: The transformed line (u',v') can be anywhere in space in the coordinate system of image 2, even though the original line (u,v) is parallel to the z-axis, necessitating the generalized locus algorithm to compute $f_2(u',v')$.

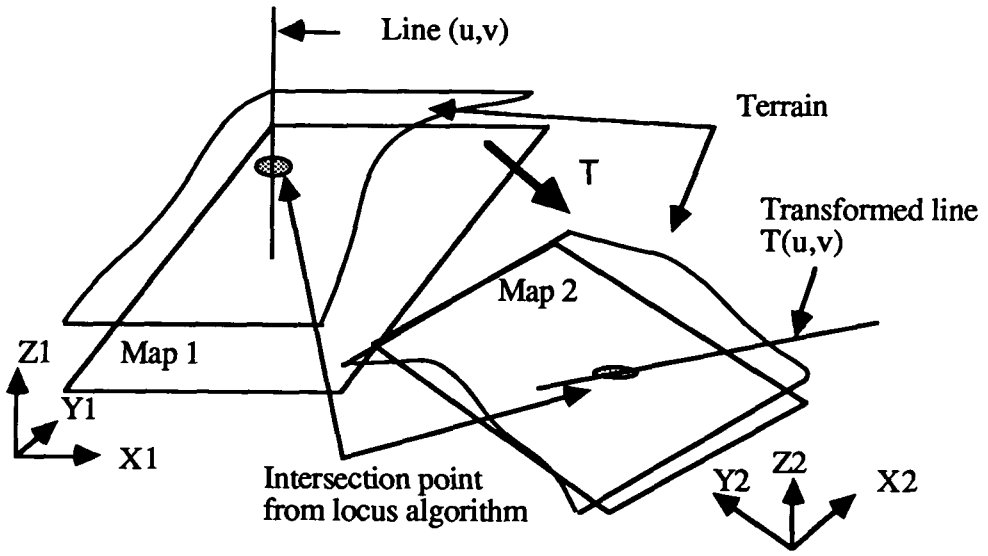

Figure 37: Principle of the iconic matching algorithm

We now have to find the displacement T for which E is minimum. If $\nu = [\alpha, \beta, \gamma, t_x, t_y, t_z]^t$ is the 6-vector of parameters of T, where the first three components are the rotation angles and the last three are the components of the translation vector, then E reaches a minimum when:

$$\frac{\partial E}{\partial \nu} = 0 \tag{23}$$

Assuming an initial estimate T_0, such a minimum can be found by an iterative gradient descent of the form:

$$\nu^{i+1} = \nu^i + k\frac{\partial E}{\partial \nu}(\nu^i) \tag{24}$$

where ν^i is the estimate of ν at iteration i. From Equation (21), the derivative of E can be computed by:

$$\frac{\partial E}{\partial \nu} = -2\sum(f_1(u,v) - g(u,v,T))\frac{\partial g}{\partial \nu}(u,v,T) \tag{25}$$

From Equation (22), we get the derivative of g:

$$\frac{\partial g}{\partial \nu}(u,v,T) = R'\frac{\partial f_2}{\partial \nu}(u',v') + \frac{\partial R'}{\partial \nu}f_2(u',v') + \frac{\partial t'}{\partial \nu} \tag{26}$$

The derivatives appearing in the last two components in Equation (26) are the derivatives of the transformation with respect to its parameters which can be computed analytically. The last step to compute the derivative of $g(u,v,T)$ is therefore to compute the derivative of $f_2(u',v')$ with respect to ν. We could write the derivative with respect to each component ν_i of ν by applying the chain rule directly:

$$\frac{\partial f_2}{\partial \nu_i}(u',v') = \frac{\partial f_2}{\partial u}\frac{\partial u'}{\partial \nu_i} + \frac{\partial f_2}{\partial v}\frac{\partial v'}{\partial \nu_i} \tag{27}$$

Equation (27) leads however to unstabilities in the gradient algorithm because, as we pointed out in Section 3.5.2, the (u,v) representation is an ambiguous representation of lines in space. We need to use a non ambiguous representation in order to correctly compute the derivative. According to equation (13), we can use interchangeably the (u,v) representation and the unambiguous (a,b,p,q) representation. Therefore by considering f_2 as a function of the transform by T, $l' = (a',b',p',q')$, of a line $l = (a,b,p,q)$ in image 1, we can transform Equation (27) to:

$$\frac{\partial f_2}{\partial \nu_i}(l') = \frac{\partial f_2}{\partial l'}\frac{\partial l'}{\partial \nu_i} \tag{28}$$

Since the derivative $\partial f_2/\partial l'$ depends only on the data in image 2, we cannot compute it analytically and have to estimate it from the image data. We approximate the derivatives of f_2 with respect to a,b,p, and q by differences of the type:

$$\frac{\partial f_2}{\partial a} = \frac{f(a+\Delta a,b,p,q) - f(a,b,p,q)}{\Delta a} \tag{29}$$

Approximations such as Equation (29) work well because the combination of the locus algorithm and the GNC image smoothing produces smooth variations of the intersection points.

The last derivatives that we have to compute to complete the evaluation of $\partial E/\partial \nu$ are the derivatives of l' with respect to each motion parameter ν_i. We start by observing that if $X = [x,y,z]^t$ is a point on the line of parameter l, and $X' = [x',y',z']^t$ is the transformed of X by T that lies on a line of parameter l', then we have the following relations from Equation (13):

$$\begin{aligned} x &= az + p, x' = a'z' + p' \\ y &= bz + q, y' = b'z' + q' \end{aligned} \tag{30}$$

By eliminating X and X' between Equation (30) and the relation $X' = RX + t$, we have the relation between l and l':

$$a' = \frac{R_x.V}{R_z.V}, \quad p' = R_x.U + t_x - a'(R_z.U + t_z) \tag{31}$$
$$b' = \frac{R_y.V}{R_z.V}, \quad q' = R_y.U + t_y - b'(R_z.U + t_z)$$

where R_x, R_y, R_z are the row vectors of the rotation matrix R, $A = [a, b, 1]^t$, $B = [p, q, 0]^t$. We now have l' as a function of l and T, making it easy to compute the derivatives with respect to ν_i from Equation (31).

In the actual implementation of the matching algorithm, the points at which the elevation is computed in the first map are distributed on a square grid of ten centimeters resolution. The lines (u, v) are therefore vertical and pass through the centers of the grid cells. E is normalized by the number of points since the since of the overlap region between the two maps is not known in advance. We first compute the $f_1(u, v)$ for the entire grid for image 1, and then apply directly the gradient descent algorithm described above. The iterations stop either when the variation of error ΔE is small enough, or when E itself is small enough. Since the matching is computationally expensive, we compute E over an eight by eight meter window in the first image. The last test ensures that we do not keep iterating if the error is smaller than what can be reasonably achieved given the characteristics of the sensor. Figure 38 shows the result of combining three high resolution elevation maps. The displacements between maps are computed using the iconic matching algorithm. The maps are actually combined by replacing the elevation $f_1(u, v)$ by the combination:

$$\frac{\sigma_1 f_1 + \sigma_2 f_2}{\sigma_1 + \sigma_2} \tag{32}$$

where σ_1 and σ_2 are the uncertainty values computed as in Section 3.5.4. Equation (32) is derived by considering the two elevation values as Gaussian distributions. The resulting mean error in elevation is lower than ten centimeters. We computed the initial T_0 by using the local feature matching of Section 4.2.2. This estimate is sufficient to ensure the convergence to the true value. This is important because the gradient descent algorithm converges towards a local minimum, and it is therefore important to show that T_0 is close to the minimum. Figure 39 plots the value of the ν_i's with respect to the number of iterations. These curves show that E converges in a smooth fashion. The coefficient k that controls the rate of convergence is very conservative in this case in order to avoid oscillations about the minimum.

Several variations of the core iconic matching algorithm are possible. First of all, we assumed implicitly that E is a smooth function of ν; this not true in general because the summation in Equation (21) is taken only over the regions in which both f_1 and g are defined, that is the intersection of the regions of map 1 and 2 that is neither range shadows nor outside of the field of view. Such a summation implicitly involves the use of a non-differentiable function that is 1 inside the acceptable region and 0 outside. This does not affect the algorithm significantly because the changes in ν from one iteration to the next are small enough. A differentiable formulation for E would be of the form:

$$E = \sum \mu_1(u, v)\mu_2(T(u, v))\|f_1(u, v) - g(u, v, T)\|^2 \tag{33}$$

where $\mu_i(u, v)$ is a function that is at most 1 when the point is inside a region where $f_i(u, v)$ is defined and vanishes as the point approaches a forbidden region, that is a range shadow or a region outside of

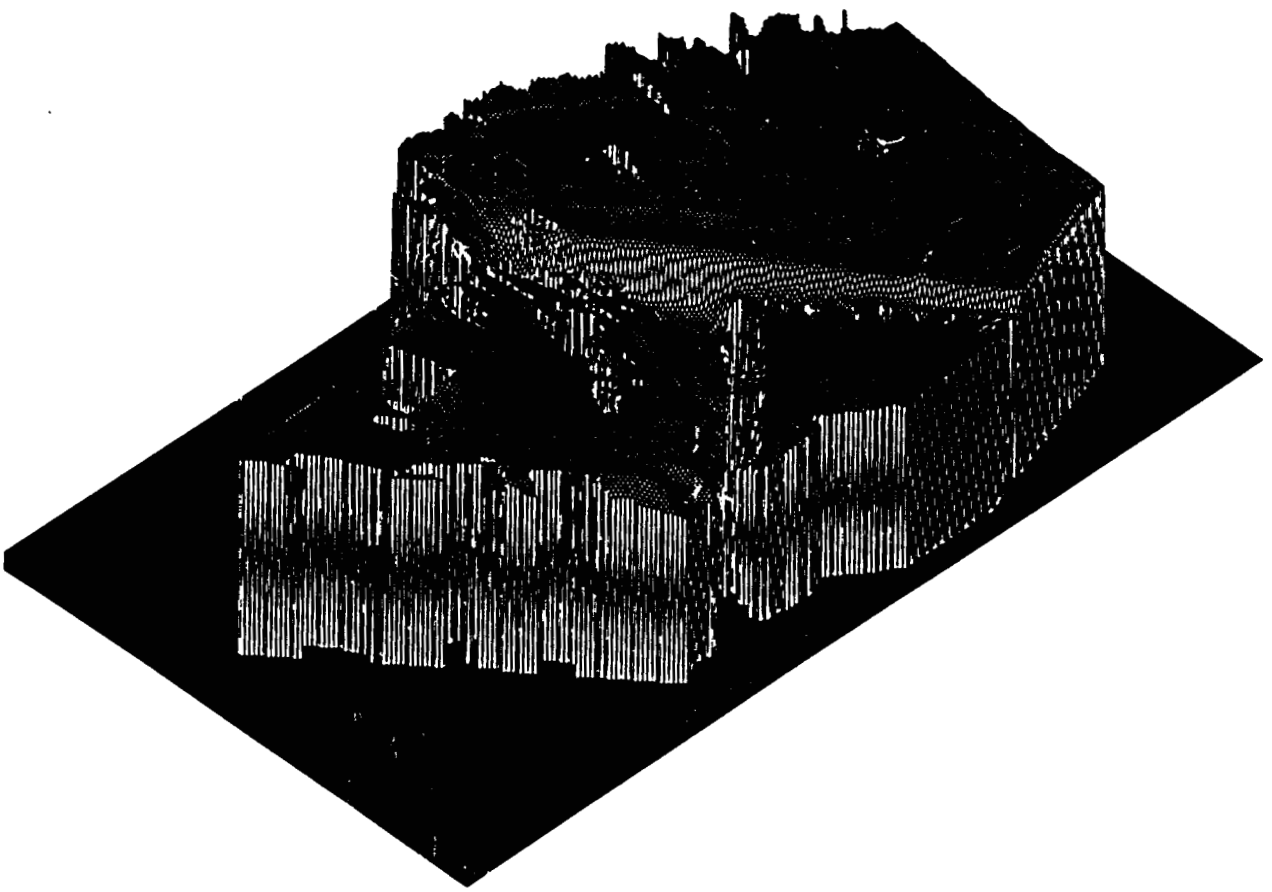

Figure 38: Combining four maps by the iconic matching algorithm

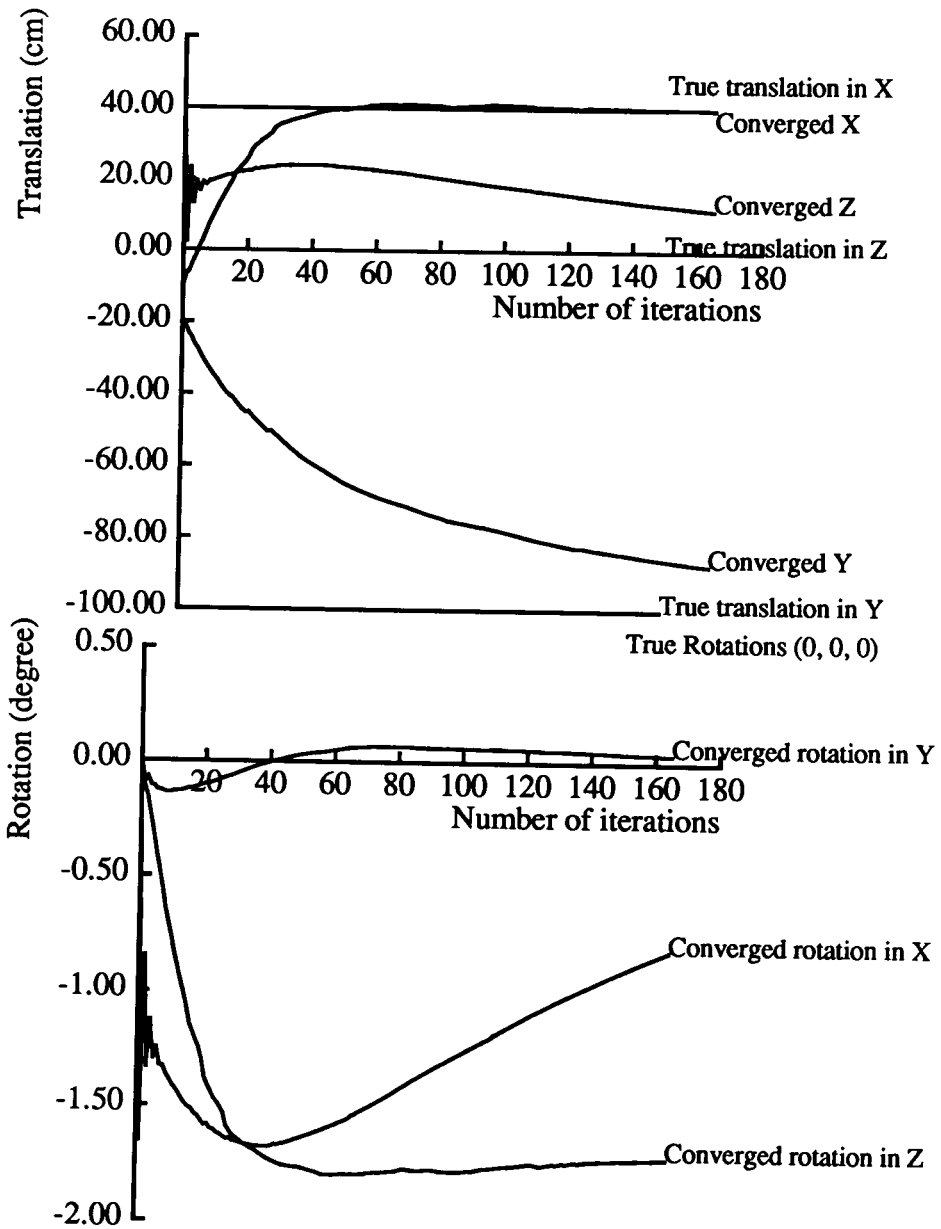

Figure 39: Convergence rate of the matching algorithm

the field of view. The summation in Eq. 33 is taken over the entire map. In order to avoid a situation in which the minimum is attained when the two maps do not overlap ($E = 0$), we must also normalize E by the number of points in the overlap region. For E to be still smooth, we should therefore normalize by:

$$\sum \mu_1(u,v)\mu_2(u,v) \tag{34}$$

In addition to E being smooth, we also assumed that matching the two maps entirely determines the six parameters of T. This assumption may not be true in all cases. A trivial example is one in which we match two images of a flat plane, where only the vertical translation can be computed from the matching. The gradient algorithm does not converge in those degenerate cases because the minimum $T(\nu)$ may have arbitrarily large values within a surface in parameter space. A modification of the matching algorithm that would ensure that the algorithm does converge to some infinite value changes Equation (21) to:

$$E = \sum ||f_1(u,v) - g(u,v,T)||^2 + \sum_i \lambda_i \nu_i^2 \tag{35}$$

The effect of the weights λ_i is to include the constraint that the ν_i's do not increase to infinity in the minimization algorithm.

5 Combining range and intensity data

In the previous Section we have concentrated on the use of 3-D vision as it relates solely to the navigation capabilities of mobile robots. Geometric accuracy was the deciding factor in the choice of representations and algorithms while we gave very little attention to the extraction of semantic information. A mobile robot needs more than just navigation capabilities, however, since it also must be able to extract semantic descriptions from its sensors. For example, we will describe a landmark recognition algorithm in Section 5. In that case, the system is able not only to build a geometric representation of an object but also to relate it to a stored model.

Extracting semantic information for landmark recognition or scene analysis may require much more than just geometric data from a range sensor. For example, interpreting surface markings is the only way to unambiguously recognize traffic signs. Conversely, the recognition of a complex man-made object of uniform color is easiest when using geometric information. In this Section we address the problem of combining 3-D data with data from other sensors. The most interesting problem is the combination of 3-D data with color images since these are the two most common sensors for outdoor robots. Since the sensors have different fields of view and positions, we first present an algorithm for transforming the images into a common frame. As an example of the use of combined range/color images, we describe a simple scene analysis program in Section 5.3.

5.1 The geometry of video cameras

The video camera is a standard color vidicon camera equipped with wide-angle lenses. The color images are 480 rows by 512 columns, and each band is coded on eight bits. The wide-angle lens induces a significant geometric distortion in that the relation between a point in space and its projection on the

image plane does not obey the laws of the standard perspective transformation. We alleviate this problem by first transforming the actual image into an "ideal" image: if (R, C) is the position in the real image, then the position (r, c) in the ideal image is given by:

$$r = f_r(R, C), c = f_c(R, C) \tag{36}$$

where f_r and f_c are third order polynomials. This correction is cheap since the right-hand side of (36) can be put in lookup tables. The actual computation of the polynomial is described in [31] The geometry of the ideal image obeys the laws of the perspective projection in that if $P = [x, y, z]^t$ is a point in space, and (r, c) is its projection in the ideal image plane, then:

$$r = fx/z, c = fy/z \tag{37}$$

where f is the focal length. In the rest of the paper, row and column positions will always refer to the positions in the ideal image, so that perspective geometry is always assumed.

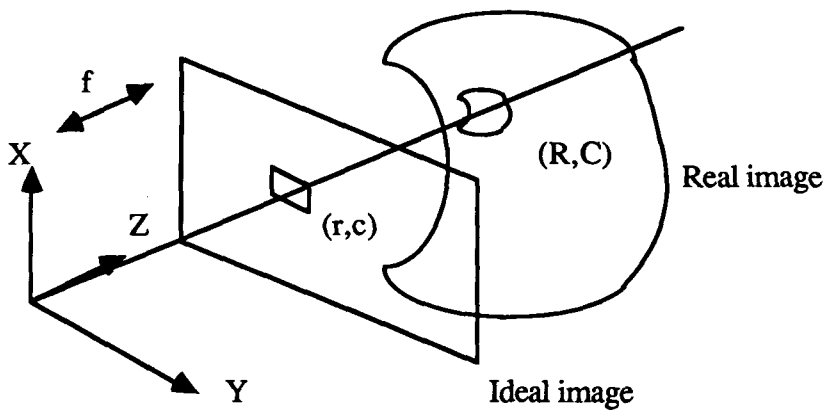

Figure 40: Geometry of the video camera

5.2 The registration problem

Range sensor and video cameras have different fields of view, orientations, and positions. In order to be able to merge data from both sensors, we first have to estimate their relative positions, known as the calibration or registration problem (Figure 41). We approach the problem as a minimization problem in which pairs of pixels are selected in the range and video images. The pairs are selected so that each pair is the image of a single point in space as viewed from the two sensors. The problem is then to find the best calibration parameters given these pairs of points and is further divided into two steps: we first use a simple linear least-squares approach to find a rough initial estimate of the parameters, and then apply a non-linear minimization algorithm to compute an optimal estimate of the parameters.

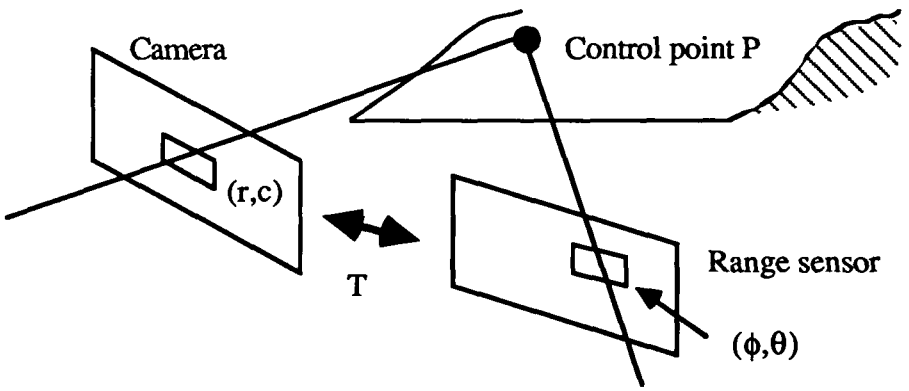

Figure 41: Geometry of the calibration problem

5.2.1 The calibration problem as a minimization problem

Let P_i be a point in space, with coordinates P_i^e with respect to the range sensor, and coordinates P_i^c with respect to the video camera. The relationship between the two coordinates is:

$$P_i^c = RP_i^e - T \tag{38}$$

where R is a rotation matrix, and T is a translation vector. R is a non-linear function of the orientation angles of the camera: pan (α), tilt (β), and rotation (γ). P_i^e can be computed from a pixel location in the range image. P_i^c is not completely known, it is related to the pixel position in the video image by the perspective transformation:

$$z_i^c r_i = f x_i^c \tag{39}$$

$$z_i^c c_i = f y_i^c \tag{40}$$

where f is the focal length. Substituting (38) into (39) and (40) we get:

$$R_z P_i^e r_i - T_z r_i - f R_x P_i^e + T_x' = 0 \tag{41}$$

$$R_z P_i^e c_i - T_z c_i - f R_y P_i^e + T_y' = 0 \tag{42}$$

where R_x, R_y, and R_z are the row vectors of the rotation matrix R, and $T_y' = fT_y$, $T_x' = fT_x$.

We are now ready to reduce the calibration problem to a least-squares minimization problem. Given n points P_i, we want to find the transformation (R,T) that minimizes the left-hand sides of equations (41) and (42). We first estimate T by a linear least-squares algorithm, and then compute the optimal estimate of all the parameters.

5.2.2 Initial estimation of camera position

Assuming that we have an estimate of the orientation R, we want to estimate the corresponding T. The initial value of R can be obtained by physical measurements using inclinometers. Under these conditions,

the criterion to be minimized is:

$$C = \sum_{i=1}^{n}[(A_i - T_z B_i - fC_i + T'_x)^2 + (D_i - T_z E_i - fF_i + T'_y)^2] \tag{43}$$

where $A_i = R_z P_i^e r_i$, $B_i = r_i$, $C_i = R_x P_i^e$, $D_i = R_z P_i^e c_i$, $E_i = c_i$, and $F_i = R_y P_i^e$ are known and T_z, T'_x, T'_y, f are the unknowns.

Equation (43) can be put in matrix form:

$$C = \|U - AV\|^2 - \|W - BV\|^2 \tag{44}$$

where $V = [T'_x, T'_y, T_z, f]^t$, $U = [A_1, .., A_n]^t$, $W = [D_1, .., D_n]^t$, $A = \begin{bmatrix} B_i & 0 & -1 & C_i \\ . & . & . & \\ B_n & 0 & -1 & C_n \end{bmatrix}$, and $B = \begin{bmatrix} E_i & -1 & 0 & F_i \\ . & . & . & \\ E_n & -1 & 0 & F_n \end{bmatrix}$. The minimum for the criterion of Equation (44) is attained at the parameter vector:

$$V = (A^t A + B^t B)^{-1}(A^t U + B^t W) \tag{45}$$

5.2.3 Optimal estimation of the calibration parameters

Once we have computed the initial estimate of V, we have to compute a more accurate estimate of (R, T). Since R is a function of (α, β, γ), we can transform the criterion from equation (43) into the form:

$$C = \sum_{i=1}^{n} \|I_i - H_i(S)\|^2 \tag{46}$$

where I_i is the 2-vector representing the pixel position in the video image, $I_i = [r_i, c_i]^t$, and S is the full vector of parameters, $S = [T'_x, T'_y, T_z, f, \alpha, \beta, \gamma]^t$. We cannot directly compute C_{min} since the functions H_i are non-linear; instead we linearize C by using the first order approximation of H_i [27]:

$$C \approx \sum_{i=1}^{n} \|I_i - H_i(S_0) - J_i \Delta S\|^2 \tag{47}$$

where J_i is the Jacobian of H_i with respect to S, S_0 is the current estimate of the parameter vector, and $\Delta S = S - S_0$. The right-hand side of (47) is minimized when its derivative with respect to ΔS vanishes, that is:

$$\sum_{i=1}^{n} J_i^t J_i \Delta S + J_i^t \Delta C_i = 0 \tag{48}$$

where $\Delta C_i = I_i - H_i(S_0)$. Therefore, the best parameter vector for the linearized criterion is:

$$\Delta S = -\sum_{i=1}^{n} (J_i^t J_i)^{-1} J_i^t \Delta C_i \tag{49}$$

Equation (49) is iterated until there is no change in S. At each iteration, the estimate S_0 is updated by: $S_0 \leftarrow S_0 + \Delta S$.

5.2.4 Implementation and performance

The implementation of the calibration procedure follows the steps described above. Pairs of corresponding points are selected in a sequence of video and range images. We typically use twenty pairs of points carefully selected at interesting locations in the image (*e.g.* corners). An initial estimate of the camera orientation is $(0, \beta, 0)$, where β is physically measured using an inclinometer. The final estimate of S is usually obtained after less than ten iterations. This calibration procedure has to be applied only once, as long as the sensors are not displaced.

Once we have computed the calibration parameters, we can merge range and video images into a colored-range image. Instead of having one single fusion program, we implemented this as a library of fusion functions that can be divided in two categories:

1. Range $\rightarrow$ video: This set of functions takes a pixel or a set of pixels (r^r, c^r) in the range image and computes the location (r^c, c^c) in the video image. This is implemented by directly applying Equations (41) and (42).

2. Video $\rightarrow$ range: This set of functions takes a pixel or a set of pixels (r^c, c^c) in the video image and computes the location (r^r, c^r) in the range image. The computed location can be used in turn to compute the location of a intensity pixel in 3-D space by directly applying Equation (3). The algorithm for this second set of functions is more involved because a pixel in the video image corresponds to a line in space (Figure 40) so that Equations (41) and (42) cannot be applied directly. More precisely, a pixel (r^c, c^c) corresponds, after transformation by (R, T), to a curve C in the range image. C intersects the image at locations (r^r, c^r), where the algorithm reports the location (r^r, c^r) that is the minimum among all the range image pixels that lie on C of the distance between (r^c, c^c) and the projection of (r^r, c^r) in the video image (using the first set of functions). The algorithm is summarized on Figure 42.

Figure 43 shows the colored-range image of a scene of stairs and sidewalks, the image is obtained by mapping the intensity values from the color image onto the range image. Figure 44 shows a perspective view of the colored-range image. In this example [16], we first compute the location of each range pixel (r^r, c^r) in the video image, and then assign the color value to the 64×256 colored-range image. The final display is obtained by rotating the range pixels, the coordinates of which are computed using Equation (3).

5.3 Application to outdoor scene analysis

An example of the use of the fusion of range and video images is outdoor scene analysis [20,26] in which we want to identify the main components of an outdoor scene, such as trees, roads, grass, etc. The colored-range image concept makes the scene analysis problem easier by providing data pertinent to both geometric information (*e.g.* the shape of the trees) and physical information (*e.g.* the color of the road).

5.3.1 Feature extraction from a colored-range image

The features that we extract from a colored-range image must be related to two types of information: the shapes and the physical properties of the observed surfaces.

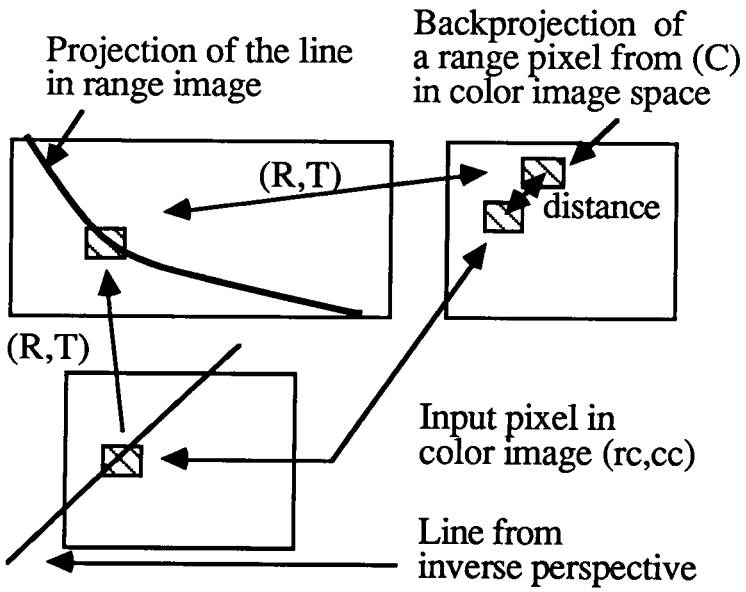

Figure 42: Geometry of the "video → range" transformation

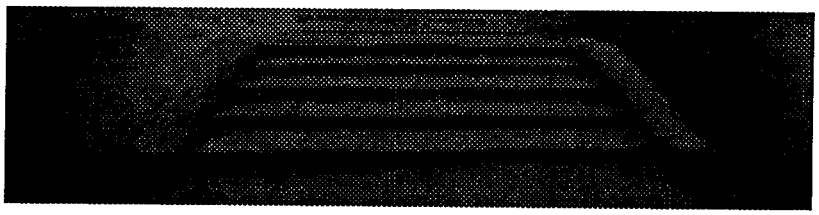

Figure 43: Colored-range image of stairs

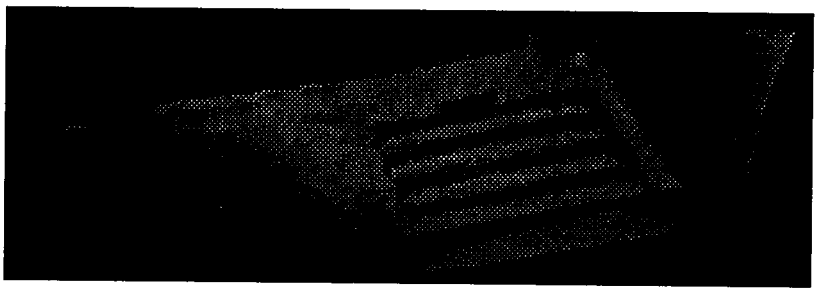

Figure 44: Perspective view of registered range and color images

The geometric features are used to describe the shape of the objects in the scene. We propose to use two types of features: regions that correspond to smooth patches of surface, and edges that correspond either to transitions between regions, or to transitions between objects (occluding edges). Furthermore, we must be able to describe the features in a compact way. One common approach is to describe the regions as quadric patches, and the edges as sets of tri-dimensional line segments. More sophisticated descriptions are possible [5], such as bicubic patches or curvature descriptors. We use simpler descriptors since the range data is relatively low resolution, and we do not have the type of accurate geometric model that is suited for using higher order geometric descriptors. The descriptors attached to each geometric feature are:

- The parameters describing the shape of the surface patches. That is the parameters of the quadric surface that approximate each surface patch.
- The shape parameters of the surface patches such as center, area, and elongations.
- The 3-D polygonal description of the edges.
- The 3-D edge types: convex, concave, or occluding.

The surface patches are extracted by fitting a quadric of equation $X^tAX + B^tX + C = 0$ to the observed surfaces, where X is the Cartesian coordinate vector computed from a pixel in the range image. The fitting error,

$$E(A,B,C) = \sum_{X_i \in patch} [X_i^t A X_i + B^t X_i + C]^2 \tag{50}$$

is used to control the growing of regions over the observed surfaces. The parameters A, B, C are computed by minimizing $E(A,B,C)$ as in [14].

The features related to physical properties are regions of homogeneous color in the video image, that is regions within which the color values vary smoothly. The choice of these features is motivated by the fact that an homogeneous region is presumably part of a single scene component, although the converse is not true as in the case of the shadows cast by an object on an homogeneous patch on the ground. The color homogeneity criterion we use is the distance $(X - m)^t \Sigma^{-1} (X - m)$ where m is the average mean value on the region, Σ is the covariance matrix of the color distribution over the region, and X is the color value of the current pixel in $(red, green, blue)$ space. This is a standard approach to color image segmentation and pattern recognition. The descriptive parameters that are retained for each region are:

- The color statistics (m, Σ).
- The polygonal representation of the region border.
- Shape parameters such as center or moments.

The range and color features may overlap or disagree. For example, the shadow cast by an object on a flat patch of ground would divide one surface patch into two color regions. It is therefore necessary to have a cross-referencing mechanism between the two groups of features. This mechanism provides a two-way direct access to the geometric features that intersect color features. Extracting the relations

between geometric and physical features is straightforward since all the features are registered in the colored-range image.

An additional piece of knowledge that is important for scene interpretation is the spatial relationships between features. For example, the fact that a vertical object is connected to a large flat plane through a concave edge may add evidence to the hypothesis that this object is a tree. As in this example, we use three types of relational data:

- The list of features connected to each geometric or color feature.
- The type of connection between two features (convex/concave/occluding) extracted from the range data.
- The length and strength of the connection. This last item is added to avoid situations in which two very close regions become accidentally connected along a small edge.

5.3.2 Scene interpretation from the colored-range image

Interpreting a scene requires the recognition of the main components of the scene such as trees or roads. Since we are dealing with natural scenes, we cannot use the type of geometric matching that is used in the context of industrial parts recognition [5]. For example, we cannot assume that a given object has specific quadric parameters. Instead, we have to rely on "fuzzier" evidence such as the verticality of some objects or the flatness of others. We therefore implemented the object models as sets of properties that translate into constraints on the surfaces, edges, and regions found in the image. For example, the description encodes four such properties:

- $P1$: The color of the trunk lies within a specific range $\Longrightarrow$ constraint on the statistics (m, Σ) of a color region.
- $P2$: The shape of the trunk is roughly cyclindrical $\Longrightarrow$ constraint on the distribution of the principal values of the matrix A of the quadric approximation.
- $P3$: The trunk is connected to a flat region by a concave edge $\Longrightarrow$ constraint on the neighbors of the surface, and the type of the connecting edge.
- $P4$: The tree has two parallel vertical occluding edges $\Longrightarrow$ constraint on the 3-D edges description.

Other objects such as roads or grass areas have similar descriptions. The properties P_{ij} of the known object models M_j are evaluated on all the features F_k extracted from the colored-range image. The result of the evaluation is a score S_{ijk} for each pair (P_{ij}, F_k). We cannot rely on individual scores since some may not be satisfied because of other objects, or because of segmentation problems. In the tree trunk example, one of the lateral occluding edges may itself be occluded by some other object, in which case the score for $P4$ would be low while the score for the other properties would still be high. In order to circumvent this problem, we first sort the possible interpretations M_j for a given feature F_k according to all the scores $(S_{ij})_i$. In doing this, we ensure that all the properties contribute to the final interpretation and that no interpretations are discarded at this stage while identifying the most plausible interpretations.

We have so far extracted plausible interpretations only for individual scene features F_k. The final stage in the scene interpretation is to find the interpretations (M_{j_k}, F_k) that are globally consistent. For example, property $P3$ for the tree implies a constraint on a neighboring region, namely that this has to be a flat ground region. Formally, a set of consistency constraints C_{mn} is associated with each pair of objects (M_m, M_n). The C_{mn} constraints are propagated through the individual interpretations (M_{j_k}, F_k) by using the connectivity information stored in the colored-range feature description. The propagation is simple considering the small number of features remaining at this stage.

The final result is a consistent set of interpretations of the scene features, and a grouping of the features into sets that correspond to the same object. The last result is a by-product of the consistency check and the use of connectivity data. Figure 45 shows the color and range images of a scene which contains a road, a couple of trees, and a garbage can. Figure 46 shows a display of the corresponding colored-range image in which the white pixels are the points in the range image that have been mapped into the video image. This set of points is actually sparse because of the difference in resolutions between the two sensors, and some interpolation was performed to produce the dense regions of Figure 46.

Only a portion of the image is registered due to the difference in field of view between the two sensors (60° for the camera versus 30° in the vertical direction for the range sensor). Figure 47 shows a portion of the image in which the edge points from the range image are projected on the color image. The edges are interpreted as the side edges of the tree and the connection between the ground and the tree. Figure 48 shows the final scene interpretation. The white dots are the main edges found in the range image. The power of the colored-range image approach is demonstrated by the way the road is extracted. The road in this image is separated into many pieces by strong shadows. Even though the shadows do not satisfy the color constraint on road region, they do perform well on the shape criterion (flatness), and on the consistency criteria (both with the other road regions, and with the trees). The shadows are therefore interpreted as road regions and merge with the other regions into one road region. This type of reasoning is in general difficult to apply when only video data is used unless one uses stronger models of the objects such as an explicit model of a shadowed road region. Using the colored-range image also makes the consistency propagation a much easier task than in purely color-based scene interpretation programs [32].

6 Conclusion

We have described techniques for building and manipulating 3-D terrain representations from range images. We have demonstrated these techniques on real images of outdoor scenes. Some of them (Sections 3.3, 3.4, and 4.2) were integrated in a large mobile robot system that was successfully tested in the field. We expect that the module that manipulates and creates these terrain representations will become part of the standard core system of our outdoor mobile robots, just as a local path planner or a low-level vehicle controller are standard modules of a mobile robot system independent of its application. This work will begin by combining the polygonal terrain representation of Section 3.4 with the path planner of [38] in order to generate the basic capabilities for an off-road vehicle.

Many issues still remain to be investigated. First of all, we must define a uniform way of representing and combining the uncertainties in the terrain maps. Currently, the uncertainty models depend heavily on the type of sensor used and on the level at which the terrain is represented. Furthermore, the displacements

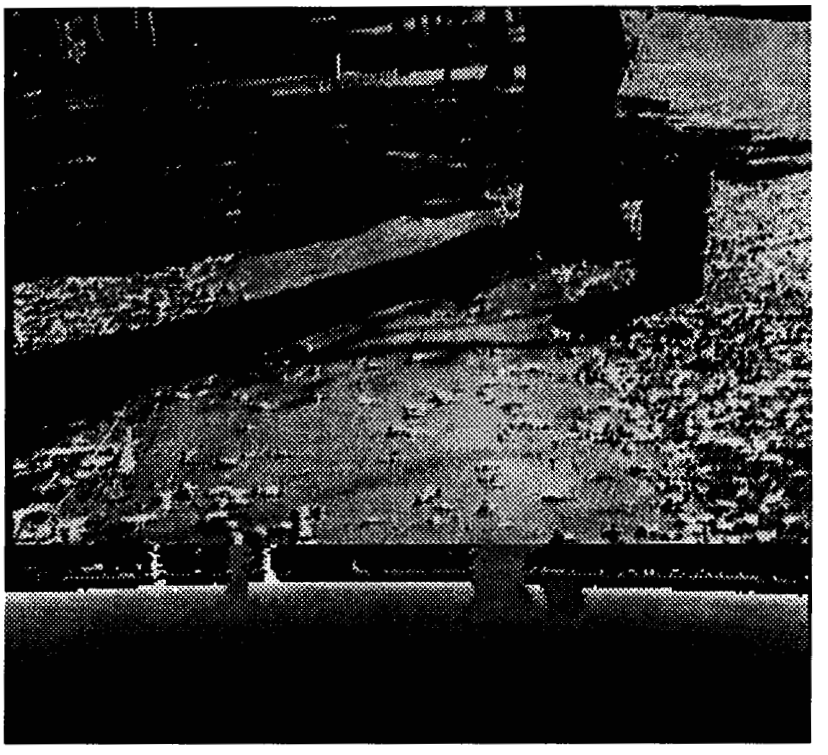

Figure 45: Color and range images of an outdoor scene

Figure 46: A view of the corresponding colored-range image

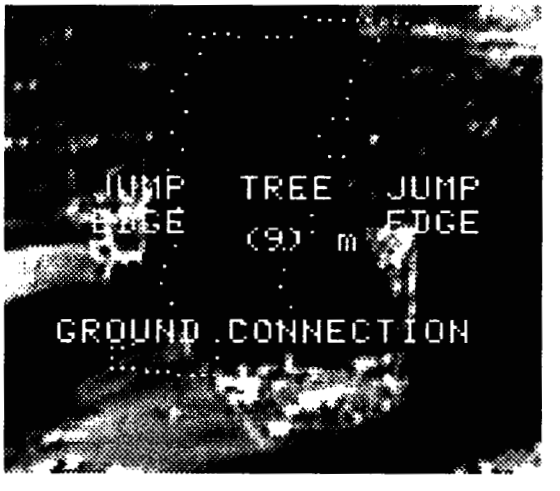

Figure 47: Edge features from the colored-range image

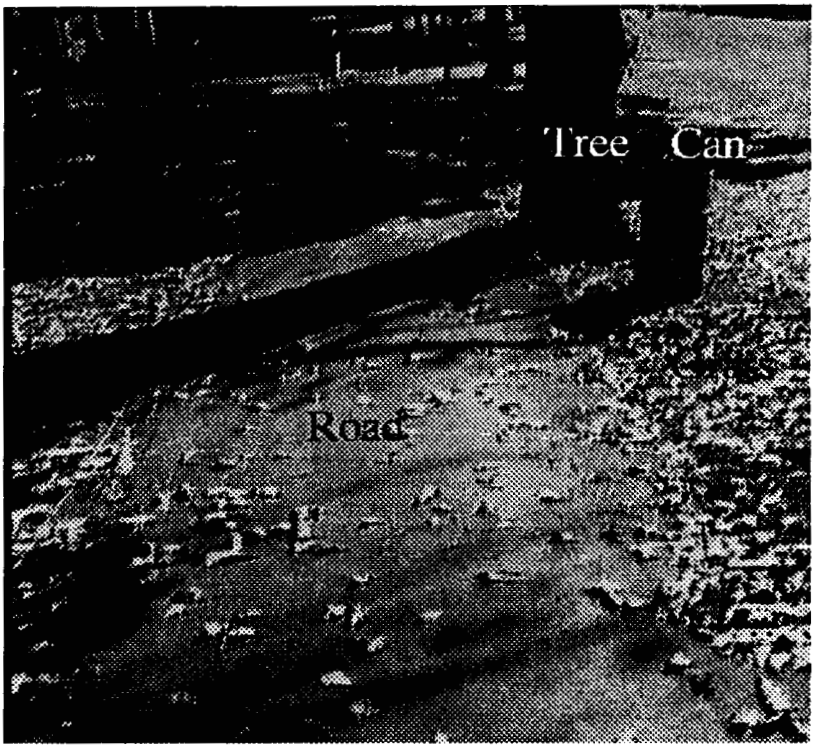

Figure 48: Final scene interpretation

between terrain maps are known only up to a certain level of uncertainty. This level of uncertainty must be evaluated and updated through the matching of maps, whether iconic or feature-based. Regarding the combination of the 3-D representations with representations from other sensors, we need to define an algorithm for sensor registration that is general enough for application to a variety of situations. The algorithms presented in Section 5 are still very dependent on the sensors that we used, and on the intended application. Registration schemes such as [17] would enable us to have a more uniform approach to the problem. An added effect of using such a registration algorithm is that we could explicitly represent errors caused by the combination of the sensors, which we did not do in Section 5. Another issue concerns our presentation of the three levels of terrain representation, the matching algorithms, and the sensor combination algorithms as separate problems. We should define a common perceptual architecture to integrate these algorithms in a common representation that can be part of the core system of a mobile robot. Finally, we have tackled the terrain representation problems mainly from a geometrical point of view. Except in Section 5, we did not attempt to extract semantic interpretations from the representations. A natural extension of this work is to use the 3-D terrain representations to identify known objects in the scene. Another application along these lines is to use the terrain maps to identify objects of interest, such as terrain regions for sampling tasks for a planetary explorer [24]. Although we have performed some preliminary experiments in that respect [19,2], extracting semantic information from terrain representations remains a major research area for outdoor mobile robots.

References

[1] M. Asada. Building a 3-D World Model for a Mobile Robot from Sensory Data. In *Proc. IEEE Robotics and Automation*, Philadelphia, 1988.

[2] J. Bares and W. Whittaker. Configuration of an Autonomous Robot for Mars Exploration. In *Proc. World Conference on Robotics*, 1988.

[3] A. Bergman and C. K. Cowan. Noise-Tolerant Range Analysis for Autonomous Navigation. In *IEEE Conf. on Robotics and Automation*, San Francisco, 1986.

[4] P. Besl. *Range Imaging Sensors.* Technical Report GMR-6090, General Motors Research Lab, Warren, MI, March 1988.

[5] P. J. Besl and R. C. Jain. Three-dimensional Object Recognition. *ACM Comp. Surveys*, 17(1), march 1985.

[6] A. Blake and A. Zisserman. *Visual Reconstruction.* MIT Press, Cambridge, MA, 1987.

[7] R. Brooks. Aspects of Mobile Robot Visual Map Making. In *Second International Robotics Research Symposium*, MIT press, 1985.

[8] M. J. Daily, J. G. Harris, and K. Reiser. Detecting Obstacles in Range Imagery. In *Image Understanding Workshop*, Los Angeles, 1987.

[9] M.J. Daily, J.G. Harris, and K. Reiser. An Operational Perception System for Cross-Country Navigation. In *Proc. Image Understanding Workshop*, Cambridge, 1988.

[10] R. T. Dunlay and D. G. Morgenthaler. Obstacle Detection and Avoidance from Range Data. In *Proc. SPIE Mobile Robots Conference*, Cambridge, MA, 1986.

[11] T. Dunlay. Obstacle Avoidance Perception Processing for the Autonomous Land Vehicle. In *Proc. IEEE Robotics and Automation*, Philadelphia, 1988.

[12] A. Elfes. Sonar-Based Real-World Mapping and Navigation. *Journal of Robotics and Automation, Vol. 3*, 1987.

[13] O.D. Faugeras, N. Ayache, and B. Faverjon. Building Visual Maps by Combining Noisy Stereo Measurements. In *Proc. IEEE Conf. on Robotics and Automation*, 1986.

[14] O.D. Faugeras and M. Hebert. The Representation, Recognition, and Locating of 3-D Objects. *International Journal of Robotics Research*, 5(3), 1986.

[15] G. Giralt, R. Chatila, and M. Vaisset. An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots. In *Proc. 1st International Symposium Robotics Research*, Cambridge, 1984.

[16] Y. Goto, K. Matsuzaki, I. Kweon, and T. Obatake. CMU Sidewalk Navigation System: a Blackboard-Based Outdoor Navigation System Using Sensor Fusion with Colored-Range Images. In *Proc. First Joint Computer Conference*, Dallas, 1986.

[17] K. Gremban, C.E. Thorpe, and T. Kanade. Geometric Calibration Using Systems of Linear Equations. In *Proc. IEEE Robotics and Automation Conf.*, Philadelphia, 1988.

[18] S.Y. Harmon. A Report on the NATO Workshop on Mobile Robot Implementation. In *Proc. IEEE Robotics and Automation*, Philadelphia, 1988.

[19] M. Hebert and T. Kanade. 3-D Vision for Outdoor Navigation by an Autonomous Vehicle. In *Proc. Image Understanding Workshop*, Cambridge, 1988.

[20] M. Hebert and T. Kanade. First Results on Outdoor Scene Analysis. In *Proc. IEEE Robotics and Automation*, San Francisco, 1985.

[21] T. Kanade and J.A. Webb. *End of Year Report for Parallel Vision Algorithm Design and Implementation.* Technical Report CMU-RI-TR-87-15, The Robotics Institute, Carnegie-Mellon University, 1987.

[22] M.G. Kendall and P.A.P. Moran. *Geometrical Probabilities.* Hafner Publishers, New York, 1963.

[23] D. Kriegman, E. Triendl, and T.O. Binford. A Mobile Robot: Sensing, Planning and Locomotion. In *Proc. IEEE Conf. on Robotics and Automation*, 1987.

[24] I. Kweon. Modeling Rugged 3-D Terrain from Multiple Range Images for Use by Mobile Robots. 1988. PhD thesis proposal.

[25] I. Kweon, M. Hebert, and T. Kanade. Perception for Rough Terrain Navigation. In *Proc. SPIE Mobile Robots*, Cambridge, MA, 1988.

[26] I. Kweon, M. Hebert, and T. Kanade. Sensor Fusion of Range and Reflectance Data for Outdoor Scene Analysis. In *Proc. Space Operations Automation and Robotics*, Cleveland, 1988.

[27] D.G. Lowe. Solving for the Parameters of Object Models from Image Descriptions. In *ARPA Image Understanding Workshop*, 1980.

[28] T. Lozano-Perez. An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles. *Communications of the ACM*, October 1979.

[29] B.D. Lucas. *Generalized Image Matching by the Method of Differences.* Technical Report CMU-CS-85-160, Carnegie-Mellon University, 1985.

[30] L. Matthies and S.A. Shafer. Error Modeling in Stereo Navigation. *Journal of Robotics and Automation, Vol. 3*, 1987.

[31] H.P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* Technical Report CMU-RI-TR-3, Carnegie-Mellon University, 1980.

[32] Y. Ohta. *Knowledge-based Interpretation of Outdoor Natural Color Scenes.* Pittman Publishing, Inc., 1984.

[33] D.J. Orser and M. Roche. The Extraction of Topographic Features in Support of Autonomous Underwater Vehicle Navigation. In *Proc. Fifth International Symposium on Unmanned Untethered Submersible*, University of New Hampshire, 1987.

[34] J. Ponce and M. Brady. Toward a Surface Primal Sketch. In *IEEE International Conference on Robotics and Automation*, St Louis, 1985.

[35] K.S. Roberts. A New Representation for a Line. In *Proc. Computer Vision and Patter Recognition*, Ann Arbor, MI, 1988.

[36] S. Shafer and W. Whittaker. *June 1987 Annual Report: Development of an Integrated Mobile Robot System at Carnegie Mellon.* Technical Report CMU-RI-TR-88-10, The Robotics Institute, Carnegie-Mellon University, 1988.

[37] R.C. Smith and P. Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *International Journal of Robotics Research*, 1986.

[38] T. Stenz. *The NAVLAB System for Mobile Robot Navigation.* PhD thesis, Carnegie-mellon University, Fall 1988.

[39] R. Szeliski. *Bayesian Modeling of Uncertainty in Low-Level Vision.* PhD thesis, Carnegie-mellon University, Computer Science Departement, July 1988.

[40] R. Szeliski. Estimating Motion from Sparse Range Data without Correspondance. In *International Conf. on Computer Vision*, Tarpon Springs, Florida, December 1988.

[41] C.E. Thorpe. *The CMU Rover and the FIDO Vision and Navigation System.* PhD thesis, Carnegie-mellon University, 1984.

[42] C.E. Thorpe, M. Hebert, T. Kanade, and S.A. Shafer. Vision and Navigation for the Carnegie-Mellon Navlab. *PAMI*, 10(3), 1988.

[43] M.A. Turk, D.G. Morgenthaler, K.D. Gremban, and M. Marra. VITS- A Vision System for Autonomous Land Vehicle Navigation. *PAMI*, 10(3), may 1988.

[44] R. Watts, F. Pont, and D. Zuk. *Characterization of the ERIM/ALV Sensor - Range and Reflectance.* Technical Report, Environmental Research Institute of Michigan, Ann Arbor, MI, 1987.

[45] J.A. Webb and T. Kanade. Vision on a Systolic Array Machine. In L. Uhr, editor, *Evaluation of multicomputers for image processing*, Academic Press, 1986.

[46] D. Zuk, F. Pont, R. Franklin, and V. Larrowe. *A System for Autonomous Land Navigation.* Technical Report IR-85-540, Environmental Research Institute of Michigan, Ann Arbor MI, 1985.