Table of Contents

Abstract

This report describes progress in vision and navigation for outdoor mobile robots at the Carnegie Mellon Robotics Institute during 1988. This research was primarily sponsored by DARPA as part of the Strategic Computing Initiative. Portions of this research were also partially supported by the National Science Foundation and Digital Equipment Corporation.

In the four years of the project, we have built perception modules for following roads, detecting obstacles, mapping terrain, and recognizing objects. Together with our sister "Integration" contract, we have built systems that drive mobile robots along roads and cross country, and have gained valuable insights into viable approaches for outdoor mobile robot research. This work is briefly summarized in Chapter 1 of this report.

Specifically in 1988, we have completed one color vision system for finding roads, begun two others that handle difficult lighting and structured public roads and highways, and built a road-following system that uses active scanning with a laser rangefinder. We have used 3-D information to build elevation maps for cross-country path planning, and have used maps to retraverse a route. Progress in 1988 on these projects is described briefly in Chapter 1, and in more detail in the following chapters.

Section I

Introduction

This report reviews progress at Carnegie Mellon from January 15, 1988 to January 14, 1989 on research sponsored by the Strategic Computing Initiative of DARPA, DoD, through ARPA Order 5351, and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-0003, titled "Road Following." Portions of this research were also partially sponsored by the National Science Foundation contract DCR-8604199, by the Digital Equipment Corporation External Research Program, and by NASA under contract NAGW-1175.

This first chapter of the report consists of an overview of accomplishments during the four years of the contract; a compendium of our insights and practical advice for building mobile robots; discussion of progress during 1988; a chronology; a list of personnel; and publications of the research group. The following chapters provide more technical detail on particular areas or projects.

Overview of Accomplishments

Outdoor mobile robot research at CMU has been funded by DARPA since January 1985. Although the contract is titled "Road Following", the research is much broader. The scope of the work has included cross-country runs and obstacle detection as well as road following; direct 3-D sensors along with video cameras; object recognition and terrain mapping; and close cooperation with the Warp group and with the Navlab Integration work, to build complete mobile robot systems. Several specific results from the Road Following contract have achieved wide recognition, and have been integrated and demonstrated at CMU and elsewhere:

- **Color-based Road Following.** The culmination of our road-following work is a reliable system that drives the Navlab along a narrow, twisting, tree-lined bicycle path. The heart of the system uses adaptive color classification, which automatically adjusts for changes in road appearance or lighting conditions. Variants of the system use two cameras, to extend the dynamic range to handle deep shadows; find intersections of known shape; incorporate additional features such as texture; and use the Warp processor for high speed. The latest version uses the Warp to achieve a 2 second processing loop, allowing vehicle speeds of 1 meter / second even on our narrow test course.

- **Terrain Representation and Obstacle Detection** We have developed three levels of terrain representation corresponding to different resolutions at which the terrain is described. At the low resolution level we describe only discrete obstacles without explicitly describing the local shape of the terrain. We used this level for fast obstacle detection and avoidance. 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. This level of representation is especially useful for cross-country navigation in which we have to deal not only with large discrete obstacles but also with the changing shape of the terrain. This representation has been successfully demonstrated in conjunction with a path planner developed under the Integration contract. Finally, the description with the highest resolution is a dense elevation map whose resolution is limited only by the sensor. The techniques we developed for this representation provide a complete description of the terrain including occluded regions and uncertainty After the low-resolution obstacle detection was demonstrated as part of the Navlab, it was ported to Martin

Marietta. Work in conjunction with Martin reduced run time to less than one half second, the frame rate of the ERIM scanner. This was the only project during the Martin ALV contract that was developed outside of Martin, integrated into the ALV, and used in one of the ALV main demos.

- **Map Building and Matching** In addition to extracting snapshot maps of the terrain from range images, we have developed algorithms for matching and merging individual maps into a single consistent representation. Again, the matching algorithms are applied to the three levels of representation: At the lowest level discrete obstacles are matched in order to compute the displacement between consecutive maps. At the medium level terrain features are matched to compute the best consistent match between maps. At the highest resolution maps are directly correlated to compute the displacement by a minimization technique. The accuracy of the resulting displacement can be as good as the resolution of the map (as low as 10 cms in our experiments).

- **Road Following by Active Sensing.** Our ERIM scanner measures not only distance to each point but also reflectance. If the road surface (e.g. asphalt) has much different reflectance than the surroundings (e.g. grass), it is straightforward to detect and track the road. For situations in which reflectances do not significantly differ, such as dirt shoulders, we have to pay attention to details of signal attenuation, grazing angle, and surface fitting in order to find the road border. Since the ERIM uses its own laser as its light source, it is insensitive to shadows or lighting changes. This system has even driven the Navlab at night. This method has also been ported to Martin Marietta, and has driven the ALV.

- **Terrain and Object Mapping.**

- **Systems.** The Road Following Contract has provided perception modules for the systems built by our Integration work. Highlights of these systems include:
  - Navigating the Schenley Park bicycle path, starting with a crude map and producing an updated map. This system included color vision for road-following; range data analysis for mapping both discrete obstacles (trees) and terrain; intersection recognition and navigation; a planner that followed the road and avoided obstacles; and sequencing to predict road appearance and to tell perception when to take an image. The system was based on CODGER, our adaptation of blackboard ideas for mobile robot navigation.
  - Navigating the CMU sidewalk network, using a preloaded map to predict object appearance and to choose between a forward-looking and an angle-mounted camera to see the next sidewalk or intersection. The map was also used to invoke a program to locate stairs, which used a "colored-range image" built by fusing camera data with xyz data from the rangefinder.

Other components have been developed and tested, but have not been integrated into complete systems. These include:

- **Sonar.** Some of our earliest successful outdoor runs used Moravec and Elfes's sonar system, originally developed for indoor use, to drive our Terregator robot in Schenley Park. The sonar system was very good at mapping and avoiding natural obstacles such as trees.

- **Stereo.** The FIDO stereo system was ported from indoor laboratory robots to the Terregator, and reimplemented on a prototype Warp. It successfully steered the Terregator around man-made outdoor obstacles, but was less successful with trees and bushes. Future systems could use the complementary strengths of sonar and stereo to build complete and reliable mapping.

- **Other Road Detection Methods.** Our early systems tracked edges, oriented edges, road cross-section profiles, correlation window outputs, and other features. Each of these methods works well, but only in particular circumstances. Current research is using several of these operators together to track the lines, shoulders, and other features of public

highways. A model-based control program will take advantage of the structure of highways to decide which features to track and how to track them. This approach should be robust as well as efficient. Other current work is exploring new methods, such as an unsupervised color classification scheme that uses shape information but does not need color data from previous images. This scheme is not susceptible to quickly changing illumination, and can find the road at the beginning of a run to initialize the color tracker.

- **Calibration.** Our multi-sensor perception experiments need to know the geometrical relationship between sensors. Even for a single sensor, it is important to know the transform from sensor to vehicle coordinates. Our best calibration system uses images of two grids of points to build transform lookup tables, or to derive traditional camera parameters such as location, piercing point, row and column vectors, etc.

- **Object recognition.** In order for a mobile robot to perform a meaningful mission, it must be able to see and recognize known objects. Examples of our object recognition work are two programs for recognizing cars, one using color data and the other using range images. Color car recognition used hierarchical grouping, in which edges are grouped into lines; lines into parallels; parallels into trapezoids; and trapezoids into connected sets that could be car roofs, windows, trunks, or hoods. Starting with range data, the 3-D system first detected flat surfaces, then applied single-surface constraints such as range of orientations allowed for a roof or door, then used surface-pair constraints such as the angle between a roof and door. Both methods work on several views of different cars.

Insights and Advice

Through the course of our work, we have developed some basic maxims of developing outdoor mobile robots. While some of these are scientific insights, most of them have the flavor of pragmatic advice. The most important include:

- Computing is a bottleneck. Our best results use the Warp, rather than a Sun, to gain processing speed. The extra computer power is mostly used not to drive the robot faster but to process images more frequently. Processing images more frequently in space means easier predictions, more objects shared between successive images, and smaller changes in apparent size and shape. Processing more quickly in time means less sensitivity to lighting changes. The 100 MFlops of the Warp help give us a 2-second loop for our current color vision algorithm. But processing remains a bottleneck. Even for the same algorithm, we could use an additional factor of 60 to get to frame rate, times an additional factor of 64 to process higher-resolution images.

- Development environments are a bottleneck. While the Warp gives us vast improvements in processing, until recently it was difficult to harness that power. Hardware developers and computer engineers tend to expect their users to have a few well-specified algorithms that can be compiled once and run many times. But it is the nature of research that programs and parameters need to be changed frequently. To be useful, a supercomputer needs to have debuggers, hardware diagnostics, easy access to display devices, and compilers that run in reasonable amounts of time. Fortunately, those are now becoming available on the Warp.

- Simplicity helps. Object models, algorithms, and systems should be no more complex than needed. A road model, for instance, that attempts to derive too many geometric parameters from a single interpreted image, may be subject to large instabilities due to small errors. We have had much greater success in modeling our road as locally planar and straight. By solving only for the x and theta of the road, we have a stable solution insensitive to minor noise. And by processing quickly, we can track the road as it does eventually turn or pitch, and compensate as we arrive at those points.

- The world changes. Our early outdoor stereo work was foiled by wind-blown trees. Early

color vision made assumptions about constant appearance, and ran afoul of variations in grass color from place to place. Fairly sophisticated vision systems can be fooled by a cloud suddenly covering the sun, which changes not only the intensity but also the color of illumination. The appearance of the road changes from one run to the next, due to our own tire tracks, oil drops, and other effects.

- Sensors are a bottleneck. Too much effort goes into overcoming insufficient dynamic range, fighting noise, and modeling errors. Our solutions include using 2 cameras mounted very close to each other, with different iris settings, to extend the dynamic range. This is an engineering solution to a technology problem, and diverts effort from science. Yet this sort of "hack" is needed to use many current sensors.
- Direct sensing helps. Reasoning in 3-D is much easier when the data starts out in 3-D, such as from a scanning laser rangefinder. Our ERIM data is not perfect, but gives us an excellent starting point for obstacle detection, terrain mapping, and 3-D object recognition.
- Image Understanding (IU) is still needed. There is no direct sensor for "road" or "tree". Furthermore, there are objects and tasks that we do not yet understand how to handle with simple algorithms and models. So even with good 3-D and color sensing, it is still necessary to do all the IU tasks of modeling and interpretation. Direct sensing may eliminate some of the messy low-level interpretation, but does not eliminate the need for fundamental work in IU.
- Integration is difficult but crucial. Capable mobile robots need multiple sensors, probably with multiple sensor interpretation methods, and have multiple goals and multiple control schemes. If the individual components are designed separately, they are not likely to work together. Much of our design and testing effort has been devoted to working with our sister Integration effort to build systems that can follow roads and avoid obstacles; that can look for landmarks while looking for roads; and that can handle other conflicting demands.
- Easy tasks are easier than expected, hard tasks are harder than expected. Following a well-lit sidewalk, bordered by green grass, is nearly trivial. Following a winding path with dirty asphalt, bordered by trees, grass, dirt, and fallen leaves, with changing lighting, is much more difficult.
- Do not trust laboratory simulations, or runs on a few canned images. Simplified or reduced test data is useful for first debugging, but success in the lab does not guarantee success outdoors. There is no substitute for lots of experimental runs.
- Mobile robot research is increasingly important. Results from our work have already been directly applied to interpreting sonar data (for design studies of an underwater autonomous vehicle) and to mapping terrain for planning footfalls for a walking planetary rover. The ideas and experience coming from our project have influenced many other mobile robots, ranging from underground mining vehicles to other road following efforts. And in general, the Road Following work is part of a paradigm shift in image understanding research, moving from generic interpretation of single frames of laboratory data to goa -driven analysis of streams of images from a real, continually moving, outdoor robot.

1988 Progress

In 1988 we neared completion of one of our road following programs, and began work on three new road followers. Our range data processing built maps and, in conjunction with NASA sponsorship, began very high resolution terrain analysis. The highlights of these projects, and of the systems that use them, are briefly described below. Further detail on the major efforts is in the following chapters.

SCARF: In 1988 we completed SCARF, our system for Supervised Classification Applied to Road

Following. SCARF is the logical continuation of a long chain of road following programs that use color classification. The first implementation of SCARF in 1986 ran on Sun workstations, with 32 by 30 pixel images, in about 12 seconds per image. Later implementations of that version ran on the prototype Warp and on production Warps, with speeds as fast as one image per 4 seconds.

Over the past year and a half, we have upgraded SCARF to use, first, higher resolution images (60 by 64), and, second, two images to increase dynamic range. This slowed our runs to tens of seconds per image, even on a Warp.

Now, taking advantage of compiler upgrades for the Warp's W2 language, and doing some code restructuring, we have reimplemented SCARF on the Warp. Our processing time is now down to 2 seconds per image. We moved almost all of the code onto the Warp cells themselves. Further, we reduced the number of calls to the Warp per image from 14 (last year) to 3 (earlier this year) to 1 (now). After initialization, we pass the Warp cells each new image, and get back only the new road location. All of the system state is saved on the cells from call to call. We also have debugging versions that can extract classification information for display, but those extra Warp calls and data movement slow down the system. Current running time is 1 second of Warp time per image.

The full formulation of the probability equation used in classification includes the log of the determinant of each class. Early implementations of SCARF on the Warp have always avoided logarithms, since there is no log function in W2. On benign data, this did not cause any problems. But running with the Navlab outside on a snowy day, the system did not work correctly. In our standard test sequences, each class had approximately the same size determinant (i.e., the classes had approximately equal variance), so we could safely ignore that term. But on a snowy day, the "snow" and "road" classes each had very small variance, while the "trees + parked cars + trash barrel" class had a much larger variance. This imbalance caused improper classifications. We worked with the Warp group to include a log macro and to compile it into our W2 code. The resulting system performs no better on most of our images, but dramatically improves performance on snowy days and under similar circumstances.

The resulting system has driven the Navlab many times, along our narrow bicycle path in Schenley Park. The top speed at which we have run is one meter per second, the length of our test course (compared with 20 cm/sec last year). With the fast processing loop and the complete formulation of probabilities, the vision results are solid. While vehicle speed has always been a secondary concern of our work, we can now drive at moderate speeds on our difficult test course, and should be able to use the same system to drive at higher speeds on wider, straighter roads.

SCARF is described in Chapter 2 of this report, "Color Vision for Road Following".

UNSCARF: One of our new road detection algorithms for this past year is UNSCARF, for UNSupervised Classification Applied to Road Following. A large problem with our early road perception work was dealing with rapidly changing illumination. If the sun is covered by a cloud, the lighting is diffuse and we

can follow roads with a single camera. If the sun is out, there are problems with camera dynamic range, but our methods that use two cameras work. But if the sun is quickly covered or uncovered by clouds, then colors change and shadows change and the brightness changes. If object appearance differs greatly between successive processed frames, current methods have a hard time tracking the road.

UNSCARF places less emphasis on colors and more on shapes. Instead of classifying each pixel according to statistics from previous images, it groups neighboring pixels using unsupervised clustering methods. We have found that clustering with 5 parameters (R,G,B and row,col) gives us classes that are both homogeneous in color and connected in the image. We then piece a road shape together out of those clusters, instead of from individual pixels. Evaluating candidate roads uses shape cues such as parallel edges, smooth edges, edges the right distance apart, and so forth. The combination of unsupervised classification and evaluation with shape cues makes UNSCARF tolerant of the large illumination changes that have given problems to previous systems.

UNSCARF is also described Chapter 2, "Color Vision for Road Following".

FERMI: FERMI deals with public highways and roads, that have more structure and variation than our Schenley Park test site. The key to handling diverse roads is explicit modeling of the colors, shapes, and features of each road type. FERMI has a representation that lists width, maximum curvature, color, surface type, location of lines, type of shoulders, presence of guard rails, type of adjacent vegetation or soil, illumination conditions (sunny or cloudy), illumination direction, and so forth. Then by having many simple experts, one for tracking each type of feature, we are able to follow many kinds of roads within the same control framework. None of the individual trackers (edges, lines, color discontinuities, etc.) that we explored in our early work were adequate in themselves for road following. But by incorporating many of them into a single system, and intelligently selecting which tracker to use to follow which feature, we expect FERMI to be reliable and flexible. In 1988, FERMI has been designed and partially constructed, and has driven the Navlab.

Details of the FERMI design are in Chapter 3, "Explicit Models for Road Following".

ERIM Reflectance: A new project for 1988 is road tracking using the ERIM reflectance data. Our ERIM laser rangefinder produces not only range at each point but also magnitude of reflectance. Since the scanner produces its own illumination, the reflectance images are not distorted by shadows or sunlight or changing cloud cover. Reflectance is affected by distance (less of the illumination is reflected back to the scanner from more distant objects), but this can be compensated for by using the range data. So many of the sources of error in standard video images are not present in active reflectance data.

There still are, however, some problems with using reflectance data. The magnitude of the reflectance changes with grazing angle: the road at larger distances appears at a shallower angle, and reflects less. Reflectance also changes from place to place along the road, as the road surface goes from dirty to clean or from wet to dry. And finally, since reflectance is only a single channel (rather than the three channels

of an RGB camera), not all objects have distinct appearances.

The solution to the grazing angle is to process each image as a series of horizontal bands, so within each band the grazing angle is approximately constant. We keep separate appearance statistics for each of the bands. We handle changes from place to place by updating our appearance models each image. The problem of multiple objects with the same appearance is more difficult. Part of the solution is to limit processing to a band around the predicted road location. Another answer is to use geometric constraints, such as expecting road edges to be locally parallel. But the effectiveness of these solutions depends on the materials that form the road and its borders. Asphalt and grass have much different reflectances, so the portion of our test path that is grass-lined is easy to segment. Dirt, however, can appear much more like asphalt, so in dirt-lined segments we have to use more detailed processing, such as tracking a single road edge when the other edge is indistinct.

Our program to follow roads using ERIM reflectance has run the Navlab many times, including runs at night. This is the first time we have had a usable day/night road following system. The program was also transferred to Martin Marietta, and successfully drove the ALV.

In addition, this work provides the first step in a new project in building and re-using maps. As we drive, we record the position of the road (from reflectance analysis) and of obstacles (from range analysis). When we later retraverse the same path, we use the detected positions of the road and obstacles to locate the Navlab on the map. The map can then be used to predict upcoming obstacles or turns in the road, and to plan paths past the current field of view.

Our work with reflectance processing and road mapping is described in Chapter 4, "Building and navigating maps of road scenes using an active sensor."

Terrain Mapping: The algorithms that build a terrain description made of polygonal regions have been implemented and demonstrated on the Navlab. The resulting description is a mesh of polygons built from an Erim image, each of which is a feature of the terrain. This terrain modeling program provides the type of information required by the new path planner. The combination of terrain modeling and path planning has been demonstrated on the Navlab and is a major step toward cross-country navigation and the implementation of the Core system.

Terrain mapping work is included as part of an overview of all our range data analysis research in the past four years in Chapter 5, "3-D Vision Techniques for Autonomous Vehicles".

Chronology

Feb	Final version of Expert System Road Finding
March	Car recognition using range data complete
April	LASSIE (car recognition with color data) complete
May	Road simulator version 1
June	ERIM reflectance used to follow roads
July	SCARF implemented in C, drives Navlab
Aug	Simple steering / planning programs "quick" and "dirty"
Aug	ERIM reflectance and mapping runs Martin Marietta ALV
Sep	Night runs of Navlab with ERIM
Oct	First offroad runs using Stentz planner with vehicle model
Oct	Retraverse route using map built on first run
Oct	Car recognition with multiple contexts ???
Nov	UNSCARF runs Navlab
Nov	FERMI runs Navlab
Dec	SCARF on Warp runs Navlab, under 1 second Warp time

Personnel

Directly supported by the project, or doing related and contributing research:

Faculty: Martial Hebert, Katsushi Ikeuchi, Takeo Kanade, Eric Krotkov, Steve Shafer, Chuck Thorpe, Jon Webb, William Whittaker.

Staff: Paul Allen, Mike Blackwell, Tom Chen, Jill Crisman, Thad Druffel, Eric Hoffman, Ralph Hyre, Bala Kumar, Jim Moody, Tom Palmeri, Jean-Christophe Robert, David Simon, Hans Thomas, Eddie Wyatt

Visiting scientists: Yoshi Goto, Taka Fujimori, Keith Gremban, Hide Kuga, Masatoshi Okutomi

Graduate students: Omead Amidi, Jennie Kay, Karl Kluge, InSo Kweon, Dean Pomerleau, Doug Reece, Tony Stentz

Publications

Selected publications by members of our research group, supported by or of direct interest to this contract.

J. Crisman and C. Thorpe
Color Vision for Road Following

In SPIE Conference on Mobile Robots, November, 1988
A different version appeared in the proceedings of SIMAP 88, University of Osaka, Japan, May 88.

J. M. Cuschieri and M. Hebert.
Sonar Applications for Underwater Vision.
In ASME Symposium on Current Practices and new Technologies in Ocean Engineering, pages 5-11. ASME, January, 1988.

Y. Goto, S. A. Shafer, A. Stentz.
The Driving Pipeline: A Driving Control Scheme for Mobile Robots.
International Journal of Robotics and Automation, Volume 4, Number 1.
Also appeared as Technical Report CMU-RI-TR-88-8, Carnegie Mellon University, The Robotics Institute, June, 1988.

K.D. Gremban, C.E. Thorpe, and T. Kanade.
Geometric Camera Calibration using Systems of Linear Equations.
In Proc. 1988 IEEE International Conference on Robotics and Automation, pages 562-567. Computer Society Press, Philadelphia, Pennsylvania, April, 1988.
Also in 1988 Proc. of Image Understanding Workshop, pages, 820-825.
Morgan Kaufmann Publishers, Inc., Massachusetts, April, 1988.

M. Hebert and T. Kanade.
3-D Vision for Outdoor Navigation by an Autonomous Vehicle.
In 1988 Proc. of Image Understanding Workshop, pages 593-601. Morgan Kaufmann Publishers, Inc., Cambridge, Massachusetts, April, 1988.

M. Hebert, T. Kanade, and I. Kweon.
3-D Vision Techniques for Autonomous Mobile Robots.
Technical Report CMU-RI-TR-88-12, Carnegie Mellon University, The Robotics Institute, August, 1988.

K. Kluge and C. Thorpe.
Explicit Models For Road Following
submitted to IEEE Conference on Robotics and Automation, 1989.

I. Kweon, M. Hebert, and T. Kanade.
Perception for Rugged Terrain.
In Proc. of SPIE Conference on Mobile Robots. SPIE, November, 1988.

I. Kweon, M. Hebert, and T. Kanade.
Sensor Fusion of Range and Reflectance Data for Outdoor Scene Analysis.

In Proc. of SOAR'88 Space Operations Automation and Robotics. NASA, Wright-State University, Dayton, Ohio, July, 1988.

D. Pomerleau
ALVINN: An Autonomous Land Vehicle In a Neural Network.
To appear in Advances In Neural Information Processing Systems, Vol. 1, 1989, D.S. Touretzky (ed.), Morgan Kaufmann.

C. Thorpe and T. Kanade.
1987 Year End Report for Road Following at Carnegie Mellon.
Technical Report CMU-RI-TR-88-4, Carnegie Mellon University, The Robotics Institute, April, 1988.

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

Chapter II
Color vision for Road Following

Jill D. Crisman and Charles E. Thorpe

Color Vision for Road Following *

Jill D. Crisman and Charles E. Thorpe
Robotics Institute, Carnegie Mellon University
Pittsburgh, PA 15213

October 12, 1988

Abstract

At Carnegie Mellon University, we have two new vision systems for outdoor road following. The first system, called SCARF (Supervised Classification Applied to Road Following), is designed to be fast and robust when the vehicle is running in both sunshine and shadows under constant illumination. The second system, UNSCARF (UNSupervised Classification Applied to Road Following), is slower, but provides good results even if the sun is alternately covered by clouds or uncovered. SCARF incorporates our results from our previous experience with road tracking by supervised classification. It is an adaptive supervised classification scheme using color data from two cameras to form a new six dimensional color space. The road is localized by a Hough space technique. SCARF is specifically designed for fast implementation on the WARP supercomputer, an experimental parallel architecture developed at Carnegie Mellon.

UNSCARF uses an unsupervised classification algorithm to group the pixels in the image into regions. The road is detected by finding the set of regions which, grouped together, best match the road shape. UNSCARF can be expanded easily to perform unsupervised classification on any number of features, and to use any combination of constraints to select the best combination of regions. The basic unsupervised classification segmentation will also have applications outside the realm of road following.

1 Introduction

At Carnegie Mellon University, we have been building successful, color vision based, road following systems for several years [6,7,9,10]. The main emphasis of our road following research is to find unstructured roads in images that are complicated by shadows, leaves or dirt lying on the road, lighting changes, and the like. We initially used edge based techniques, that searched for edges in the image to correspond with road edges in the scene. This proved inadequate for our Schenley Park test site, since often image edges caused by shadows were more distinctive than edges formed from road boundries. Currently we have been using a color classification system, SCARF (Supervised Classification Applied to Road Following), where each pixel in the image is labeled as road or non-road based on the match of its color to previously learned colors. The road is found by looking for the road shape that contains the most 'road' labeled pixels. We also use an unsupervised classification algorithm, UNSCARF (UNSupervised Classification Applied to Road Following), that groups pixels that have similar color and location, and then searches for the combination of groups that best matches the road shape. This paper discusses these two systems.

Other groups have also been working on road-following. In Germany, Dickmanns and Grafe [3,4] view road following as a control problem. They have developed an elegant control formulation that incorporates a simple road edge detector with the vehicle model to drive their vehicle at speeds up to 100 kph. They also use constraints of the autobahn shape

*This research is sponsored by the Strategic Computing Initiative of DARPA, DoD, through ARPA Order 5351, and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-0003, titled "Road Following." Portions of this research were also partially sponsored by the National Science Foundation contract DCR-8604199 and by the Digital Equipment Corporation External Research Program.

and markings. The autobahns are of constant width and are either straight, constant curvature, or clothoid in shape. The rapid processing and structured road model help to limit a search window in the image, and discard the extraneous edges normally found by edge detectors. However, it seems that their trackers could be distracted by the shadows, puddles and road imperfections that plague our test site.

The University of Maryland [2] has also been working on road following. Their system drove an autonomous vehicle based on edge detection. Image edges where tracked from the bottom of the image to the top using an edge detector in a window of the image. Once an edge is located, it is used to constrain the position and orientation of the next window. Then the edges were grouped using a Hough transform to determine which image edges form the best road edge. This system worked well when the dominant edges in the image are road edges, but similar systems at CMU have failed when tracking edges in strong shadows or when leaves or dirt lie on the roads.

At Martin Marietta, the VITS system [8] has achieved impressive speeds on fairly unstructured roads. It projects the three-dimensional color space (red, green, blue) into one dimension, or in later systems two dimensions. It then differentiates the road from non-road by a linear discriminant function. The road/non-road threshold is selected by sampling a part of the image that is guaranteed to be road. This system is similar to CMU road following, but emphasizes speed rather than general capability. Their system works fast, up to 20 kph, on their test site, but it is doubtful that it will work on other test sites, since the color projection is tuned for the features that are best to discriminate their road from their non-road.

Our goal is to build general color vision algorithms that work in a wide variety of situations. In particular, we are working on recognizing unstructured roads in various types of illumination and weather conditions. To give our system general capabilities, we must address the following problems:

- *The objects in the scene undergo spatial changes in appearance.* For example, under sunlight, roads appear to be a different color than they appear in the shade.
- *Objects in the scene undergo temporal changes in appearance.* This may occur when clouds pass over the sun for instance. The change in illumination will cause identical road segments to have different colors from frame to frame.
- *The dynamic range of our cameras is limited.* We cannot digitize meaningful data in dark shadows of a brightly sunlit image, nor can we capture data in the brightly sunlit regions of a dark image.
- *The roads in Schenley Park are very unstructured.* There are no center or bordering lines painted on our roads, as on highways. Many of the road edges are obscured or uneven. The pavement of our roads is deteriorating in places, and the pavement may be covered with the same leaves, dirt, or snow, that appear off road.

Our two new systems, SCARF and UNSCARF, were built to address these problems. Both systems deal with the limited dynamic range of the cameras by using two cameras with different iris settings to capture both dark and bright areas of the scene. SCARF is designed to be a fast, adaptive system. Even though algorithm speed is not a goal of our research, faster algorithms have the advantage of more overlap between frames, if the vehicle speed is constant. When the images are processed closer in time and distance, the lighting conditions are less likely to change dramatically and the road position in the image will not move far between frames. UNSCARF tackles the temporal and spatial changes by processing each image independently of the others. No color models are tracked from frame to frame, making this algorithm insensitive to spatial or temporal changes in color.

In the next section of this paper, we describe the SCARF algorithms and discuss results. UNSCARF is detailed and discussed in the following section, and finally, general conclusions are drawn in the final section of this paper.

2 SCARF

SCARF has evolved by adding more and more capabilities to a simple road following system. A block diagram of this system is shown in figure 1. SCARF uses two cameras to digitize one frame of data. The two color images are reduced by an averaging filter and sent to the classifier. For each pixel in the reduced images, the classifier calculates a likelihood value that describes how well the pixel matches remembered road and non-road colors. The interpreter uses the likelihood values to find the road shape that contains the most likely road pixels. The road location is then used to update the remembered colors for the next frame. The road location is also used to steer the vehicle. This system has been implemented on the WARP supercomputer.

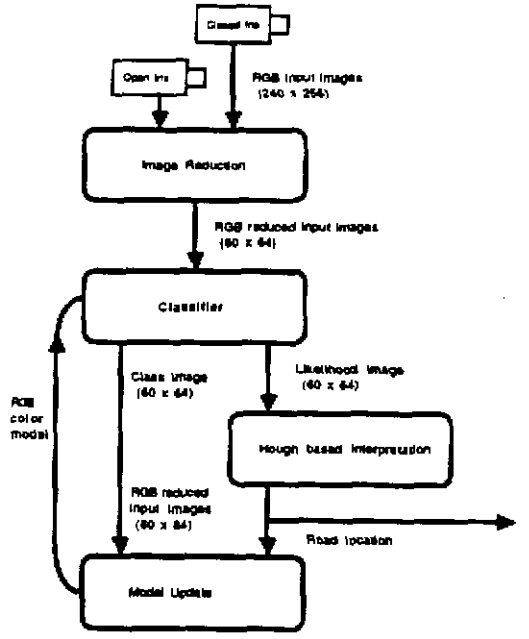


Figure 1: Block diagram of SCARF

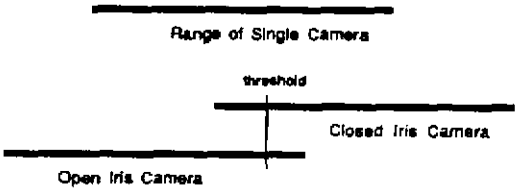


Figure 2: Extending the dynamic range using 2 cameras

2.1 Two Camera System

To extend the dynamic range of a single camera, we are using two images of the same scene digitized from the two cameras mounted on our test vehicle, the Navlab. The cameras were positioned as closely together as possible, and bore-sighted, minimizing the difference between the camera images. To avoid calibrating the two cameras, we treat the images as if they were taken from the same camera. This approximation is adequate for our purposes since the baseline of our cameras is much smaller than the distance to the road.

The improvement in dynamic range results from the different iris settings of the two camera as shown in figure 2. One of the cameras is set to capture the shadowed area of the scene by opening its iris. The second camera captures the sunny areas by using a closed iris.

When the two color images are digitized, they are first reduced by using an averaging filter. This not only reduces the data size, but will also reduce the noise content of the image. The reduced input images are used throughout the program, to increase the speed of the processing.

Two different methods for using the two input images were tested. The first approach is to combine the two reduced images into one. We used a simple thresholding technique to extend the dynamic range as shown in figure 2. If the closed iris image pixel was too dark, then the pixel was selected from the open iris image, otherwise it was copied from the closed iris image. The second approach was to use both reduced input images to form a six-dimensional color space. Then all six features, the red, green, and blue bands from the two images, are used in SCARF.

2.2 Classifier

In standard pattern recognition theory, a classifier takes a d-dimensional measurement vector, x, and chooses the best class label, ω_j, from a set of K classes, using a previously computed, *class conditional* probability, $P(\mathbf{x}|\omega_j)$, for each class [5]. The best class is the class that maximizes the *a posteriori* probability, $P(\omega_j|\mathbf{x})$. The expression for the *a posteriori* probability is normally derived from Bayes rule:

$$P(\omega_j|\mathbf{x}) = \frac{P(\mathbf{x}|\omega_j)P(\omega_j)}{P(\mathbf{x})}.$$

In our case, each pixel provides a 6 dimensional measurement vector (d = 6), $\mathbf{x} = [R_1 G_1 B_1 R_2 G_2 B_2]^T$, formed by concatenating the red, green, and blue bands of the two reduced input images. We use several classes to model road and non-road colors, typically 12 road models and 12 non-road models, giving 24 total color models (K = 24). We assume that the *class conditional* probability models for each class are Gaussian distributions, therefore, $P(\mathbf{x}|\omega_j)$ can be completely characterized by the mean vector and the covariance matrix of the sampled points for class ω_j. We also assume that the $P(\omega_j)$ is the ratio of the number of samples in ω_j, N_j, over the total number of samples, N. Therefore,

$$P(\omega_j|\mathbf{x}) = \frac{2\pi^{-d/2}||\mathbf{C}||^{-1/2} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)}}{P(\mathbf{x})} \frac{N_j}{N}.$$

Rather than calculating $P(\omega_j|\mathbf{x})$ at each pixel, we simplify the calculations by chosing the class, ω_j, that has the maximum $\ln P(\omega_j|\mathbf{x})$. This can be further simplified by noticing that $P(\mathbf{x})$ is identical for all of the classes, so that it can be eliminated from all of the terms. Therefore our classifier selects the class that maximizes the following likelihood measure:

$$\lambda_j = \ln(\frac{N_j}{N}) - \frac{1}{2}\ln((2\pi)^d||\mathbf{C}_j||) - \frac{1}{2}(\mathbf{x} - \mathbf{m_j})^T \mathbf{C}_j^{-1}(\mathbf{x} - \mathbf{m_j}).$$

2.3 Interpretation

The interpretation receives a likelihood image, containing λ_j, and a classification image, containing ω_j, from the classifier. By looking at the classification image, we can label each pixel in the image as either 'road' or 'non-road'. The interpreter searches for the road having the highest combined likelihood using a voting scheme similar to the Hough technique. The standard Hough algorithm searches for a line by voting for all of the lines passing through an edge point. However, we find a road by voting for all the possible roads containing 'road' pixels and by voting against all possible roads containing 'non-road' pixels. The main difference is that all of our pixels vote, not just pixels lying on the edge of the road. We also use the likelihood measure to determine the weight of each vote.

We assume the road is locally nearly straight, and can be parameterized using v, the column where the road center line crosses the horizon row, (or the vanishing point) and θ, the angle between the road center line and a vertical image line. These two parameters are the dimensions of an accumulator used for collecting votes. Each pixel in the likelihood image votes for all the roads that contain that pixel by adding its likelihood to the proper positions in the accumulator.

For each angle θ_i, a given pixel location (r, c) will vote for a set of vanishing points lying between v_s and v_e as shown in figure 3. The starting column position, v_s, corresponds the interpretation that pixel (r, c) lies on on the right hand edge of the road in the image, and the ending column position, v_e corresponds to the (r, c) pixel lying on the left hand edge of the road. The v positions are calculated by:

$$v_s = c + (r - horiz)\tan\theta - (w/l)(r - horiz)$$

$$v_e = c + (r - horiz)\tan\theta + (w/l)(r - horiz).$$

where *horiz* is the horizon row in the image, w is the road width at the bottom of the image, and l is the length from the horizon row to the bottom of the image. The maximum value of the accumulator is chosen to be the road.

2.4 Model Formation

The new model can be calculated using standard statistical equations for mean and covariance:

$$\mathbf{m}_j = \frac{1}{N_j}\sum_{i=1}^{N_j} \mathbf{x}_i \qquad (1)$$

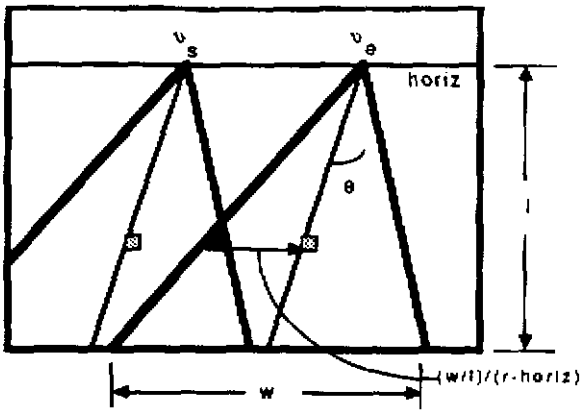


Figure 3: Hough voting scheme

$$\mathbf{C}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{x}_i^T \mathbf{x}_i - \mathbf{m}_j^T \mathbf{m}_j. \tag{2}$$

First we have to decide which pixels belong to each class. Once we have a 'road'/'non-road' labeling, we calculate statistics for the road and non-road classes. Then we reclassify each 'road' or 'non-road' pixel using only the road classes for 'road' pixels and only the non-road classes for 'non-road' pixels. We iterate the calculate statistics and reclassify steps until the classes converge.

The road location is given from the user in initialization or from the interpretation on subsequent steps. Using this location, each pixel can be identified as road or non-road. The road region and the non-road region of the image are shrunk, forming a safety margin at the edge of the road. This is important so that the new color model is not corrupted due to the discretization of road locations or inaccurate fitting of a straight road model to a gently curving road. The reduced road and non-road areas are used to sample the colors of road and non-road.

An iterative clustering technique is applied to the road region and an identical procedure is followed for the non-road region. First, the road pixels are arbitrarily given one of the road class labels. We assign the classes by indexing through the road pixels and assigning the next road class. The color model, consisting of $\{N_j, \mathbf{m}_j, \mathbf{C}_j\}$ is calculated for each of the classes using equations (1) and (2). Then all of the road pixels are relabeled by the class whose mean color is closest to the pixel value, and a new color model is calculated using the new labeling. This 'label/sample' loop is repeated until most of the pixels remain in the same class.

2.5 WARP Implementation

We have implemented one of our supervised classification systems on the wire-wrapped, prototype WARP supercomputer [1]. The increase in processing speed, although significant, was limited by the small memory on each cell. Much time was spent down-loading code and data, at each function call, typically 14 calls per step. Our new PC WARP has more memory on each processing unit, allowing larger programs and larger global data structures. Therefore, we have one large WARP function rather than multiple WARP functions, taking advantage of the larger program memory. This results in greater speeds since data is only downloaded once and the WARP start-up sequence is executed once per image frame.

The inputs to the WARP program are the six reduced images and the statistical model for each class. The WARP function segments, interprets, and produces the new color model. It outputs the road location and the new color model.

2.6 Discussion

This program has been tested on several sequences of images. The SCARF system has driven over all of the roads of our test site successfully. We have driven the system a variety of weather conditions, on different road surfaces, and under different lighting conditions. It adapts very well to different road surface types and differing off-road objects. Figure 4 shows SCARF running through severe shadow conditions from our test site.

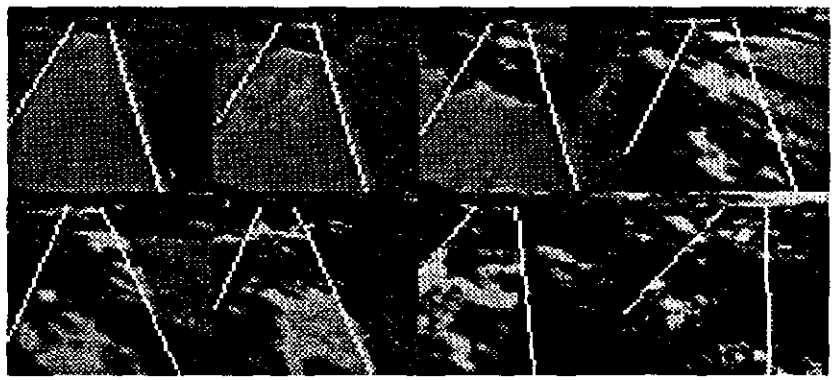


Figure 4: SCARF examples in dark shadows: The lines show the resulting road location.

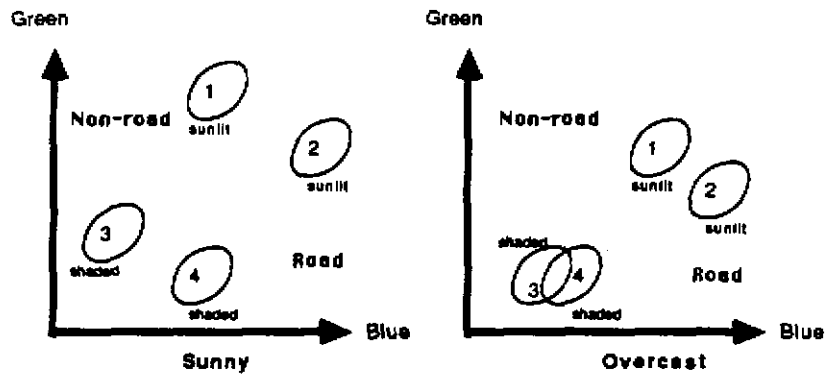


Figure 5: Effects of rapid illumination change

Using two input images does increase the dynamic range beyond that of a single camera. We found that combining the data into a single image provided a fast means of extending the dynamic range, however, using both input images was more reliable. Not only does the use of two images increase the effective dynamic range, it also increased the data available to classify each pixel, thereby increasing classification confidence and accuracy. Moreover, using all the data from both cameras avoids the potential problems of picking a threshold for selecting data to form a single image.

The classification works well as long as the lighting conditions or the road type does not change drastically between frames. As the time and distance between frames decrease, the results from the classification improve.

The Hough interpretation provides the robustness of the SCARF system. Since it is an area based technique, there is more data used in the interpretation than an edge based technique. This makes the system less sensitive to noise. Using this interpretation, we have been able to drive our Navlab vehicle in a variety of weather conditions. The results are good even when the road may be partially covered with the same leaves, dirt, or snow that is on the non-road parts of the image.

3 UNSCARF

UNSCARF was designed to attack the problem with temporal and spatial color appearance changes. In SCARF, models of road and non-road colors, taken from the previous image, were used to label pixels in the current image. However, if the color appearance of these objects changed dramatically, for any reason, then the color models no longer represented the colors of the objects in the new image. SCARF performed well as long as the illumination did not change too quickly. An example of a failure situation due to rapid illumination changes is shown in figure 5. A classifier is calculated for sunny and shaded road and non-road classes in a sunlit scene as shown on the left. In the next image, that classifier will fail, since the sun is hidden by clouds and the colors of the road and non-road classes have shifted.

UNSCARF does not use pre-computed color models, instead it groups pixels that are similar in color and location in the image using an unsupervised classification algorithm as shown in figure 6. Then the pixels with the same labels, or

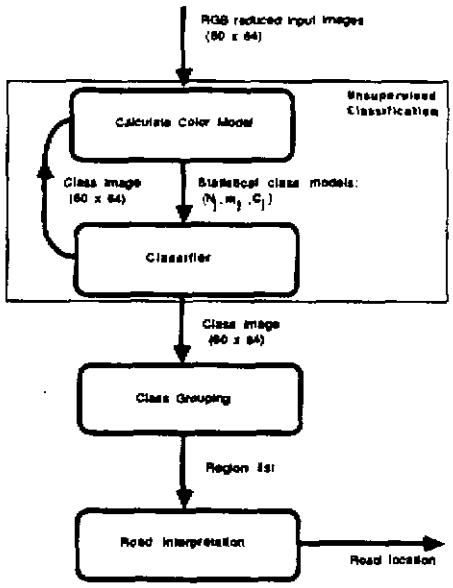


Figure 6: UNSCARF block diagram

classes, are collected into regions using a connected components algorithm, and polygon approximations are fit to the pixel regions. Finally we search for the combination of polygons that, taken together, forms the best road shape.

3.1 Unsupervised Classification Segmentation

The unsupervised classification algorithm groups pixels having similar colors and locations by an iterative clustering technique similar to the model update of the SCARF system. The main difference is that none of the pixels have a 'road' or 'non-road' label. The pixels are given an initial classification. Then a statistical model is calculated for each class, and the pixels are reclassified using the new model. This is repeated until the classes converge.

Each pixel of the input image has five features (d = 5) that are used in the clustering:

$$\mathbf{x} = [RED, GREEN, BLUE, row, column]^T.$$

This can easily be expanded to a eight dimensional space by using the RGB bands of a second image. The system has a fixed number of classes in each image, in our typically 24. First it labels each pixel of the image with a class, insuring that the classes are evenly scattered throughout the image. Next, a statistical color model, $\{N_j, \mathbf{m}_j, C_j\}$, is formulated for each class, ω_j, for this class assignment using equations (1) and (2). Then the pixels are labeled using a classification scheme similar to that of SCARF and a new statistical model is calculated. The 'classify/sample' loop is repeated until most of the pixels in the image remain in the same class. This usually converges quite quickly, taking between 8 and 15 iterations until 95% of the pixels remain in the same class.

The classification scheme can have several different flavors. The first scheme used was a nearest mean classifier. In other words, the pixels were labeled with the class whose mean was closest to the pixel value. This has a tendency to form spherical clusters in the feature space. Since we were using the spatial parameters of $(row, column)$ all of the regions formed from the final class labeling have a approximately circular shapes in the image. To allow elliptical shapes to represent elongated or linear objects we used the nearest class as given by the Mahalanobis distance:

$$d_j = (\mathbf{x} - \mathbf{m}_j)^T C^{-1} (\mathbf{x} - \mathbf{m}_j).$$

This distance metric needed to be normalized since once one of the classes had a larger covariance than any of the other classes, then in the reclassification, even more pixels would be classified as the large class. This would balloon until all of

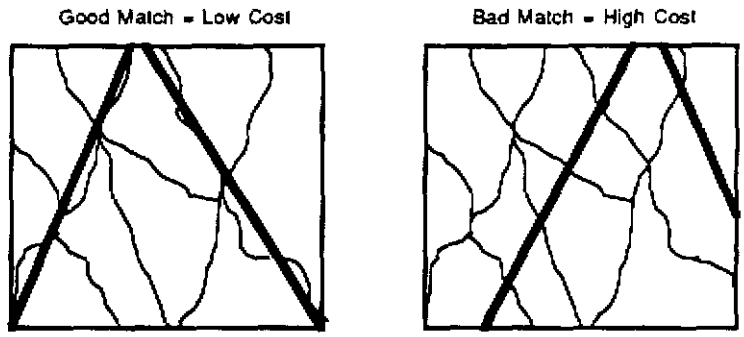


Figure 7: Road costs

the pixels were described by one class. To avoid this, we normalize the distance metric, by dividing each element of the covariance matrices of each class by the d^{th} root of the determinant of that matrix :

$$C'_{i,j} = C_{i,j}/\|C\|.$$

Then the Mahalanobis distance metric is calculated using C':

$$d_j = (\mathbf{x} - \mathbf{m}_j)^T C'^{-1} (\mathbf{x} - \mathbf{m}_j).$$

This allows each class to have a different elliptical shape, while maintaining the same size for each class and thus preventing one class from dominating the others.

Selecting the initial classes scattered throughout the image, causes the (*row*, *column*) parameter statistics to be almost identical for all classes. Therefore the initial classification is based solely on color. In later steps, however the (*row*, *column*) parameters are valuable. By clustering with color, we assume that an object will have a similar color throughout. By adding the (*row*, *column*) parameters, we are exploiting the constraint that objects are localized in the image. The positional constraint made segmentations cleaner than strictly color constraints, by eliminating small noisy regions.

3.2 Interpretation

The interpreter is based on evaluating all possible roads that will could appear in the image. The evaluation function looks at the difference between the road shape and the region edges in the image. The interpreter uses the same two road parameters as in SCARF: v, the column at which the road crosses the horizon, and θ, the angle between the center line and a vertical line in the image. However, instead of building an accumulator, we step through all of the interpretations and evaluate how well that interpretation fits the regions of the image.

To evaluate a candidate road, we first decide which regions would be part of theat candidate road. This is done by testing if the center of mass of the region lies on the road. All of the regions lying in the candidate are then grouped together and approximated with a conglomerate polygon. The area between the road model and the edge of the conglomerate polygon is used as a cost metric of the interpretation. The candidate road whose conglomerate polygon has the lowest cost is selected as the result. Figure 7 shows the cost metric of a good fit and a bad fit for the road.

3.3 Discussion and Future Work

In this system, the low-level segmentation uses mainly color constraints to segment the image, while the high-level interpretation uses only geometric constraints to localize the road. Therefore, the different levels of the system are using completely different constraints. Figure 8 shows an example of the unsupervised classification segmentation running on a road scene. The images to the right is the class image, where each class label is represented by a different intensity value. The left image shows the pixels colored by the mean values of their class labeling. The top pair of images is the initial scattered classification used to build the initial models. The middle pair of images shows the classification and mean class colors after two 'classify/sample' loops, and the bottom pair shows the results after five 'classify/sample' steps.

The advantage of the cost evaluation scheme of our interpretation is that new constraints can easily be added to the total cost. For example, we could add costs so that all of the road regions sould have a similar color, different that those

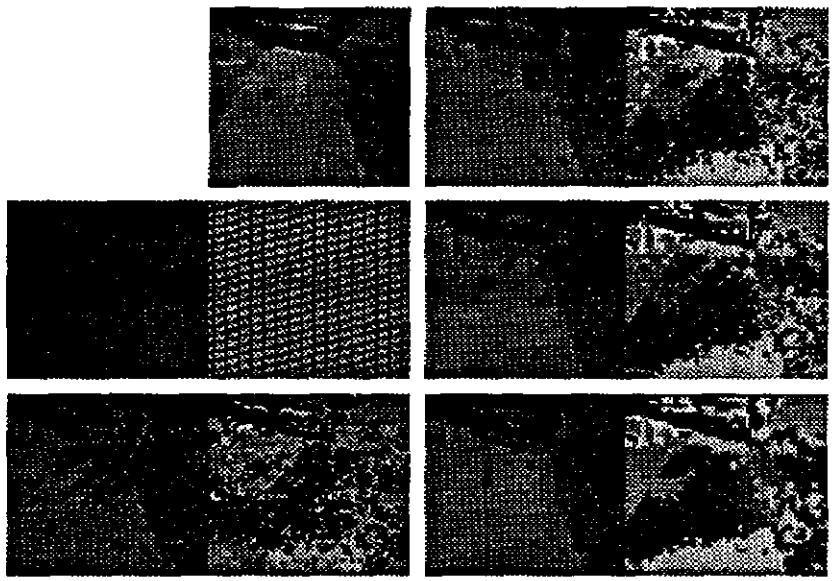


Figure 8: Example Clustering: Top left image is the original image. Each pair of class images corresponds to an iteration of the clustering algorithm. The right images have each class is assigned a different intensity, and the left images have each pixel is colored by the mean RGB class value. The bottom right pair is the final class images.

of the non-road regions. We could also add a cost insuring that the road region is similar in color to the road seen in the previous image. A cost can be added so that conglomerate polygons with straight edges are preferred over those with jagged edges. Although these additional cost have not been necessary on the images tested, they may become more important as we become more experienced with this interpretation system.

The system takes about 3-20 minutes to process one frame of the sequence. To speed up the processing, we have implemented the unsupervised clustering algorithm distributed on a multiple number of Sun workstations. Using this method, we have achieved a speed up proportional to the number of Suns used.

We will expand the unsupervised classification algorithm in several ways.

- First, if the system could decide the number of classes needed to characterize the data, rather than having a fixed number of classes specified, then the regions would be more representative of the data. As an initial attempt, we will split and merge regions after each reclassification step. Large regions will be split, and regions with very close mean values will be merged. This way, the system will decide how many regions it needs to adequately represent the data.
- We will expand the road interpretation to detect intersections. We will apply the road searching that we have currently implemented. Then we will enumerate all of the possible branches from the road, and search for intersection branches with the same cost evaluation method used for the main road. We may need to add a color constraint to the algorithm, since in our test site sometimes the shadows of the trees can form intersection shapes.
- We believe that the basic unsupervised segmentation algorithm described here can be used for many different vision applications. To show this we will use this system to do terrain typing for our cross-country navigation experiments.

4 Results and Conclusions

SCARF and UNSCARF have been prototyped and tested individually. Our current efforts include speed and algorithmic improvements to each system. We are also considering cooperation between SCARF and UNSCARF. One idea is to use

UNSCARF as a bootstrapping program and use SCARF as the general road-follower. If SCARF should realize that its results are not good, then control can be returned to UNSCARF.

The second possibility is to use lessons learned from one system to improve the other. We intend, for instance, to track the colors of the road regions detected by UNSCARF. Analyzing the changes in colors over time may provide cues which can improve the supervised classification in SCARF. This combination of different methods for scene segmentation will continue to expand the Navlab's capabilities for paved roads, dirt roads, and for terrain typing for cross-country navigation.

References

[1] E. Arnould, H. T. Kung, O. Menzilcioglu, and K. Sarocky. A systolic array computer. In *Proceedings of 1985 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 232–235, March 1985.

[2] D. DeMenthon. *Inverse Perspective of a Road from a Single Image*. Technical Report CAR-TR-210, University of Maryland, 1986.

[3] E. Dickmanns and A. Zapp. Autonomous high-speed road vehicle guidance by computer vision. In *Proc. 10th IFAC*, Munich, 1987.

[4] E. Dickmanns and A. Zapp. A curvature-based scheme for improving road vehicle guidance by computer vision. In *Proc. SPIE Conference 727 on Mobile Robots*, Cambridge, 1986.

[5] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, Inc., 1973.

[6] T. Kanade, C. Thorpe, and W. Whittaker. Autonomous land vehicle at CMU. In *Proc. ACM Computer Conference*, Cincinnati, February 1986.

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

[8] M. Turk, D. Morgenthaler, K. Gremban, and M. Marra. VITS–a vision system for autonomous land vehicle navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 1988.

[9] R. Wallace, K. Matsuzaki, Y. Goto, J. Webb, J. Crisman,.and T. Kanade. Progress on robot road following. In *IEEE Conf. on Robotics and Automation*, San Francisco, 1986.

[10] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, and T. Kanade. First results in robot road following. In *Proceedings of IJCAI 85*, August 1985.

Chapter III
Explicit Models for Robot Road Following

Karl Kluge and Chuck Thorpe

Explicit Models for Robot Road Following

Karl Kluge and Chuck Thorpe

Abstract

Robots need strong explicit models of their environment in order to reliably perceive and navigate. An *explicit* model is information directly available to the program itself, used for reasoning about what to look for, how to look for it, and how to interpret what it has seen. We discuss the need for explicit models in the context of road following, showing how road followers built by our own and other groups have suffered by not having explicit models. Our new road tracking system, FERMI, is being built to study explicit models and their use. FERMI includes explicit geometric models and multiple trackers, and will use explicit models to select features to track and methods to track them.

Implicit Road Models Considered Harmful

We claim that vision systems need to have strong explicit models in order to do reliable recognition. This is especially true in difficult situations, such as perception for an outdoor robot operating in an environment with no control over objects or illumination. Our particular domain is color vision for road following.

During the last four years there has been intense research on robot vision for following roads. Several different systems have been developed, many of them under sponsorship of DARPA as part of the Autonomous Land Vehicle program. Although there have been many solid contributions to road following, there is still no reliable general purpose road tracking vision system. Most existing road trackers work well for only a particular road, or only under benign illumination. They have impoverished models that do not allow them to reason about failures in their low level feature trackers. Weak models and weak or nonexistent high levels make them brittle in the presence of disturbances such as disappearing features or illumination changes.

Each system has a model of the road, including expectations about road shape and appearance, and the changes in shape, location, and appearance from one location to the next. The models are used to guide recognition, predicting how and where a road should appear and what methods should be used to find it. The models are also used for vehicle guidance, providing continuity while digitizing and processing each image.

A complete model of the road encompasses assumptions made by the programmer, and procedural knowledge for road recognition, as well as the data structure used by the program for road description. The assumptions made in road modeling fall into three loose categories: *subconscious* models which are implicit to the programmer; *implicit* models, representing decisions made by the programmer but not available to the program; and *explicit* models which the program itself can access and modify.

Each kind of assumption is appropriate in some circumstances. However, the more information is

made explicitly available to the program, the wider the range of circumstances the program can handle autonomously. This is especially true for models of highly structured roads, such as well-marked streets and highways.

Typical subconscious assumptions, for instance, are that the road doesn't move, doesn't change color at any one location, is continuously connected, doesn't bend so sharply that it goes entirely out of the camera field of view, doesn't fold violently in 3-D. Many of these assumptions derive from the functionality of a road: if a narrow road makes a sudden right-angle bend, it is impossible for a vehicle to follow, and therefore is no longer a "road". Assumptions at that level are safe, and are applicable to a wide variety of roads. Other subconscious assumptions are much more insidious. One road following program begins with the (correct) implicit assumption that road edges are locally parallel, then (incorrectly) makes the subconscious assumption that feature-extraction routines will find the correct edges. This leads to drastic errors in inferred geometry when the subconscious assumptions are violated. Such an assumption may be not only wrong, but also hard to pinpoint and eliminate, since it was never consciously made or documented.

Implicit models show up in papers and in documentation, but not in code or data structures in any form that the program itself can access or modify. Typical implicit models are that the road is locally nearly straight, that the road is always brighter than its surroundings, or that the dominant edges in the scene are the road borders. Such implicit assumptions are often used by the programmer to select a single algorithm for recognizing that particular road type, or for calculating road geometry under that assumption. Well-constructed programs that rely on those road models are understood by their authors to only work in those cases where the underlying assumptions hold. In particular, for unstructured roads that do not have lane or edge markings and that do not follow rules of curvature or shape, the road model is very limited. With such a limited road model, it may not be possible or practical for the program itself to use an explicit model. If there is only one feature that can be tracked reliably, and only one algorithm for feature tracking, then there is no need for explicit program reasoning.

Explicit models are most useful in the opposite case, in which the road follows strong rules of shape and appearance, and there are many possible features and a variety of recognition algorithms. Then the program itself can select the correct features and algorithms, and can watch for changes in the road and change its strategy accordingly. Moreover, an explicit model that includes road semantics can help tie together separate phenomena. By "semantics" we mean labels such as "intersection" or "right turn lane", and the associated rules and descriptions that prescribe road appearance and shape in those situations. For instance, a program with only implicit models may notice that a feature it had been tracking has now disappeared. Only with an explicit model will it be possible for the program to understand that the feature was a double yellow line, that its disappearance might mean an approaching intersection, and that it is now past time to start looking for crossing traffic.

Road following programs to date use only subconscious and implicit models. This is due partly to the kinds of roads being tracked, which often do not have enough structure to make strong models necessary or possible. In other cases, however, the road has strong structure, but the designer has made all the decisions implicitly. Many road following systems have only a single road-tracking algorithm, and have a fixed road model. The designer uses an implicit model to pick the "best" method for following the road. The resulting system appears relatively simple and efficient, since it has only one algorithm to code and needs no higher-level reasoning.

Simple appearances are deceptive. Such implicit road models detract from system performance, and contribute to brittleness, and to difficulty in debugging and making enhancements. Furthermore, systems based on such a preprogrammed model of the world tend *not* to be as simple as they would at first appear. Since the world is rarely as static as an implicit, preprogrammed, model, those programs need many special cases, exceptions, recovery mechanisms, and other complications.

We contend that it is possible and advantageous to make the road model explicit, and to not only model appearance and shape information but also to include semantics in the model. Moreover, using such a model will make it easier to program and debug a road follower, and will lead to efficient programs. The bulk of the processing can be done by simple operators that needn't be concerned with special cases, while the costlier recovery procedures and switching between operators will occur infrequently.

The first half of this paper reviews other road followers, and outlines the road models and hidden assumptions used by each program. In the second half, we introduce FERMI, the Following Explicit Road Models Intelligently, and describe its construction and performance.

Systems, Models, and Assumptions

In this section, we describe several systems, describe their road models, and critique the implicit models in each.

SCARF: Color Classification

Implicit model: road colors mostly constant from one image to the next, known road shape (either known width, locally straight and parallel for Hough interpretation, or arbitrary but known for ground search)

Subconscious model: constant lighting and cameras so that constant road colors map to constant road images

SCARF, for Supervised Classification Applied to Road Following, has been developed over the last three years at Carnegie Mellon [5]. SCARF keeps color models for typical road and nonroad features, typically 8 road and 8 nonroad features. Each color model represents the means, and covariances of the color values for that feature, and the number of pixels in that class. An image is classified by comparing each pixel to each class, and determining the most likely classification for that pixel as well as the probability of that classification. The most likely road is found by convolving a known road shape with the classification output, looking for the road position that has the greatest sum of road probabilities inside the road shape and nonroad probabilities outside the road shape. In practice, this can be done efficiently using an adaptation of the Hough transform that votes for areas instead of lines.

Once the most likely road has been found, SCARF builds new color models by supervised classification. The area inside the road is used to build new road models, and the area outside the road for the new nonroad classes.

SCARF was designed for use on a narrow, twisting, tree-lined bicycle path near the CMU campus. With constant illumination, it works well. Various color classes typically represent shady road, sunny road, leaves, wet patches, dirt, and so forth. As the vehicle moves onto a new type of road, classes adjust to represent the new appearances, as long as there is enough overlap between scenes that the

majority of the road has been seen and modeled in the previous image.

The biggest weakness of SCARF is in changing illumination. If the sun goes behind a cloud between images, the appearance of road and nonroad features can change, rendering color models incorrect. A second weakness is the reliance on known road shape. If the road curves sharply, or if it changes width, the assumed shape model (locally nearly straight, known width) is invalid. Finally, SCARF suffers from the lack of features in its environment. It is difficult to build explicit models, since its environment has few features: the bicycle path has no lines, stripes, guard rails, or shoulders.

UNSCARF: Unsupervised Classification

Implicit model: road is a collection of homogeneous regions that together form a "road shape" (currently known width, straight edges)

Subconscious model: road edges are clean

UNSCARF, for UNSupervised Classification Applied to Road Following, was designed by Crisman at CMU to overcome the problems of SCARF with rapid illumination change [1]. UNSCARF does not keep color models from image to image, and does not classify pixels as road or nonroad. Instead, for each image, it starts from scratch and finds the classes that best describe the image. It uses the classes to divide the image into regions of similar appearance, then searches for the combination of regions that forms the best road. "Best", in this case, currently means closest match to known road shape. Other heuristics being considered include shape constraints, such as edge smoothness and straightness, and color constraints.

UNSCARF uses a weaker model than SCARF. By eliminating the subconscious assumption that lighting is constant, UNSCARF successfully finds roads in cases where that assumption is violated. But UNSCARF also gives up a great deal of useful information for the many occasions when illumination does not change between successive images. A better solution would start by detecting illumination changes explicitly, and using colors from previous images if illumination is constant. This is one of the themes of our current work. The best solution would be to improve the illumination model from a binary decision (changed / constant) to a quantitative analysis of how colors change with changing illumination. A complete analysis requires understanding the interactions of direct lighting; diffuse illumination from sky, clouds, and leaves; object colors and highlights; camera sensitivity; and digitizer effects. While work has begun in those directions [4], it is far from being applicable to unconstrained outdoor scenes.

Maryland

Implicit model: small-scale road edges dominate the scene, edges are parallel, vehicle motion is accurate

Subconscious model: edge-finding is accurate, edges are clean and linked, limited curvature

Davis, LeMoigne, Kushner, and Waxman, at the University of Maryland, have a long history of research in perception for road following. Their strongest system, and the only one to actually drive an autonomous vehicle, is based on finding edges and grouping them into lines with Hough transforms [8]. During road tracking, an initial window is placed at the bottom of the image on the predicted road location. The search for the road edge in this window has two degrees of freedom, for location and orientation. Once this edge is located, other windows are placed above the initial window. In each succeeding window the road edge position is constrained by the lower window, so the Hough search need only look

for orientation. This technique can work adequately for scenes in which the dominant edges are road borders. Similar techniques at CMU were defeated by strong, straight, shadow edges from trees and buildings, and by scenes in which road edges were obscured by leaves or dirt.

Besides road tracking, the Maryland research also considered 3-D shape reconstruction. The higher-level attempts at 3-D interpretation of road scenes were extremely sensitive to noise. DeMenthon [2] shows how Waxman's model can lead to perceived roads folding back over themselves, and proposes a new geometry that ameliorates some of those problems.

VITS

Implicit model: consistent colors within one image (road has at most 2 classes, for sunny and shaded), known vehicle motion and road model to seed process

Subconscious model: The color combination chosen is assumed to be always adequate despite changing illumination & dirt on road; this implies road appearance is constant from day to day

The Martin Marietta VITS system [6] has achieved some impressive goals. It has followed roads at speeds up to 20 kph, and detected and avoided obstacles on the road. Their system projects the 3-D color of each pixel onto a single dimension or, in later systems, onto a 2-D plane. Pixels are classified into road or nonroad based on a linear discriminant function. Once each pixel is classified, blob coloring gives the location of the road. The most interesting part of the Martin Marietta research is in selecting the road / non-road threshold. In each new scene, vehicle motion is combined with the previous road model to calculate the portion of the image guaranteed to contain road. This road area, called a power window, is sampled to determine the typical road color for this image. The Martin system is a tightly-engineered combination of perception, control, modeling, and highly tuned hardware. In many ways, their system is similar to some of the CMU road-following, but driven by speed constraints rather than generality of experiments. Where CMU's SCARF uses full color (or even 6 channels of color from two cameras) to track a variety of road appearances, they have selected the best combination of colors for their particular road. SCARF keeps many different possible appearances for both road and offroad, while VITS has at most two, again sacrificing general capability for speed.

Dickmanns and Grafe

Implicit models: gray-level edges of roads dominate the scene, road follows clothoid shape, physical constraints and fast processing limit feature motion, known relationship between road features to be tracked.

Subconscious models: all interesting features are oriented edges, no simultaneous distractions

Dickmanns and Grafe have demonstrated road following with a Mercedes van equipped with special-purpose computing [3]. They have achieved impressive performance, tracking a new section of the autobahn at speeds up to 100 kph. The heart of their system is an elegant control formulation, in which road geometry, vehicle turning radius and speed, and the location of visually tracked features are all fed into a single filtered state model. When running at high speeds, their system takes advantage of the geometry of the German autobahn. The road consists of straight lines, constant radius curves, and clothoids smoothly connecting curves and straights. German roads have known lane widths and well-defined markings.

The major weakness of this system is its extremely simple perception model. They use a monochrome

camera and do simple edge detection. Their rapid processing and structured road model help them to detect and discard anomalous edges, but it nevertheless appears that their trackers could get distracted by shadows, puddles, road imperfections, or changing illumination.

FERMI

All of the above road followers have implicit and subconscious models of the road. But none of them has more than one means of tracking the road, or does any higher-level reasoning about the road, or has any explicit road model available to the program. Yet it is important to build and to use explicit road models. Highways, freeways, rural roads, even suburban streets have strong constraints. Modeling these explicitly makes reasoning easier and more reliable. When a line tracker fails, for instance, an explicit model of road and shoulder colors adjacent to the line will help in deciding whether the line disappeared, became occluded, turned at an intersection, or entered a shadow. This kind of geometric and photometric reasoning is vital for building reliable and general road trackers. We are now building the FERMI road tracking system to study explicit modeling, and to study the use of those models in building reliable vision.

Explicit Models

Our goal in constructing FERMI is to follow structured roads as reliably as possible. Our central principle is to make explicit as much as possible: road features, geometry, and other effects. We are first of all building individual knowledge sources that know how to model and track specific features:

- road edge markings (white stripes)
- road center lines (yellow stripes)
- guard rails
- shoulders
- type and color of road surface

We also have an explicit geometric model of the road. This model consists of a series of *generalized stripes*. A generalized stripe is the 2-D analog of a generaized cylinder. It consists of a spine curve (currently restricted to arcs of constant curvature), and the description of a cross-section which is translated along the spine. The model of the road in Figure 1, for instance, looks something like

- Spine: Curvature = 0.0.
- Feature 1: starts —304 cm ± 15 cm, height 0 cm, type shoulder, description asphalt.
- Feature 2: starts —273 cm ± 0 cm, height cm, type line, description white.
- Feature 3: starts —262 cm ± 0 cm, height 0 cm, type road, description asphalt.
- Feature 4: starts —24 cm ± 0 cm, height 0 cm, type line, description yellow.
- Feature 5: starts —7 cm ± 0 cm, height 0 cm, type road, description asphalt.
- Feature 6: starts 7 cm ± 0 cm, height 0 cm, type line, description yellow.
- Feature 7: starts 24 cm ± 0 cm, height 0 cm, type road, description asphalt.
- Feature 8: starts 262 cm ± 0 cm, height 0 cm, type line, description white.
- Feature 9: starts 273 cm ± 0 cm, height 0 cm, type shoulder, description asphalt.
- Feature 10: starts 304 cm ± 15 cm, height 0 cm, type offroad, description grass.

The program will explicitly note transient road phenomena such as

- shadows
- local changes in road surface, e.g. patches
- global illumination changes, such as the sun going behind a cloud
- camera changes (auto-iris, auto-gain)

- 3-D effects such as going up and down hills

Explicitly modeling all these different features will be the basis for efficiency and reliability. The system will be efficient because the geometric constraints can specify subwindows of the image for each feature tracker, and tracker history from frame to frame can predict appearance and shape. Another reason for efficiency is that many simple trackers can be easily implemented on parallel hardware. Reliability will come first because of the strong geometric constraints among trackers, and the ability to detect and ignore anomalous outputs. The ability to use a strong geometric model of the road to focus on a small area of the image to look for a feature reduces the chances of being misled by extraneous image features. More importantly, the system will be reliable because one tracker, on discovering a shadow edge or road curvature change, can pass that information to other trackers and keep them from being caught by the same phenomenon.

Trackers

Many of the individual feature trackers have already been developed. We have done some preliminary experiments using, for example, the oriented edge operator used to drive the Terregator in 1986 and a simplified version of the color classifier developed in 1987. Customizing these feature trackers to follow lines, stripes, and edges will make them faster and more robust than the general-purpose trackers needed for our park scenes.

We currently have four trackers implemented:

- Oriented edge profile: Intensity profiles are extracted from a training window oriented perpendicular to the direction of the feature. These oriented templates are matched by correlation with intensity profiles from later images. The implicit model is that the color intensity profiles of an edge are roughly uniform along the length of the edge.
- Extended linear feature tracker: Intended for use tracking such features as white and yellow road stripes. An unsupervised clustering algorithm is used on the RGB pixel values in a training window to split the image window into two clusters: the line and the background. The mean RGB values for the two clusters are used in later images to classify the pixels in a search window. A line is fit to the pixels which are classified as being part of the linear feature, giving an estimate of the location and orientation of the linear feature in the image. Implicit model: that the dominant color phenomenon in the training and search windows arises from the contrast between the line and the background, and remains approximately constant from image to image.
- Color boundary tracker: Used on ragged edges such as a grass/road boundary. Performs the same sort of clustering as the previous tracker, splitting the pixels in the training window into two classes. The pixels which have neighbors that have a different label are marked, and a line fit to these boundary points to estimate the edge position and orientation. Implicit model: assumes that the dominant color phenomenon in the training and search windows is the contrast between the colors of the two features whose boundary is being tracked.
- Matched filter tracker: A small training window is selected. In later search windows the training window is correlated with the search window. The maximum correlation value in each row of the search window is selected as an edge point, and a line is fit to the edge points. Implicit model: the appearance of the feature is constant enough for correlation to be used.

Our current method of selecting a tracker looks at the size in the image of each feature. If the feature is narrow (i.e. a line or stripe), it selects a linear feature tracker. If the feature is wide (e.g. a lane of the road), it chooses to track the edge of that feature, and selects an edge operator such as the oriented edge tracker. Figure 1 shows the road described earlier, with boundary and oriented edge trackers

tracking the white lines on the left and right side of the road and the left edge of the right lane.

Tracker fusion

It is necessary to merge the estimates of feature locations and orientations returned by the trackers placed on various features at various points in the image into a single estimate of where the vehicle is relative to the spine of the generalized stripe that is currently being transversed. The method of fusion needs to take into account the possibility of trackers failing or returning erroneous estimates.

The current method of tracker fusion is a Hough technique. Let us suppose that the spine of the current stripe is a straight line (the technique extends in a straightforward way to arcs of known constant curvature). Since the road is likely to be almost straight ahead of the vehicle, let's represent it as a line of the form $y = m * x + b$, where the x-axis points straight ahead of the vehicle and the y-axis points to the left. Let's suppose we have a feature tracker tracking a white stripe whose center is offset from the road spine by $offset_{stripe}$, and that the tracker has returned (x_i, y_i) as it's estimate of the location of the center of the stripe. For a given m value, the y-intercept of the white stripe center line is given by $b_{stripe} = y_i - m * x_i$, and the y-intercept of the spine by $b_{spine} = b_{stripe} + offset_{stripe} / \cos(\text{atan}(m))$. Figure 2 shows the relationship between the feature position and the spine of the associated generalized stripe.

Each tracker votes for all possible spines that are consistent with its position estimate for its feature. The largest peak in the accumulator array is taken as the position of the road spine. Trackers whose position estimates are not consistent with that spine estimate are anomalies which need to be explained. Currently the program does not try to explain tracker failures.

Interpretations

At a higher level, we can use the semantics of the model to interpret tracker failure. Tracker failure may be noticed by the tracker itself, or the tracker may give a response that is inconsistent with the output of other trackers. In either case, the monitoring system will notice the failure and will try to explain the underlying cause, and use that explanation to update its model. Examples of such reasoning include:

- double yellow -> single dashed yellow: no change
- double yellow -> none: intersection appearing, predict all other lines disappear, start intersection-traversal behavior
- white line disappears -> <many possibilities>
  - road / shoulder: nothing
  - all road with no border -> possible side road turning off
  - dark scene: check for shadow
  - uninterpretable: check for occlusion

Current Status

The program which currently exists contains

- Code for dealing with an explicit road model described as generalized stripes with spines which are arcs with constant curvature.
- The four trackers described above.
- A simple tracker selection mechanism to decide which tracker should track which feature.
- Prediction code that positions each tracker correctly based on the perceived position of the road in the previous image and the vehicle's motion.
- Tracker fusion using a Hough technique to determine the vehicle position relative to the spine of the current road stripe.
- A simple facility for producing synthetic road images in order to test the effects of errors in

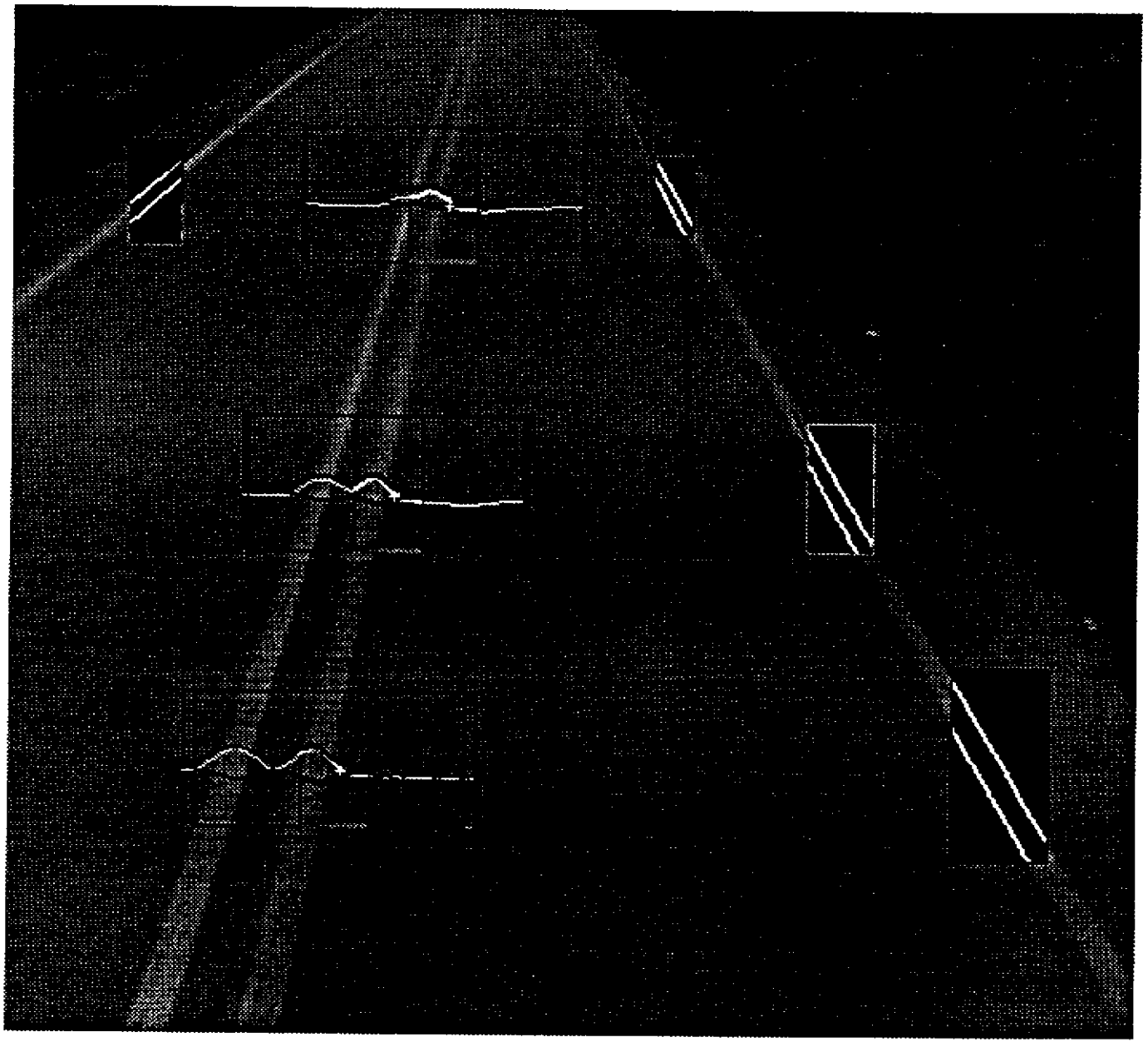


Figure 1: Road with oriented edge and boundary trackers

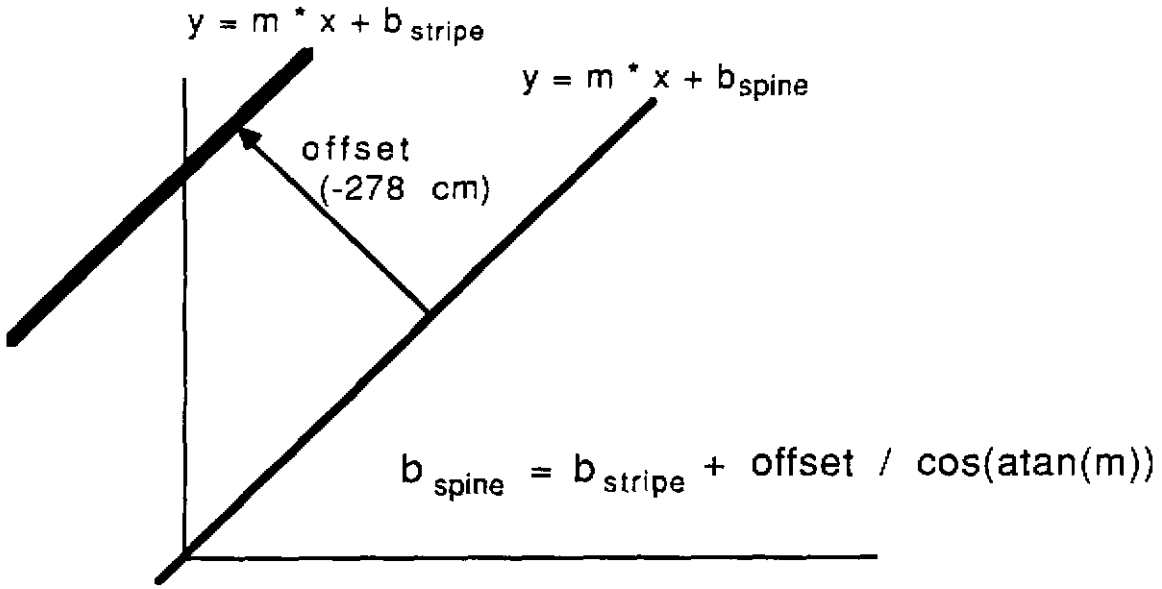


Figure 2: Stripe feature and spine locations in the Hough space

calibration and the road model separate from the image processing problems.

We have run this program on the CMU Navlab on a section of path in the park near campus. Each digitize-track-fusion-steer cycle took about 20 seconds, running on a single Sun 3/180. The main goals of our initial work have been to develop a family of trackers that work well in many situations, and to check auxiliary functions such as path planning and camera calibration.

We are just beginning the second phase of our programming, which will exploit our explicit feature models. The first item on our agenda is fully automatic selection of features to track and tracker type and placement. The user currently decides which features of the road model should be tracked. Possible heuristics for automatic feature selection include both a priori reasoning (what is the expected contrast between these two adjacent features?) and run time reasoning (what is the actual contrast in the initialization image?).

A second step is detection of trackers which returned erroneous values, and explanation of their failure. A circular arc has only two degrees of freedom. We typically use eight or more trackers, some of which return x,y position and some of which also include perceived feature orientation. This gives us a greatly overconstrained system, and will make it possible to detect malfunctioning trackers. Trackers can also in some instances give internal evidence of difficulties, for instance correlation values or residuals of line fits. Once the program determines that a tracker is failing, the next step is determining why it failed, and using that diagnosis to prevent other trackers from falling into the same trap.

We also need to model the semantics of road markings. Cues such as a double yellow line turning into a dashed yellow line can predict the road becoming straight and flat.

We also will build and test additional simple feature trackers as we gain experience with failure modes. No one tracker is likely to be reliable in all circumstances, so the greater variety of trackers available the

greater the chance of having one that works for a particular condition. Perhaps more important than the proliferation of operators is implementing them efficiently on the Warp, our high-speed experimental parallel processor [7]. Most of our processing time is consumed in local image processing operations which are relatively easy to implement on parallel hardware.

Acknowledgements

Our thanks to Jill Crisman, who developed the oriented edge tracker, and Thad Druffel, who developed much of the code for the other trackers.

This research is sponsored by the Strategic Computing Initiative of DARPA, DoD, through ARPA Order 5351, and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-0003, titled "Road Following." Portions of this research were also partially sponsored by the National Science Foundation contract DCR-8604199 and by the Digital Equipment Corporation External Research Program.

References

[1] J. Crisman and C. Thorpe.
Color Vision for Road Following.
In *Proc. SPIE Conference 1007 on Mobile Robots.* Cambridge, 1988.

[2] D. DeMenthon.
Inverse Perspective of a Road from a Single Image.
Technical Report CAR-TR-210, University of Maryland, 1986.

[3] E. Dickmanns and A. Zapp.
A curvature-based scheme for improving road vehicle guidance by computer vision.
In *Proc. SPIE Conference 727 on Mobile Robots.* Cambridge, 1986.

[4] Klinker, Gudrun J.
A Physical Approach to Color Image Understanding.
PhD thesis, Carnegie Mellon University, May, 1988.

[5] C. Thorpe, M. Hebert, T. Kanade and S. Shafer.
Vision and navigation for the Carnegie-Mellon Navlab.
PAMI 10(3), 1988.

[6] M. Turk, D. Morgenthaler, K. Gremban and M. Marra.
VITS--A Vision System for Autonomous Land Vehicle Navigation.
IEEE Transactions on Pattern Analysis and Machine Intelligence , May, 1988.

[7] E. Arnould, H. T. Kung, O. Menzilcioglu and K. Sarocky.
A Systolic Array Computer.
In *Proceedings of 1985 IEEE International Conference on Acoustics, Speech and Signal Processing*; pages 232-235. March, 1985.

[8] A. Waxman, J. LeMoigne, L. Davis, B. Srinivasan, T. Kushner, E. Liang and T. Siddalingaiah.
A visual navigation system for autonomous land vehicles.
Journal of Robotics and Automation, Vol. 3 , 1987.

Chapter IV
Building and navigating maps of road scenes using an active sensor

Martial Hebert

Building and navigating maps of road scenes using an active sensor [1]

Martial Hebert
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents algorithms for building maps of road scenes using an active range and reflectance sensor and for using the maps to traverse a portion of the world already explored. Using an active sensor has some attractive advantages: It is independent of the illumination conditions, it does not require complex calibration in order to transform observed features to the vehicle's reference frame, and it provides 3-D terrain models as well as road models. Using the map built from sensor data facilitates navigation in two respects: The vehicle may navigate faster since less perception processing is necessary, and the vehicle may follow a more accurate path since the navigation system does not rely entirely upon inaccurate visual data. We present a complete system that includes road following, map building, and map-based navigation using the ERIM laser range finder. We report on experimentations of the system both on the CMU NAVLAB and the Martin Marietta ALV.

[1]This research was sponsored by the Defense Advanced Research Projects Agency, DoD, through ARPA Order 5351, monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-003

1 Introduction

Autonomous road following using visual information is an important application of mobile robots. In addition to navigating on roads, the visual information can be used to build maps of the observed environment. An area of research that has not been explored is to close the loop by using the map built from previous observations to guide the navigation on a portion of the world already explored. Such a capability of map based navigation would enable us to improve the performances of the vehicle in three directions:

- Faster navigation: Perception is typically the bottleneck in autonomous mobile systems because images have to be processed as often as possible to compensate for the lack of knowledge about the world. If apriori knowledge of the environment is available from previous observations, perception is needed only to periodically check that the vehicle stays on the path prescribed by the map. The perception bottleneck is therefore reduced, thus leading to faster navigation.
- More reliable navigation: Autonomous navigation is unreliable because of the uncertainty associated with any sensor data and processing. Relying more on a map means relying less on sensor data acquired during the execution of a navigation plan. Map based navigation should therefore provide more accurate navigation.
- Simpler perception: A map can provide the expected appearance of the environment at any location. That includes the expected location of objects, and the expected position and appearance of the road. This additional knowledge allows for simpler perception processing.

Athough map based navigation algorithms could be used with a man made map (*e.g.* from surveying), using a map built from sensor information does not make any assumptions on the amount of knowledge available to the system, thus leading to a fully autonomous system. This is also important since it is difficult to obtain the resolution of a map built from sensor data by using surveying alone.

Most of the existing road following systems are based on intensity or color image processing [14,18,15]. In this paper, we investigate the use of active sensing, namely laser range finding, for both road following and map building. Using such a sensing modality has some attractive features such as its stability with respect to illumination conditions and the direct conversion to world coordinates without calibration. Our goal is therefore to build a complete system from road following to map building using active sensing, whereas previous research on active sensing for autonomous vehicles focused on 3-D map building or obstacle detection [2,3,6,4].

The images used in the experiments reported in this paper are range and reflectance images from a laser range finder, the ERIM scanner [17]. The images are 64 rows by 256 columns 8-bit images. The maximum range is 64 feet corresponding to a pixel value of 255. The vertical (resp. horizontal) field of view is 30° (resp. 80°). Figure 1 shows a range image (top image) and the corresponding reflectance image of a simple scene consisting of a road and two trees.

Even though the road following programs were demonstrated on the Martin Marietta vehicle (the ALV), all results presented in this paper were obtained using the Carnegie-Mellon vehicle (the NAVLAB) [10].

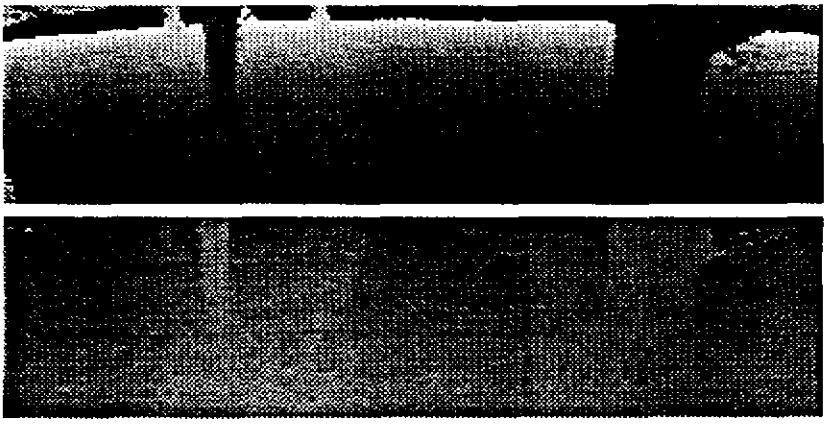


Figure 1: Range and reflectance images

2 Following roads using active reflectance images

Early work on road following from active sensing focused on the use of range data to find the edges of the road [11,1]. The drawback of this approach is that it assumes that the road is limited by edges that correspond to discontinuities of the terrain surface. This assumption limits severely applicability of the algorithms. An alternative approach is to use the active reflectance images for road following. Active reflectance images have two characteristics that make them attractive for road-following applications: First, they are insensitive to outside illumination, that is no shadows are cast by objects in reflectance images and the influence of the level of ambiant light on the image is minimal (in fact, any program using reflectance images would work as well under night conditions). Second, each pixel in the reflectance image is also a range pixel whose position in space can be derived from the geometry of the scanner. This allows us to compute the position of the edges of the road found in a reflectance image in the vehicle's 3-D world without any of the calibration procedures that are typical of the video-based road following algorithms [5].

A significant drawback of using reflectance images is that the value of the reflectance at each pixel depends on the value of the range at that pixel [7,17,19]. In other words, the reflectance values decrease as the square of the range to the measured surface. This effect can be corrected to some extend by calibrating the sensor with respect to a surface of constant reflectance, that is to fit a function $refl_{corrected} = f(refl_{observed}, range_{observed})$ over a portion of a training image of constant reflectance [7]. The function f is then used to build a correction lookup table. Such a calibration reduces the effects of the reflectance but does not completely remove them because of approximations in the sensor model and because the surface portion used for the training does not exactly have constant reflectance.

Edge detection would be the natural way of finding road edges in grey level images. The nature of the reflectance data, however, suggests the use of a region-based technique for two reasons: First, the dynamic range of the image is low, many spurious edges that are of similar strength as the road edges will be found. Second, the intensity of the road in reflectance images is very stable because it is insensitive to shadows and changes in illumination. This is to be compared with video images in which the appearance of the road region varies significantly, thus requiring the use of multiple classes of road and non-road

regions [14]. Instead of extracting the road edges directly, a road region extractor identifies the pixels that are part of the road based on the road location and appearance predicted from a previous image.

The principle of the road region extractor is to keep the mean m_i and variance σ_i of the reflectance values inside the road region for each group i of four scanlines in the image. The statistics are computed on groups of scanlines instead of the entire image in order to account for the intensity attenuation at long range and for the presence of small markings on the road that would have an effect on a few scanlines only instead of propagating to the entire road region. The statistics are computed on the first image by selecting the road region interactively. The road region is extracted from the reflectance image by thresholding the pixels in swath i that are between $m_i + \sigma_i$ and $m_i - 2 * \sigma_i$, where (m_i, σ_i) are the values computed on the previous image. The resulting binary image is then processed to remove small isolated regions. The road region is extracted from the set of remaining regions by using three criteria: the shape of the boundary that is the value of the elongation, the size of the region knowing the average with of the road, and the position of the region in the image as predicted from the previous image. Once the road region is extracted, the values m_i and σ_i are computed for each swath in order to process the next image. This algorithm is similar to [14,15], except that is uses only one road class, that is only one set of statistics, and that it computes and predicts the road appearance over small swaths instead of the entire image.

The final output of the road finder program is the direction of the center line of the road. The line is computed by fitting two parallel lines to the left and right edges of the road polygon. If the two lines are parametrized by a direction $\vec{v}$, which is the direction of the road common to both edges, and the signed distances of the lines to the origin d_l and d_r (Figure 2), the best fit is computed by minimizing:

$$\sum_{leftedge} (p_l.\vec{v} + d_l)^2 + \sum_{rightedge} (p_r.\vec{v} + d_r)^2 \qquad (1)$$

The center line of the road is the middle line of $(\vec{v}, d_l)$ and $(\vec{v}, d_r)$, the width of the road is $w = |d_l - d_r|$. Figure 3 shows the result of the road following program on a typical sequence of reflectance images. The left part of Figure 3 shows the sequence of reflectance images, the right part shows the road edges and the center line of the road projected on the ground plane.

In order to drive the vehicle, two points on the center line are sent to a local path planner. The path planner generates a sequence of circular arcs using a "pure pursuit" algorithm derived from [16]. The road following program drove successfully two vehicles, the CMU NAVLAB [9] and the Martin Marietta ALV, over several hundred meters at a speed of 40cm/s. In both cases, the road following is implemented on a Sun3 workstation. The average computation time is 3 seconds per reflectance image which allows for enough overlap between consecutive images.

3 Building maps from range and reflectance images

We have so far addressed the problem of building a representation of the environment from individual range and reflectance images. In the case of a mobile robot, however, we have to deal with a stream of images taken along the vehicle's path. Merging those individual representations into a coherent map of the world is important for three reasons: First of all, merging representations from successive viewpoints

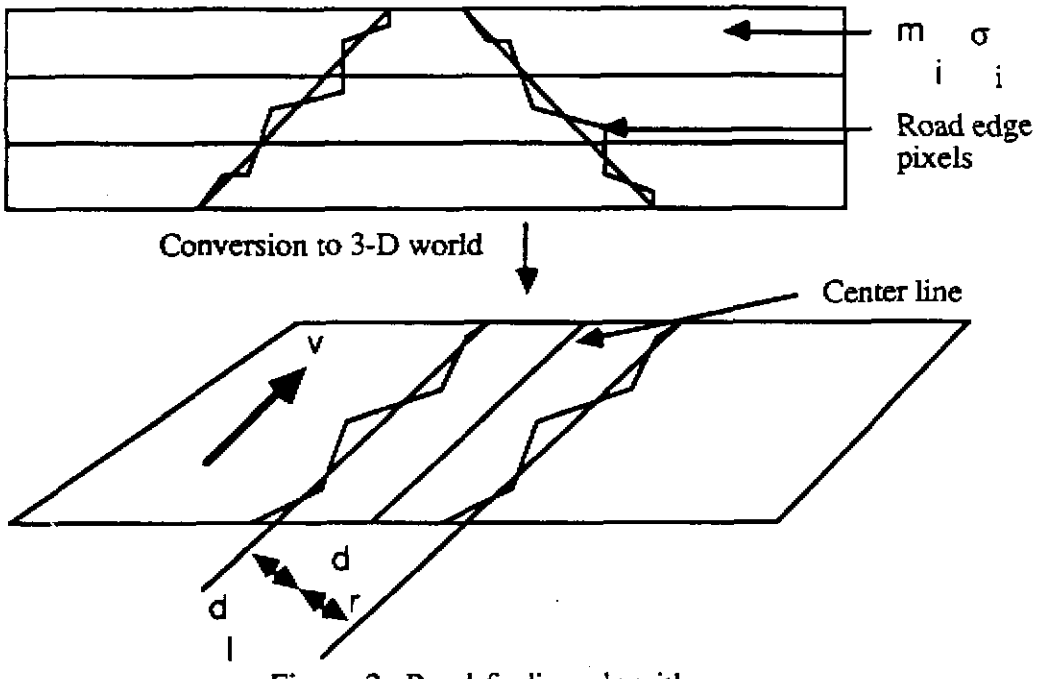


Figure 2: Road finding algorithm

produces 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. 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. Another reason why merging maps increases the resolution of the resulting representation is 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 is to compute the position with respect to features observed in the world instead of a fixed coordinate system [12,8]. That requires the identification and fusion of common features between successive observations in order to estimate the displacement of the vehicle. The third motivation is that having a map would enable the vehicle to navigate more efficiently a portion of the world that has been already explored. We will focus on this aspect in section 4.

The main problem in building a map from a sequence of consecutive images is to compute the relative positions of features observed from different vantage points in order to merge them in a consistent map expressed in a single coordinate system. Two types of information may be used to compute the relative positions: The matching of geometric features from image to image, and the best estimate of the current position of the vehicle as given by the dead reckoning. The position estimate from the motion of the vehicle cannot be used alone unless a sophisticated navigation system is used as in [3] since positional errors

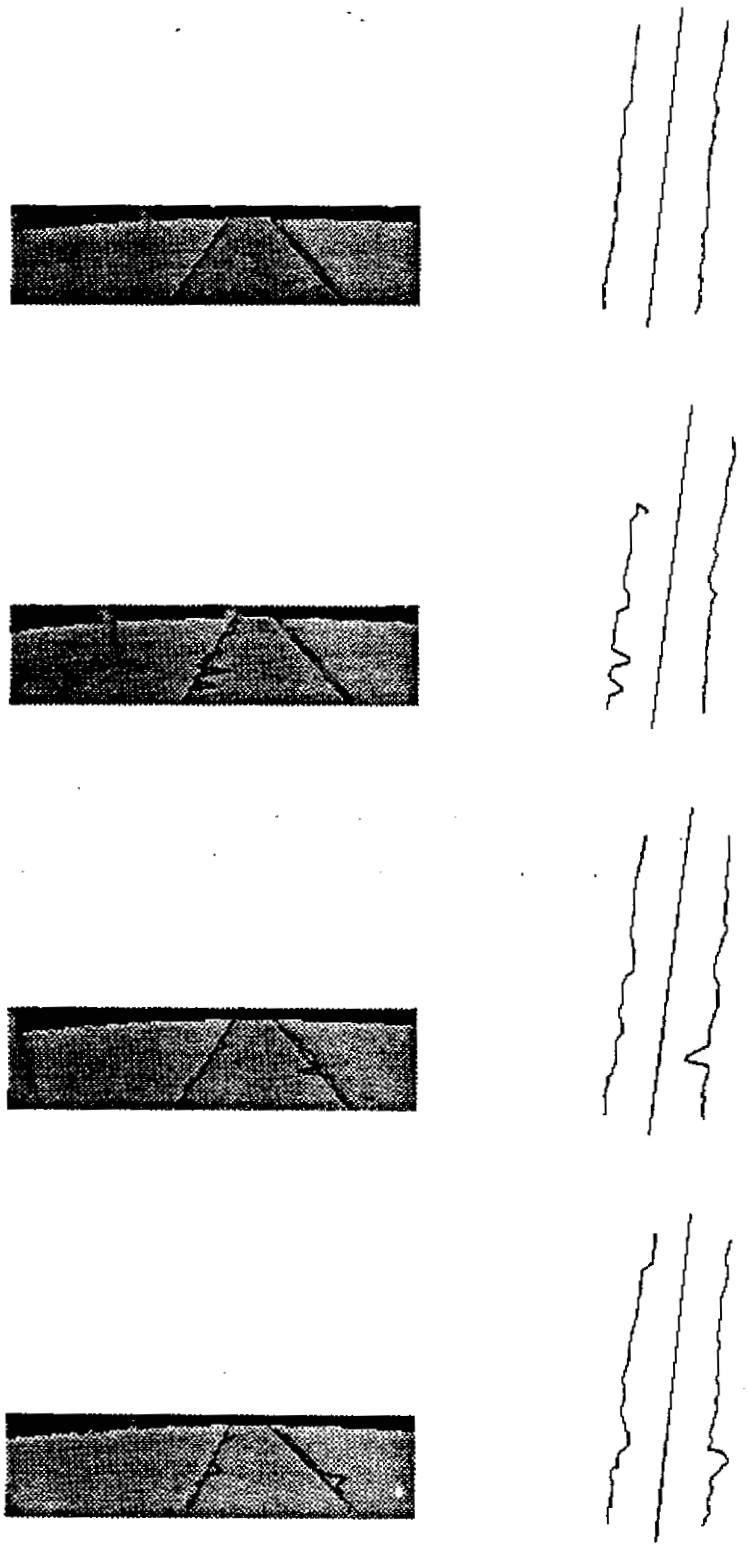


Figure 3: Road following on a sequence of reflectance images

do accumulate in time, thus leading to unacceptable errors in the position estimate. The final position estimates should be a combination in which the estimate from the dead reckoning is used to predict matches between features, and a set of consistent matches is used to estimate the resulting displacement between images. In general, if F_i^1 and F_j^2 are two sets of features extracted from two images, 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. 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 are not known accurately 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^2_{ij}. 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^1_i, F^2_{ij}) and $(F^1_{i'}, F^2_{ij'})$, are compatible only if the angle between the vectors $w^1 = \overrightarrow{F^1_i F^1_{i'}}$ and $w^2 = \overrightarrow{F^2_{ij} F^2_{ij'}}$ 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 $S = (F^1_{i_k}, F^2_{j_k})_k$ 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:

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

The distance $d(.)$ used in Equation (2) 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 (2), 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 itself becomes large.

In addition to the road edges, the features that we consider for map building are polygons that describe the surface of the terrain and the discrete obstacles. The algorithms for extracting the polygonal description are reported in [7] and [6]. To summarize, the features used in the matching are:

- 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.

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, and the new features are added to the map while updating the connectivity between features.

In the current implementation, the initial estimates of the displacement T_0 are taken from the central database that keeps track of the vehicle's position using dead reckoning. 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 6 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 4 shows the original sequence of raw range and reflectance images, Figure 5 shows perspective views of the corresponding individual maps, and Figure 6 is a rendition of the combined maps using the displacement and matches computed from the feature matching algorithm. This display is a view of the map rotated by 45° about the x axis and shaded by the values from the reflectance image.

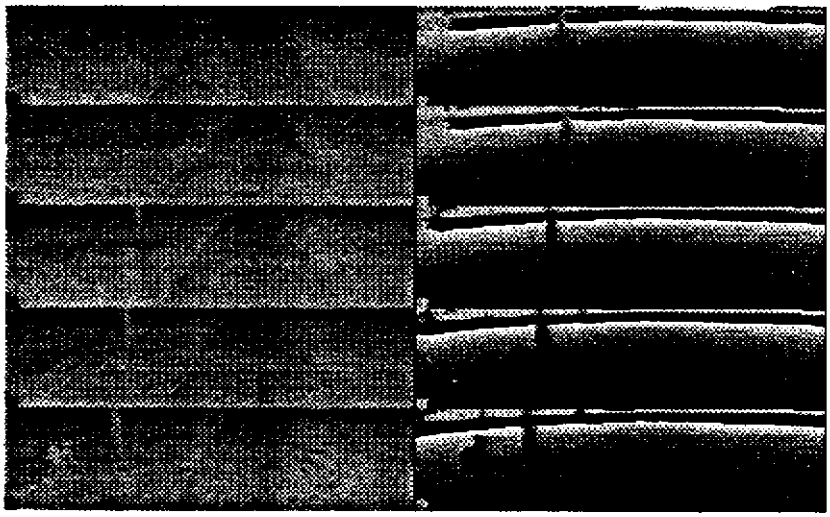


Figure 4: A sequence of range and reflectance images

Figure 8 shows a map built from twenty images over sixty meters. In this display, only the road edges, the center line of the road, and the discrete obstacles are shown. To obtain this result, the vehicle was driven by the road following algorithm of Section 2 at a continuous speed of 20 cm/s. The road following and map building modules are separated because the map building module requires an average of fifteen seconds of computation time per image which would prevent stable continuous motion. The overall structure of the map building/road following system that was used in this experiment is shown in Figure 7. The map building and road following modules are executed on two separate processors (Sun3's). They both access the ERIM scanner through a network interface. The road following module sends a new path that is a sequence of arcs to a separate helm module running on a three processor. The helm

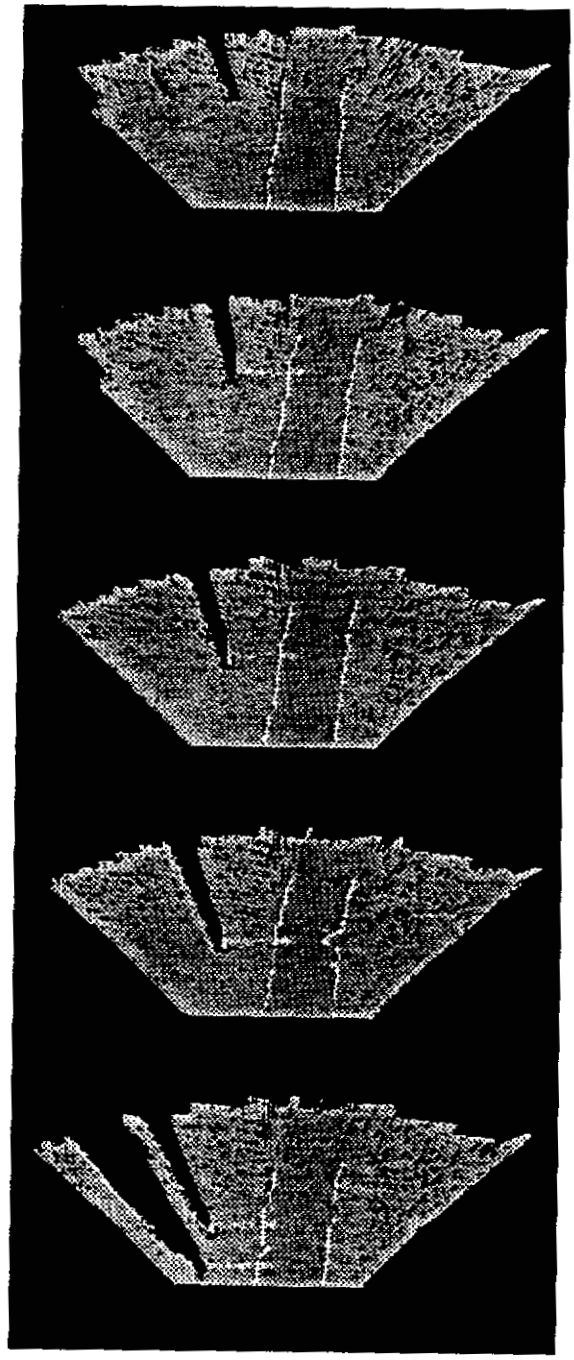


Figure 5: Individual maps

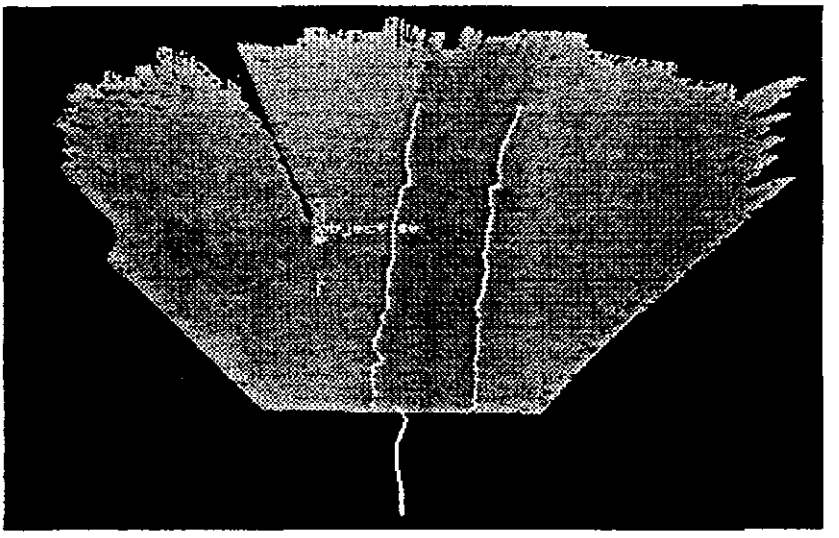


Figure 6: Perspective view of the combined map

also provides initial estimates of the vehicle position to the map building module. The communications between the helm and the two other modules are handled through the CODGER system [10].

4 Map-based road following

In this Section, we investigate the last part of the system, that is the use of the map built from road following to traverse the same portion of the world.

The map-based road following must proceed in three steps: computation of the starting position, path planning in the map, path execution and correction. The first step is needed to avoid constraining the starting position and heading of the vehicle at the beginning of the traversal of the map to those used to initiate the map building stage. The position and heading of the vehicle with respect to the map are computed by matching the features, road edges and objects, observed in an image taken at the starting position with the features of the map that are predicted to be visible given a rough initial guess of starting position. The matching algorithm is basically the same as the one used for the map building except that in the current implementation, only road edges and discrete obstacles are used. For example, Figure 9 shows the initial guess of the starting position (marked by a cross) and the portion of the road and the obstacle that are used for the matching. The map features are predicted by intersecting the sensor field of view with the map.

Given the starting position, the second step is to compute a path that follows the road using the map. This step is the most straightforward in that any path planner that provides for smooth paths can be used. For example, Figure 10 shows a path (solid line) composed of a sequence of circular arcs. The path is computed by dividing the center curve of the road (dotted line) into small segments over which the pure pursuit path planning algorithm of Section 2 is applied.

Once a path is computed, the vehicle is ready to follow the road based on the map. Ideally, the vehicle

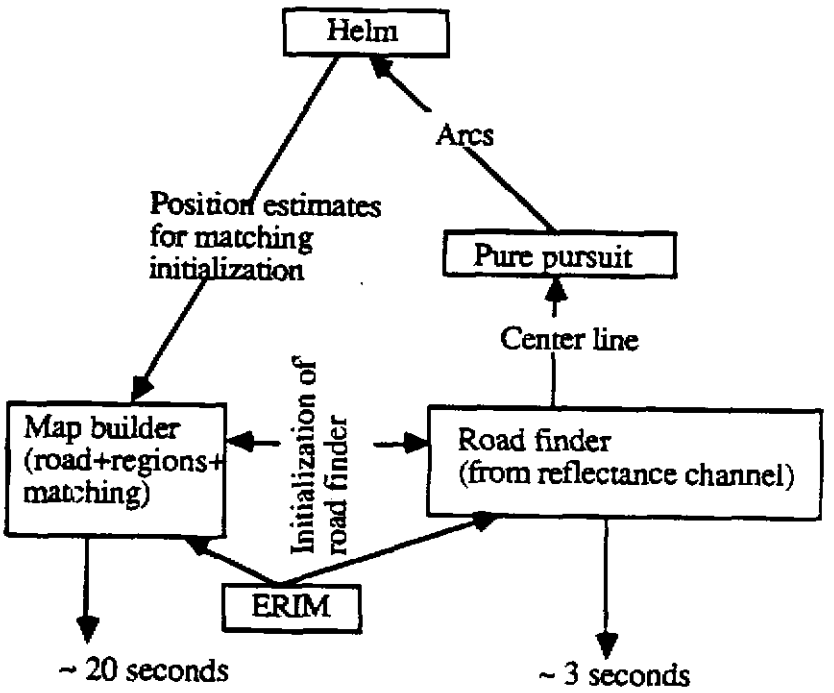


Figure 7: The map building/road following system

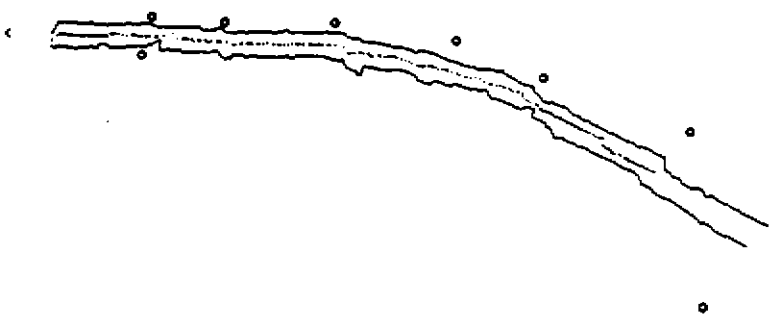


Figure 8: Complete map of a road scene

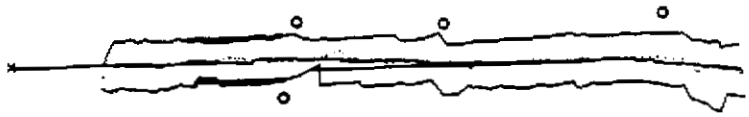


Figure 9: Estimation of the starting position and heading

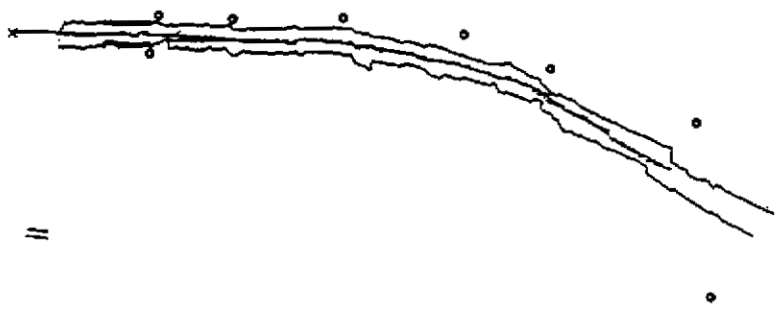


Figure 10: Path planned using a map

should be able to correctly execute the path without any perception at all. In practice, however, the vehicle will drift away from the ideal path due to wheel slippage, and the accumulation of small controller errors and numerical errors. Therefore, the position and heading of the vehicle with respect to the map must be recomputed periodically by comparing the features that are actually observed while executing the path and the features that are predicted from the map given the current estimate of the vehicle's position. The question now is how often should we make a position correction, that is take an image, extract road edges and objects, and match them with the map, in order to stay within reasonable bounds of the original path. This problem is the key to map-based navigation: If the corrections are performed too often we are back to the original road following approach and we loose the benefit of having a map. If, on the other hand, we do not perform enough corrections along the path, we may drift significantly far from the nominal path and eventually run off-road. Furthermore, the corrections should be meaningful in the sense that enough features should be present at the time of the correction to ensure that the newly computed position is indeed closer to the truth than the currently available estimate. Several strategies are possible to choose the locations at which corrections should be performed. An attractive strategy is to estimate the uncertainty on the position and heading as the vehicle moves, a new correction is requested whenever the uncertainty reaches a threshold that indicates that the vehicle is too far from its nominal path [13]. This approach guarantees that the distance between the vehicle's path and the nominal path always lies within preset bounds. It does not, however, guarantee that the images taken at the time at which a correction is needed contain enough features of interest. Another possible approach is to make a correction whenever the map predicts that features of interest may be observed from the current position. In our case, it is important to guarantee that the corrections are performed when objects are visible, since otherwise the correction would be computed on the basis of the road edges only and would therefore be ambiguous. A correction is therefore computed whenever at least one object is predicted to be visible from a position along the path. Matching the predicted objects and road edges from the map with the observed road and objects provides an unambiguous new estimate of the vehicle's position and heading. Figure 11 shows the locations at which new images are taken for computing the corrections along the path of Figure 10. The road edges and objects that are matched with the corresponding observed features are shown as bold segments of the road edges and dark circles respectively. The crosses along the path indicate the successive positions of the vehicle at regular intervals of one second (at a speed of 20 cm/s). The position is not displayed if an image is being processed, therefore the gaps in the stream of positions in this display illustrate the time spent in processing images while executing the initial path (The percentage is in reality a bit lower than what appears on display because the map, range image processing, and helm modules normally run on different processors whereas this display was produced with all the modules executing on one Sun).

Computing a correction gives an offset $\Delta = (\Delta x, \Delta y, \Delta\theta)$ between the nominal position and heading and the actual values at the time the image is taken. This offset must be used to correct the current course of the vehicle. This is achieved by shifting the path that has been executed while the image was being processed by Δ, by replanning from the current position as given by the shifted, and by replacing the pending set of motion commands by this new path. Figure 12 illustrates such a sequence of events: As the vehicle comes into view of the first objects, an image is taken and matched against the map, the new position is shown as a cross on the left of the initial path, a new path is planned that takes the vehicle back to its original course.

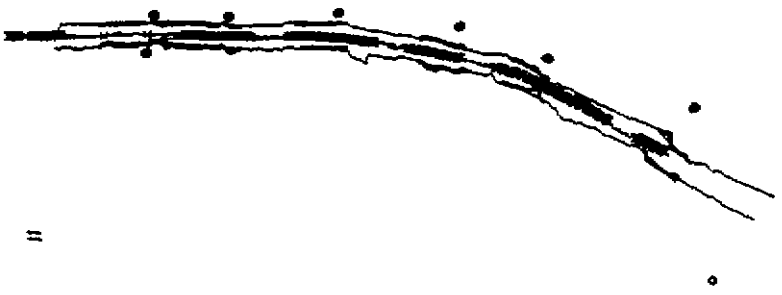


Figure 11: Locations at which images are taken along the path

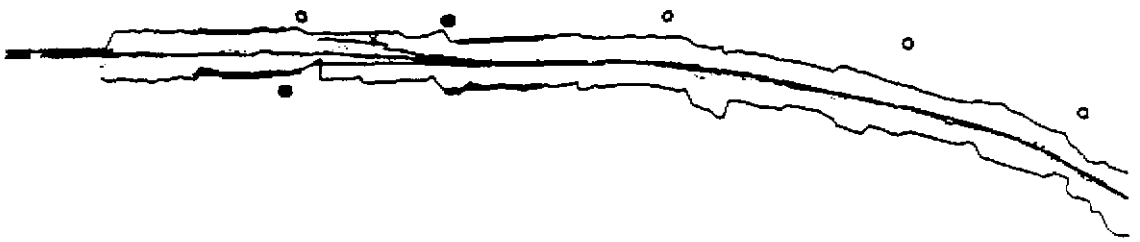


Figure 12: Corrected path

These results show that it is possible to use a map to efficiently guide the navigation of an autonomous vehicle. The main benefit is that considerably fewer images have to be processed while retraversing the map. For example, The map of Figure 11 requires seven images to be processed. Following the same road at the same speed without the support of a map would require at least 25 images for a displacement of two meters between consecutive images. The reason for the discrepancy is that even if the position of the road were computed perfectly from each individual image, the path planner would not have information far enough in front the vehicle to plan a stable path that is guaranteed to remain on the road. Although the same results could be obtained by using a map that is entered manually, it is important to note that the combination of map building from sensor data and map-based navigation results in a fully autonomous system that can learn its environment and use its new knowledge to navigate it.

5 Conclusion

The road following and map building system shows that road environments can be efficiently navigated and mapped using an active sensor such as a laser range finder. The map based navigation system shows that the information gathered during a initial traversal of the road can be used to improve the navigation over a portion of the stretch of road already explored. Specifically, using the map provides an initial path to follow, and a list of optimal locations at which visual data should be processed in order to correct the vehicle position that drifts over time. The combination of those three components provide a basis for autonomous navigation of roads including 3-D terrain modeling and knowledge gathering and utilization through map building and map based navigation.

We are currently extending the ideas used in those systems to the case of cross-country navigation and combined on road/off road navigation in which the map contains a representation of terrain regions in addition to the road model and the discrete obstacles. This type of information is currently extracted but it is not used for the map based navigation. The system presented here uses a simple path planner based on the pure pursuit control scheme. Our plan is to use the path planner described in [13] to take into account vehicle model and uncertainty, and to be able to apply our approach to cross-country navigation.

Acknowledgements

Mike Blackwell, James Frazier, and David Simon made the NAVLAB experiments possible. Chuck Thorpe provided the path planner used in this system. Keith Gremban ported and demonstrated the road following program on the Martin Marietta ALV.

References

[1] J. Beyer, C. Jacobus, and F. Pont. Autonomous Vehicle Guidance using Laser Range Imagery. In *SPIE Vol. 852, Mobile Robots II*, Cambridge, 1987.

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

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

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

[5] K.D. Gremban, C.E. Thorpe, and T. Kanade. Geometric Camera Calibration Using Systems of Linear Equations. In *Proc. Image Understanding Workshop*, Cambridge, 1988.

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

[7] M. Hebert, I. Kweon, and T. Kanade. *3-D Vision Techniques for Autonomous Vehicles*. Technical Report CMU-RI-TR-88-12, The Robotics Institute, Carnegie-Mellon University, 1988.

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

[9] 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.

[10] S. A. Shafer, A. Stentz, and C. E. Thorpe. *An Architecture for Sensor Fusion in a Mobile Robot.* Technical Report CMU-RI-TR-86-9, Carnegie-Mellon University, the Robotics Institute, 1986.

[11] U.K. Sharma and L.S. Davis. Road Following by an Autonomous Vehicle using Range Data. In *SPIE Vol. 727, Mobile Robots II*, Cambridge, 1986.

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

[13] T. Stentz. *The NAVLAB System for Mobile Robot Navigation.* PhD thesis, Carnegie-Mellon University, Fall 1988.

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

[15] 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.

[16] R. Wallace, K. Matsuzaki, Y. Goto, J. Crisman, J. Webb, and T. Kanade. Progress in robot road-following. In *IEEE International Conference on Robotics and Automation*, 1986.

[17] 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.

[18] A.M. Waxman, J.J. LeMoigne, L.S. Davis, B. Srinivasan, T.R. Kushner, E. Liang, and T. Siddalingaiah. A Visual Navigation System for Autonomous Land Vehicles. *Journal of Robotics and Automation, Vol. 3*, 1987.

[19] 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.

Chapter V
3-D Vision Techniques for Autonomous Vehicles

Martial Hebert, Takeo Kanade, Inso Kweon

Chapter V
3-D Vision Techniques for Autonomous Vehicles

Martial Hebert, Takeo Kanade, Inso Kweon

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.

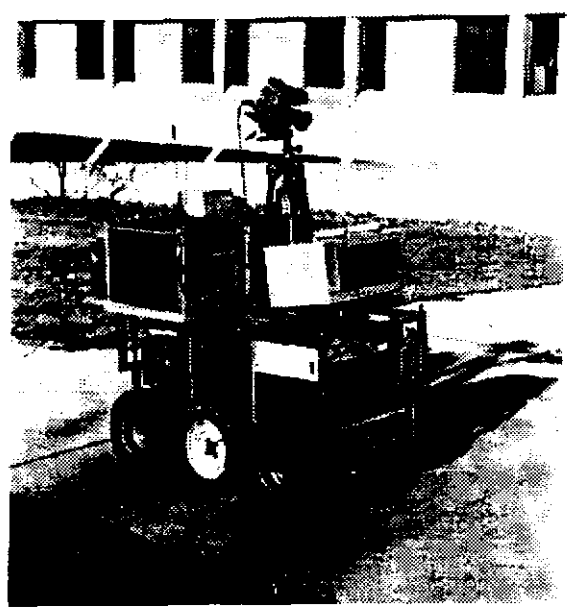


Figure 1: The Terregator

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

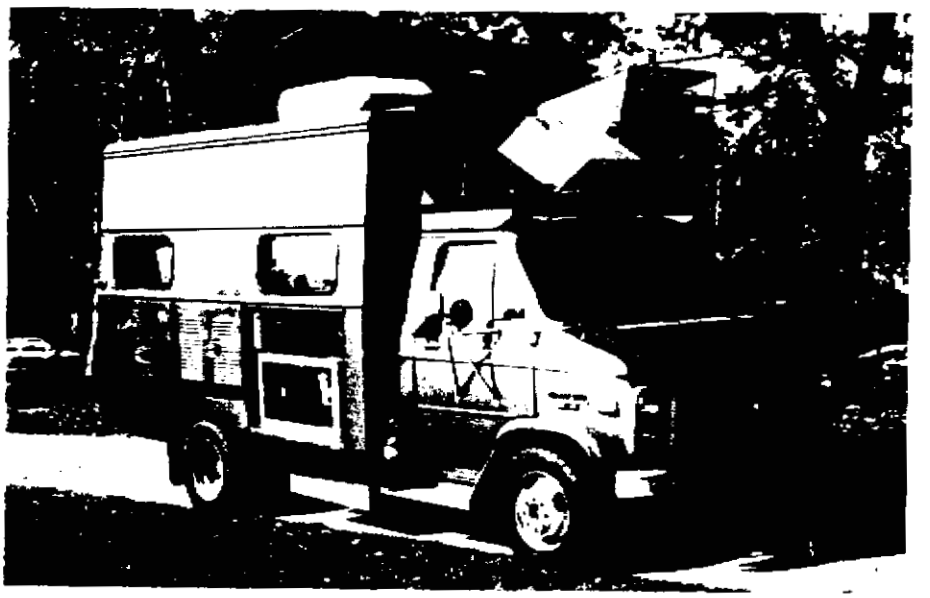


Figure 2: The Navlab

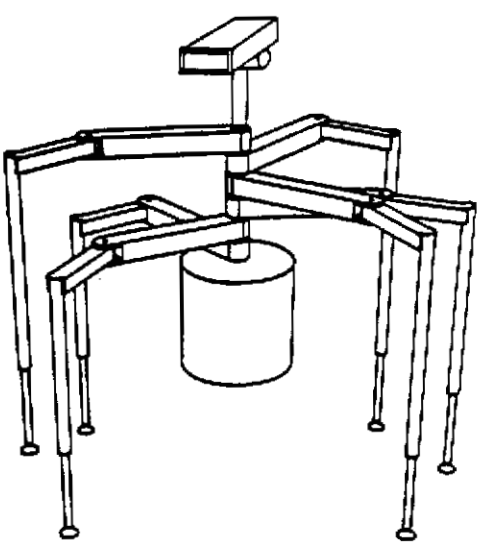


Figure 3: The Ambler

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° x 80° field of view. The data produced by the ERIM sensor is a 64 x 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 x 256 x 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).

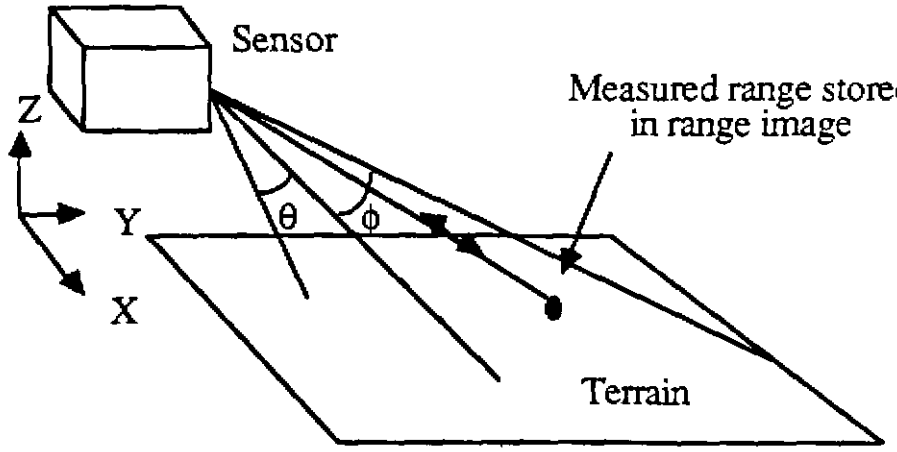


Figure 4: Geometry of the range sensor

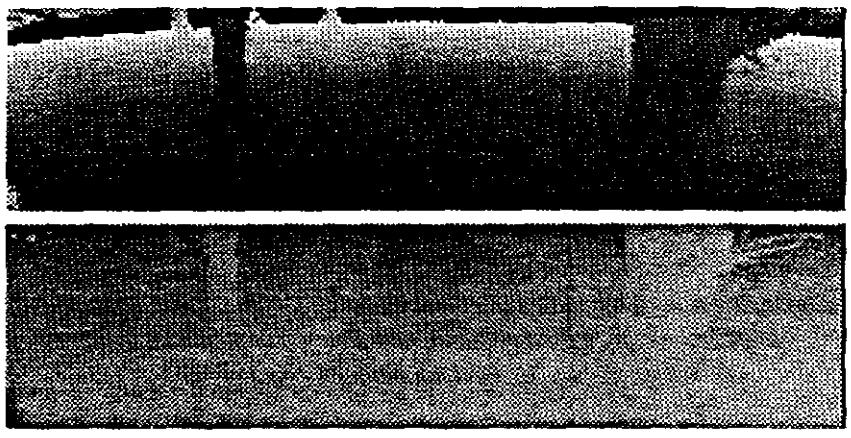


Figure 5: Range and reflectance images

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

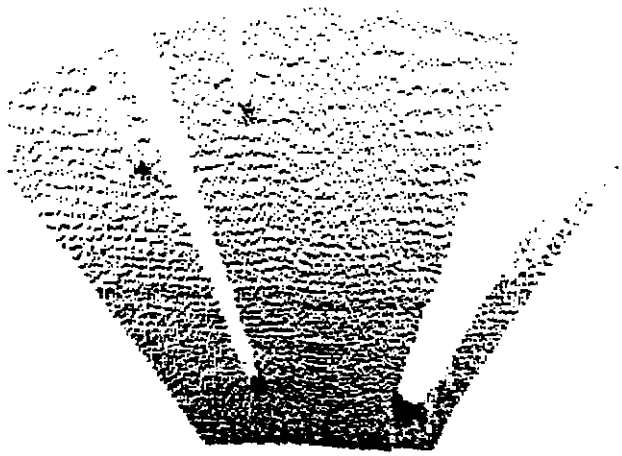


Figure 6: Overhead view

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.

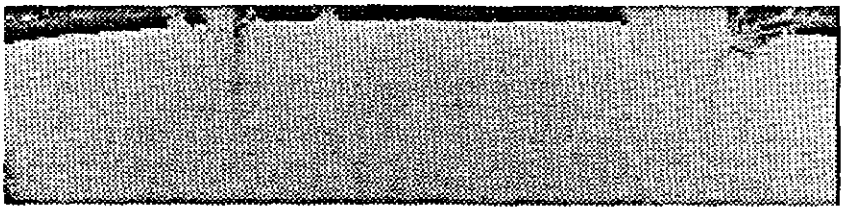


Figure 7: Corrected reflectance image

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].

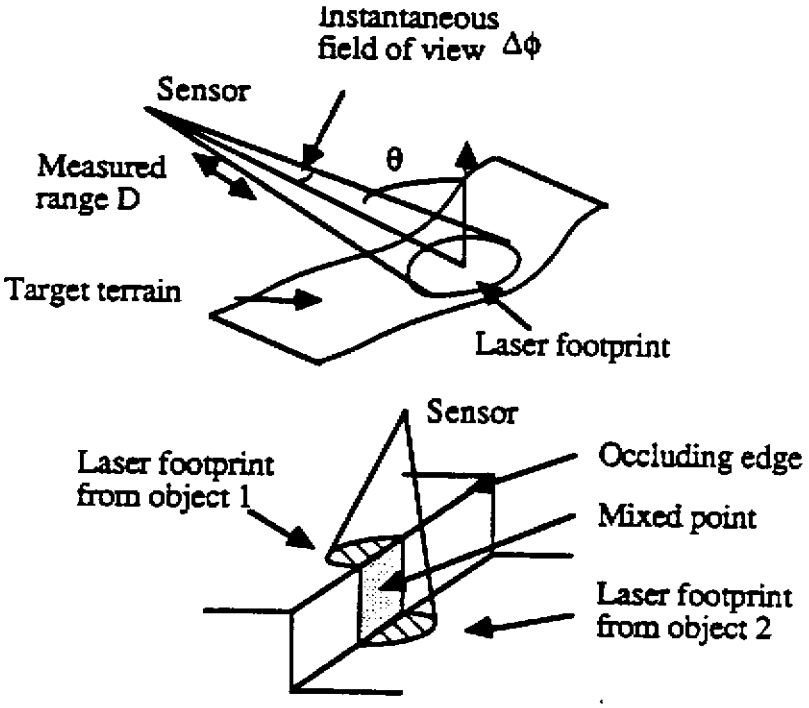


Figure 8: Sources of noise in range data

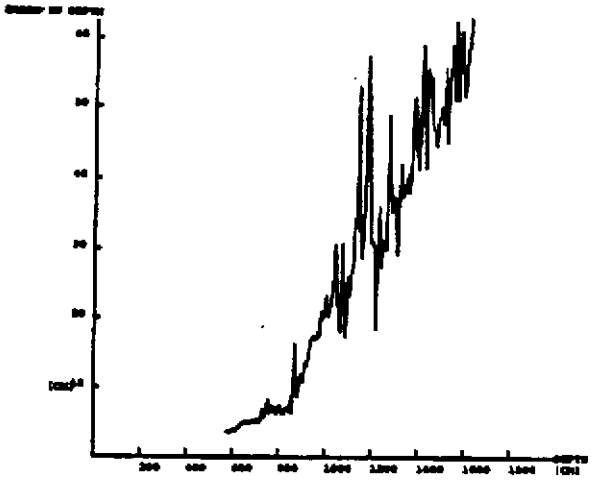


Figure 9: Noise in range data

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

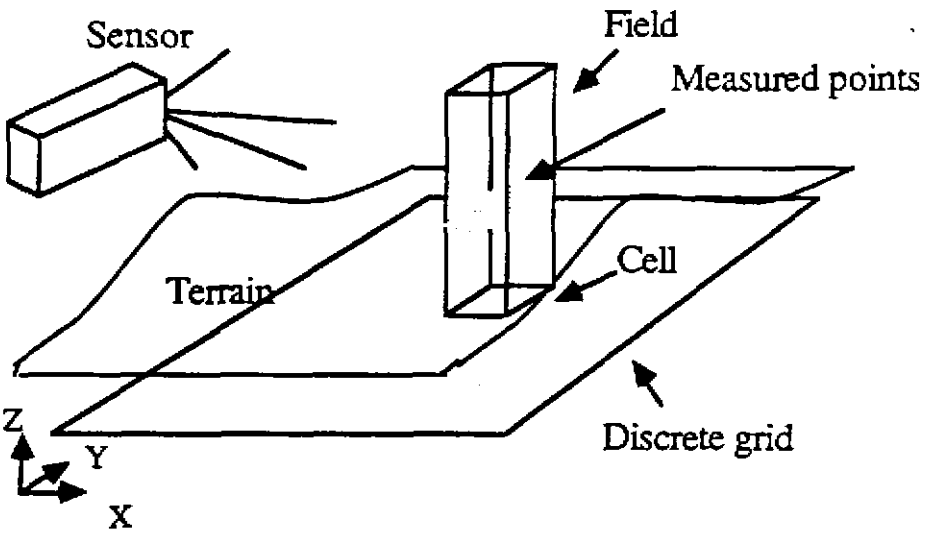


Figure 10: Structure of an elevation map

(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.

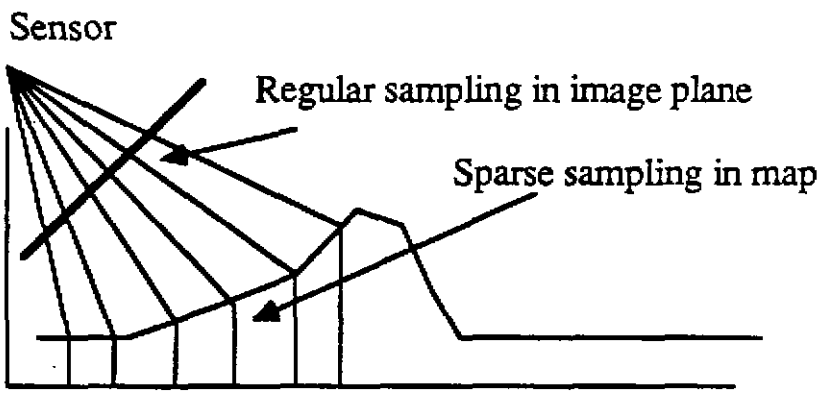


Figure 11: The sampling problem

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

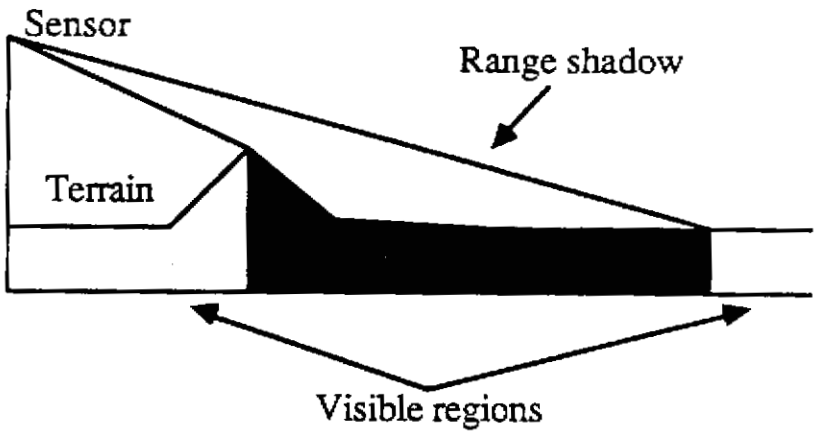


Figure 12: An example of a range shadow

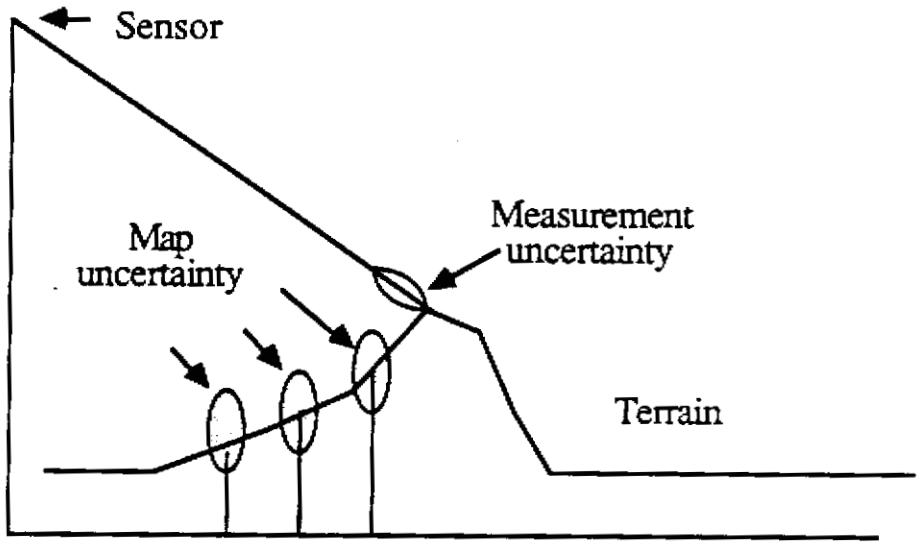


Figure 13: Representing uncertainty

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.

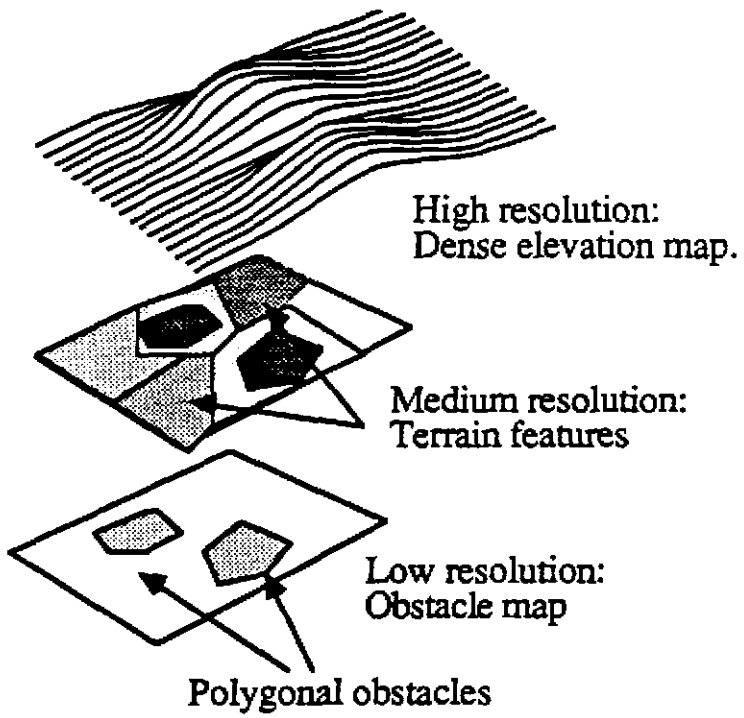


Figure 14: Levels of terrain representation

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.

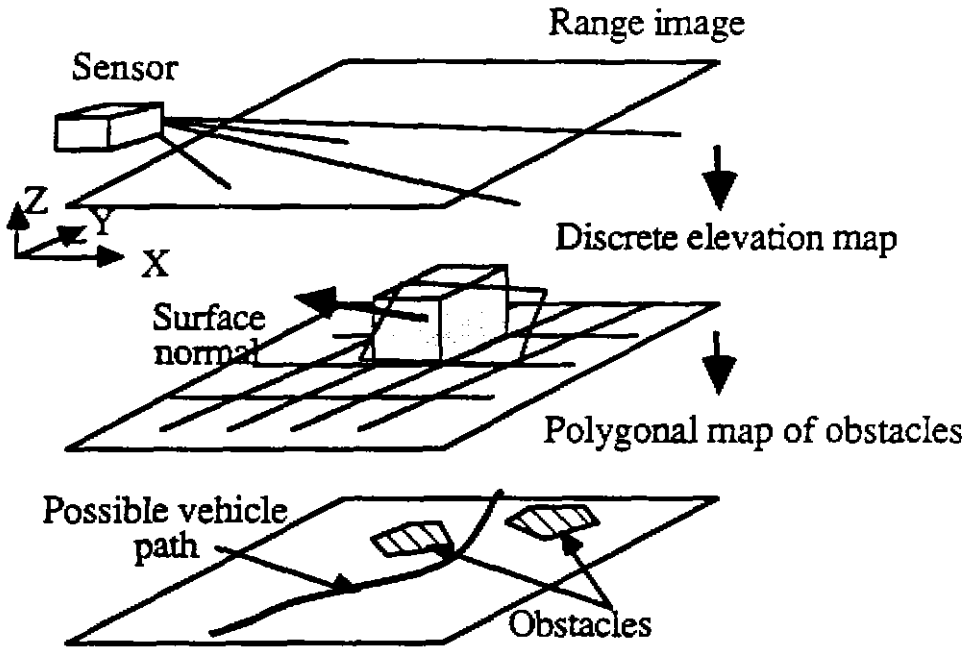


Figure 15: Building the obstacle map

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

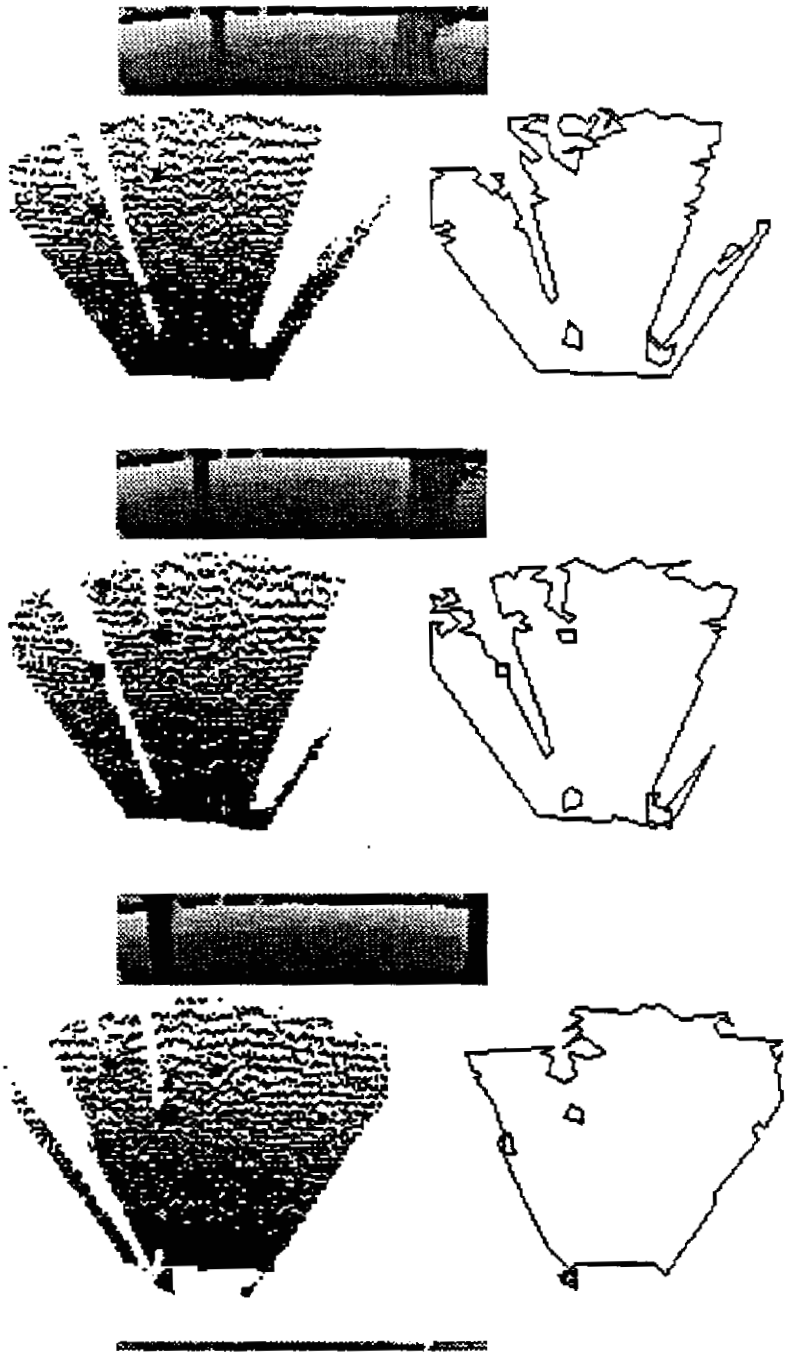


Figure 16: Obstacle detection on a sequence of images

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.

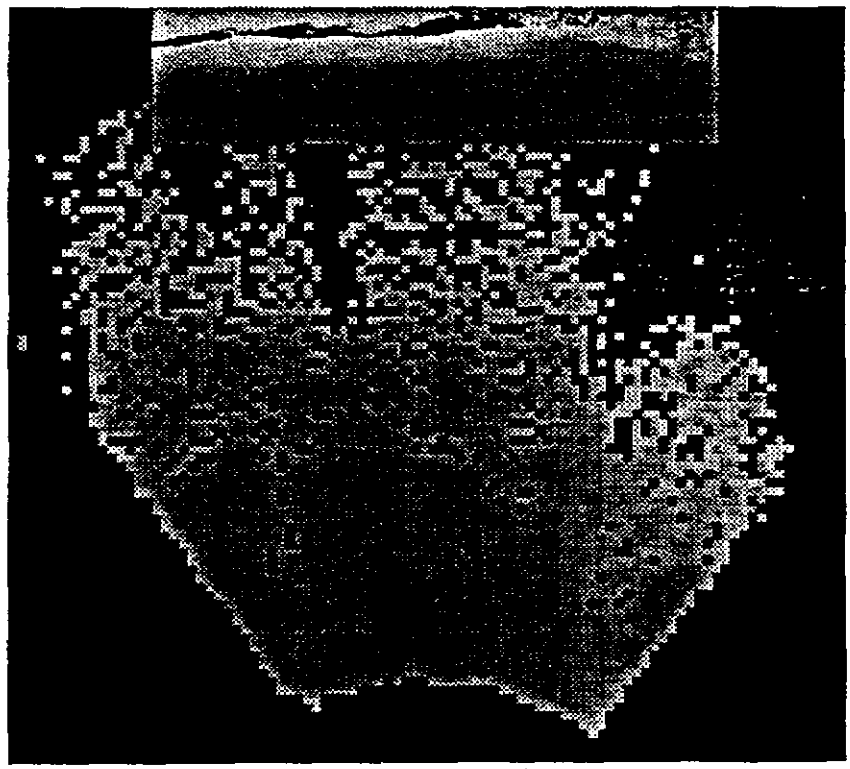


Figure 17: Range image and elevation map

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

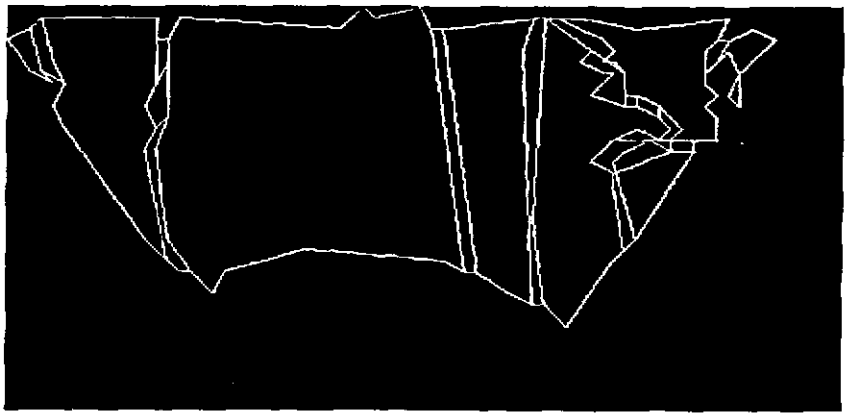


Figure 18: Polygonal boundaries of terrain regions

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, $Diff(\phi_1) = D_I(\phi_1) - I(\phi_1, \hat{\theta}_I(\phi_1))$ and $Diff(\phi_2) = D_I(\phi_1) - I(\phi_2, \hat{\theta}_I(\phi_2))$ have opposite signs, where $\hat{\theta}_I(\phi)$ is the image column that is the closest to $\theta_I(\phi)$. 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 $\theta_I(\phi)$ and $D_I(\phi)$ 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 $\phi, \theta_I(\phi), D_I(\phi)$. 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:

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

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

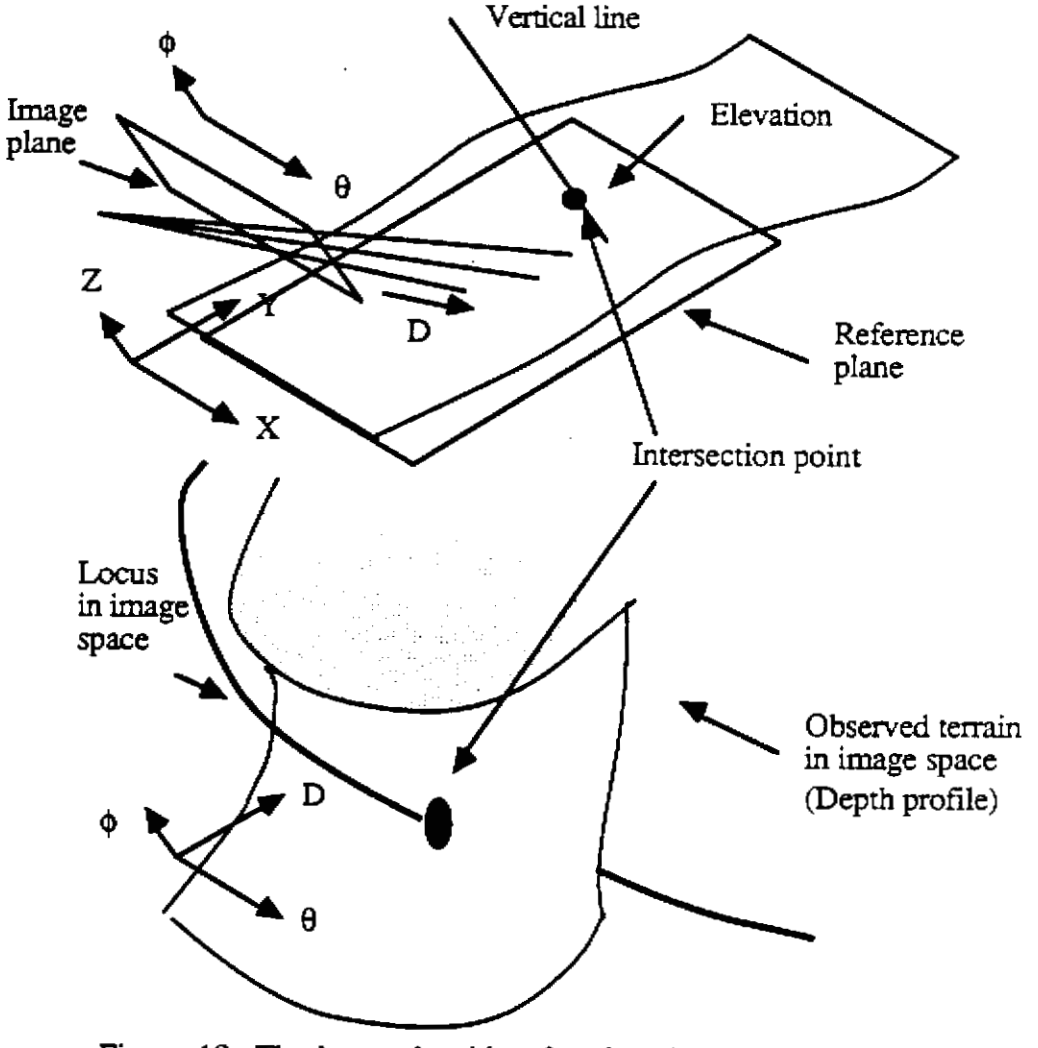


Figure 19: The locus algorithm for elevation maps

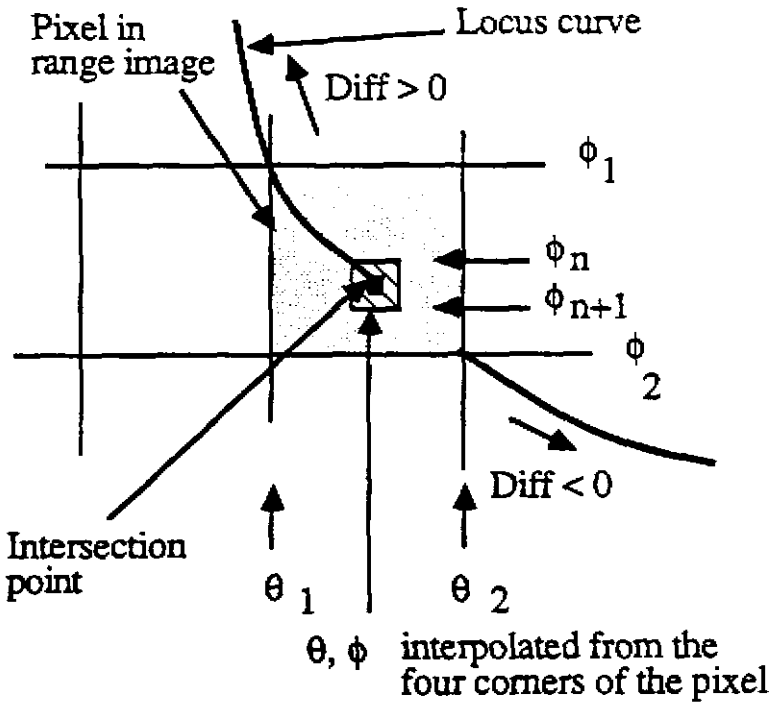


Figure 20: Image interpolation around the intersection point

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.

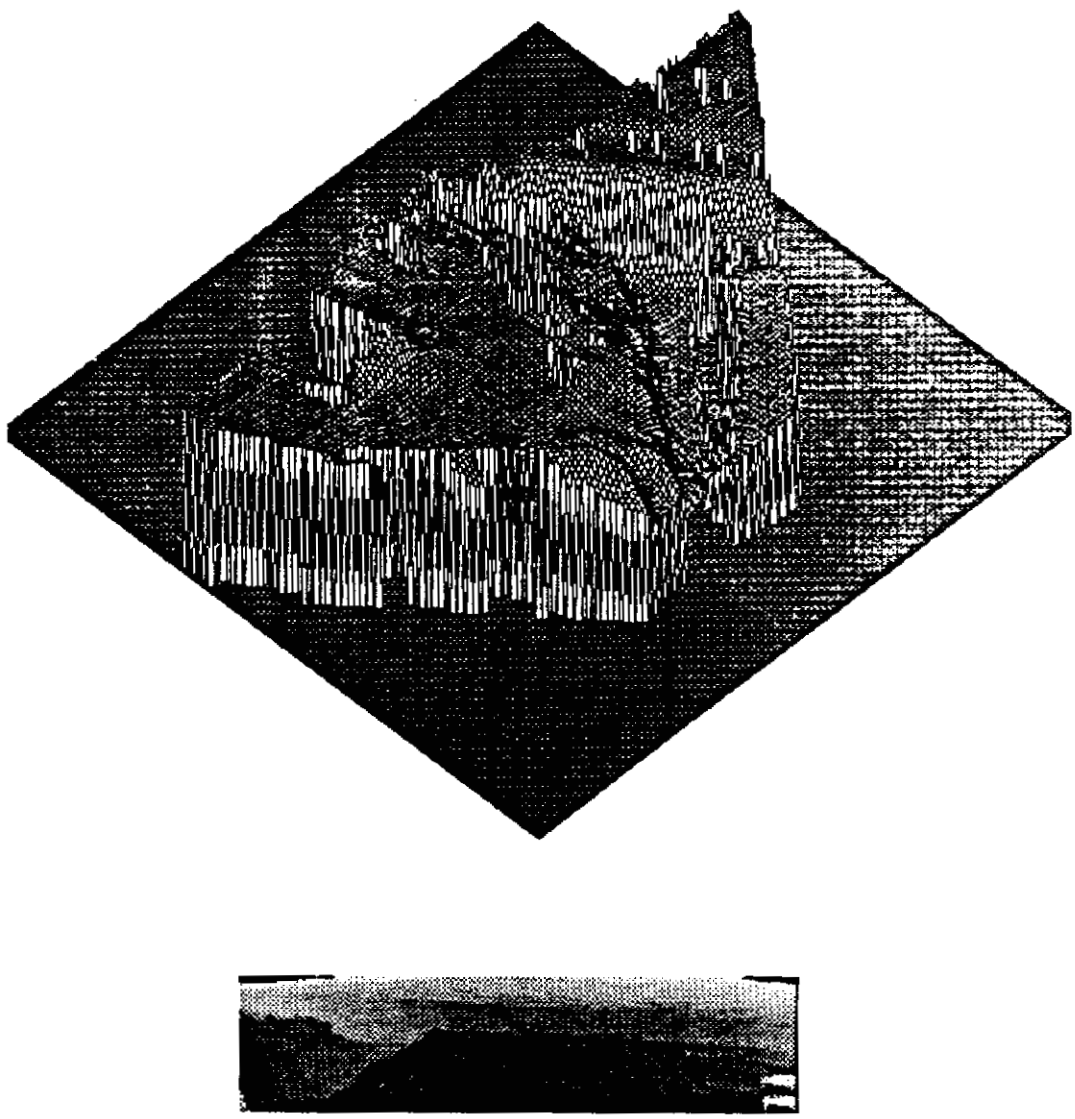
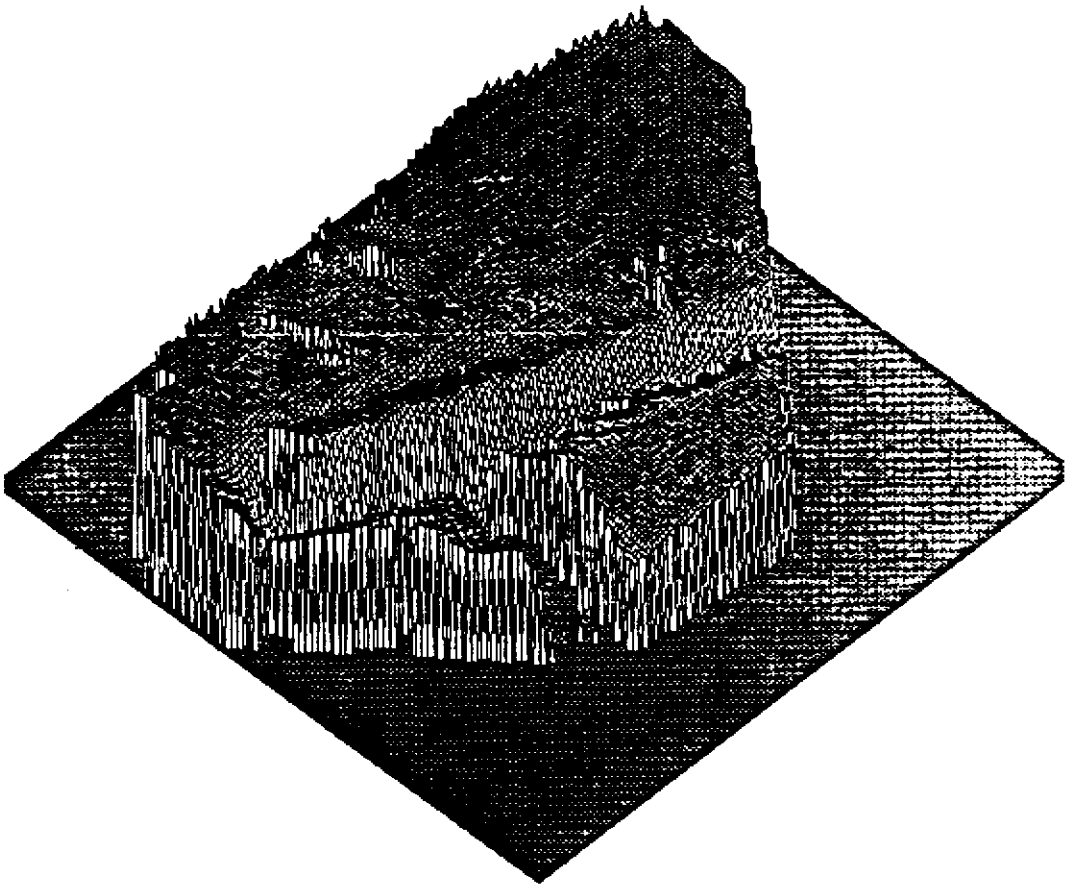


Figure 21: The locus algorithm on range images



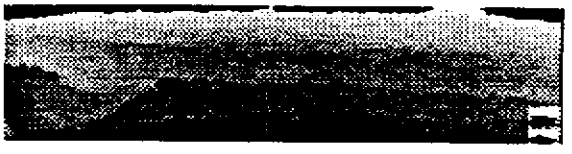


Figure 21: The locus algorithm on range images (Continued)

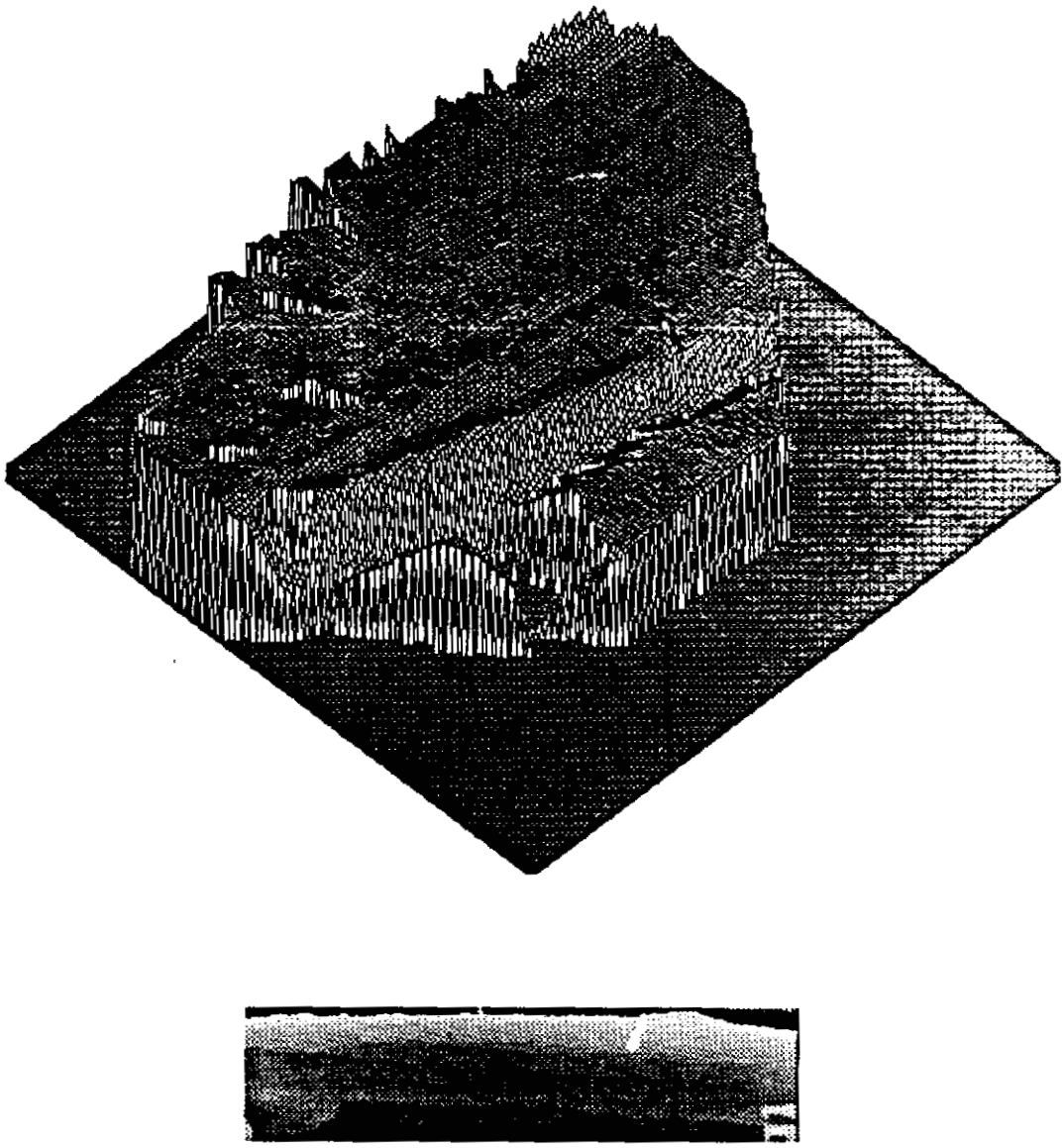


Figure 21: The locus algorithm on range images (Continued)

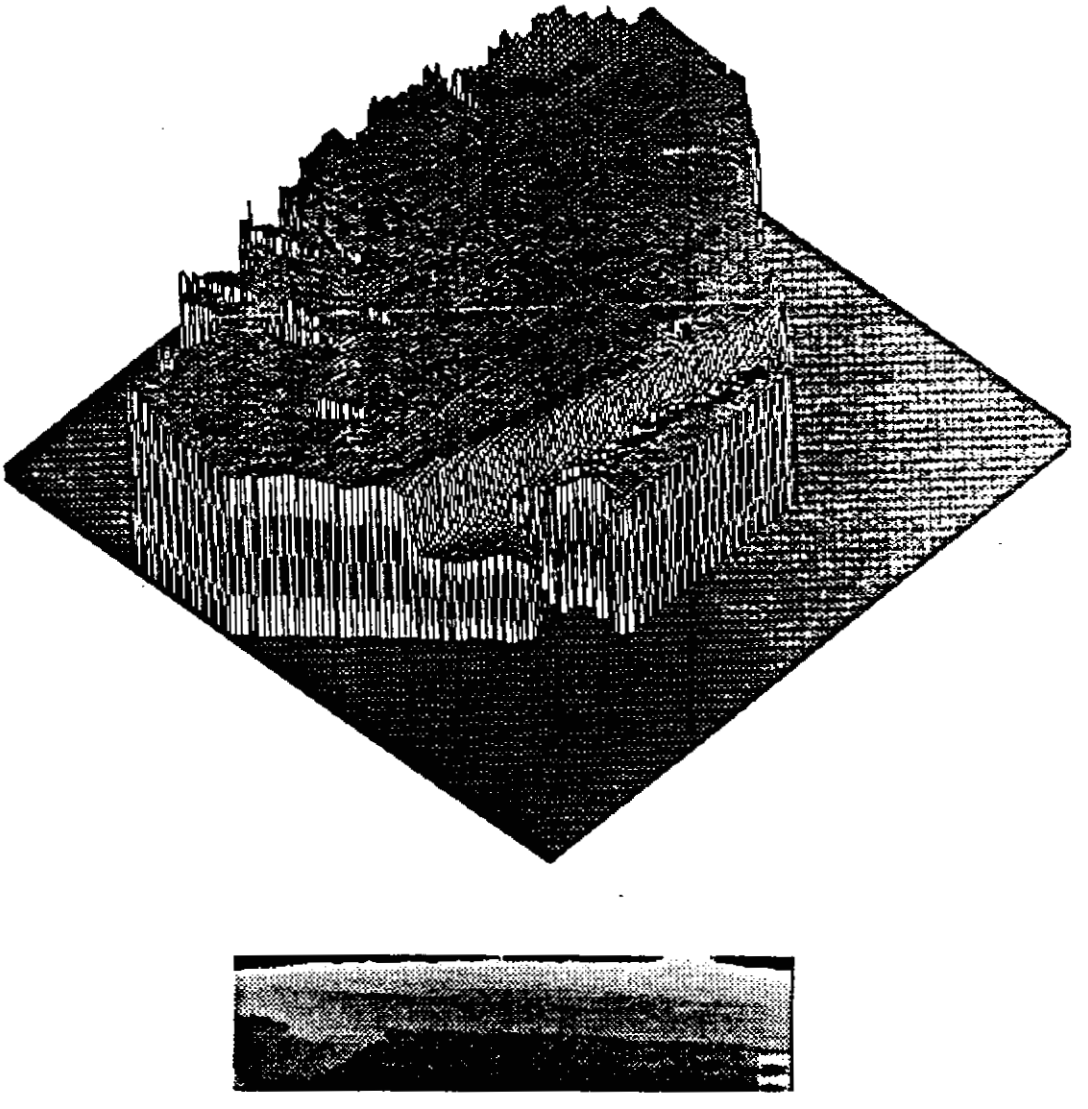


Figure 21: The locus algorithm on range images (Continued)

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:

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

where h_i is the true elevation value and $\bar{h}_i$ 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.

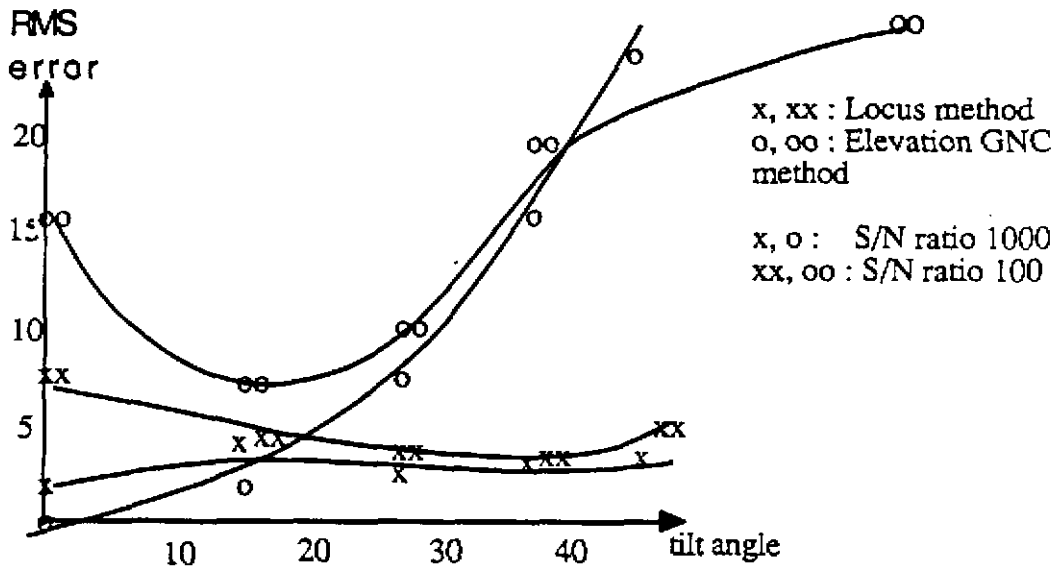


Figure 22: Evaluation of the locus algorithm on synthesized images

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:

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

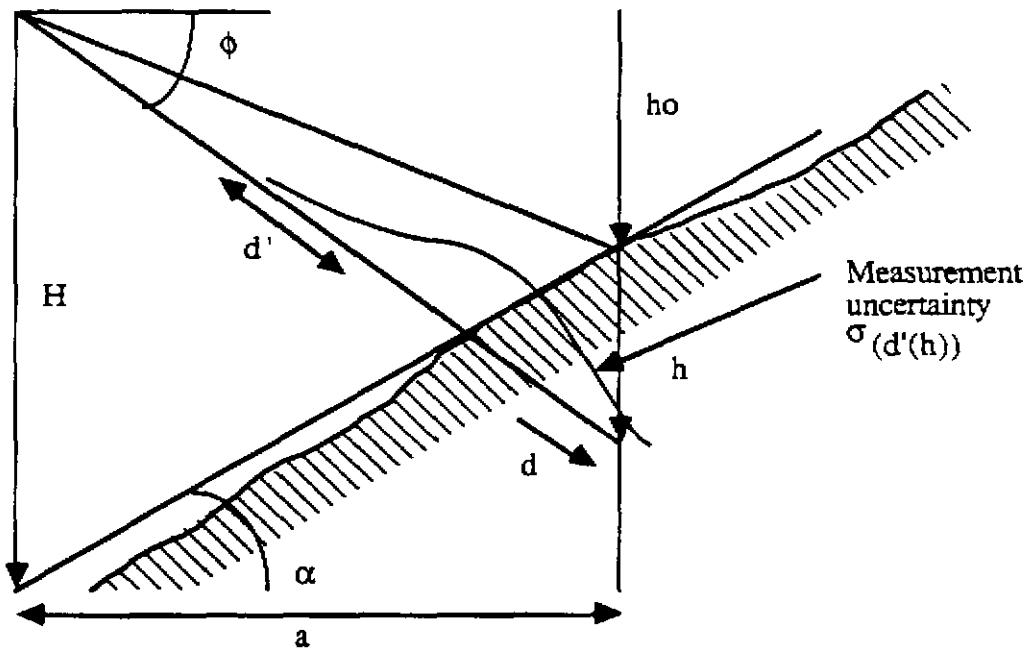


Figure 23: Computing the uncertainty from the locus algorithm

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).

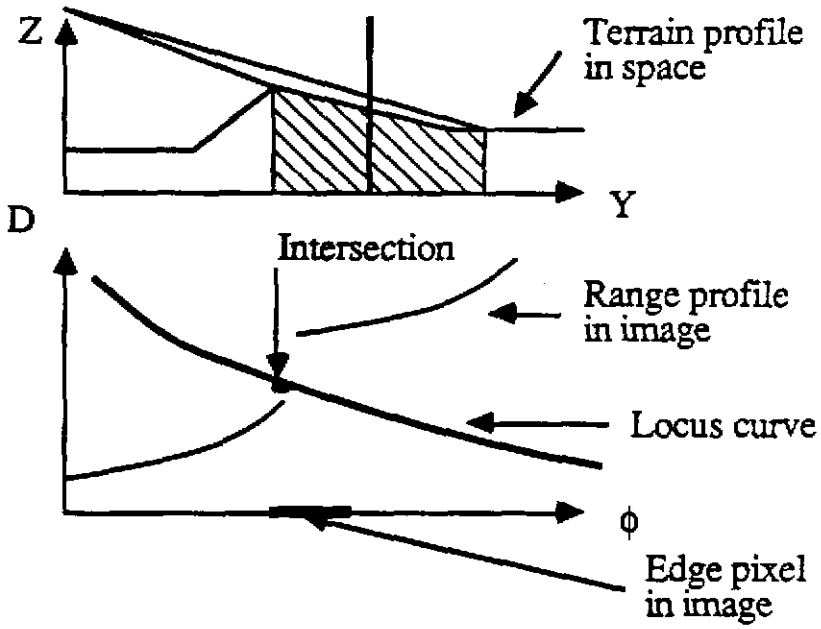


Figure 24: Detecting range shadows

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

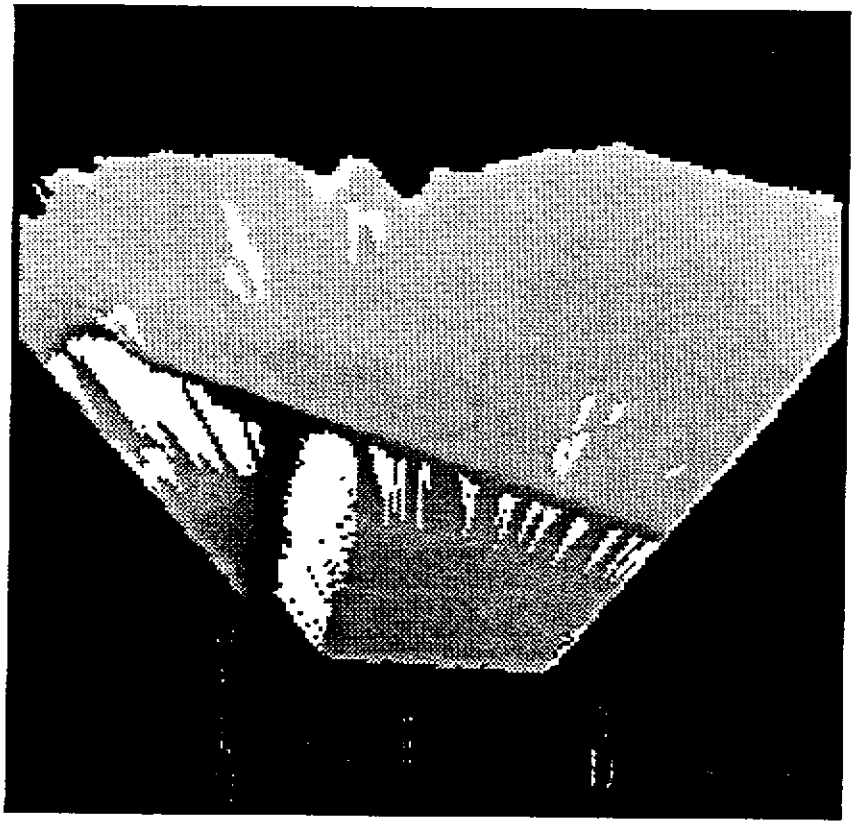


Figure 25: Shadow regions in an elevation map

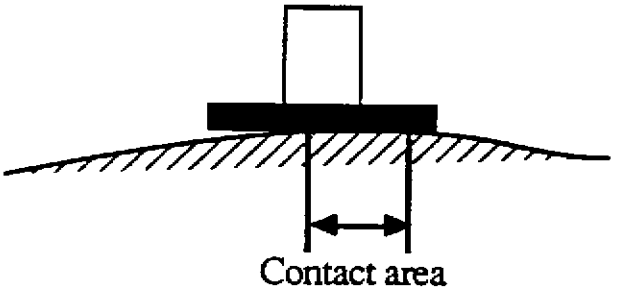


Figure 26: Footfall support area

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.

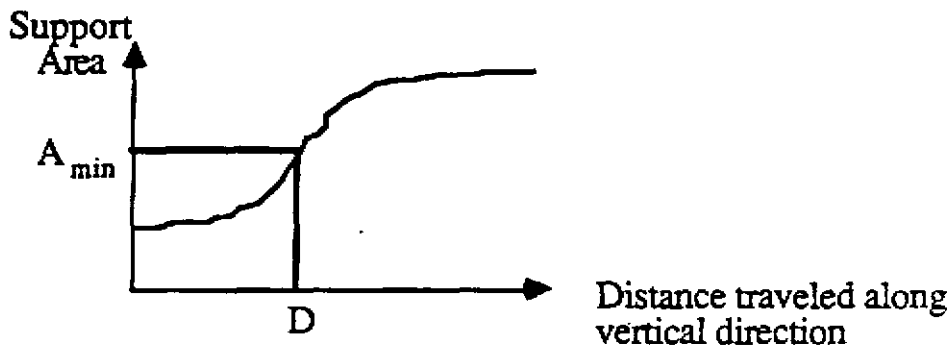


Figure 27: Support area versus elevation

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.

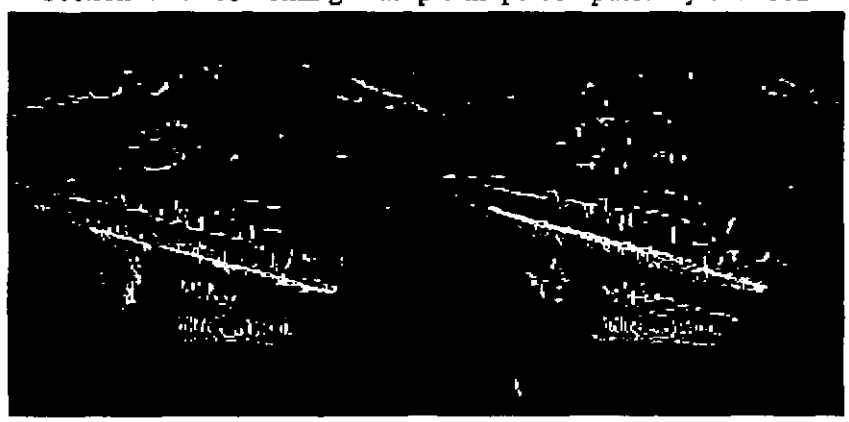


Figure 28: The high curvature points of an elevation map

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

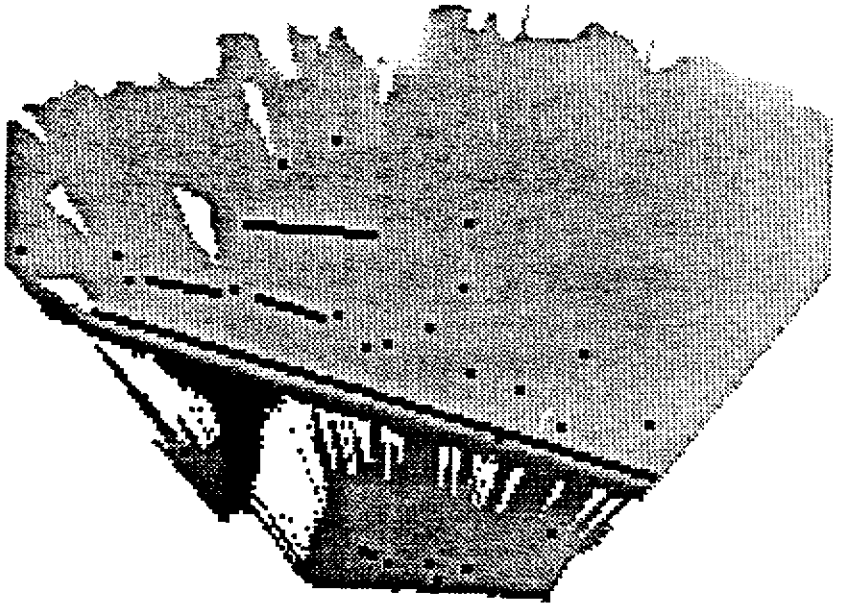


Figure 29: Local features from a high resolution elevation map

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.

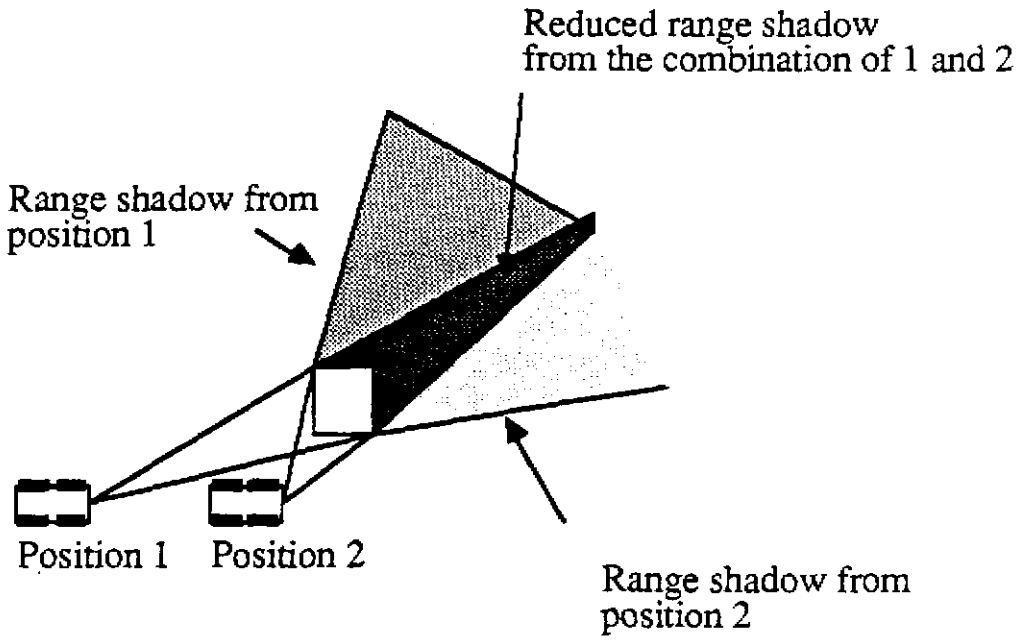


Figure 30: Reducing the range shadow

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

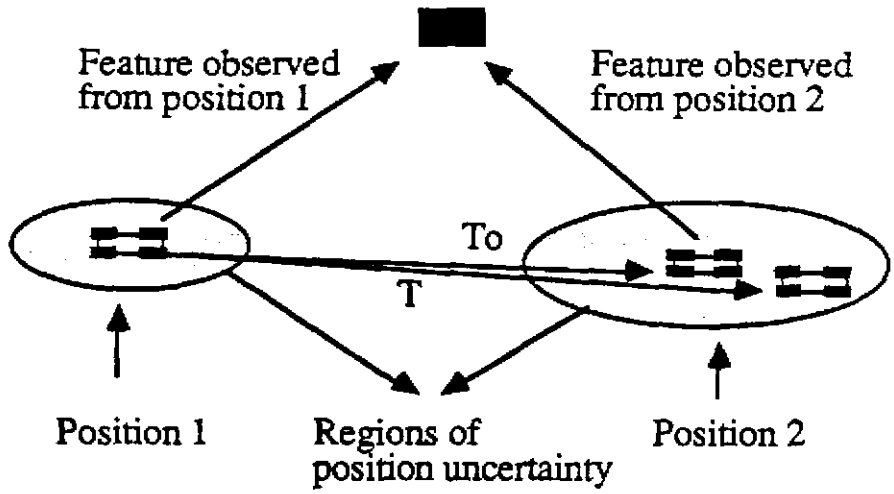


Figure 31: Matching maps for position estimation

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.

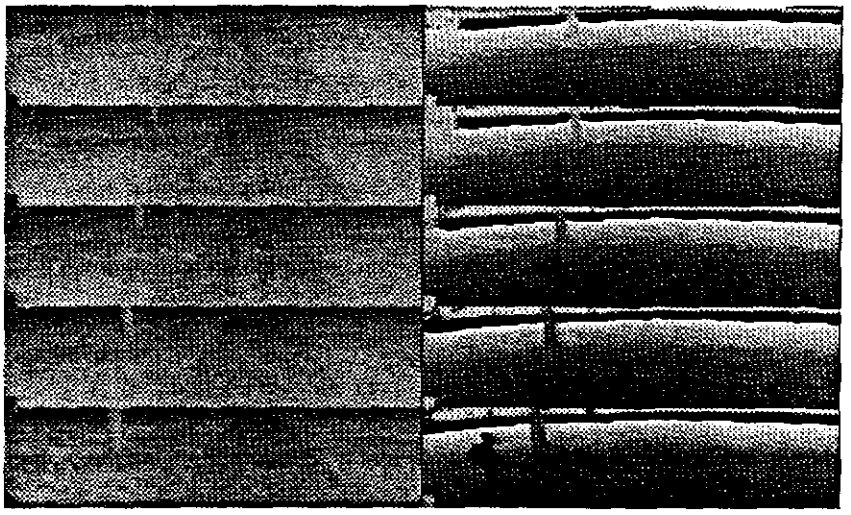


Figure 32: A sequence of range and reflectance images

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

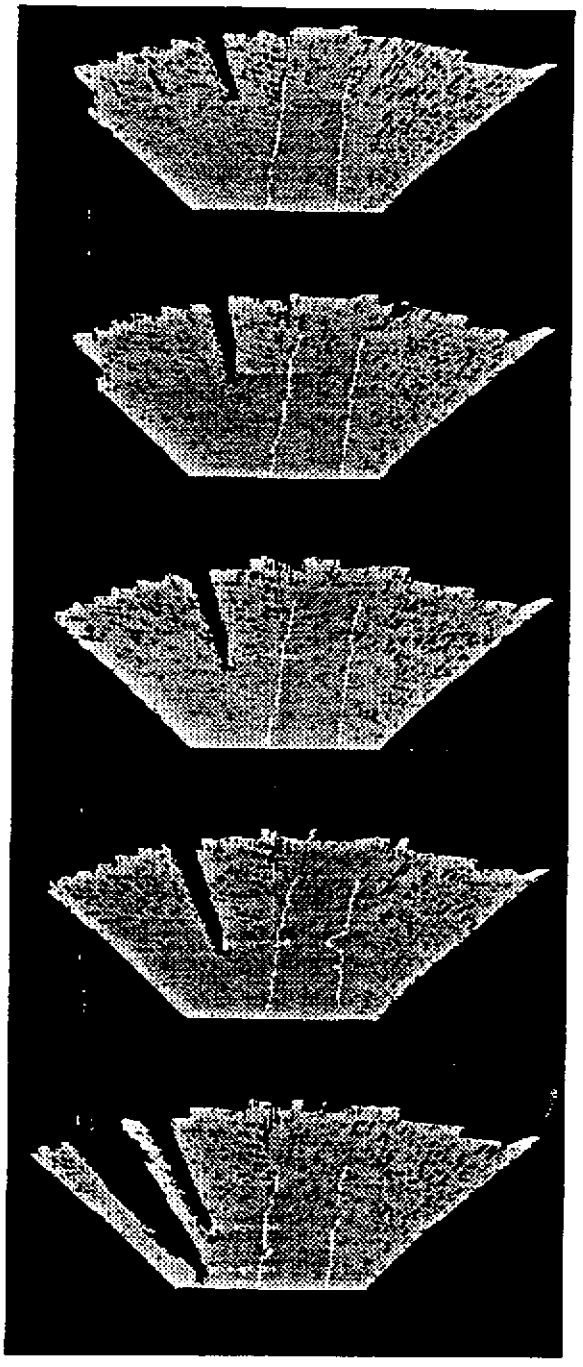


Figure 33: Individual maps

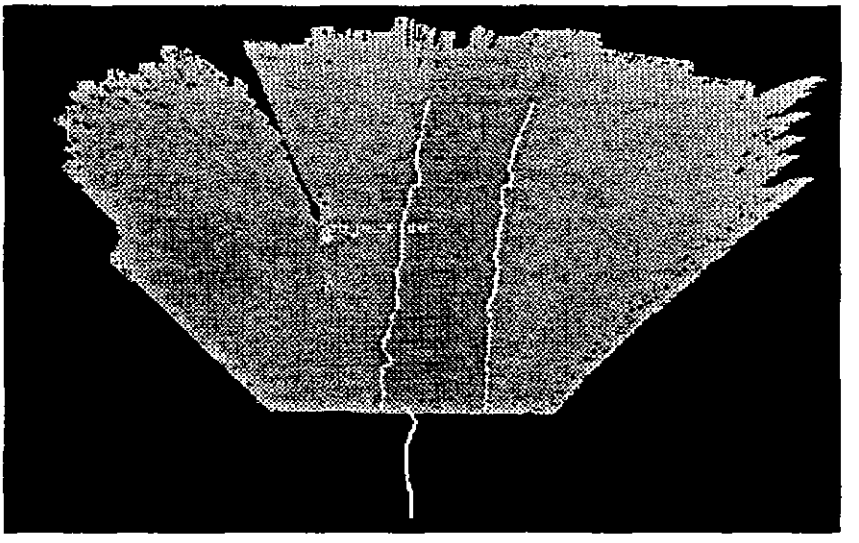


Figure 34: Perspective view of the combined map

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

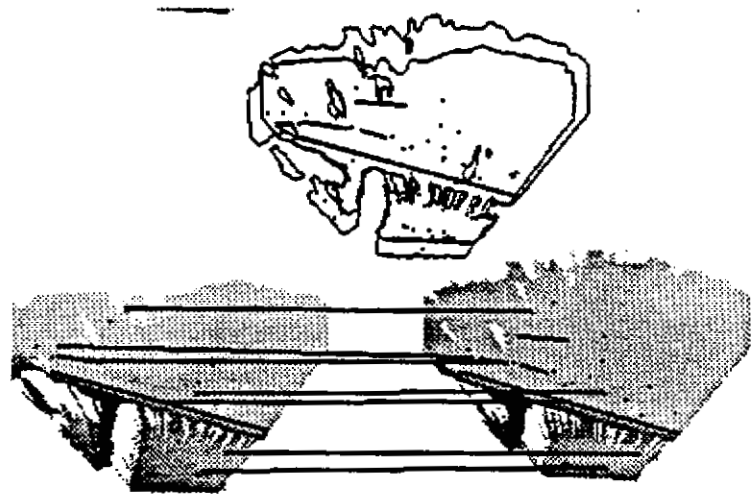


Figure 35: Matching maps using local features

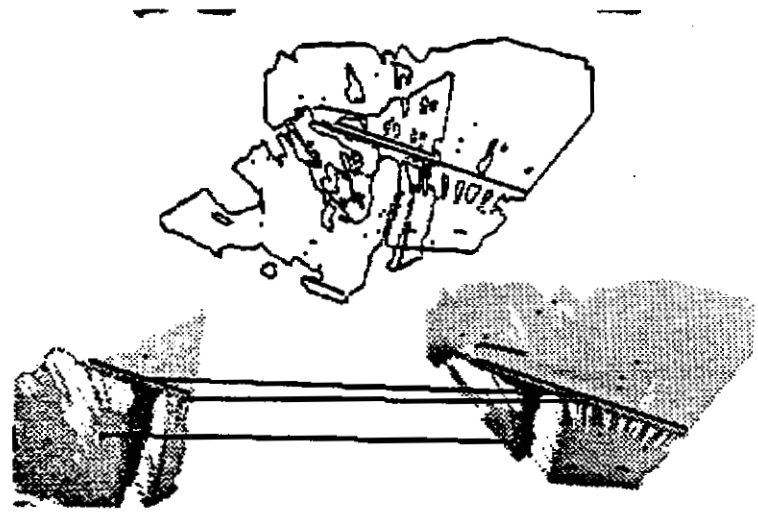


Figure 36: Matching maps using local features (large rotation component)

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 $f_2(u',v')$.

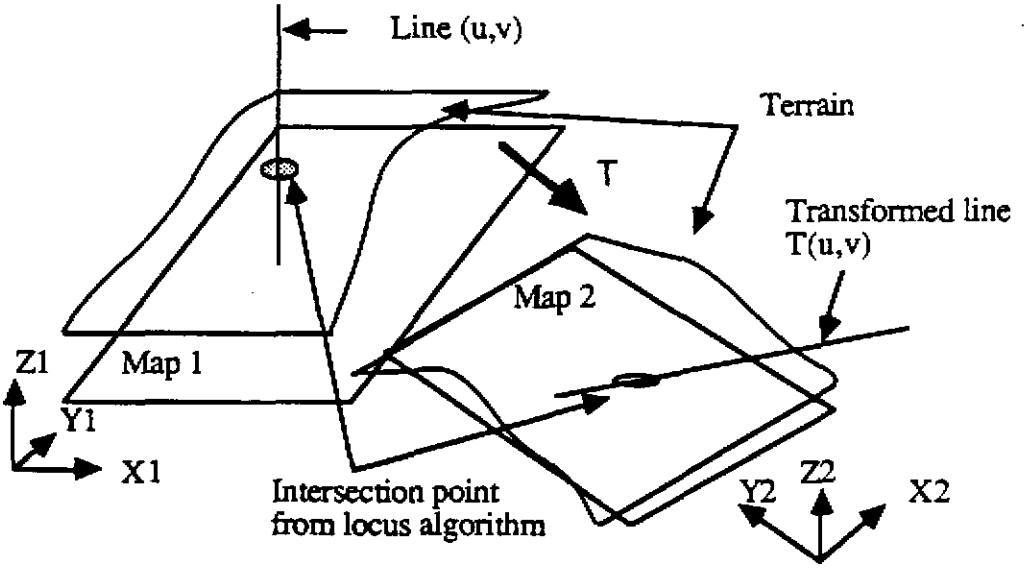


Figure 37: Principle of the iconic matching algorithm

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

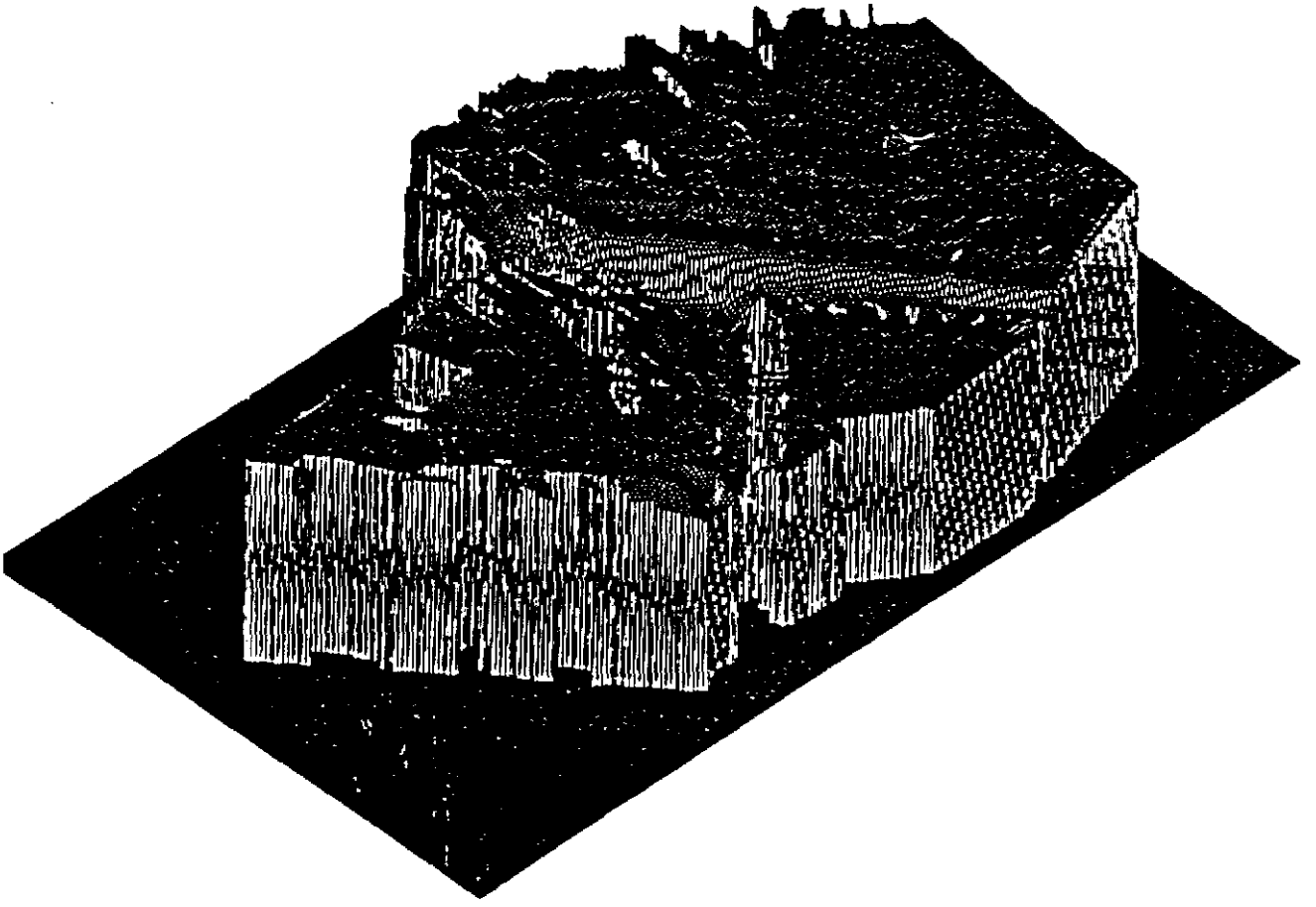


Figure 38: Combining four maps by the iconic matching algorithm

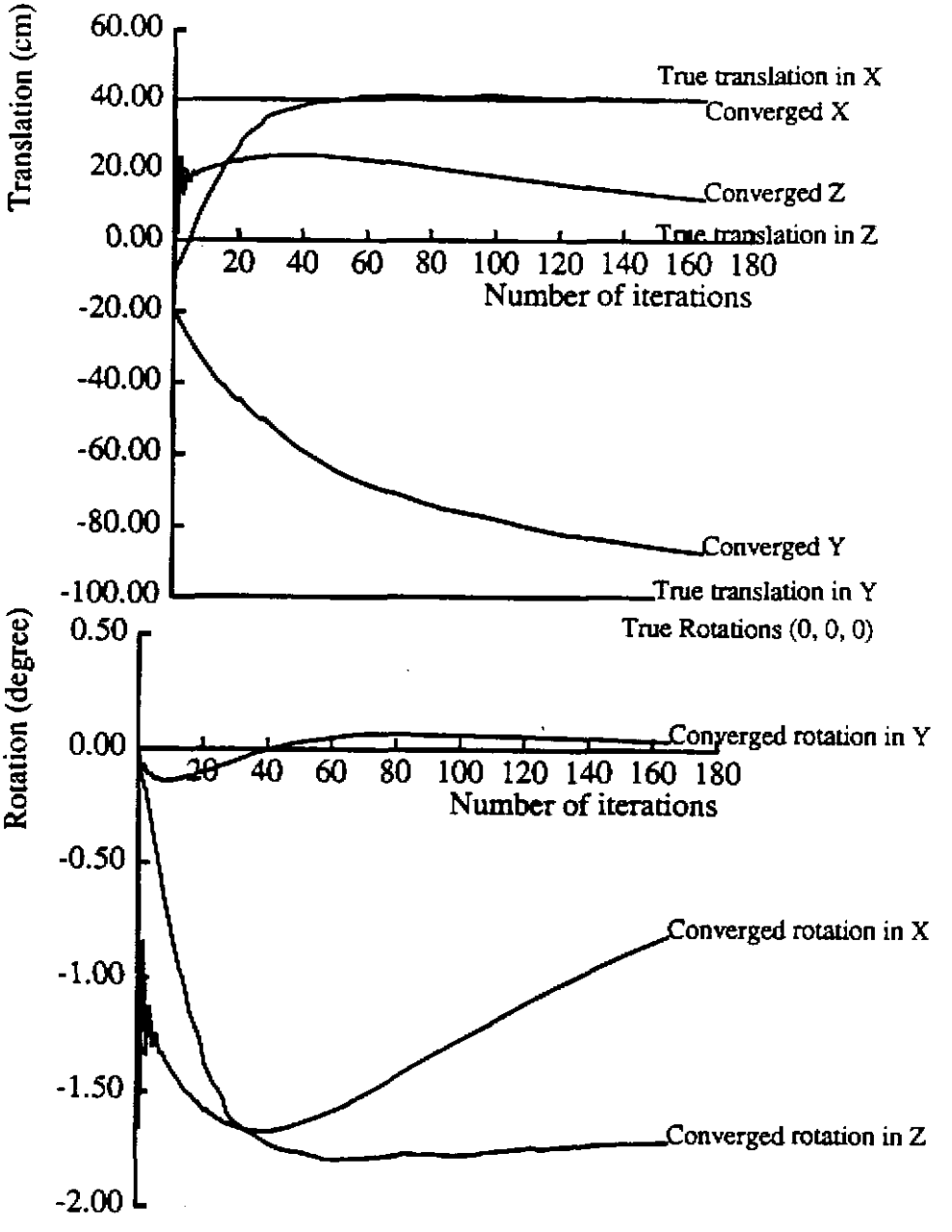


Figure 39: Convergence rate of the matching algorithm

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.

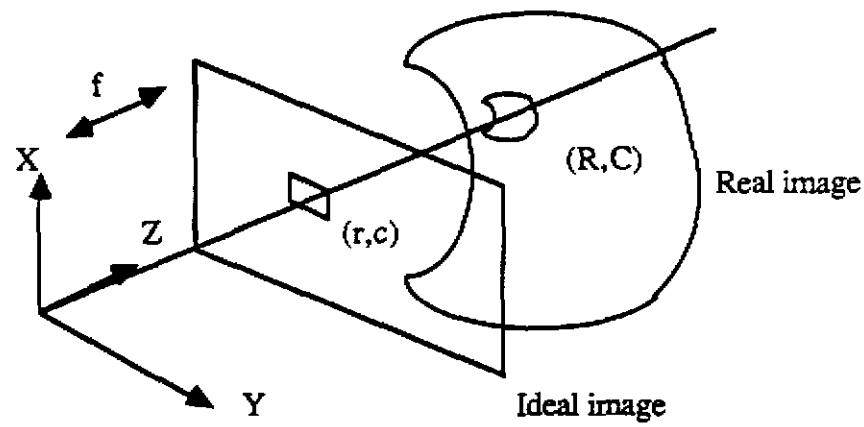


Figure 40: Geometry of the video camera

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.

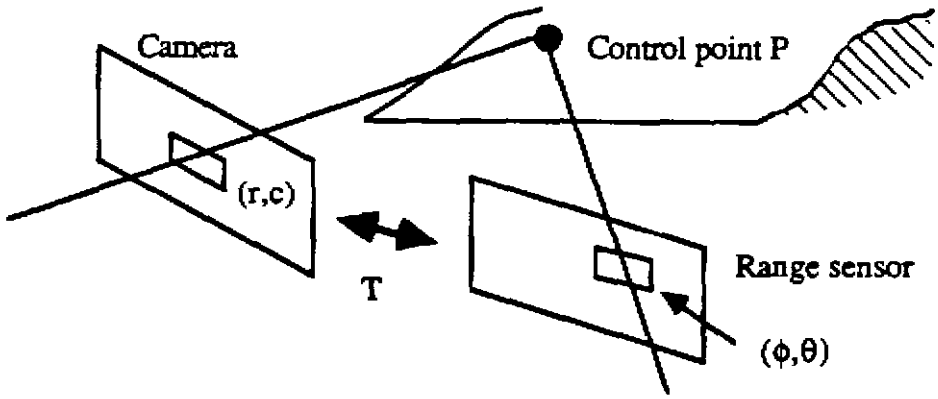


Figure 41: Geometry of the calibration problem

5.2.1 The calibration problem as a minimization problem

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

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

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^s 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:

$$R_x P_i^s r_i - T_z r_i - f R_x P_i^s + T_x' = 0 \tag{41}$$
$$R_z P_i^s c_i - T_z c_i - f R_y P_i^s + T_y' = 0 \tag{42}$$

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 $A_i = R_z P_i^c r_i$, $B_i = r_i$, $C_i = R_x P_i^c$, $D_i = R_z P_i^c c_i$, $E_i = c_i$, and $F_i = R_y P_i^c$ 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 → 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 → 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.

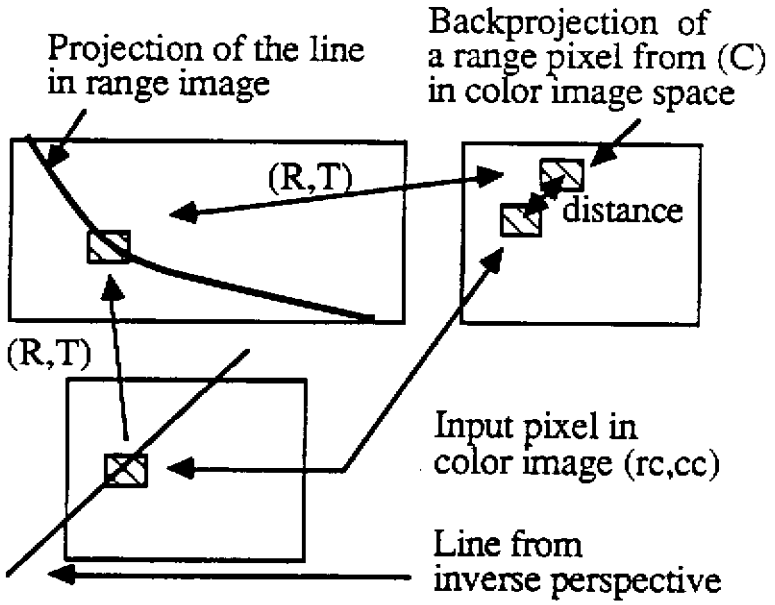


Figure 42: Geometry of the "video → range" transformation

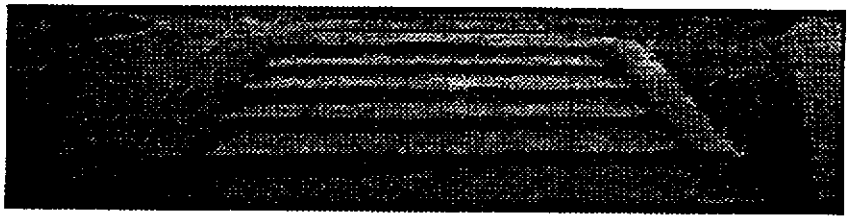


Figure 43: Colored-range image of stairs

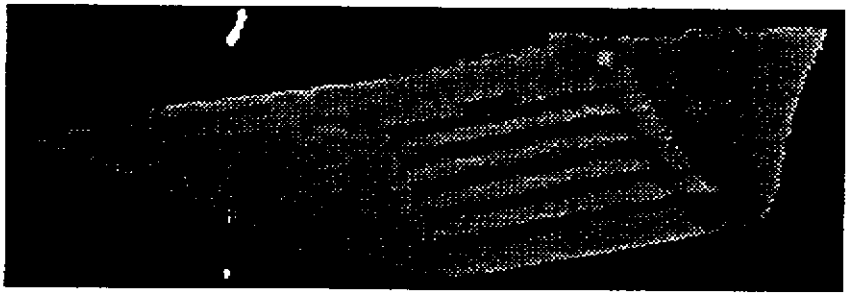


Figure 44: Perspective view of registered range and color images

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,

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

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

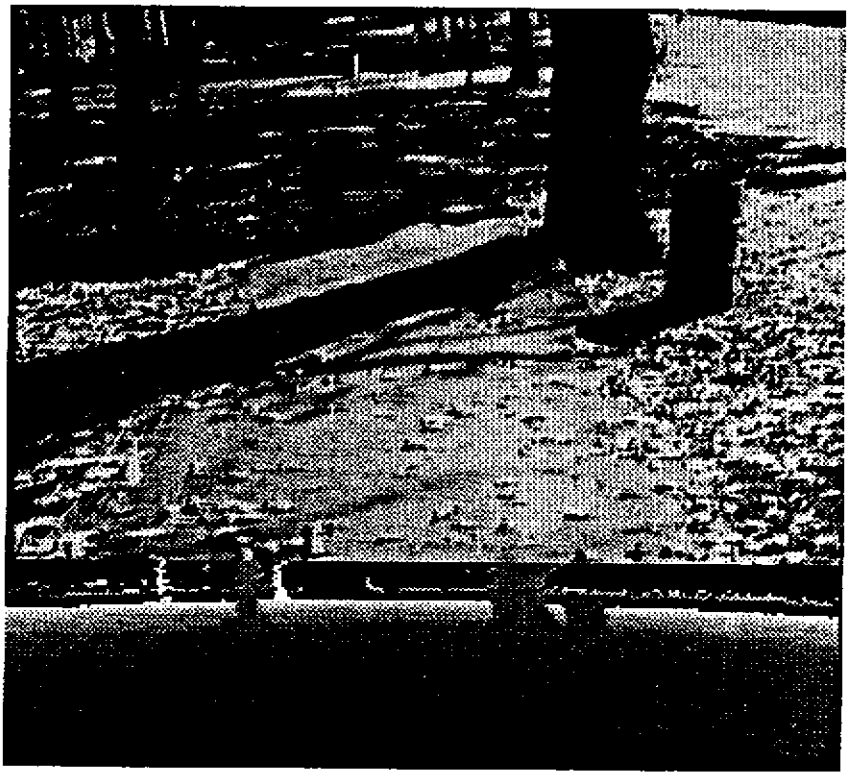


Figure 45: Color and range images of an outdoor scene

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

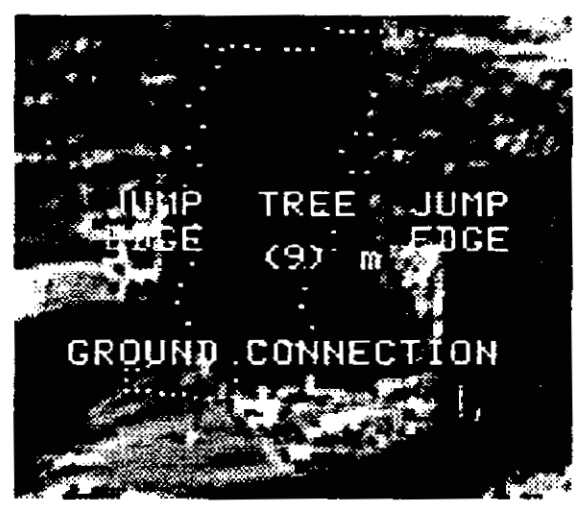


Figure 47: Edge features from the colored-range image

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.