

Toward Autonomous Driving: The CMU Navlab

Part I — Perception

Charles Thorpe, Martial Hebert, Takeo Kanade, and Steven Shafer
Carnegie Mellon University

DESIGNING A ROBOT THAT CAN navigate in a laboratory setting is quite different from designing one that can deal with the real world. The Navlab project at Carnegie Mellon University seeks to build complete autonomous systems capable of outdoor navigation, both on roads and cross-country. Our emphasis on real systems operating in real environments has forced us to move computer vision techniques from controlled environments to the real world. In the process, we quickly discovered that standard perception techniques were not reliable enough. As much as we would have liked to, we could not tolerate any of the simplifying assumptions under which many perception systems are built. In the real world, ambient illumination is unpredictable, objects cast shadows, and the appearance of the world does not remain constant from one vantage point to another. This has led us to the development of new perception techniques that can deal with real-world conditions and imperfect sensors. While the logistical costs of performing such experiments have sometimes been significant, our resulting algorithms and systems are calibrated to reality.

For outdoor navigation, the biggest challenge, and our main area of research, has

been image understanding in difficult conditions. Instead of operating a robot at high speeds over cleanly marked expressways, we have worked on unstructured roads (including dirt roads and a winding asphalt bicycle path), on the changing appearance of structured roads in dappled shadows and at intersections, and on off-road navigation over rough terrain. The perception techniques we have developed for the Navlab include road-following techniques using color classification and neural nets, and three-dimensional terrain modeling. (Our companion article on p. 43 describes complete autonomous systems built around these techniques, as well as the physical configuration of the Navlab itself.)

We have seen significant results in many of these areas. Our Navlab robot van drives

itself at slow speeds along unmarked, unmapped trails, locating and traversing intersections. On more typical structured roads, the Navlab drives up to its mechanical limit of 28 kilometers per hour. It can run without a map or use maps it has built, along with information from previous runs, to select different behaviors at different locations. Off road, the Navlab can move slowly over moderately rough terrain and can map large areas as it drives. The resulting software has been transferred to other projects, including the Martin Marietta Autonomous Land Vehicle (see the sidebar on the ALV and other related work) and our own NASA-sponsored Ambler, a walking machine for planetary exploration (see "Pushing the Envelope," *IEEE Expert*, June 1990, pp. 2-6).

*DESIGNING AN OUTDOOR MOBILE ROBOT THAT FOLLOWS
FLAT, STRAIGHT, WELL-ILLUMINATED, AND CLEARLY
MARKED ROADS IS ONE THING. OPERATING SUCH A
ROBOT IN A REALISTIC ENVIRONMENT WITH BAD
WEATHER, BAD LIGHTING, AND BAD OR CHANGING
ROADS IS ANOTHER.*

Related work

Our work is part of the broader framework of the Defense Advanced Research Projects Agency's Strategic Computing Initiative, including the Autonomous Land Vehicle project that began in 1984. Several of the contractors from the Strategic Computing program's vision component worked on perception and planning for autonomous navigation. Among others, the University of Massachusetts and Honeywell developed motion-tracking software.^{1,2} SRI developed tracking using 3D data,³ and Advanced Decision Systems investigated qualitative navigation.⁴ Martin Marietta built and operated the ALV itself and developed its own road-following software⁵

and obstacle-avoidance system.^{6,7} Hughes (see the article on p. 16) and the University of Maryland⁸ contributed off-road and on-road navigation, respectively, directly to the ALV testbed. Our role was to build a "new-generation" vision system. We were tasked to look beyond the immediate problems of getting the ALV through its first demonstrations, and to address the issues of more difficult perception and integration.

Beyond the DARPA community, the past five years have seen several other outdoor mobile robot projects. In the US, Texas A&M has begun work on visual tracking for convoy-following and obstacle avoidance.⁹ Gen-

eral Motors is working on lane-following at high speeds under relatively constant illumination.^{10,11} The FMC Corporation has built a road-following system based on color thresholding,¹² and the University of Bristol has developed a system that follows roads by tracking lane markings.¹³ In Germany, Dickmanns and Graefe¹⁴ have built an elegant control formulation for driving on expressways. Fujitsu and Nissan in Japan have built prototype road-following software.¹⁵ The research at CMU and within the DARPA community is distinguished from all of these by its concentration on more difficult vision problems in realistic environments.

References

1. P. Anandan, "A Computational Framework and an Algorithm for the Measurement of Visual Motion," *Int'l J. Computer Vision*, Vol. 2, No. 3, 1989, pp. 283-310.
2. B. Bhanu et al., "Qualitative Target Motion Detection and Tracking," *Proc. Image Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1989, pp. 370-398.
3. A. Bobick and R. Bolles, "Representation Space: An Approach to the Integration of Visual Information," *Proc. Image Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1989, pp. 263-272.
4. T. Levitt et al., "Qualitative Navigation," *Proc. Image Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1987.
5. M. Turk et al., "VITS: A Vision System for Autonomous-Land-Vehicle Navigation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 10, No. 3, May 1988, pp. 342-360.
6. R. Dunlay and D. Morgenthaler, "Obstacle Detection and Avoidance from Range Data," *Proc. SPIE Mobile Robots Conf.*, Soc. of Photooptical Instrumentation Engineers, Bellingham, Wash., 1986, pp. 912-917.
7. R. Dunlay, "Obstacle Avoidance Perception Processing for the Autonomous Land Vehicle," *Proc. IEEE Robotics and Automation Conf.*, CS Press, Los Alamitos, Calif., 1988, pp. 912-917.
8. A. Waxman et al., "A Visual Navigation System for the Autonomous Land Vehicle," *IEEE J. Robotics and Automation*, Vol. RA-3, Apr. 1987, pp. 124-141.
9. N. Kehtarnavaz and N. Griswold, "Establishing Collision Zones Under Uncertainty," *Proc. Mobile Robots IV*, Soc. of Photooptical Instrumentation Engineers, Bellingham, Wash., 1989, pp. 66-76.
10. S. Kenue, "Lanelok: Detection of Land Boundaries and Vehicle Tracking Using Image-Processing Techniques, Part I: Hough-Transform, Region-Tracing, and Correlation Algorithms," *Proc. Mobile Robots IV*, Soc. of Photooptical Instrumentation Engineers, Bellingham, Wash., 1989, pp. 221-233.
11. S. Kenue, "Lanelok: Detection of Land Boundaries and Vehicle Tracking Using Image-Processing Techniques, Part II: Template-Matching Algorithms," *Proc. Mobile Robots IV*, Soc. of Photooptical Instrumentation Engineers, Bellingham, Wash., 1989, pp. 234-245.
12. D. Kuan, G. Phipps, and A. Hsueh, "Autonomous Land Vehicle Road Following," *Proc. Int'l Conf. Computer Vision*, IEEE, Piscataway, N.J., 1987.
13. L. Shaaser and B. Thomas, "Finding Road Lane Boundaries for Vision-Guided Vehicle Navigation," *Proc. Roundtable Discussion on Vision-Based Vehicle Guidance*, IEEE, Piscataway, N.J., 1990.
14. B. Mysliwetz and E. Dickmanns, "Distributed Scene Analysis for Autonomous Road Vehicle Guidance," *Proc. SPIE Conference on Mobile Robots*, Soc. of Photooptical Instrumentation Engineers, Bellingham, Wash., 1987.
15. T. Ozaki, M. Ohzora, and K. Kurahashi, "Image Processing System for Autonomous Vehicle," *Proc. Mobile Robots IV*, Soc. of Photooptical Instrumentation Engineers, Bellingham, Wash., 1989, pp. 256-266.

Color vision for road-following

Roads that are nearly straight, evenly illuminated, and well-marked can be tracked easily in color or monochrome video images. Finding the edges of a clean sidewalk or tracking freshly painted white stripes on an empty expressway are both relatively easy vision problems. Following a road becomes much more difficult when the road runs

through dappled shadows, when illumination suddenly changes as the sun goes behind a cloud, or when the "road" is a meandering bicycle path with no lines or stripes and with broken and uneven borders. The challenge for a truly autonomous road-following system is handling a variety of road conditions and changing illumination. To illustrate the problem, our first road-following software ran a simple edge

detector (Roberts' operator, followed by thresholding) over the image and looked for edge fragments that had strong contrast, were parallel, and pointed in roughly the correct direction. This worked well for clearly marked sidewalks. Unfortunately, when we took our robot onto a bicycle path, the highest-contrast edges in the scene were shadow edges.

Navlab test sites include a variety of

road conditions, from dirt roads to freeways. A single perception system cannot address all possible configurations, so our approach is to build different systems for roads that are structured (such as highways and city streets) and unstructured (such as dirt roads). This lets us take advantage of road structures when they are available, while retaining the ability to deal with unstructured roads when needed.

One system, SCARF (for "supervised classification applied to road following"), deals with unstructured roads using adaptive color classification. It deals with changing illumination and road appearance by updating its color models for each new image. It handles poorly defined roads by classifying all the pixels in the image and applying a voting scheme to a simple road model to find the most probable road in the image.

YARF ("yet another road follower"), our second vision system, deals with structured roads. It takes advantage of the lines and stripes of structured roads and uses an explicit model of those features to guide individual trackers and to filter and validate its detected road model.

Other than when bootstrapping the system, SCARF and YARF require only video images as input. But if we could manually train a system on a portion of the road to be followed, that system should be faster and more robust, since more prior information on road appearance and geometry would be available. To investigate this idea, we have built ALVINN ("autonomous land vehicle in a neural net"), the Navlab's third main color vision system, which uses a connectionist architecture. It achieves its power by being trained directly on the current road and by processing quickly so that small imperfections tend to be smoothed out.

SCARF. Mobile systems have used three approaches in following unstructured roads: edge extraction, thresholding, and classification. Systems that use edge extraction apply gradient operators to the image of the road. The system assumes that strong edges correspond to road edges, and it groups them to yield road geometry. Edge-based systems can be very fast and can work well on clearly delineated roads with no shadows. As soon as strong shadows appear, however, they break down rapidly because strong edges now correspond to shadow edges.

Systems that use thresholding use some combination of the color bands — such as red-blue — and threshold the resulting image.¹ These systems label as road all pixels with similar intensities, but when shadows are present, shaded road and shaded off-road often have very similar features, thus confusing the classification.


SCARF uses adaptive color classification to avoid the shortfalls of previous systems.² SCARF runs in a loop: classify image pixels, find the road model that best matches the classified data, update the col-

pixel belongs to a particular class and says intuitively whether a particular color variation is significant. Since sunlit asphalt tends to be homogeneously colored, it is represented by a class with small variance; grass has more variety and is represented with correspondingly larger variances.

Classified pixels vote for all road locations that would contain them, with votes weighted by classification confidence. The road with the most votes is used both for steering and for recalculating the color classes using nearest-mean clustering to collect new road and off-road color statistics. SCARF's simple model of road geometry represents roads as triangles in the image. The apex is constrained to lie on a particular image row, corresponding to the horizon, and the base has a fixed width, dependent on road width and camera calibration. There are two free parameters: the column in which the apex appears and the skew of the triangle. While this simple two-parameter model does not represent curves, hills, or road-width variations, it does approximate the road shape well enough to allow reliable driving. It is especially effective because the voting procedure uses all pixels, not just those on the edges, and therefore is relatively insensitive to misclassifications. Furthermore, the simple model allows for fast voting, and it functions well with small amounts of data, so SCARF can process highly reduced images (typically 60x64 or 30x32 pixels) at high sampling rates (approximately two seconds per frame). By processing images that are closely spaced along the road, SCARF can correct small errors in road representations before the vehicle arrives at the mistaken locations. By processing images that are closely spaced in time, SCARF perceives even the drastic illumination changes caused by clouds covering the sun as gradual shifts in road appearance.

The basic SCARF system runs on Sun workstations and was demonstrated on a number of roads. SCARF has driven the Navlab along bicycle paths, dirt roads, gravel roads, and suburban streets, and it has been integrated into several of our Navlab systems. Figure 1 shows SCARF correctly finding a road, even through deep shadows where the road is not obvious even to a human observer.

We have built several extensions of SCARF. The first uses parallel hardware



MOBILE SYSTEMS HAVE USED THREE APPROACHES IN FOLLOWING UNSTRUCTURED ROADS: EDGE EXTRACTION, THRESHOLDING, AND CLASSIFICATION.

or models for classification, classify image pixels, and so on. The simple models of road color and geometry make very few assumptions about the road and let SCARF run robustly, even when following unstructured roads.

SCARF's first strength comes from representing multiple color classes as Gaussian distributions in full red-green-blue color and from calculating probabilities instead of using binary thresholds. SCARF typically uses four color classes to describe road appearance, and four to describe off-road objects. During classification, SCARF compares each pixel to all eight classes. The result is both the label of the most probable class and its probability. Multiple classes let SCARF represent the different colors of the road (such as asphalt, wet patches, shadowed pavement, and leaves) and off-road objects (such as trees, sunlit grass, shaded grass, and leaves). Using full color, instead of monochrome images or some other combination of colors, keeps all the image information that might be useful in discrimination. The Gaussian representation of each color class lets SCARF calculate the likelihood that a

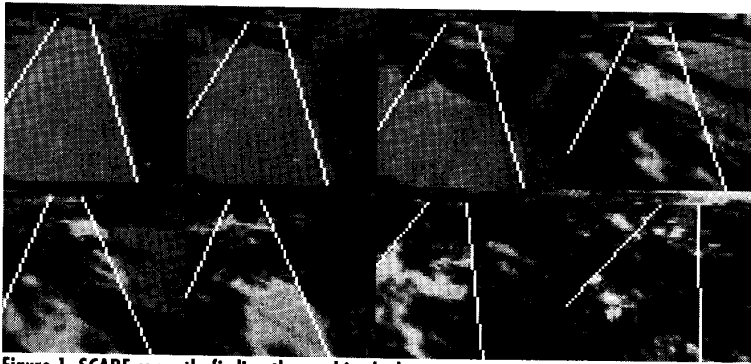


Figure 1. SCARF correctly finding the road in shadows.

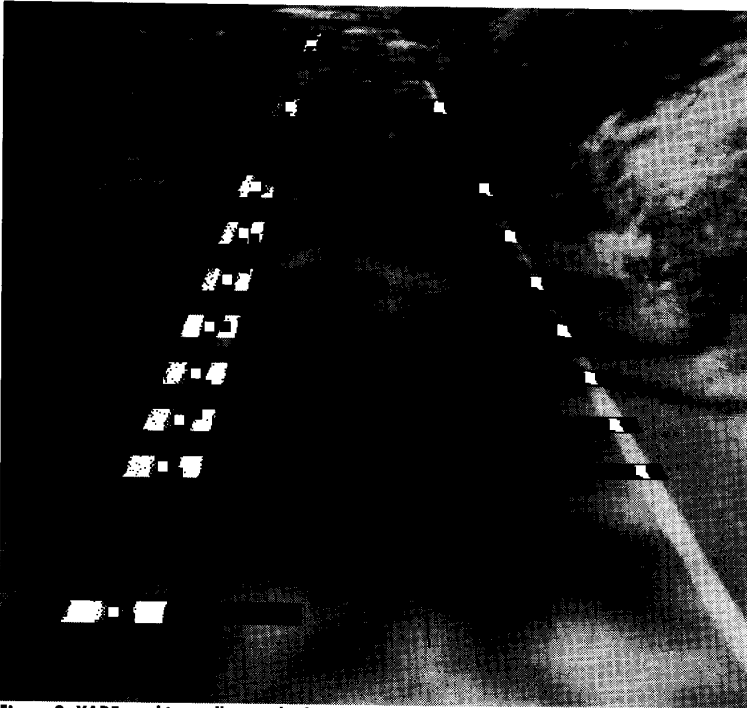


Figure 2. YARF tracking yellow and white lines in shadows.

instead of conventional workstations to improve performance. Using the Warp computer, a 10-cell systolic array, SCARF is parallelized by dividing the image into strips and by processing each strip on a separate cell. The second extension adds to the road model by checking for intersections as well as for the main road.

YARF. Following roads in urban environments requires specific techniques to take advantage of prior knowledge of the

environment. The Vamors system combines specialized hardware with a control formulation of the problem to achieve runs at up to 96 kilometers per hour. Vamors uses simple feature trackers to track the position of the road's center line and side markings. However, it can be sensitive to changes in illumination, shadows, and changes in road structure such as intersections. The Lanelok system can use three different types of image operators to track road edges: Sobel edge detection followed

by Hough transform, region extraction, and template matching. Lanelok has been demonstrated off line on thousands of images. The University of Bristol's system extracts lane markings as regions of the image that are brighter than a given threshold and limited by edges of appropriate geometry. A circular arc is fitted to the regions after back projection on the ground plane.

By using a single segmentation technique to locate the road, each of these systems lacks a recovery mechanism if its particular segmentation technique fails. To address this problem, we designed YARF to explicitly model as many aspects of road-following as possible.^{3,4} Highways, freeways, rural roads, even suburban streets have strong constraints and easily identifiable features. For instance, the road's center line is yellow and has a constant known width, and its curvature is lower than a known threshold. YARF explicitly models each constraint and feature. From feature models we build specialized image operators that find features such as road markings. From constraint models, we build specialized trackers that apply the image operators to long sequences of images, predicting a feature's location in an image from its location in the previous image. A tracker for the center line finds a yellow stripe in a small window in a color image and predicts its location in the next image using a model of road geometry and current vehicle motion. Constraint models also provide a way to detect and recover from errors in feature detection. This makes reasoning easier and more reliable. In addition to geometric constraints, YARF uses an explicit model of the noise of feature detectors and vehicle position to yield optimal performance.

YARF has four main components: feature trackers, geometric modeling, error detection and recovery, and noise modeling.

Specialized feature trackers. YARF has individual operators that know how to model and track specific features, such as edge markings (white stripes), center lines (yellow stripes), and shoulders. YARF also uses an explicit geometry model of the road, consisting of the vehicle's location on the road, the location of stripes, the type of stripes (such as broken or solid), and the maximum and current road curvature. Other features, which are not yet modeled but might be helpful, include the locations

of shadows, 3D effects as the road goes through valleys and over hills, and global illumination changes.

The yellow-line tracker, for instance, uses the hue of the lines for segmentation. The hue is calculated for all pixels in a window around the predicted line location. Pixels with a hue between 40 (reddish-yellow) and 100 (greenish-yellow) are set to 1, others to 0. After filtering the thresholded image using a "shrink and grow" operator, the resulting image is normally dominated by one or two blobs, corresponding to the yellow line or lines. The blob descriptors are considered the line locations. Figure 2 shows the yellow-line tracker and a separate white-line tracker, finding road lines even in complex shadows.

Road geometry. YARF is designed for higher speeds than SCARF, and it runs in a more predictable environment. This requires and allows a more complex road model, one that encodes curvature as well as position. YARF models the road as a generalized stripe, that is, as a one-dimensional feature that is swept perpendicular to a spine curve. The spine is modeled locally by a circular arc, assuming that the road lies on flat ground. We find the spine's equation by fitting a circular arc to the detected features. Since the equation of a circle

$$\text{radius}^2 = (x - x_{\text{center}})^2 + (y - y_{\text{center}})^2$$

is nonlinear and sensitive to noise, we can approximate a circular arc with a parabola, similar to Mysliwetz and Dickmanns' approach:

$$x = \text{half_curvature} \times y^2 + \text{slope} \times y + \text{offset}$$

The least-squares values of curvature, slope, and lateral offset are easily computed using the matrix pseudoinverse. In practice, this approximation is adequate for the sorts of curvatures and slopes of roads within the Navlab's field of view. To improve the estimation's stability, we fit the curvature model to features detected over a few frames.

Error detection and recovery. Occasionally, the system finds features incorrectly. YARF detects these mistakes both locally — based on the results of a single operator — and globally — when checking for consistency. Local error detection depends on the specific operator. For example, the blob

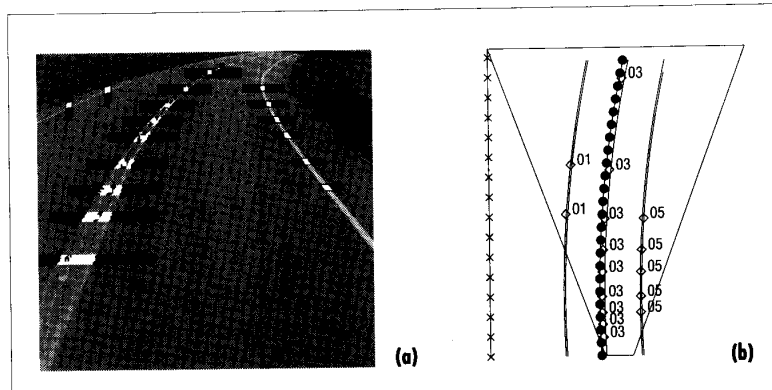


Figure 3. YARF tracking result: (a) extracted center and edge features; (b) road model derived from features extracted above. Diamonds show detected feature locations, solid circles show derived center line.

detector usually finds a light blob (the white line) against a dark background (asphalt). It maintains statistics of the mean, variance, and covariances of red, green, and blue for both feature and background colors. If all pixels in its prediction window are the background color, the color blob detector reports a missing feature. If a light blob is found, but only at the edge of the window, the detector reports a clipped feature. If all pixels are much lighter or darker than modeled by either color, it reports an illumination shift. It is up to the higher-level calling program to decide whether the road has widened, the white stripe is temporarily missing, or the lighting really has changed. Error detection and recovery is a critical component of YARF. It allows for robust navigation in the presence of changing illumination, shadows, and noisy road features while using fast and simple specialized trackers.

Noise modeling. We have achieved good results by fitting curves to the points detected over the three most current frames, with no error weighting or filtering. For instance, Figure 3a shows features (center line and edge markings) extracted from a road scene, and Figure 3b shows the road that the road model fit to the features using straight least-squares. The diamonds in Figure 3b show tracked lane lines on the left and right, and the solid circles show the estimated road position and heading. For relatively slow speeds (up to about 15 kilometers per hour), the errors in detected positions are probably dominated by image-processing noise rather than vehicle motion noise. Future runs, at higher speeds, will probably require more elaborate filtering

schemes. We are working on two approaches: Kalman filters, which compute optimal estimates of states (such as the positions) by combining measurements and their associated uncertainty models, and robust statistics, which address the problem of removing outliers that cannot be handled by linear filtering techniques.

Performance. YARF has driven the Navlab at speeds of up to 15 miles per hour on a public road. The environment in the experiments contained large shadows generated by surrounding trees. It also included relatively high-curvature curves, thus demonstrating the performance of the road model. YARF works because of the integration of all its modeled constraints: Explicitly modeling tracker performance and feature appearance, and using specialized trackers, allows high speeds, high accuracy, and local failure detection; explicitly modeling road geometry allows accurate predictions and enables global error detection; explicitly modeling errors and uncertainty lets YARF correctly size its prediction windows; explicitly modeling changes in road geometry lets YARF handle urban streets with intersections and other discontinuities in lane structure and appearance, as well as the cleaner environment of highway lanes.

ALVINN. Built by CMU's connectionist group, ALVINN is trained by having a human driver steer it over a portion of the road to be followed.⁵ The system inputs the camera image and the steering angle at each time step. The image is preprocessed to enhance road contrast. The enhanced image and the steering angle are fed to a

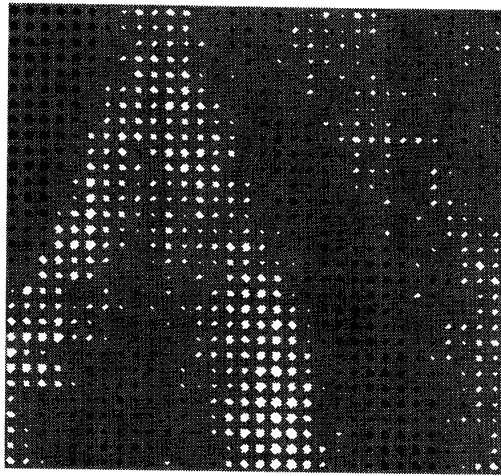


Figure 4. ALVINN weights for one hidden unit. Input weights are at the bottom, output weights are at the top. Positive weights are white, negative are black.

back-propagation algorithm that adjusts weights in the hidden units of ALVINN's neural network until the weights settle to values that give the correct steering response for each input image. Typically, training requires less than 100 input images and takes less than five minutes.

This training scheme lets ALVINN directly learn how to follow roads. It is more difficult to train the network to recover from errors, such as when it is not quite aligned on the road. To provide examples of images from slightly different vantage points, and the proper steering commands, each input image is reused in several positions. The images are shifted to simulate a variety of errors, and the steering command is shifted to generate the command that would bring the Navlab back onto the road.

When ALVINN runs, it preprocesses the input images, gives them to the neural net, and directly outputs the steering-wheel angles as dictated by the network, with no reasoning about road location. ALVINN uses reduced-resolution images (typically 30×32 or 45×48 pixels) and runs at about a fifteenth of a second per image.

ALVINN uses a compiled representation, going straight from images to steering with no intermediate geometric or symbolic representation. During its learning phase, the back-propagation algorithm automatically compiles this knowledge by selecting

the features that discriminate between different steering angles, which correspond to different road locations. Since ALVINN starts with no preconceived idea of what the road looks like, it learns different sets of weights to follow many different types of roads with no change in the underlying algorithms.

The disadvantage of such a compiled representation is that it cannot take advantage of geometric or symbolic input. If ALVINN is trained to run on a particular road, we cannot tell it that a second road is just like the first, only twice as wide. The advantage is that it is fast and easy to train for a particular road. The weights learned by ALVINN tend to be large, low-frequency edge masks, or matched filters that look for the road in general locations. Thus, local imperfections in the road or in lighting do not greatly distort the output steering direction. Figure 4 shows the weights for one of ALVINN's hidden units. The square shows the weights coming into one hidden unit from the input image, and the line at the top shows the weights going out to different steering angles. White indicates positive weights, and dark indicates negative weights. This unit mainly looks for a road on the left edge of the image, and mainly votes for turning left. There is also a secondary pattern that would match a road further to the right, and there are

slightly positive weights supporting straight-ahead steering.

3D perception

An outdoor mobile robot needs information derived from appearance (such as road location in a color image, or terrain type), but it also needs to know the geometry of the observed environment. In some tasks, such as cross-country navigation, the most important information is the geometry of the terrain, the set of 3D surfaces observed or traversed by the vehicle. The first step toward building geometric representations of a terrain is choosing a suitable sensor. Clearly, one color camera is not enough to collect 3D data. An alternative is to use passive techniques for recovering 3D data, such as stereo vision, but these techniques have significant drawbacks, including high computational demand, difficulty in ranging bland surfaces, and reliance on ambient lighting. Instead, we use an active sensor: a laser range scanner that can generate a high-resolution depth image of the terrain in front of the vehicle. Such a sensor alleviates the need to infer 3D information from 2D information and, since it is active, it is also less sensitive to outside illumination.

The terrain can be described at different levels of resolution depending on the task, the environment, and the amount of computation time allocated in the system for 3D perception. For example, a system that follows roads known to be locally flat and mostly obstacle free requires a completely different representation than a system that navigates through rugged open terrain. In the latter case, vehicle safety is the overwhelming issue, and vehicle speed is much less important. This is consistent with the general approach that a mobile robot's components must be tailored to its environment and task. Here, we discuss three types of representation: obstacle maps, terrain feature maps, and high-resolution maps.

Range sensing. The Navlab uses the Environmental Research Institute of Michigan (ERIM) scanner, which is typical of laser range finders that measure the phase shift of an amplitude-modulated laser.⁶ Table 1 lists characteristics of the ERIM, one of the earliest scanners, and the Perceptron, a later model.

A laser range finder has considerable

Table 1. Relative performance of example range scanners.

	ERIM	PERCEPTRON
EYE SAFE	Yes (?)	Yes
FIELD OF VIEW	80x30 degrees	60x60 degrees (programmable tilt)
PIXELS	256x64	256x256
AMBIGUITY INTERVAL	20 meters	40 meters
DEPTH	8 bits (8 cm)	12 bits (1 cm)
INTENSITY	8 bits	8 bits
MAX RANGE	40 meters (?)	50 meters
SCAN RATE	2 frames/sec	2 frames/sec
SCAN DIRECTION	Top to bottom	Programmable
INTERFACE	VME to Sun	VME to Sun
TEMPERATURE	Narrow range	"Pittsburgh"
CONSTRUCTION	Wire wrap	Printed circuit
COMPONENTS	All custom	Most off the shelf
SIZE	90x35x45 cm	45x35x35 cm
WEIGHT	50 kg	< 25 kg
POWER	26 VDC	110 VAC Figure 8. Four levels of the terrain quadtree.

advantages over more traditional techniques such as stereo vision: It is insensitive to outside illumination, it is fast compared to standard passive techniques, and it provides a high-resolution range map instead of the sparse map produced by most passive techniques. Laser range finders also have limitations, however. For one, mixed measurements are generated at the boundary between two objects separated by a large distance. Also, surfaces with different reflectivity properties can lead to unexpected variations in measured range values. The main limitation is the acquisition rate, which is too slow for high-speed highway driving, for example.

Discrete objects and obstacle detection. The lowest-resolution terrain representation is an object map containing a small number of objects represented by their trace on the ground plane. Several techniques have been proposed for obstacle detection.

The Martin Marietta ALV detects obstacles by computing the difference between the observed range image and precomputed images of ideal ground planes at several different slope angles. The system groups points that are far from the ideal ground planes into regions and reports them as obstacles to the path planner. A very fast implementation of this technique is possible, since it requires only image differences and region grouping. However, it makes strong assumptions about the shape of the terrain. Specifically, it restricts the terrain shape to a few admissible slopes and elevations. The technique also considers only the absolute positions of the potential obstacle points, not relative positions and slopes. As a result, the system would overlook a short, sharp ridge or step, even though it might be an obstacle.

Another approach, proposed by the Hughes AI group, is to detect 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 discontinuities in range, while the second detects the portion of the terrain with a high slope. This approach has the advantage of taking a vehicle model into account when deciding whether a point is part of an obstacle.

The Navlab uses an elevation map approach to detect obstacles. Each cell of the terrain contains the set of data points that fall within its field. We can estimate the

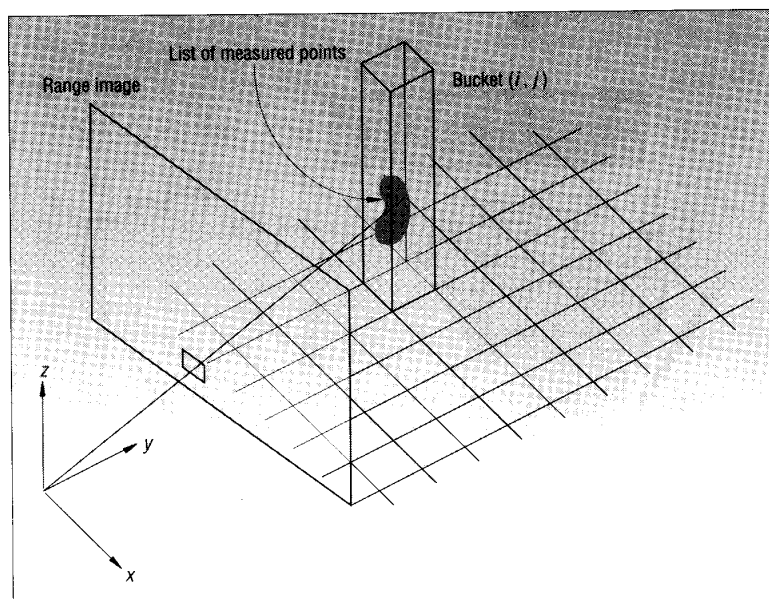


Figure 5. Building the obstacle map. 3D data points are projected into discrete buckets on a horizontal grid.

surface normal (unit vector perpendicular to the surface) at each cell of the elevation map by fitting a reference surface to the corresponding set of data points. When the system finds cells whose surface normal is

far from the vehicle's idea of the vertical direction, it reports them as part of its projection of an obstacle (Figure 5) and then groups them into regions corresponding to individual obstacles.

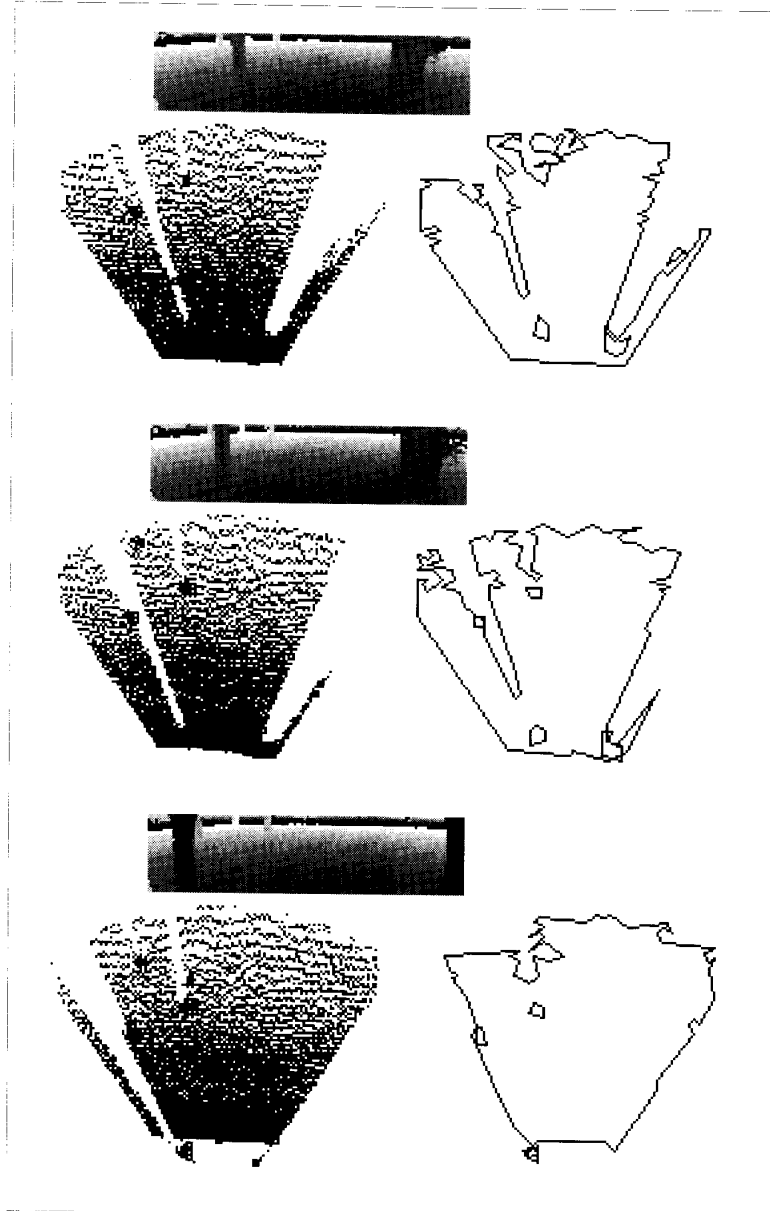


Figure 6. Obstacle detection on a sequence of images. In each group, the top is the original range image, the left is the overhead view, and the right is the segmented elevation map.

Figure 6 shows the result of applying the obstacle-detection algorithm to a sequence of ERIM images. The top of each group shows the original range images, the left shows the range pixels projected in the elevation map, and the right shows the resulting polygonal obstacle map. The large enclosing polygon in the obstacle map is the limit of the visible portion of the world.

In fast obstacle-detection mode, several

improvements can decrease the computation time:

- We can look for obstacles only in a narrow stripe in front of the vehicle.
- We do not need to detect all objects; it is sufficient to raise an alarm as soon as one object is found.
- We do not need high spatial resolution at close range, so the data can be subsampled close to the vehicle.

Taking these improvements into account, we achieved obstacle detection in 600 milliseconds on a Sparc workstation. This is fast enough for now, since acquisition time still averages 500 milliseconds.

Another application of object detection is to build object maps by combining observations, which is critical to improving object localization and to removing spurious objects. Matching objects is not very expensive in our case, since we have only a few objects to match in each frame and since we can assume that we have a reasonable estimate of the displacement between frames from the inertial navigation system or from dead-reckoning, so that the locations of objects detected in one image can be easily predicted in the next image. The main issues are removing spurious objects — which might be due to noise in the range image or to moving objects (such as people) crossing the field of view — and computing the location of the objects as accurately as possible so that position corrections computed using the object map are also accurate.

We deal with spurious objects by calculating a confidence measure for each object. Once an object has been seen in one image, it should appear in subsequent images, as predicted by vehicle motion, object position, and sensor field of view. If it appears as predicted, its confidence is increased; otherwise its confidence decreases. Objects with low confidence are discarded. The system obtains accurate object locations by updating the location's uncertainty (mean and covariance matrix) each time the object is observed in a new image. The uncertainties are combined using standard maximum-likelihood techniques. Figure 7 shows a sequence of six images, collected at intervals of about 50 centimeters. The white lines connect objects that match between images. The white dots indicate the locations of the detected objects.

Terrain modeling for cross-country navigation. Obstacle detection is sufficient for navigating in flat terrain with discrete obstacles, such as a road bordered by trees. We need a more detailed description when the terrain is uneven. For that purpose, a path planner could use an elevation map directly. This approach is costly because of the amount of data to be handled by the planner, which usually does not

need such a high-resolution description. For example, the planner should not need to scan a full elevation map if the terrain is completely flat: the terrain representation should provide enough information to quickly determine that no search is needed. An alternative to elevation maps is grouping smooth portions of the terrain into regions and edges — the basic units manipulated by the planner. This set of features provides a compact terrain representation, but the planner might still need information at a higher resolution. For example, in cluttered environments the planner has to examine small portions of the terrain to decide which areas are traversable.⁷ Therefore, a compromise representation should include both high-resolution elevation data and feature information and should allow for efficient access to large chunks of terrain.

We can implement such a compromise by organizing elevation and feature maps in a quadtree structure. Each node of the tree contains information that describes the portion of the terrain covered by the corresponding quadrant: minimum and maximum elevation, maximum slope, average elevation, and maximum discontinuity within the quadrant. Discontinuities and slopes are computed by applying a gradient operator to the elevation map. This representation saves a considerable amount of computation time both in building the terrain representation and in using it for path planning: We can now build a complete terrain representation in two seconds on a Sparc workstation. Getting information about a patch of terrain using the terrain pyramid is more efficient than using the raw elevation map, in the same way that representing an object with a quadtree is more efficient than representing it with pixels. For a small initial cost, the amount of duplicated computational effort needed by the cross-country planner is vastly reduced. Figure 8 shows several levels of the quadtree representation built from the range image of Figure 9.

Because of the sensor's limited field of view, the terrain model derived from a single range image might not be sufficient. In our case, the map is accurate enough up to six meters in front of the vehicle. However, we can fuse quadtree representations from consecutive images to yield a larger model of the terrain. In the current Navlab configuration, we use the inertial navigation system's readings to register images in

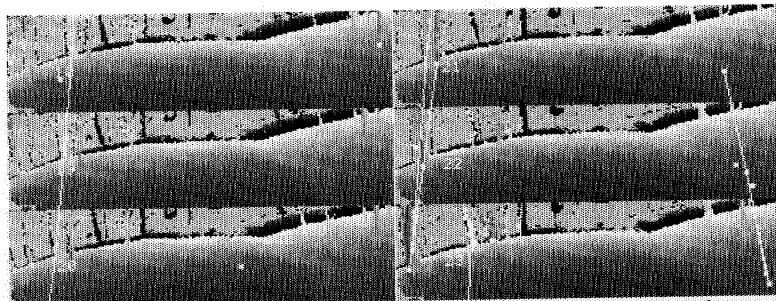


Figure 7. Matching objects in a sequence of range images. White lines show corresponding objects in sequential images.

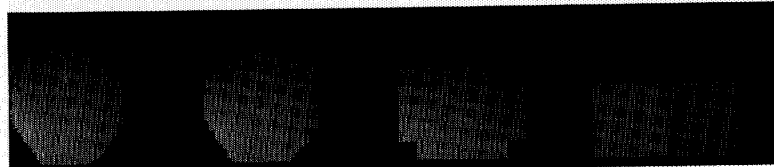


Figure 8. Four levels of the terrain quadtree.

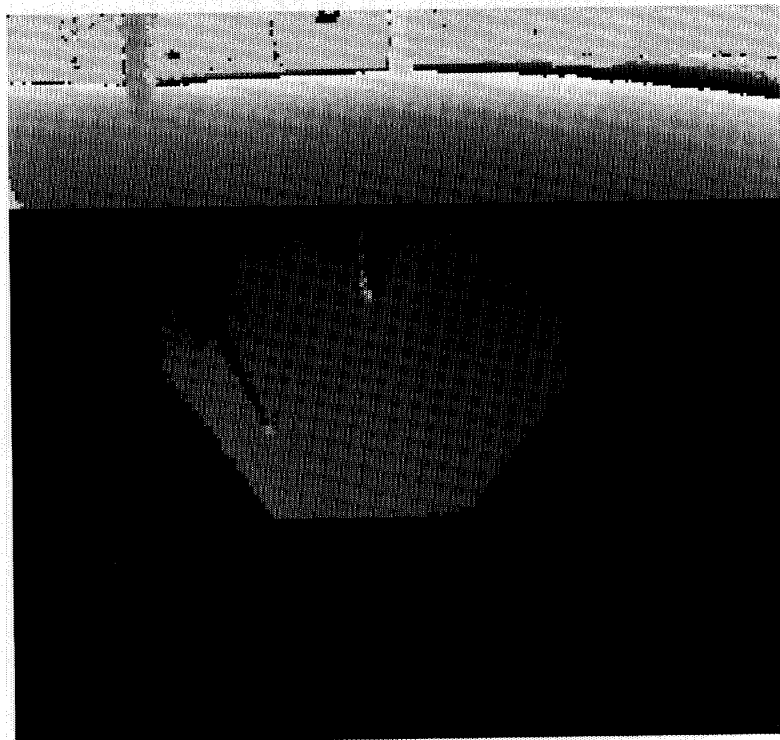


Figure 9. A range image and corresponding elevation map.

x and y axes, plus roll, pitch, and yaw. We achieve registration in the z axis by calculating the z offset between maps as the mean difference of z values.

Map building from terrain features. We can also build composite maps from terrain descriptions. The basic problem is matching terrain features between successive

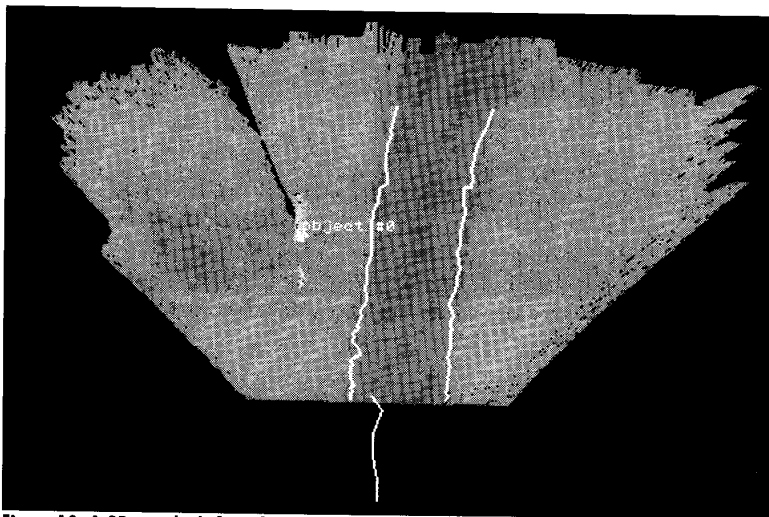


Figure 10. A 3D map built from five range images.

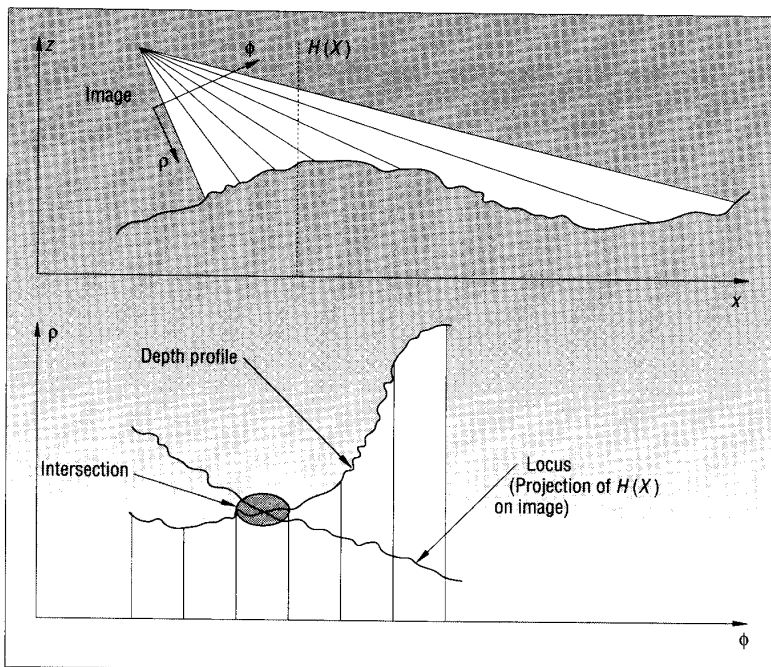


Figure 11. The locus method. The top graph shows the intersection of a surface with a vertical line in world space. The bottom graph shows the same intersection in image space.

images and computing the transformation between features. In this case, the features are the regions that describe the terrain, parameterized by their areas, the equation of the underlying surface, and the center and the main directions of the region. If objects are detected, they are used in the matching in the same way as before.

Finally, if the vehicle is traveling on a road, the edges of the road can also be used in the matching. As in object matching, the system uses an initial estimate of the displacement between successive frames to predict the matching features. A search procedure then finds the most consistent set of matches. As before, the search is

very fast due to the small number of features and the fact that the initial guess of the transformation between images is usually quite close to the actual value. The features are weighted in the search according to how reliably they can be detected. A feature's reliability depends on its type; discrete objects are more reliable than terrain regions and road edges. Once a set of consistent matches is found, the transformation between frames is recomputed and the common features are merged.

This map-building approach has been tested on sequences of images with errors in position estimation of up to one meter in translation and 20 degrees in rotation. Figure 10 shows a composite map in which a sequence of five maps is merged using feature matching.

High-resolution terrain models. The high-resolution terrain representation is an elevation map, that is, a function $z = f(x, y)$ represented by a regular grid of values (x_i, y_i) . The most straightforward way to convert a range image to an elevation map would be to map each pixel (row, column, range) of the range image to an (x, y, z) location in map coordinates. There are several problems with this approach:

- Since this approach is similar to image warping, the distribution of data points in the elevation map is not uniform. The map gets sparser farther from the sensor.
- Objects create range shadows — regions of space that are not visible, even though they lie within the sensor's field of view. Shadow regions must be explicitly identified and represented separately, since no information is available in those regions.
- Range measurements are corrupted by noise due to electronic noise, surface material, laser footprint, and so on. Range noise translates into uncertainty on the elevation map and must be explicitly represented.

These problems could be solved by applying a standard interpolation technique to the sparse elevation map, providing a dense elevation that is a reasonable interpolation of the sparse input. However, such a technique would not account for the geometry of the sensor, making it difficult to identify shadows or to convert sensor uncertainty to map uncertainty.

The locus algorithm overcomes many of these problems by explicitly accounting for

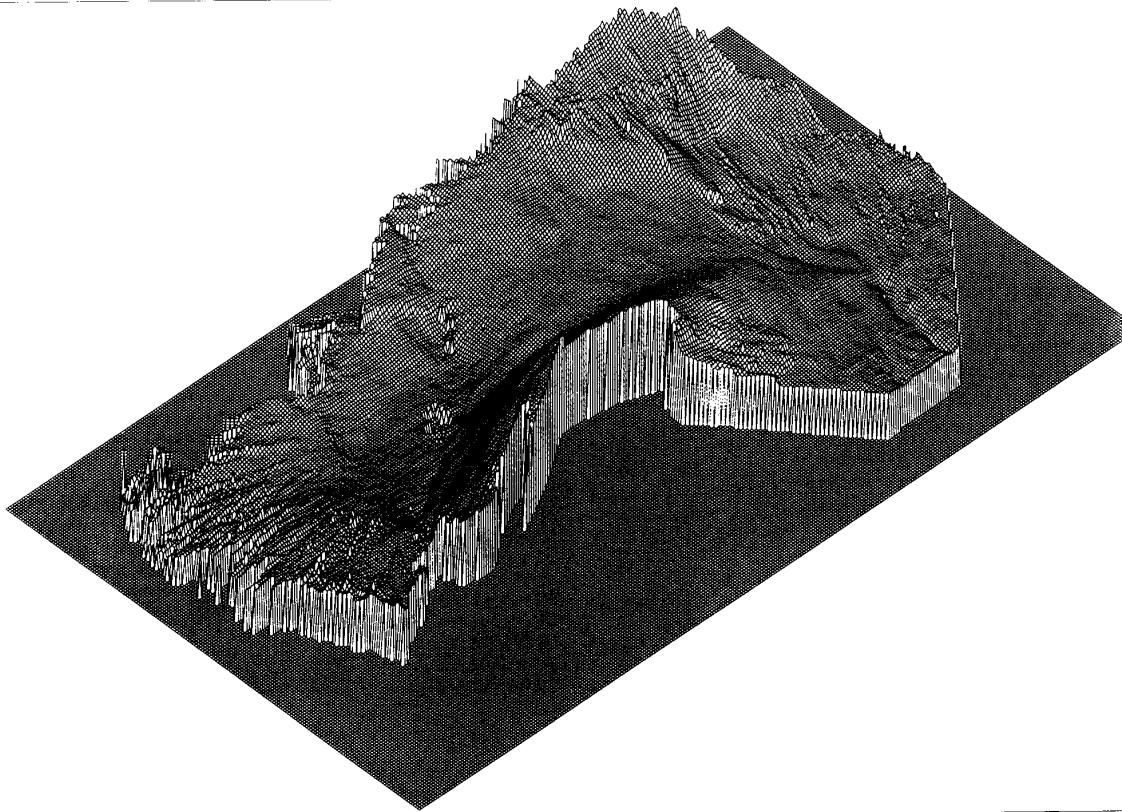


Figure 12. An elevation map built by the locus algorithm from 122 range images, covering 250 meters.

the sensor geometry in building a dense elevation map. The idea is illustrated in Figure 11: Finding the elevation z of a point (x, y) is equivalent to computing the intersection of the surface observed by the sensor with a vertical line passing through (x, y) . Knowing the geometry of the sensor, we can represent the line in image space by an analytical equation of the form $range = f(r, c)$, where r and c are the row and column coordinates in the image. (The projection of this line into the image defines a locus of points that gives the algorithm its name.) The intersection between the line and the observed surface is found between two adjacent pixels (r_1, c_1) and (r_2, c_2) , such that $range_1 < f(r_1, c_1)$ and $range_2 > f(r_2, c_2)$, where $range_1$ and $range_2$ are the values in the image. We obtain the final value of z by interpolating the range between (r_1, c_1) and (r_2, c_2) .

The key point of the locus algorithm is that the interpolation takes place in the image instead of in the map. This lets us explicitly account for the sensor model:

The uncertainty on z is computed by combining the known uncertainties at (r_1, c_1) and (r_2, c_2) . We can detect the unknown regions in the map by observing that (x, y) belongs to an unknown region of the map if (r_1, c_1) or (r_2, c_2) are on a range discontinuity in the image. Another important consequence is that we can compute the elevation at any point of the map without having to recompute the entire map, whereas standard map interpolation would have to compute the entire sparse map before interpolating the dense map.

We have used this technique to build terrain maps that have resolutions as fine as 10 centimeters and include uncertainty and explicit representation of unknown regions. The locus algorithm can also be used to build large maps by matching maps from individual images. Two images of the same area taken from two different locations are related by a transformation T (rotation and translation) between the two locations. The matching problem is essentially to compute T as accurately as

possible. Once this is done, the maps can easily be merged into a larger composite map. Given some value of T , we can compute from the images two elevations z_1 and z_2 for each point (x, y) in the map. The sum, $E(T)$, of squared differences $(z_1 - z_2)^2$ over the part of the map that is visible in both images measures how good our knowledge of T is. $E(T)$ is minimum when T is the exact transformation between the locations at which the two images were taken. Starting with an initial estimation of T , we can therefore apply a minimization algorithm (gradient descent) to $E(T)$ to compute the best possible value of T . In practice, the initial value of T is computed from feature matching or the vehicle's positioning system. We have applied this matching technique to the building of large maps (several hundred meters) using range images collected as the vehicle travels. Figure 12 shows a map built by combining 122 range images.

The locus algorithm provides a basis for a number of other map operations, such as

matching local maps for the vehicle with low-resolution aerial elevation maps. Detailed terrain features, such as ridges and valleys, can be extracted from high-resolution maps.

PERCEPTION CONTINUES TO BE the bottleneck. That is not to say that the other problems of mobile robots (such as path planning and map representation) are solved, but rather that they cannot be properly explored until robust perception components are built. The performance of a mobile robot system depends on the performance of the perception components. It is often assumed that robots are control systems, and that perception will provide clean numerical input; or that robots are cognitive problem solvers, and that perception will provide clean symbolic scene descriptions. Neither assumption is justified by current perception techniques and technology. Robots will not fulfill their potential unless we continue to improve perception capability.

While the most important scientific bottlenecks to perception involve inadequate algorithms, current abilities in sensor design are also a stumbling block. Too much effort has been spent in overcoming sensor limitations, which is necessary for doing real experiments but makes no lasting scientific contribution. For example, laser-scanning technology is a great advent in 3D sensing, but it still has considerable limitations: slow image acquisition (which severely limits vehicle speed), ambiguity intervals, bad behavior on certain material types, and so on. Color cameras also have problems, including limited field of view, inadequate dynamic range for mixed sun/shadow conditions, and unpredictable response from automatic irises and gains. We do not believe that any one magic sensor will "solve" the outdoor robot problem, but advances in sensors will certainly enable and encourage advances in image-understanding algorithms. We continue to build better algorithms, but their full power will not become useful until we have adequate sensors.

Much remains to be done to build perception systems reliable enough for high speed on-road and off-road navigation. Specifically, we continue to pursue vision for road tracking, including work on ALVINN and YARF. Other researchers at

CMU are working on strategies for interacting with other traffic and on tracking moving objects. We continue to need new sensors, both for road tracking and for longer-range obstacle detection. Off road, we are working with new range sensors, with inertially stabilized sensor platforms, and with new computer architectures, to build faster and more accurate systems.

Acknowledgments

Besides the authors, the Navlab group includes Omead Amidi, Didier Aubert, Jill Crisman, Jim Frazier, Jay Gowdy, Eric Hoffman, Karl Kluge, Dirk Langer, Jim Moody, Dean Pomerleau, Bill Ross, Ken Rosenblatt, David Simon, Tony Stentz, and Eddie Wyatt.

This research is sponsored in part by contracts from DARPA (titled "Perception for Outdoor Navigation" and "Development of an Integrated ALV System"), by NASA under contract NAGW-1175, by National Science Foundation contract DCR-8604199, and by the Digital Equipment Corporation External Research Program.

References

1. Y. Goto et al., "CMU Sidewalk Navigation System: A Blackboard-Based Outdoor Navigation System Using Sensor Fusion with Color-Range Images," *Proc. First Joint Conf. ACM/IEEE*, 1986.
2. J. Crisman and C. Thorpe, "Color Vision for Road Following," in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer, Norwell, Mass., 1990, pp. 9-23.
3. K. Kluge and C. Thorpe, "Explicit Models for Robot Road Following," in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer, Norwell, Mass., 1990, pp. 25-38.
4. D. Aubert and C. Thorpe, "Color Image Processing for Navigation: Two Road Trackers," Tech. Report CMU-RI-TR-90-09, Robotics Inst., Carnegie Mellon Univ., Pittsburgh, 1990.
5. D. Pomerleau, "Neural Network-Based Autonomous Navigation," in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer, Norwell, Mass., 1990, pp. 83-94.
6. P. Besl, "Range Imaging Sensors," Tech. Report GMR-6090, General Motors Research Labs, Warren, Mich., 1988.
7. A. Stentz, "Multiresolution Constraint Modeling for Mobile Robot Planning," in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer, Norwell, Mass., 1990, pp. 912-917.
8. I. Kweon, *Modeling Rugged Terrain by Mobile Robots with Multiple Sensors*, doctoral dissertation, Carnegie Mellon Univ., Pittsburgh, 1990.



Charles Thorpe is a senior research scientist at Carnegie Mellon University's Robotics Institute, where he directs research on the Navlab. He is also involved with robots for planetary exploration and underwater mapping. His research interests include

computer vision, planning, and the control of robot vehicles in unstructured outdoor environments. He is also editor of the book, *Vision and Navigation: The Carnegie Mellon Navlab*.

Thorpe received his PhD in computer science from CMU in 1984.



Martial Hebert is a research scientist at Carnegie Mellon's Robotics Institute, where his current projects include modeling environments from range data for autonomous vehicles, supervised mapping of plants, and object modeling for manipulation

in unstructured environments. His research interests include building and recognizing 3D environment models from sensor data.



Takeo Kanade is a professor of computer science, codirector of the Robotics Institute, and chair of the PhD program in robotics at Carnegie Mellon. He is the principal or coprincipal investigator for the university's projects on image understanding, vi-

sion systems for autonomous land vehicles, and the NASA Mars Rover.

Kanade received his PhD in electrical engineering from Kyoto University, Japan, in 1974. He is an AAAI fellow and a founding editor of the *International Journal of Computer Vision*.



Steven Shafer is an associate professor at Carnegie Mellon, and associate director of the PhD program in robotics. His work in the Calibrated Imaging Lab of the Robotics Institute examines how a computer can analyze images using optical models of illumination, reflection, and the imaging process. His research interests also include architectures for mobile robot perception, planning, and control.

Shafer received his PhD in computer science from Carnegie Mellon in 1983.

Readers can contact the authors at the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213.