Vision and Navigation for the Carnegie-Mellon Navlab

CHARLES THORPE, MARTIAL H. HEBERT, TAKEO KANADE, MEMBER, IEEE, AND STEVEN A. SHAFER, MEMBER, IEEE

Abstract—**This paper describes results on vision and navigation for mobile robots in outdoor environments. We present two types of vision algorithms: color vision for road following, and 3-D vision for obstacle detection and avoidance. In order to evaluate these algorithms in a realistic outdoor environment, we have integrated the vision capabilities into a complete system including a self-contained mobile platform, and a software architecture for the real-time control of distributed perception and navigation modules. The resulting system is able to navigate continuously on roads while avoiding obstacles.**

Index Terms—**Blackboards, color vision, mobile robots, range data.**

I. Introduction

MOBILE robot systems provide a unique opportunity to develop perception and navigation techniques in complex real-world environments. The tools a robot uses to bridge the chasm between the external world and its internal representation include sensors, image understanding to interpret sensed data, geometrical reasoning, and a concept of time and of the vehicle's motion over time. This paper presents a mobile robot system that includes both perception and navigation tools. The rationale for developing a whole system is that it allows us to confront our vision techniques with real-world environments in the context of actual autonomous navigation missions.

We have developed a testbed for the study of mobile robots, the Navigation Laboratory. The testbed is used for integrating perception and navigation capabilities. The sensory capabilities of the system are color vision, and 3-D vision using an active sensor. Color vision is used for finding roads in color images. 3-D vision is used for obstacle detection and terrain modeling. We have integrated the perception modules into a system that allows the vehicle to drive continuously in an actual outdoor environment. In order to integrate perception modules, navigation modules, and hardware interface, we propose a distributed architecture articulated around a knowledge database "Communication Database with Geometric Reasoning" (CODGER). The CODGER tools handle much of the modeling of time and geometry, and provide for synchronization of multiple processes. The architecture coordinates control and information flow between the high-level symbolic processes running on general purpose computers, and the lower-level control running on dedicated real-time hardware.

Manuscript received December 15, 1986; revised May 15, 1987. This work was supported by the Strategic Computing Initiative of the Defense Advanced Research Projects Agency, ARPA Order 5351, monitored by the U.S. Army Engineer Topographic Laboratories under Contract DACA760-85-C-0003, DARPA Contract DACA76-86-C-0019, and National Science Foundation Grant DCR-8604199.

The authors are with the Department of Computer Science and the Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 8820104.

II. Navlab: Navigation Laboratory

The Navlab [11] is a self-contained laboratory for navigational vision system research with on-board computing facilities. Figs. 1 and 2 show the vehicle.

Navlab is based on a commercial van chassis, with hydraulic drive and electric steering. Computers can steer and drive the van by electric and hydraulic servos, or a human driver can take control to drive to a test site or to override the computer. The Navlab has room for computers on board including Sun workstations and a Warp systolic computer [1]. Navlab is currently equipped with two sensors mounted above the cab: a TV camera, and a laser range finder. The interface between the computers and the driving hardware is provided by a controller which can steer the Van along circular arcs. The vehicle has room for up to four researchers in the back. This gets the researchers close to the experiments, and eliminates the need for video and digital communications with remote computers.

III. Road Following

The first application of a mobile robot is to find and follow roads using a video camera. Two approaches could be used: line tracking and region analysis. The first approach attempts to extract the edges of the road as seen by a camera, and backprojects them on the ground plane in order to compute the appropriate steering commands for the vehicle [4], [15]. This approach assumes an accurate model of the road shape. Unfortunately, in practice the strongest edges are often the shadow edges, whereas the road edges are much weaker and do not necessarily fit the straight line model. The line tracking approach relies heavily on inferring the road geometry from the road edges. The geometric inference assumes that the road edges are parallel [15], and therefore it is very unstable when applied to noisy data. In fact, it can be shown that the inferred road may fold over or under the ground plane [5].

Fig. 1. The Navlab.

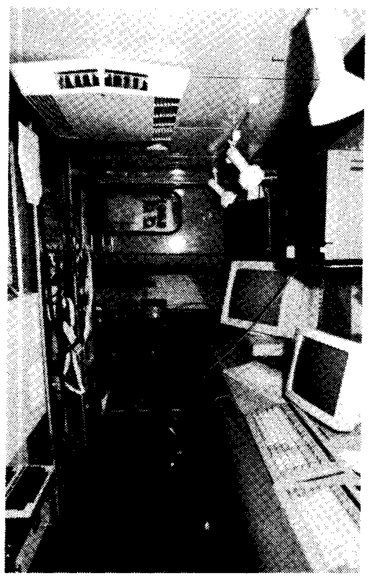

Fig. 2. Navlab interior.

The second approach classifies the pixels of a color image as on-road or off-road pixels based on the color characteristics of the road [13]. This approach does not require a geometric model as accurate as the line tracking approach. The main problem is that the dominant color features are dark (shadows) and light (sun). As a result road and nonroad pixels cannot be separated by a single threshold in RGB space. Furthermore, characteristic RGB values for a given feature (e.g., sunny road) drift, so that algorithms based on fixed threshold fail.

Previous work shows that the road following problem is difficult; existing systems work adequately for their particular environments, e.g., even illumination or straight edges. We use the second approach to find and follow roads [12]. We use multiclass adaptive color classification to classify image points as "road" or "nonroad" on the basis of their color. Since the road is not a uniform color, color classification must have more than one road model, or class, and more than one nonroad class. Because conditions change from time to time and from place to place over the test course, the color models must be adaptive. In addition to color classification, we added texture information to the classification algorithm. Once the image is classified, the road is identified by means of an area-based voting technique that finds the most likely location for the road in the image.

A. Overview of the Road Following Algorithm

Fig. 3 shows a simple scene which we will use to explain our algorithm. As shown in Fig. 4, the algorithm involves three stages:

1) Classify each pixel.

2) Use the results of classification to vote for the best-fit road position.

3) Collect new color statistics based on the detected road and nonroad regions.

Pixel classification is done by a standard pattern classification method [6]. Each class i is represented by the means m_i and covariance matrix, Σ_i, of red, green, and blue values, and by its *a priori* likelihood, f_i, based on expected fraction of pixels in that class. Assuming Gaussian distribution, the confidence that a pixel of color X belongs to a class is computed based on the distance $(X - m_i)^t \Sigma^{-1} (X - m_i)$. Each pixel is classified with the class of highest probability. Figs. 5 and 6 show how each pixel is classified and its confidence.

Once each point has been classified, we must find the most likely location of the road. We assume the road is locally flat, straight, and has parallel sides. The road geometry can be described by two parameters as shown in Fig. 7:

1) The intercept P, which is the image column of the road's *vanishing point*. This is where the road centerline intercepts the vanishing line of the locally flat plane of the road. In other words, the intercept gives the road's direction relative to the vehicle. Since the camera is fixed to the vehicle this vanishing line is constant assuming that the vehicle's pitch and roll relative to the road are small. In the following, $r_{horizon}$ is the row position of the vanishing line in the image.

2) The orientation θ of the road in the image, which tells how far the vehicle is to the right or left of the centerline.

We set up a two-dimensional parameter space, with intercept as one dimension and orientation as the other. Each point classified as road votes for all road (P, θ) combinations to which it could belong, while nonroad points cast negative votes, as shown in Fig. 9 and Fig. 8. For a given pixel (row, col), the corresponding set of pairs (P, θ) is the curve:

$$\left(col + \left(row - r_{horizon}\right) \times \text{tg}\ \theta,\ \theta\right)$$

in parameter space. The (P, θ) pair that receives the most votes is the one that contains the most road points, and it is reported as the road. For the case of Fig. 3, the votes in (P, θ) space look like Fig. 10. Fig. 11 shows the detected position and orientation of the road. It is worth noting that since this method does not rely on the exact local

Fig. 3. Original image.

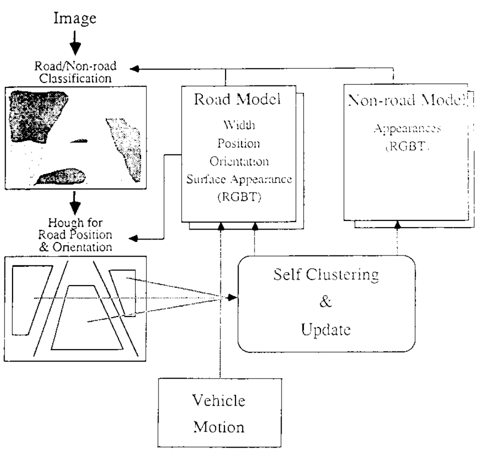

Fig. 4. Color vision for road following, including color classification, Hough transform for road region detection, and updating multiple road and nonroad models.

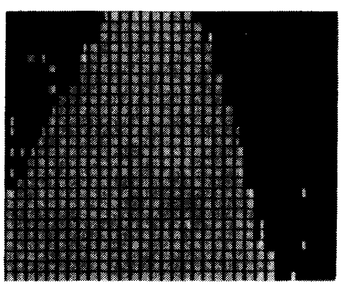

Fig. 5. Segmented image. Color and texture cues are used to label points below the horizon into two road and two off-road classes.

geometry of the road, it is very robust. The road may actually curve or not have parallel edges, or the segmentation may not be completely correct, but still the method outputs approximate road position and orientation. The main limitation is the field of view of the camera which limits the size of the parameter space. Currently, the intercept lies within the parameter space as long as the road has a curvature radius greater than 60 meters assuming a 60 degree field of view. If the road curve is sharper, the algorithm returns the closest interpretation in the parameter space.

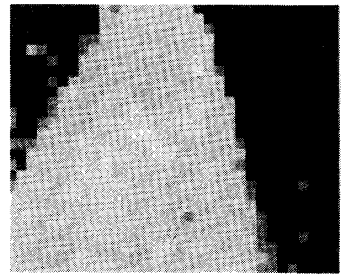

Fig. 6. Road probability image. The pixels that best match typical road colors are displayed brightest.

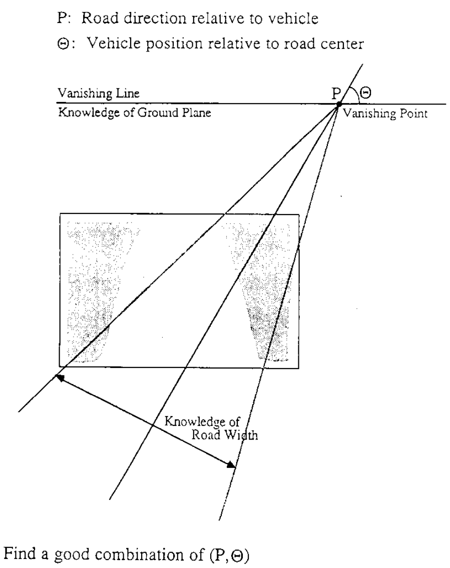

Fig. 7. Road transform that considers the geometry of road position and orientation. Geometry of locally flat, straight, and parallel road regions can be described by only P and θ. Point A classified as road could be a part of the road with the shown combination of (P, θ), and thus casts a positive vote for it. Point B classified as off-road, however, will cast a negative vote for that (P, θ) combination.

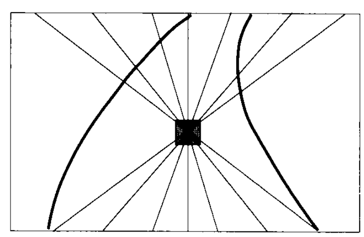

Fig. 8. A road point could be a part of roads with different orientations and vanishing points.

Once the road has been found in an image, the color statistics of the road and off-road models are modified for each class by resampling the detected regions (Fig. 12) and updating the color models. The updated color statistics will gradually change as the vehicle moves into a different road color, as lighting conditions change, or as the colors of the surrounding grass, dirt, and trees vary. As

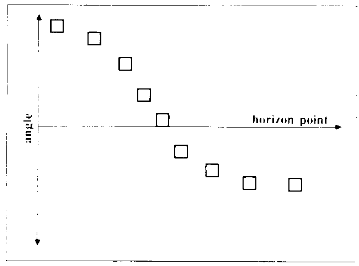

Fig. 9. The point from Fig. 8 would vote for these orientation/intercept values.

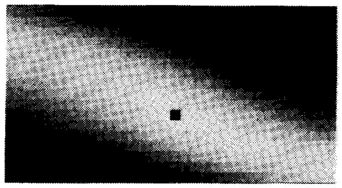

Fig. 10. Votes for best road orientation and intercept, and point with most votes (dark square), for road in Fig. 3.

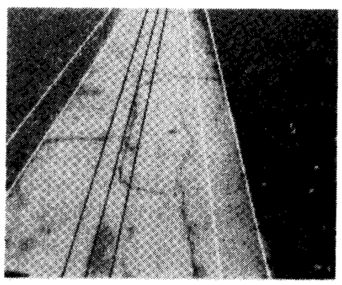

Fig. 11. Detected road, from the point with the most votes shown in Fig. 10.

long as the processing time per image is low enough to provide a large overlap between images, the statistics adapt as the vehicle moves. The road is picked out by hand in the first image. Thereafter, the process is automatic, using the segmentation from each image to calculate color statistics for the next.

Actually several important additional processing steps are possible. One is to smooth the images first. This removes outliers and tightens the road and nonroad clusters. Another is to have more than one class for road and for nonroad, for instance one for wet road and one for dry, or one for shadows and one for sun. Other variations change the voting for best road. Besides adding votes for road pixels, we subtract votes for nonroad points. Votes are weighted according to how well each point matches road or nonroad classes. Finally, an image contains clues other than color, such as visual texture. Roads tend to be smooth, with less high-frequency variation than grass or leaves, as shown in Fig. 13. We calculate a normalized texture measure, and use that in addition to color in the road classification.

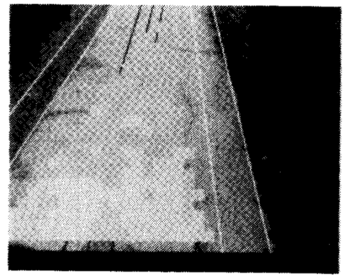

Fig. 12. Updating road and nonroad model colors, leaving a safety zone around the detected road region.

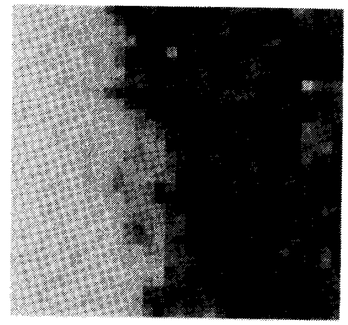

Fig. 13. Zoomed picture of road-nonroad boundary. The road (at left) is much less textured than the grass (at right).

B. Implementation

The implementation of the road following algorithm presented above runs in a loop of six steps: image reduction, color classification, texture classification, combining color and texture results, voting for road position, and color update. These steps are shown in Fig. 14, and are explained in detail below.

Image Reduction: We create a pyramid of reduced resolution R, G, and B images. Each smaller image is produced by simple 2×2 averaging of the next larger image. We found that other reduction methods, such as median filtering, are more expensive and produce no noticeable improvement in the system. We start with 480×512 pixel images, and typically use the images reduced to 30×32 for color classification. We use less reduced versions of the images for texture classification. Image reduction is used mainly to improve speed, but as a side effect the resulting smoothing reduces the effect of scene anomalies such as cracks in the pavement.

Color Classification: Each pixel (in the 30×32 reduced image) is labeled as belonging to one of the road or nonroad classes by standard maximum likelihood classification. We usually have four road and four nonroad classes. The number of classes is a compromise between having only two classes [13] (one for road and one for nonroad), which leads to poor performance in a changing environment, and having too many classes, which leads to an unreliable classification. Each class is represented by the mean R, G, and B values of its pixels, by a 3×3 covariance matrix, and by the fraction of pixels expected

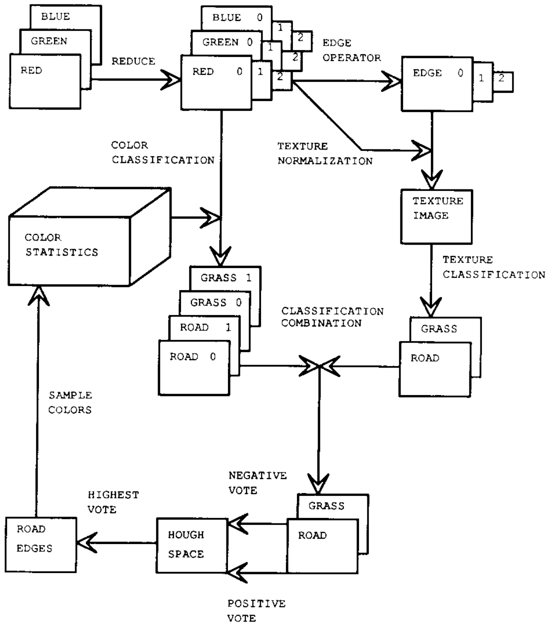

Fig. 14. Processing cycle for color vision.

a priori to be in that class. The classification procedure calculates the probability, P_i^C, that a pixel belongs to each of the classes.

Texture Calculation: The regions of paved roads tend to appear smoother in the image than nonroad regions of grass, soil, or tree trunks. The texture calculation combines a low-resolution (low-frequency) gradient image and a high-resolution (high-frequency) gradient image into a texture image. The algorithm is composed of six substeps:

- Calculate gradient at high resolution by running a Roberts operator over the 240×256 image.
- Calculate a low resolution gradient by applying a Roberts operator to the 60×64 image.
- Normalize the gradient by dividing the high resolution gradient by a combination of the average pixel value for that area and the low resolution gradient. Dividing by the average pixel value handles shadow interiors, while dividing by the low resolution gradient removes the shadow boundary.
- Normalized gradient

$$= \frac{\text{high-freq gradient}}{\alpha \times \text{low-freq gradient} + \beta \times \text{mean pixel value}}$$

Typical values for the coefficients are $\alpha = 0.2$ and $\beta = 0.8$.

- Threshold. Produce a 240×256 binary image of "microedges" by thresholding the normalized gradient. A fairly low threshold, such as 1, is usually adequate.
- Count Edges. Count the number of edges in each pixel block of 8×8 pixels. This gives a 30×32 pixel texture magnitude image. Fig. 15 shows the texture image derived from Fig. 3. Each texture pixel has a value between 0 and 255, which is the number of microedge pixels in the corresponding block of the full-resolution image.

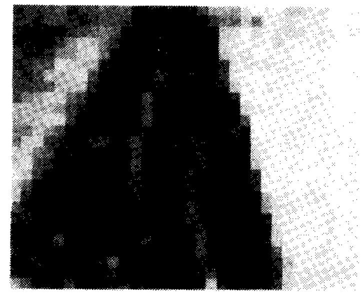

Fig. 15. Low resolution texture image, showing textures from Fig. 3. The brighter blocks are image areas with more visual texture.

- Texture Classification. Classify each pixel in the 30×32 image as road or nonroad on the basis of texture, and calculate a confidence P_i^T for each label. We found experimentally that a fixed mean and standard deviation for road and nonroad textures were better than adaptive texture parameters. Our best results were with road mean and standard deviation of 0 and 25, and nonroad values of 175 and 100. Effectively, any pixel block of the image with more than 35 microedges above threshold is considered textured, and is therefore classified as nonroad.

The weights and thresholds used in the texture calculation have been initially determined from a training set of 50 images.

Combination of Color and Texture Results: For each road or nonroad class i, confidence measures from color and texture are combined into one confidence P_i using the formula:

$$P_i = (1 - \alpha) P_i^T + \alpha P_i^C.$$

The weight α takes into account the fact that color is more reliable, so that the color probabilities should be weighted more than the texture probabilities. The final result is a classification of the pixels into road and nonroad, and a confidence calculated by:

$$C = Max\ \{P_i,\ i\ road\ class\} - Max\ \{P_i,\ i\ nonroad\ class\}.$$

The final confidence C is negative for nonroad pixels, and positive for road pixels.

Vote for Best Road Position: This step uses a 2-D parameter space similar to a Hough transform. Parameter P is the column of the road's vanishing point, quantized into 32 buckets because the image on which the classification and voting are based has 32 columns. Parameter θ is the road's angle from vertical in the image, ranging from -1 to 1 radian in 0.1 radian steps. A given road point votes for all possible roads that would contain that point. The locus of possible roads whose centerlines go through that point is an arctangent curve in the parameter space. Because the road has a finite width, the arctan curve has to be widened by the width of the road at that pixel's image row. Road width for a given row is not a constant over all possible road angles but is nearly constant enough that it does not justify the expense of the exact calculation. Each pixel's vote is weighted by its calculated confidence. Pixels classified as nonroad cast negative votes (with their weights reduced by a factor of 0.2) while road pixels add votes. In pseudo C code, the voting for a pixel at (row, col) is

```
for (theta = -1; theta <= 1; theta += 0.1) {
      center = col + tan (theta) * (r - r_horizon);
      for (c = center - width/2; c <= center + width/
        2; c++) {
            parameter_space[theta] [c] += confidence;
            }
      }
```

At the end of voting, one road pair (P, θ) will have the most votes. That intercept and angle describe the best road shape in the scene.

Color Update: The parameters of the road and nonroad classes need to be recalculated to reflect changing colors. We divide the image into four regions plus a "safety zone": left off-road, right off-road, upper road, and lower road. We leave a 64-pixel wide "safety zone" along the road boundary, which allows for small errors in locating the road, or for limited road curvature. For each of the four regions, we calculate the means of red, green, and blue. We use the calculated parameters to form four classes, and reclassify the image using a limited classification scheme. Limited reclassification is based on distance from a pixel's values to the mean values of a class, rather than the full maximum likelihood scheme used in classifying a new image. This tends to give classes based on tight clusters of pixel values, rather than lumping all pixels into classes with such wide variance that any pixel value is considered likely. The limited reclassification allows road pixels to be classified as either of the two road classes, but not as nonroad, and allows nonroad pixels to be reclassified only as one of the nonroad classes. The reclassified pixels are used as masks to recalculate class statistics. Since the limited reclassification is based on the distances between pixels and mean values, the loop of classify pixels/recalculate statistics is guaranteed to converge to a classification in which no pixels can switch classes. In practice, the loop is repeated three times. The final reclassified pixels are used to calculate the means, variances, and covariances of R, G, and B for each of the classes, to be used to classify the next image.

The color update technique cannot handle sudden changes in color statistics, it is therefore important to ensure enough overlap between images. In the current implementation, the speed of the vehicle is adjusted so that there is at least 75 percent overlap between consecutive images.

Calculation of Road Position in Vehicle Coordinates: The pair (P, θ) gives the position of the road in image coordinates. The last step of the algorithm is to convert the image position into vehicle coordinates. In order to perform the conversion, the system is first calibrated by computing for two rows r_1 and r_2 the number of pixels per meters, ppm_i and the column corresponding to the center of the road c_i. The calibration assumes that the width of the road w is known, and that the distance d_i between the vehicle and each row r_i is known. The location in vehicle coordinates of a road of image coordinates (P, θ), is given by the vehicle coordinates of the intersections Q_i, between the line of parameters (P, θ) and the rows r_i (Fig. 16). If Q_i is at column C_i in the image, then the distance between Q_i and the center of the vehicle is given by $x_i = (C_i - c_i)/ppm_i$.

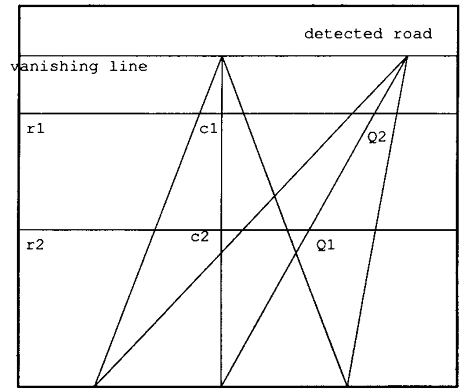

Fig. 16. Calibration procedure.

C. Performance

We have run this algorithm on the Navlab in an outdoor environment. The failure rate is close to 0. The occasional remaining problems come from one of three causes:

- The road is covered with leaves or snow, so one road color class and one nonroad color class are indistinguishable.
- Drastic changes in illumination occur between successive pictures (e.g., the sun suddenly emerges from behind a cloud) so all the colors change dramatically from one image to the next.
- The sunlight is so bright and shadows are so dark in the same scene that we hit the hardware limits of the camera. It is possible to have pixels so bright that all color is washed out, and other pixels in the same image so dark that all color is lost in the noise.

Not every image is classified perfectly, but all are good enough to result in the detection of correct road position and orientation for navigation. We sometimes find the road rotated in the image from its correct location, so we report an intercept off to one side and an angle off to the other side. But since the path planner looks ahead about the same distance as the center of the image, the steering target is still in approximately the correct location, and the vehicle stays on the road. This algorithm runs in about 10 s per image on a dedicated Sun 3/160, using 480 × 512 pixel images reduced to 30 rows by 32 columns. We currently process a new image every 4 m, which gives about three fourths of an image overlap between images. Ten seconds is fast enough to balance the rest of the sys-

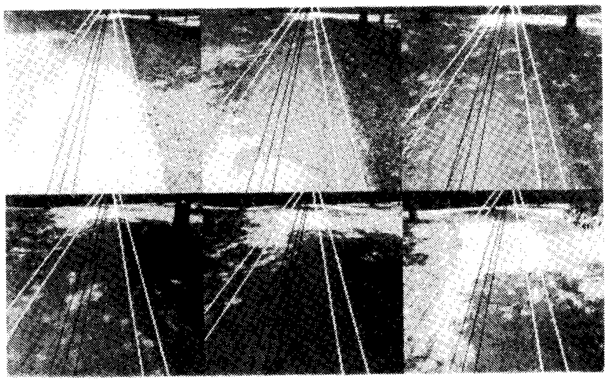

Fig. 17. Road following on a sequence of images.

tem but is slow enough that clouds can come and go and lighting conditions change between images. Fig. 17 shows the detected road on a typical sequence of nine images. This example is a typical case in which the presence of large shadow areas would preclude the use of both road edge tracking and classification based on two classes, road and nonroad.

D. Evaluation

In the course of our study of the road following problem, we have identified the following principles:

Assume Variation and Change: The appearance of a road varies from place to place and from time to time. For example, the road may be locally covered with leaves, or the lighting conditions may change in time due to cloudiness. We therefore need more than one road color model at any one time. The color models must adapt to changing conditions. In addition, we need to process images frequently so that the change from one image to the next will be moderate.

Use Few Geometric Parameters: A complete description of the road's shape in an image can be complex. The road can bend gently or turn abruptly, can vary in width, and can go up- or downhill. However, the more parameters there are, the greater the chance of error in finding those parameters. Small misclassifications in an image could give rise to fairly large errors in perceived road geometry. We found that two free parameters, orientation and distance from the vehicle, are sufficient to locally describe the road. Using only those two parameters implies that road width is fixed, the world is flat, and that the road is straight. While none of these assumptions is true over a long stretch of the road, they are nearly true within any one image; and the errors in road position that originate in oversimplifications are balanced by the smaller chance of bad interpretations.

Work in the Image: The road can be found either by projecting the road shape into the image and searching in image coordinates, or by back projecting the image onto the ground and searching in world coordinates [14]. The problem with the latter approach comes in projecting the image onto an evenly spaced grid in the world. Unless one uses a complex weighting scheme, some image pixels (those at the top that project to distant world points) will have more weight than other (lower) points. On the other hand, working directly in the image makes it much easier to weight all pixels evenly. Moreover, projecting a road shape is much more efficient than back projecting all the image pixels.

Calibrate Directly: A complete description of a camera must include its position and orientation in space, its focal length and aspect ratio, lens effects such as fisheye distortion, and nonlinearities in the optics or sensor. The general calibration problem of trying to measure each of these variables is difficult. It is much easier, and more accurate, to calibrate the whole system than to tease apart the individual parameters. The easiest method is to take a picture of a known object and build a lookup table that relates each world point to an image pixel and vice versa. Projecting road predictions into the image and back projecting detected road shapes onto the world are done by means of table lookup (or table lookup for nearby values with simple interpolations). Such a table is straightforward to build and provides good accuracy, and there are no instabilities in the calculations.

IV. Perception in 3-D

Color vision is not sufficient for the navigation of a mobile robot. Information such as the location of obstacles requires the availability and processing of 3-D data. 3-D vision has two objectives: obstacle detection, and terrain analysis. Obstacle detection allows the system to locally steer the vehicle on a safe path. Terrain analysis provides a more detailed description of the environment which can be used for more accurate path planning or for object recognition [7].

In order to study 3-D vision for a mobile robot, we used an ERIM scanning laser range finder. The scanner produces, every half second, an image containing 64 rows by 256 columns of range values; an example is shown in Fig. 18. The scanner measures the phase difference between an amplitude-modulated laser and its reflection from a target object, which in turn provides the distance between the target object and the scanner. The scanner produces a dense range image by using two deflecting mirrors, one for the horizontal scan lines and one for vertical motion between scans. The volume scanned is 80 degrees wide and 30 degrees high. The range at each pixel is discretized over 256 levels from 0 to 64 feet.

A. Preprocessing

Our range processing begins by smoothing the data and undoing the peculiarities of the ranging geometry. The *ambiguity intervals*, where range values wrap around from 255 to 0, are detected and unfolded. Two other undesirable effects are removed by the same algorithm. The first is the presence of mixed points at the edge of an object. The second is the meaninglessness of a measurement from a surface such as water, glass, or glossy pigments. In both cases, the resulting points are in regions limited by con-

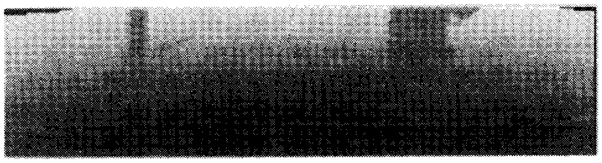

Fig. 18. Range image of two trees on flat terrain. Gray levels encode distance; nearer points are painted darker.

siderable jumps in range. We then transform the values from angle-angle-range, in scanner coordinates, to x-y-z locations. These 3-D points are the basis for all further processing.

B. Obstacle Detection and Terrain Analysis

We have two main processing modes: obstacle detection and terrain analysis. Obstacle detection starts by calculating surface normals from the x-y-z points. Flat, traversable surfaces will have vertical surface normals. Obstacles will have surface patches with normals pointed in other directions. This analysis is relatively fast, running in about 5 s on a Sun 3/75, and is adequate for smooth terrain with discrete obstacles.

Simple obstacle maps are not sufficient for detailed analysis. For greater accuracy we do more careful terrain analysis and combine sequences of images corresponding to overlapping parts of the environment into an *extended obstacle map*. The terrain analysis algorithm first attempts to find groups of points that belong to the same surface and then uses these groups as seeds for the region growing phase. Each group is expanded into a smooth connected surface patch. The smoothness of a patch is evaluated by fitting a surface (plane or quadric). In addition, surface discontinuities are used to limit the region growing phase. The complete algorithm is:

1) Edges: Extract surface discontinuities, pixels with high jumps in x-y-z.

2) Clustering: Find clusters in the space of surface normals and identify the corresponding regions in the original image.

3) Region growing: Expand each region until the fitting error is larger than a given threshold. The expansion proceeds by iteratively adding the point of the region boundary that adds the minimum fitting error.

The clustering step is designed so that other attributes such as color or curvature can also be used to find potential regions on the object. The primitive surface used to compute the fitting error can be either a plane or a quadric. The decision is based on the size of the region. Fig. 19 shows the resultant description of 3-D terrain and obstacles for the image of Fig. 18. The flat, smooth, navigable region is the meshed area, and the detected 3-D objects (the two trees) are shown as polyhedra.

Obstacle detection works at longer range than terrain analysis. When the scanner is looking at distant objects, it has a very shallow depression angle. Adjacent scanlines, separated by 0.5 degrees in the range image, can strike the ground at widely different points. Because the grazing angle is shallow, little of the emitted laser energy returns to the sensor, producing noisy pixels. Noisy range values, widely spaced, make it difficult to do detailed analysis of flat terrain. A vertical obstacle, such as a tree, shows up much better in the range data. Pixels from neighboring scanlines fall more closely together, and with a more nearly perpendicular surface the returned signal is stronger and the data cleaner. It is thus much easier for obstacle detection to find obstacles than for terrain analysis to certify a patch of ground as smooth and level.

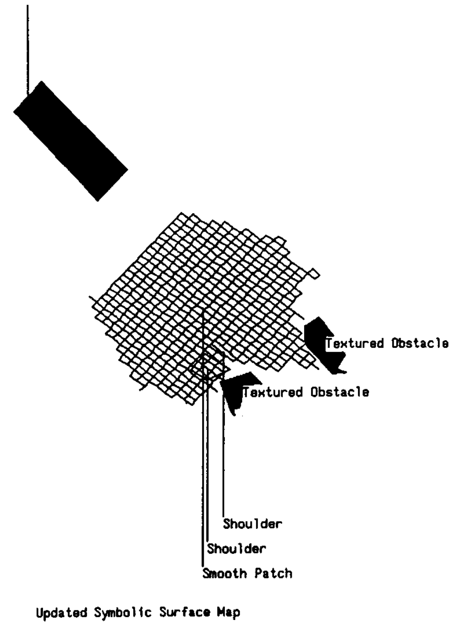

Fig. 19. The resultant description of 3-D terrain and obstacles from the image in Fig. 18. The navigable area is shown as a mesh, and the two trees are detected as "textured obstacles" and shown as black polygons.

V. System Building

When computer vision is to be used as part of a larger system, the architecture of the system influences the design of the vision components. A variety of design approaches for mobile robot architectures have been discussed [2], [3], [8], [9], [15]. The main feature our approach has in common with them is the need for a distributed architecture integrating perception and navigation. We propose a system architecture called CODGER that meets the needs of the NAVLAB and provides a framework within which our vision programs are executed. The CODGER system has been designed to integrate the algorithms of Sections III and IV, and to demonstrate them on the testbed of Section II.

A. Architecture Principles for Real World Robots

Artificial Intelligence systems, including intelligent mobile robots, are symbol manipulators. In most AI systems, the symbol manipulation is based on inference, either by the logic of predicate calculus or by probabilities. In contrast, the bulk of the work of a mobile robot is based on modeling geometry and time because these are the properties utilized by knowledge sources for object modeling, motion, path planning, navigation, vehicle dynamics, and so forth. Inference may be a part of a mobile robot system, but geometry and time are pervasive. Consequently, intelligent robots need a new kind of expert system shell that provides tools for handling 3-D locations and motion.

Based on this observation, we have developed and followed the following tenets of mobile robot system design in building our system:

Modularity: Much of the deep knowledge can be limited to particular specialist modules. The effects of lighting conditions and viewing angle on the appearance of an object, for instance, are important data for color vision but are not needed by path planning. So one principle of mobile robot system design is to break the system into modules and minimize the overlap of knowledge between modules. Furthermore, modularity allows the system to have different components written in different languages or executing on specialized hardware as appropriate for each module.

Virtual Vehicle: As many as possible of the details of the vehicle should be hidden. We therefore hide the details of sensing and motion in a "virtual vehicle" interface, so a single "move" command, for instance, will use the different mechanisms of two vehicles but will produce identical behavior.

Synchronization: A system that has separate modules communicating at a fairly coarse grain will be loosely coupled. Lock-step interactions are neither necessary nor appropriate. However, there are times when one module needs to wait for another to finish, or when a demon module needs to fire whenever certain data appear. The system should provide tools for several different kinds of interaction and for modules to synchronize themselves as needed.

Real-Time Versus Symbolic Interface: A mobile robot requires a blend of high-level reasoning processes and low-level real-time processes within a single system. The high-level processes are typically event-driven and require highly variable amounts of time depending on the vehicle's state and the environment. The low-level processes, however, typically control vehicle functions such as locomotion and must run in real time. The system should provide for both real-time and asynchronous symbolic processes, and for communications between them.

Geometry and Time: Much of the knowledge that needs to be shared between modules has to do with geometry, time, and motion. An object may be predicted by one module (the lookout), seen separately by two others (color vision and 3-D perception), and used by two more (path planner and position update). During the predictions, sensing, and reasoning, the vehicle will be moving, new position updates may come in, and the geometrical relationship between the vehicle and the object will be constantly changing. Moreover, there may be many different frames of reference: one for each sensor, one for the vehicle, one for the world map, and others for individual objects. Each module should be able to work in the coordinate frame that is most natural; for instance, a vision module should work in camera coordinates and should not have to worry about conversion to the vehicle reference frame. The system should provide tools that handle as many as possible of the details of keeping track of coordinate frames, motion, and changing geometry.

Distributed Control: In some systems (notably early blackboards) a single master process "knows" everything. The master process may not know the internal working of each module, but it knows what each module is capable of doing. The master controls who gets to run when. The master itself becomes a major AI module and can be a system bottleneck. In contrast, the individual modules in a mobile robot system should be autonomous, and the system tools should be slaves to the modules. The module writers should be free to decide when and how to communicate and when to execute.

We have followed these tenets in building the Navlab system. At the bottom level, we have built the CODGER "whiteboard" to provide system tools and services [10]. On top of CODGER we have built an architecture that sets conventions for control and data flow. CODGER and our architecture are explained below.

B. Blackboards and Whiteboards

The program organization of the NAVLAB software is shown in Fig. 20. Each of the major boxes represents a separately running program. The central database, called the *Local Map*, is managed by a program known as the Local Map Builder (*LMB*). Each module stores and retrieves information in the database through a set of subroutines called the *LMB Interface* which handle all communication and synchronization with the LMB. If a module resides on a different processor than the LMB, the LMB and LMB Interface will transparently handle the network communication. The Local Map, LMB, and LMB Interface together comprise the CODGER (COmmunications Database with GEometric Reasoning) system.

The overall system structure—a central database, a pool of knowledge-intensive modules, and a database manager that synchronizes the modules—is characteristic of a traditional blackboard system. Such a system is called "heterarchical" because the knowledge is scattered among a set of modules that have access to data at all levels of the database (i.e., low-level perceptual processing ranging up to high-level mission plans) and may post their findings on any level of the database. We call CODGER a *whiteboard* because it implements a heterarchical system structure, but it differs from a blackboard in several key respects. In CODGER, each module is a separate, continuously running program; the modules communicate by storing and retrieving data in the central database. Synchronization is achieved by primitives in the data retrieval facilities that allow, for example, for a module to request data and suspend execution until the specified data appears. When some other module stores the desired data,

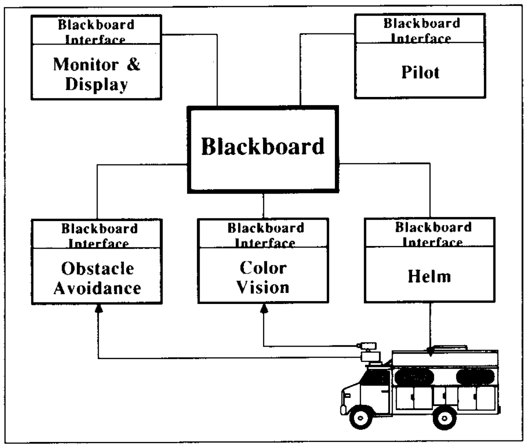

Fig. 20. Navlab software architecture.

the first module will be reactivated and the data will be sent to it. With CODGER a module programmer thus has control over the flow of execution within his module and may implement real-time loops, demons, data flows among cooperating modules, etc. Like other recent distributed AI architectures, whiteboards are suited to execution on multiple processors.

C. Data Storage and Retrieval

Data in the CODGER database (Local Map) is represented in *tokens* consisting of classical *attribute-value pairs*. A module can store a token by calling a subroutine to send it to the LMB. Tokens can be retrieved by constructing a pattern called a *specification* and calling a routine to request that the LMB send back tokens matching that specification. The specification is simply a Boolean expression in which the attributes of each token may be substituted; if a token's attributes satisfy the Boolean expression, then the token is sent to the module that made the request. For example, a module may specify:

tokens with **type** *equal to "intersection" and* **traffic-control** *equal to "stop-sign"*

This would retrieve all tokens whose **type** and **traffic-control** attributes satisfy the above conditions. The specification may include computations such as mathematical expressions, finding the minimum value within an array attribute, comparisons among attributes, etc. CODGER thus implements a general database. The module programmer constructs a specification with a set of subroutines in the CODGER system.

One of the key features of CODGER is the ability to manipulate geometric information. One of the attribute types provided by CODGER is the *location*, which is a 2-D or 3-D polygon and a reference to a *coordinate frame* in which that polygon is described. Every token has a specific attribute that tells the location of that object in the Local Map, if applicable, and a specification can include geometric calculations and expressions. For example, a specification might be

tokens with **location** *within 5 units of (45,32)* [*in world coordinates*]

or

tokens with **location** *overlapping X*

where X is a description of a rectangle on the ground in front of the vehicle. The geometric primitives currently provided by CODGER include calculation of centroid, area, diameter, convex hull, orientation, and minimum bounding rectangle of a location, and distance and intersection calculations between a pair of locations.

CODGER also provides for automatic coordinate system maintenance and transformation for these geometric operations. In the Local Map, all coordinates of location attributes are defined relative to **WORLD** or **VEHICLE** coordinates; **VEHICLE** coordinates are parameterized by time, and the LMB maintains a time-varying transformation between **WORLD** and **VEHICLE** coordinates. Whenever new information (i.e., a new **VEHICLE**-to-**WORLD** transform) becomes available, it is added to the "history" maintained in the LMB; the LMB will interpolate to provide intermediate transformations as needed. In addition to these basic coordinate systems, the LMB Interface allows a module programmer to define *local coordinates* relative to the basic coordinates or relative to some other local coordinates. Location attributes defined in a local coordinate system are automatically converted to the appropriate basic coordinate system when a token is stored in the database. CODGER provides the module programmer with a conversion routine to convert any location to any specified coordinate system.

D. Module Architecture

Several modules use the CODGER tools and fit into a higher level architecture. The modules are:

- *Pilot:* Looks at the map and at current vehicle position to predict road location for Vision. Plans paths.
- *Map Navigator:* Maintains a world map, does global path planning, provides long-term direction to the Pilot. The world map may start out empty, or may include any level of detail up to exact locations and shapes of objects.
- *Color Vision:* Waits for a prediction from the Pilot, waits until the vehicle is in the best position to take an image of that section of the road, returns road location.
- *Obstacle Detection:* Gets a request from the Pilot to check a part of the road for obstacles. Returns a list of obstacles on or near that chunk of the road.
- *Helm:* Gets planned path from Pilot, converts polyline path into smooth arcs, steers vehicle.
- *Graphics and Monitor:* Draws or prints position of vehicle, obstacles, predicted and perceived road.

These modules use CODGER to pass information about *driving units*. A driving unit is a short chunk of the road or terrain (in our case 4 m long) treated as a unit for perception and path planning. The need for driving units comes from the limited field of view of the sensors. The environment must be divided into chunks small enough to be viewed in one frame of each sensor. The sizes and positions of the driving units is computed based on the field of view of the sensors and the speed of the vehicle. The Pilot gives driving unit predictions to Color Vision, which returns an updated driving unit location. Obstacle Detection then sweeps a driving unit for obstacles. The Pilot takes the driving unit and obstacles, plans a path, and hands the path off to the Helm. The whole process is set up as a pipeline, in which Color Vision is looking ahead 3 driving units, Obstacle Detection is looking 2 driving units ahead, and path planning at the next unit. If for any reason some stage slows down, all following stages of the pipeline must wait. So, for instance, if Color Vision is waiting for the vehicle to come around a bend so it can see down the road, Obstacle Detection will finish its current unit and will then have to wait for Color Vision to proceed. In an extreme case, the vehicle may have to come to a halt until everything clears up. All planned paths include a deceleration to a stop at the end, so if no new path comes along to overwrite the current path the vehicle will stop before driving into an area that has not been seen or cleared of obstacles.

VI. Conclusions and Future Work

The system described has successfully driven the Navlab many tens of times, processing thousands of color and range images without running off the road or hitting any obstacles. CODGER has proved to be a useful tool, handling many of the details of communications and geometry. Module developers have been able to build and test their routines in isolation, with relatively little integration overhead. Yet there are several areas that need much more work.

We drive the Navlab at 10 cm/s, a slow shuffle. Our slow speed is because our test road is narrow and winding, and because we deliberately concentrate on competence rather than on speed. But faster motion is always more interesting, so we are pursuing several ways of increasing speed. One bottleneck is the computing hardware. We have mounted a Warp, Carnegie Mellon's experimental high-speed processor, on the Navlab. We have implemented the vision and range algorithms of the Warp machine, allowing us to drive the vehicle at 50 cm/s. At the same time, we are looking at improvements in the software architecture. We need a more sophisticated path planner, and we need to process images that are more closely spaced than the length of a driving unit. Also, as the vehicle moves more quickly, our simplifying assumption that steering is instantaneous and that the vehicle moves along circular arcs becomes more seriously flawed. We are looking at other kinds of smooth arcs, such as clothoids. More important, the controller is evolving to handle more of the low-level path smoothing and following.

One reason for the slow speed is that the Pilot assumes straight roads. We need to have a description that allows for curved roads, with some constraints on maximum curvature. The next steps will include building maps as we go, so that subsequent runs over the same course can be faster and easier. Travel on roads is only half the challenge. The Navlab should be able to leave roads and venture cross-county. Our plans call for a fully integrated on-road/off-road capability. Current vision routines have a built-in assumption that there is one road in the scene. When the Navlab comes to a fork in the road, vision will report one or the other of the forks as the true road depending on which looks bigger. It will be important to extend the vision geometry to handle intersections as well as straight roads. We already have this ability on our sidewalk system and will bring that over to the Navlab. Vision must also be able to find the road from off-road. Especially as we venture off roads, it will become increasingly important to be able to update our position based on sighting landmarks. This involves map and perception enhancements, plus understanding how to share limited resources, such as the camera, between path finding and landmark searches.

Acknowledgment

The Navlab was built by W. Whittaker's group in the Field Robotics Center, and the Warp group is led by H. T. Kung and J. Webb. The real work gets done by an army of eight staff, nine graduate students, five visitors, and three part time programmers.

References

[1] M. Annaratone, F. Bitz, J. Deutch, L. Hamey, H. T. Kung, P. C. Maulik, P. S. Tseng, and J. A. Webb, "Applications experience on Warp," in *Proc. AFIPS Nat. Comput. Conf.*, 1987.
[2] R. Bhatt, D. Gaw, and A. Meystel, "A real-time guidance system for an autonomous vehicle," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987.
[3] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics and Automation*, vol. RA-2, no. 1, 1986.
[4] L. S. Davis, T. R. Kushner, J. Le Moigne, and A. M. Waxman, "Road boundary detection for autonomous vehicle navigation," *Opt. Eng.*, vol. 25, no. 3, Mar. 1986.
[5] D. DeMenthon, "Inverse perspective of a road from a single image," Center for Automation Research, Univ. Maryland, College Park, Tech. Rep. 210, July 1986.
[6] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.
[7] M. Hebert and T. Kanade, "Outdoor scene analysis using range data," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1986.
[8] D. Kuan and U. K. Shorna, "Model-based geometric reasoning for autonomous road following," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987.
[9] M. Parodi, J. J. Nitao, and L. S. McTamaney, "An intelligent system for an autonomous vehicle," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1986.
[10] S. Shafer, A. Stentz, and C. Thorpe, "An architecture for sensor fusion in a mobile robot," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1986.
[11] J. Singh *et al.*, "NavLab: An autonomous vehicle," Carnegie Mellon Robotics Inst., Pittsburgh, PA, Tech. Rep., 1986.
[12] C. Thorpe, "Vision and navigation for the CMU Navlab," in *Proc. SPIE*, Oct. 1986.
[13] M. A. Turk, D. G. Morgenthaler, K. D. Gremban, and M. Marra, "Video road-following for the autonomous land vehicle," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987.
[14] R. Wallace, K. Matsuzaki, Y. Goto, J. Crisman, J. Webb, and T. Kanade, "Progress in robot road-following," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1986.
[15] 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," *IEEE J. Robotics and Automation*, vol. RA-3, no. 2, 1987.

Charles Thorpe received the B.A. degree in natural science from North Park College, Chicago, IL, in 1979, and the Ph.D. degree from Carnegie-Mellon University, Pittsburgh, PA, in 1984. His doctoral dissertation described FIDO, a stereo vision and navigation system for a robot rover.

He is a Research Scientist in the Robotics Institute of Carnegie-Mellon University. He is one of the Principal Investigators on the Navlab project. His research focuses on perception, and on the interactions between perception and planning in mobile robots.

Dr. Thorpe is a member of the Association for Computing Machinery and the American Institute of Aeronautics and Astronautics.

Martial H. Hebert received the Master's degree and Doctorate degree in computer science from the University of Orsay, France, in 1981 and 1984, respectively.

He is a Research Scientist at the Robotics Institute at Carnegie-Mellon University, Pittsburgh, PA. His past experience includes working as a Research Scientist at the Institut National de Recherche en Informatique et Automatique from 1982 to 1984. His current research includes 3-D vision for an autonomous land vehicle.

Takeo Kanade (M'80) received the Ph.D. degree in information science from Kyoto University, Japan, in 1974.

He is Professor of Computer Science and Robotics at Carnegie-Mellon University, Pittsburgh, PA. Currently he is Acting Director of the Robotics Institute. Before his 1980 appointment with Carnegie-Mellon, he was Associate Professor of Information Science at Kyoto University. He has worked on several areas in robotics and artificial intelligence: theoretical and practical aspects of computer vision, 3-D range sensing and analysis, development and control of new direct-drive arms which were first conceived and prototyped at CMU (DD Arm I and DD Arm II), and mobile robot systems. He has authored and edited three books and over sixty papers and technical reports in these areas. His current research includes: the Navlab (a van with on-board sensors and computers) vision system, the Mars Rover development, and Reconfigurable Modular Manipulator System.

Dr. Kanade served as a General Chairman of IEEE International Conference on Computer Vision and Pattern Recognition in 1983, and a Vice Chairman of IEEE International Conference on Robotics and Automation in 1986. He is the Editor of the *International Journal of Computer Vision*.

Steven A. Shafer (M'84) received the B.A. degree in computer science from the University of Florida in 1977 and the Ph.D. degree in computer science from Carnegie-Mellon University, Pittsburgh, PA, in 1983.

He is a Research Scientist studying computer vision and mobile robots in the Department of Computer Science of Carnegie-Mellon University, where he has been working since 1983. He is primarily interested in the analysis of images by computer, using optical properties of illumination, objects, and cameras. By analyzing properties such as color, gloss, and shadows, he is developing methods to base computer vision on an understanding of physics rather than the current ad hoc statistical and pattern classification techniques. His view of computer vision as a problem in measurement of physical quantities is also reflected in his work on geometric, photometric, and spectral calibration of cameras. This work is carried out in the Calibrated Imaging Laboratory at CMU, and is primarily oriented toward robotics tasks in navigation and object manipulation. He is also studying architectures for mobile robot perception, planning, and control. He is one of the authors of CODGER, the distributed blackboard system used by the NAVLAB autonomous robot van at Carnegie-Mellon University, and has been active in designing the software for control of the NAVLAB. He is also coauthor of the PHAROS "microscopic" traffic simulator, which will be used to develop a computer program to control a robot vehicle driving through traffic. He is conducting the effort at CMU to integrate into the NAVLAB system modules that are being developed at other universities and research labs. In addition to his research work, he has been active in the enhancement of the UNIX operating system at Carnegie-Mellon. He is the author and maintainer of over one hundred system programs and library subroutines, including a relational database manager (DAB) and a software upgrade program (SUP). He is the author of several software reference manuals and instructional documents, and has taught the use of UNIX at CMU and elsewhere.

Dr. Shafer has been active in professional activities in the fields of computer vision and optics (appearance measurement). He has produced two programs to perform digital image segmentation, one each at Carnegie-Mellon and at the University of Hamburg in Germany; the former program (PHOENIX) is a component of the Darpa Image Understanding Testbed compiled by SRI International. He is the author of one book and numerous papers, appearing in both the computer science and optics literature, and served as a consultant for the *Handbook of Artificial Intelligence*, the *Encyclopedia of Artificial Intelligence*, and the Time-Life book series *Understanding Computers*. He has taught several courses in computer science, including computer vision, and organized and instructed in the CMU Robotics Institute's Tutorial on Computer Vision. He is a member of professional societies for computer science, optics, and color science; he is chairman of Committee 42 of the Inter-Society Color Council, "Terminology for the Optics of Materials."