NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

1987 Year End Report
for Road Following at Carnegie Mellon

Charles Thorpe and Takeo Kanade
Principal Investigators

CMU-RI-TR-88-4

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

April 1988

© 1988 Carnegie Melton University

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

Table of Contents

Abstract 1

Introduction *Charles Thorpe and Takeo Kanade* 3

- Introduction and Overview 3
- Chronology 5
- Personnel 6
- Publications 6

Color Vision for Road Following 11
Charles Thrope and Jill Crisman

3-D Vision for Outdoor Navigation by an Autonomous Vehicle 29
Martial Hebert and Takeo Kanade

Knowledge-Based Interpretation of Outdoor Road Scenes 45
Takahiro Fujimori and Takeo Kanade

Car Recognition for the CMU Navlab 99
Karl Kluge and Hideki Kuga

Geometric Camera Calibration Using Systems of Linear Equations 119
Keith D. Gremban, Charles E. Thorpe, and Takeo Kanade

University Libraries
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This report describes progress in vision and navigation for outdoor mobile robots at the Carnegie Mellon Robotics Institute during 1987. This research was primarily sponsored by DARPA as part of the Strategic Computing Initiative. Portions of this research were also partially supported by the National Science Foundation and Digital Equipment Corporation.

We are pursuing a broad range of perception research for guiding outdoor autonomous vehicles. In 1987 we concentrated on five areas:

1. Road following. We expanded our road tracking system to better handle shadows and bright sunlight. For the first time, we added capabilities for recognizing intersections.
2. Range data interpretation. Our range interpretation work has expanded from processing a single frame, to combining several frames of data into a "terrain map." Building terrain maps requires finding features in each frame of data, and matching those features between frames to get precise data registration. We also used range data for recognizing cars.
3. Expert systems for image interpretation. We explored finding roads in very difficult scenes, without relying on strong a priori road color or shape models. The expert system that resulted from our research is very competent, and breaks new ground in expert system design.
4. Car recognition. We recognize cars in color images by a hierarchy of grouping image features, and predicting where to look for other image features. We use only weak 2-D shape constraints, rather than searching for a particular 3-D pose.
5. Geometric camera calibration. Our new method for calibration avoids complex non-linear optimizations found in other calibration schemes. We have variants for cameras with near-linear lenses and for cameras with significant local distortions.

This report begins with an introduction, chronology, and lists of personnel and publications. It also includes papers describing each of the research areas.

Introduction and Overview

This report reviews progress at Carnegie Mellon from January 15, 1987 to January 14, 1988 on research sponsored by the Strategic Computing Initiative of DARPA, DoD, through ARPA Order 5351, and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-0003, titled "Road Following." Portions of this research were also partially sponsored by the National Science Foundation contract DCR-8604199 and by the Digital Equipment Corporation External Research Program. This report consists of an introduction and overview, and detailed reports on specific areas of research.

During 1987 we extended our work on perception for road following and mapping, and started new efforts in expert systems for road finding, in expert systems for car recognition, and in camera calibration. The road following, mapping, and calibration modules were used to drive our Navlab robot. Each of the modules is described briefly below, and at length in the following chapters of this report.

Road following. In 1987 we attacked problems of limited camera dynamic range and of intersection recognition. The previous vision system successfully drove the Navlab on our narrow, twisting, test path on Flagstaff Hill. Its main limitations were that it could not deal adequately with scenes containing both very bright and very dark areas, and that it used a single road model and could not properly handle intersections.

The dynamic range of a camera is not sufficient to see both heavily shadowed areas and areas in full sunlight. Therefore in our 1987 system we used two video cameras, one with its iris set open for seeing into the shadows, and one with its iris closed for seeing sunny areas. The two images were combined pixel by pixel in a preprocessing phase which required keeping separate color statistics for shaded and sunny areas, and modifying our classification scheme accordingly.

Having a single road model meant that our system relied on precisely known road width, and that it could not recognize an intersection. We overcame those limitations in two ways. One approach used a map and the vehicle's position to predict the shape of the road or intersection in the field of view. If the vehicle's location was poorly known, it was often necessary to try both a road model and an intersection model, and decide which model matched best. Another approach eliminated specific models, and used generic road cues such as straightness of edges, edges parallel to each other, connectivity of the road region, and so forth.

During 1987 we installed a Warp machine on our Navlab testbed vehicle. Our first Warp vision system used a prototype Warp, running the 1986 algorithms, to increase vehicle speed by a factor of five. Later, we installed a General Electric production version of the Warp. Rather than push for speed, in our 1987 system we used the increased processing power for processing two images (for shady and sunny areas), for searching for intersections as well as roads, and for processing at higher resolutions to be able to see intersections at a distance.

The 1987 color vision system is described by Crisman and Thorpe in the report "Color Vision for Road Following."

ERIM interpretation and map building. There were three major advances in range data processing this year: map building by active registration, object recognition, and efficient Warp implementation.

Individual frames of range data can be processed to describe the local terrain, either by building a depth map or by extracting features such as smooth patches or edges. The maps from a single frame, though, cover only a limited area, may contain noisy data, and have "shadows" of unseen area behind tall objects. This year's research developed methods to combine data from multiple scans to fill in the shadows, extend the coverage, and increase map precision. The difficulty in combining data is getting precise registration between frames of data taken at different vehicle locations. The techniques we developed match extracted features from successive frames to get precise registration from frame to frame. If there are many features, the results are quite good. If there are few or no features, registration must depend on the vehicle's dead reckoning; but in featureless areas, a higher degree of error can be tolerated.

We have demonstrated the use of range data to recognize cars. The range image is segmented, using cues from the reflectance channel to keep regions from growing across edges. The extracted surfaces are then matched with a generic car model. The model contains both surfaces and constraints between surfaces, such as "the roof must be roughly perpendicular to the side." Matching proceeds by finding candidate surface matches, and checking their consistency with other surface matches by constraint checking.

Range data processing can be expensive. It is largely floating point calculations, and requires many steps ranging from removal of spurious data to surface fitting. We have implemented our range analysis on the Warp processor on the Navlab. We have used two systems, one with a single Warp shared between color vision and range processing, and the other system with a dedicated warp for each of the two perception modules. This code has been transferred to Martin Marietta and used in their 1987 demo.

More details about range data processing are in the attached paper "3-D Vision for Outdoor Navigation by an Autonomous Vehicle*, by Martial Hebert and Takeo Kanade.

Road finding by expert system. In very bad lighting conditions, or in cases where there is no a priori mad location or shape model, finding a road in an image can be very difficult. We developed an expert system road finder that uses a wide array of knowledge. This system, written in OPS, alternates between model fitting ami region evaluation. At each step, it assigns tentative labels (road, grass, tree, sky, etc.) ami weights to the regions it thinks i ran label reliably. It then fits a geometric road model to the labeled regions. The road model helps constrain region labeling, and region labels help constrain model fitting. Region labeling uses mostly local information, about a region and its neighbors. Model fitting uses global information, about the labels and locations of al regions in the image. Solutions tend to start with a few confidant regions, then grow to neighboring regions, and finally settle on a complete image interpretation. If there is an intersection In the scene, the model fitting module wilt detect a problem with the best fit and wil propose an additional road *model* to explain the discrepancy. This road finding expert system successfully labeled many difficult scenes, wifh much less geometric constraint than used in road following. It contributes not only to road finding but also explores a new paradigm in expert systems, atemating between labeling and model fitting*

The OPS-based road finding expert system Is further described by Taka Fujimori and Takeo Kanade in "Knowtodge-Based Interpretation of Outdoor Road Scenes."

Car recognition. We have created a program that recognizes cars in color images without the use of a strong 3-D model. The .program detects image features and groups of Image features (lines, trapezoids, ellipses), and uses approximate 2-D constraints. This is in contrast to the vast majority of programs which perform recognition based on image edges, and which require a precise model of the geometry of the lines in the object. The Landmark Acquisition SybSystem Improving Exploration (LASSIE) begins by extracting edges, and linking them to form lines with the Miwa line finder, developed at CMU. LASSIE then proceeds through a series of grouping steps, looking for parallel lines, parallel lines plus two others to complete a trapezoid, sets of trapezoids that share common edges, and so forth. It then hypothesizes interpretations of the groupings as windows and the roof of a car. Given a trial interpretation, it predicts where to search for wheels, lines from other parts of the car, and other features to confirm the match.

Karl Kluge and Hideki Kuga describe LASSIE in "Car Recognition for the CMU Navlab."

Geometric camera calibration. Cameras are notoriously hard to calibrate. First of all, it is difficult to measure lens and sensor distortions, which can vary from skewing and pin-cushion distortions, to rotations caused by misaligned CCD chip mounting. Beyond distortions, it is also difficult to get exact positions of the "center of focus" of a typical camera with compound lens. Most solutions either assume precise measurements that are not possible, or depend on nonlinear optimizations that are unreliable and do not always converge to the correct solution.

We have developed a reliable, simple method for camera calibration. By taking two pictures of a grid of points, it is possible to extract all the calibration parameters. The grid itself needs to be accurately measured, and its motion between the two pictures needs to be accurately known, but all other parameters are treated as unknowns. Thus, the output is not only the camera's lens parameters, but also its location in space and its orientation. Two different methods of extracting the parameters have been derived, a linear method that smooths errors in near-linear cameras, and a local method that is best for cameras with significant distortions across their field of view.

Camera calibration is described by Keith Gremban, Chuck Thorpe, and Takeo Kanade in the report "Geometric Camera Calibration Using Systems of Linear Equations."

Chronology

Mar: ERIM: System runs on the WARP.
Apr: WARP: Prototype installed on Navlab.
Apr: VISION: Road Following running on WARP.
Apr: WARP: WPE interface to GIL installed.
May: WARP/Navlab Demo runs at 50 cm/sec.
May: VISION: First OPS based road scene interpreted.
May: SIDEWALK II.V: Parallel execution of pipeline.
Jun: CODGER: Performance monitoring tools.
Jul: SIDEWALK III: Automatically switches between cameras.
Jul: SIDEWALK III: Speed changes for perception.
Jul: VISION: Ten OPs based road scenes interpreted.
Aug: VISION: Complete calibration model developed.
Aug: VISION: First car identified.
Aug: VISION: Segmentation of open & closed iris images.

Aug: NAVLAB: Power system upgrade.
Sep: VISION: Park intersection recognition.
Sep: WARP: GE machine installed on Navlab.
Oct: CODGER: Runs with uncertainty model.
Oct: VISION: New interpretation using hough space.
Nov: VISION: Reduced resolution digitization runs.
Dec: PARK 87: New path planner.
Dec: PARK 87: Map revision

Personnel

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

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

Staff: Paul Allen, Mike Blackwell, Tom Chen, Jill Crisman, Ralph Hyre, Bala Kumar, Jim Moody, Tom Palmeri, Jean-Christophe Robert, Eddie Wyatt

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

Graduate students: Karl Kluge, InSo Kweon, Doug Reece, Tony Stentz

Publications

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

Kevin Dowling, Rob Guzikowski, Jim Ladd, Henning Pangels, Jeff Singh, and William Whittaker.
NAVLAB: An Autonomous Navigation Testbed.
Technical Report CMU-RI-TR-87-24, Carnegie Mellon University,
The Robotics Institute, November, 1987.

Y. Goto and A. Stentz.
Mobile Robot Navigation: The CMU System.
IEEE Expert :44-54, Winter, 1987.

Martial Hebert, C.E. Thorpe, S.E. Dunn, J.M. Cushieri, P.O.
Rushfeldt, W.H. Girodet, and P. Schweizer.
A Feasibility Study for Long Range Autonomous Underwater Vehicle.
In Proceedings of Fifth International Symposium on Unmanned Untethered Submersible Technology, pages 1-13. University of New Hampshire, June, 1987.

K. Ikeuchi and T. Kanade.
Modeling sensor detectability and reliability in the sensor configuration space.
Technical Report CMU-CS-87-144, Carnegie-Mellon University, Computer Science Department, 1987.

Takeo Kanade (ed.).
Three-Dimensional Vision Systems.

Kluwer Academic Publishers, Boston, 1987.

Takeo Kanade.
Image Understanding Research at CMU.
In Proceedings of Image Understanding Workshop, pages 32-40. SAIC, Los Angeles, California, February, 1987.

T. Kanade and M. Fuhrman.
A Noncontact Optical Proximity Sensor for Measuring Surface Shape.
Three Dimensional Vision.
Kluwer Academic Publishers, Boston, 1987, pages 151-194.

Larry Matthies and Takeo Kanade.
The Cycle of Uncertainty and Constraint in Robot Perception.
To appear in Robotics Research 4 , 1987.

Larry Matthies and Steven A. Shafer.
Error Modeling in Stereo Navigation.
IEEE Journal on Robotics and Automation :239-248, June, 1987.

Matthies, R. Szeliski, and T. Kanade.
Kalman Filter-based Algorithms for Estimating Depth from Image Sequences.
Technical Report CMU-CS-87-185, Carnegie Mellon University, Computer Science Department, December, 1987.

T. Okada and T. Kanade.
A Three-Wheeled Self-Adjusting Vehicle in a Pipe, FERRET-1.
International Journal of Robotics Research 6(4):60-75, Winter, 1987.

D. Reece and S. Shafer.
An Overview of the PHAROS Traffic Simulator.
In Proc. of the 2nd International Conference on Road Safety. Groningen, The Netherlands, September, 1987.

Charles Thorpe, Martial Hebert, Takeo Kanade, and Steven Shafer.
Vision and Navigation for Carnegie Mellon Navlab.
Annual Review of Computer Science.
Annual Reviews Inc., California, 1987, pages 521-556.

C. Thorpe and T. Kanade.
1986 Year End Report for Road Following at Carnegie Mellon.
Technical Report CMU-RI-TR-87-11, Carnegie Mellon University, The Robotics Institute, May, 1987.

Color Vision for Road Following *

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

April 4, 1988

Abstract

This paper discusses the research conducted at CMU in several topic areas: increasing the dynamic range of current cameras to better handle outdoor scenes, detecting roads in heavily shadowed scenes and in rapidly changing illumination, and identifying intersections. This research is necessary to add new capabilities to our road following vision system. We built a new system, using the results from our research, that could identify roads and intersections in shaded scenes. We also transferred some of our algorithms to the WARP, an experimental supercomputer developed at CMU, to test the capabilities of current parallel machines.

1 Introduction

Our long-term objective is to create color vision systems capable of finding and following roads under all conditions. Our ambitions include modeling and perceiving dirt roads, city streets, and expressways; handling days with sun as well as rain, snow, or dark of stormy night; and both driving on familiar, well-mapped roads, or exploring new areas without the aid of a map.

In our navigation system, the vision module is just one of many modules. The vision module is responsible for detecting roads in the camera images, the pilot module uses the road position to plan a path for the vehicle to travel, a map of the world is updated by the map update module, and the helm module is responsible for sending driving commands to the vehicle. This paper is concerned only with the vision module and does not discuss the functions of the rest of our navigation system.

Our previous research [1] has built the PARK I system capable of driving our Navlab robot van, shown in Figure 1, on a narrow, twisting, asphalt path under a variety of lighting conditions, with no a priori map knowledge but with a known road model. During the past year, we have concentrated on new capabilities in four areas:

- increasing the dynamic range capabilities of our cameras to handle lighting extremes,
- decreasing our reliance on a fixed road model, to allow us to track roads that change and curve,
- using a map to predict intersection locations and shapes, thus allowing us to recognize intersections and successfully navigate a road network,
- using the CMU Warp supercomputer to increase the processing speed.

Section 2 of this paper discusses our PARK I system, including both a description of processing and a discussion of the points we decided to improve. Section 3 presents specific vision research towards our objectives. Several topics were researched in isolation, some of which were used in a PARK II system. In section 4, we describe the PARK II system incorporating the new vision algorithms, and its performance. This section also discusses the WARP implementation of our PARK I system, and the final section lays out the plan for our next round of research tasks.

"This research is sponsored by the Strategic Computing Initiative of DARPA, DoD, through ARPA Order 5351, and monitored by &e US Army Engineer Topographic Laboratories under contract DACA76-85-C-0003, tilled "Road Following."

2 Previous Work

2.1 The PARK I Demo System

The PARK I demo system digitizes images and tries to find the best straight road in the image, using color classification and a Hough voting scheme. This is done in five phases: System Interface, Pre-processing, Labeling, Interpretation, and Model Update. The data and control flow between the phases is shown in Figure 2. The interface phase tells the pre-processing phase when to digitize an image. The pre-processing phase transforms the camera data into reduced resolution color and texture images. These images are used by the labeling phase to produce a road probability image, in which each pixel describes how well the color and texture of that image pixel match known colors and textures of road pixels. This probability image is passed to the interpretation phase that finds the best straight road in the image. The road location is back-projected onto the ground plane and passed to the navigation system. The road location in the image plane is given to the model update phase where the vision module's idea of road and grass colors are updated. Each of these phases is described in more detail below.

System interface. The interface between the vision module and the navigation system defines the function of the vision module. The navigation system tells perception where the vehicle should be when it takes the next picture. The vision system continuously polls the navigation system for the vehicle position, and digitizes an image when the vehicle is closest to the specified location. The image is then processed to determine the road location in the image.

The required output of the vision system is the location of the road on the ground plane. The ground is assumed to be locally flat, with the vehicle sitting on the same plane as the road in the image. The coordinate frame of the vision output is defined by the vehicle location when the image is digitized as shown in Figure 2. The road is assumed to be straight and of known width. It can therefore be represented as a rectangle on the ground. Furthermore, this rectangle can be described completely by its center line. The location of this rectangle on the ground plane can be described by two parameters, (x, θ): the x position of the center line as it crosses the x axis, and the angle at which the center line crosses this axis. These two parameters are the description of the road that the vision module returns to the system.

Pre-processing. The pre-processing phase transforms the camera data into images that can be processed by the labeling phase. In our case, labeling cannot process the full size images from the camera due to time constraints. Therefore, the pre-processing phase first reduces the color images from (480 x 512) to (30 x 32) pixels. This is done in a series of averaging reduction stages that create a pyramid of reduced resolution color images.

Preprocessing also calculates a texture value for each pixel in the (30 X 32) image, to give the labeling phase another cue besides color. Texture is calculated by counting edges found by a high-resolution Robert's operator, normalized by local image intensity and large-scale edges. Normalizing by intensity assures that the texture operator will have similar responses to similar textures, regardless of the illumination of the textured region. Normalization by large-scale edges reduces the response of the texture operator to shadow edges. Grassy regions and trees tend to have many local variations, which produce many local edges, which in turn produce a strong response in the texture operator. Road surfaces are usually more evenly colored, and evoke less response from the texture operator.

At the end of this phase, the four (30 x 32) images representing red, green, blue, and texture, are passed to the labeling phase.

Labeling. The goal of labeling is to label each pixel as either road or grass, and to record a confidence that the pixel is correctly labeled. We compute the label and confidence by comparing each pixel's color and texture with multiple road and grass appearance models. Since sunny and shaded road have very different colors, we need a separate color model for each illumination. Moreover, since colors change gradually as the illumination changes from sunny to dark shadows, we have extra models to represent these intermediate colors. We found that the system ran well using four road models and four grass models.

Labeling uses these color models in a standard pattern recognition technique. At each pixel, a likelyhood is calculated for each model, and the model with the highest likelyhood is selected. The probability of this model is recorded in the road-probability image. Positive probabilities in this image signify that one of the road models had this highest probability, while negative pixels signify one of the grass models. The road-probability image is used by the interpretation phase.

Interpretation. The interpretation phase is responsible for transforming the road probabilities into a (x, θ) interpretation. This ground plane interpretation can be transformed into the image plane, resulting in an equivalent set of parameters: c, the column position where the road centerline crosses the horizon row, and ϕ, the angle with which the centerline crosses the horizon row. These two image plane parameters, (c, ϕ), are used in a Hough space interpretation scheme. Each road pixel votes, using its probability, into the Hough accumulator, for all road shapes that contain that road pixel. Each grass pixel votes, using its negative probability, against all road shapes that contain that grass pixel. The best road location in the image is given by the coordinates of the peak of the Hough space. This location is passed directly to the model update phase, and is transformed to the ground plane and sent to the navigation system interface for path planning and vehicle guidance.

Model Update. The model update phase is responsible for adjusting the statistical color model of the different road and grass models. This adjustment is necessary when the illumination conditions are changing, or if the road or grass colors change. The texture model is not adjusted, since texture remains constant regardless of the lighting or the road or grass colors. After the road is found by the labeling phase, we know exactly which pixels in the image are road pixels and which are grass pixels. Although we know the road/grass type of each pixel, we use only those road and grass pixels that are not near the road edge for updating the model. This prevents the corruption of the color model due to errors of modeling curved roads with straight road models. For each of the road and grass regions, we perform a nearest mean clustering technique for all of the data in the road and grass regions. This technique groups color data into a set of models where the color data in each individual model is described by color means and covariances. The clustering technique selects the set of models whose mean values are the most separated, and whose clusters are the most compact. The means and variances of the resulting classes are then used by the labeling phase to label the next image.

2.2 Lessons Learned from the PARK I Demo System

Under many conditions, the PARK I system was rather robust. On days when the light was fairly constant the vision module worked perfectly almost all of the time. In the times that the solution was imperfect, it was not far from the correct solution, so that the Navlab drove on the edge of the road instead of down the center. However, there were some limitations with this system that needed to be addressed.

Increased Dynamic Range. The need for an increased dynamic range in the camera is especially apparent on bright sunny days, when the vehicle is in the sunlight and the vision module is processing images containing dark shadows. In these dark shadows, the pixels have such low values that the road and grass pixel values are indistinguishable. Likewise, when the vehicle is in the shade, sunlit regions of the image are saturated, and again, road and grass are indistinguishable. We need some way of increasing the dynamic range of the perception system to handle these difficult scenes.

Changing Illumination. The labeling phase depends on knowing the colors of road and grass from the previous image. At the end of each step, we determine new road and grass colors, so that the system can adapt to changing conditions. When illumination changes gradually or when the system slowly enters shadowed regions from sunny regions, the system can update the color model and successfully track the road. In this sense, the system is capable of adapting to the environment. However, when the illumination changes rapidly, as when the sun goes behind a cloud, the color model used for labeling can be inaccurate, and the road can be misclassified. Therefore, we want reliable methods of distinguishing road from grass regions that do not solely rely on color models derived from the previous image.

Intersection Detection. As in any vision system, the interpretation phase needs a model for the system to match in the image. In the PARK I system, the model is implicitly coded as a straight road. We needed to add models and algorithms to interpret intersections and curving roads.

3 Research and Algorithm Development

To address the problems discussed above, and to expand our road following capabilities, we researched in the following topics independently:

- As an attempt to increase the dynamic range of the cameras, we developed a two camera pre-processing phase.
- Unsupervised learning was tried for dealing with rapidly changing lighting conditions.
- Two different ideas for recognizing intersections, a convolution interpretation scheme and a region searching algorithm, were developed.

The algorithms that were successfully developed and that met speed and accuracy requirements were integrated into a PARK II demo system.

3.1 Two Cameras

Since the dynamic range of available video cameras is not large enough for bright sunny days with dark shadows, we are trying a two camera system with fixed iris settings. One of the cameras has its iris open to peer into the shaded areas of the scene. The image from this camera is called the *bright* image. The other camera has a closed iris which is used for looking into sunlit areas. The closed iris image, the *dark* image, has an average intensity less than the open iris image. By combining the bright and dark images, we can indirectly increase the dynamic range of the sensor.

To test this idea, we replaced the pre-processing phase of the PARK I system with a new pre-processing phase that digitizes two color images and produces one reduced resolution color image and a texture image. These images receive some of their pixel data from the bright image, and some of their pixel data from the dark image. This combined image is fed into the labeling phase.

First the digitized images are reduced to form bright and dark color image pyramids. Next, each image pyramid is run separately through the texture subroutine, producing a bright and dark texture image. Then the two color images are combined, and the two texture images are combined. For each pixel in these combined images, we select a value either from the dark image or from the bright image. This is better than trying to do any sort of pixel-level averaging or combination since these techniques would distort color values of the images. So the combination step chooses, for each pixel, which image to use, and gets the red, green, blue, and texture values from that image. It also produces a mask image that records from which image, bright or dark, each pixel was selected.

For selecting which pixels should be placed in the combined images, we use a simple thresholding technique. We first apply a threshold to each pixel in the dark image. If its pixel value is less than the threshold value, then the data in the dark image is inadequate. Therefore, this combined pixel is copied from the bright image. If the dark image pixel value is greater than the threshold, then this pixel is usable, and is copied from the dark image into the combined image. A more sophisticated algorithm could perhaps improve system performance by dividing the image into sunny and shaded pixels instead of just dark and bright pixels. However, separating sunny from shaded may be just as difficult as the original problem of separating road from non-road.

By running this combination step, and carefully setting the threshold parameters, we are able to succeed in increasing the dynamic range of cameras to handle images containing dark shadows and bright sunlight. Unfortunately, these parameters need to be individually set for the conditions on each day that we run the vehicle. Therefore, we would like this threshold to adapt automatically. Even with the correct threshold, there are problems with the colors changing within a shadow and with rapid and large illumination changes, such as the sun going behind a cloud. Sunlit areas are fairly evenly illuminated, and the same surface type produces approximately the same image colors. But shaded areas are illuminated by reflections from the sky, from clouds, and from surroundings like trees and buildings. The color of the illumination, and thus the color of the image pixels, changes noticeably depending on the color of the reflecting objects. More research, including fundamental work in optics of materials, is needed to be able to properly classify road and nonroad under all illumination conditions.

3.2 Unsupervised Learning Labeling

When the lighting conditions change rapidly, our PARK I system fails, since it relies on color models from the previous image. Because illumination can change rapidly and unpredictably outdoors, models calculated from a previous image may not correspond to the colors in the next image. We would like to build a labeling phase which does not rely on color models from the previous image.

The input to this labeling module is the same as the PARK I system, except, of course, there is no input color model. The output should be a a set of regions in the image, a collection of which would completely and exclusively cover the road in the image. Our basic approach is to use a standard clustering algorithm to group regions that have similar colors [2].

To run this clustering algorithm, the number of classes in the image must be pre-selected. First the pixels of the image are divided randomly into each of the classes. The mean color value is computed for each class. Next, each pixel is labeled as belonging to the class whose mean value is closest to its pixel value. The mean color is then recomputed for each class. The sequence of *label by closest mean* and *calculate mean* is repeated until few or no pixels change their class. In the first labeling step, the majority of pixels change their class, however, in the second labeling step, only a minority of the pixels change. In successive steps, the number of pixels that change classes continues to decrease. If run long enough, this will converge until no pixels change; in practice, three iterations is adequate. Now all of the adjacent pixels, having the same class label, can be grouped into regions using a connected components algorithm.

The biggest problem with this procedure is that the number of regions found in the image is quite large. Searching for a road shape becomes impractical if there are too many regions. Simple methods for reducing the number of regions, such as shrinking and growing or merging small regions, either still left too many regions or distorted the region contours to too great an extent. For unsupervised labeling to be applicable to real time road following, we need to improve both the labeling and the searching methods (see Section3.4 below). This segmentation system is not directly applicable in our PARK I system, since its interpretation phase depends on road/grass labeled pixels. There are no semantic labels associated with the unsupervised learning scheme. We are in the process of developing interpretation methods for this scheme.

3.3 Interpretation Using Convolution

One of the main reasons for the robustness of the PARK I system was the road model fitting. We used a Hough transform to globally calculate most likely position of the road, regardless of local misclassifications.

We want to retain that robustness, but extend the class of shapes from just straight roads, in the earlier system, to also include curved roads and intersections. For these more complicated shapes we replaced Hough transforms with convolution, using a template mask in the shape of the predicted road or intersection.

Each road segment or intersection is modeled, based on either map data or previous scene interpretations, as a set of polygons. The vehicle planning system chooses the most likely set of roads and intersections in the field of view, and passes those shape descriptions to the vision module. The vision module does not directly search for those shapes in the image. As the distance changes between the vehicle and an intersection, for instance, perspective projection will change not only the location of the intersection in the image but also its size and shape. So instead the pixels from the image are projected onto the ground, and the search for the predicted shape is carried out in ground coordinates.

The first input to this system is a probability image which contains positive probabilities for road, and negative probabilities for grass. The magnitudes of these probabilities correspond to the confidence that that pixel is actually road or grass. The second input is a shape description of the expected road or intersection in the scene. The output of this system is the most likely location of the specified road shape.

First, this algorithm back projects the road probability image onto the assumed ground plane. Then it convolves a mask formed from the description of the road or intersection with the ground plane probability image. To find a straight road, only the x dimension and the angle of the road need be found to describe the position of the road, since the appearance does not change along the y direction. However, to locate an intersection, the (x, y, θ) position of the intersection on the ground plane are needed.

To test this idea, we replaced the interpretation module of the PARK I system. We provided test models for the roads and intersections by selecting the borders of the road in our test images. These points are back-projected onto the ground plane, giving us a perfect model of the intersection seen in the image. The results are very good, matching the road in almost all situations. Matching with distorted models, or with models picked from a different view of the same scene, gave less accurate but still satisfactory results.

3.4 Region Searching Interpretation

The main motivation for this algorithm is to apply road information, such as edge shape, color, road shape, and predicted location, to help decide the location of the road in the image. This scheme frees us from the exact shape description that is required by the convolution interpretation scheme discussed above. In real navigational situations, an exact model, in general, is unknown. This algorithm is an attempt to interpret road and intersection scenes using general constraints, without knowing the exact shape of the roads.

The input to this algorithm is a list of regions. Each region includes the following information: road/grass label, road probability, size, neighboring regions, and polygonal approximations. The output is the collection of regions that make the *best* road. Best, in this case, is evaluated using geometric and heuristic information. We search various combinations of regions in the image to see which collection forms the best road. Each collection of regions is called the candidate road. An exhaustive search over all candidate roads would be much too expensive, so we use a *Hill Climbing* algorithm to constrain the search space.

This algorithm is tested by replacing the interpretation phase of the PARK I system. In the PARK I system, the labeling phase labels pixels in the image. This algorithm needs these pixels to be grouped into regions, so we added a region growing step to the labeling phase to test this algorithm.

Region Growing. Before we can apply the region searching technique, we first need to convert the classification and probability images into a list of regions. This is done in the following steps:

1. Use a connected components algorithm for region extraction.
2. Merge each small region into its most similar neighbor.
3. Approximate the remaining regions with polygons, retaining neighbor information.
4. Calculate descriptors, such as size and road/grass probability, for each region.

Region Searching Algorithm. Searching starts by considering the initial candidate road, C_0 consisting of the regions that were labeled as road in the input image. We first evaluate the cost, ν of the initial candidate C_0. The algorithm proceeds by either adding neighboring grass regions to the candidate road or deleting road border regions from the candidate road. A neighboring grass region touches the candidate road, and the road border regions touch at least one grass region. The algorithm can be described as follows:

1. *Try expanding road* by evaluating the costs of all the candidate roads that can be formed by adding one neighboring grass region to the initial candidate road, C_i. Remember which of the grass regions G, when added to the initial candidate, gave the lowest cost of ν_G.
2. Likewise, *try shrinking road* by evaluating the costs of all of the candidate roads that can be formed by deleting one of the road border regions from the initial candidate road, C_i. Remember which of the road regions R, when deleted from the initial candidate, gave the lowest cost of ν_R.
3. If ν_R is lower than ν_G and ν, then removing one region from the road improves the road model fit. $C_{i+1} = C_i - R$ and $\nu = \nu_R$. Go to 1.
4. If ν_G is lower than ν_R and ν, then adding a grass region to the road improves the model fit. $C_{i+1} = C_i + G$ and $\nu = \nu_G$. Go to 1.

5. Else ν is lower than ν_R and ν_G, switching the label of any one region will not lower the cost, so **exit**.

The final result region C_{final} is the collection of regions in the image that form the best road.

Evaluation of Candidate Roads. The confidence of a road interpretation can be determined by a combination of the following constraints. Notice that each constraint tries to enforce a particular attribute.

- The *confidence* measure comes from color classification of the pixels, and their resulting road or grass probabilities. It prefers regions that have high road probabilities.
- The *edge straightness* metric prefers candidate roads whose edges form straight lines.
- The *road width* constraint reinforces candidate roads that have the correct width.
- The *parallelness* measure supports candidate road that have parallel edges.
- The *prediction* constraint prefers candidate roads that are spatially close to the prediction.

Any deviation from the ideal for each of these features adds to the cost. We form a weighted linear combination of these costs to calculate the total cost of the candidate road, ν. The candidate road that is selected by the region searching algorithm will have the best compromise between all of these constraints. This cost function can be easily expanded if additional constraints are needed and available.

Results. This approach is very promising and powerful. When given good predictions, it successfully classifies all our test images, except the single worst shaded intersection image. Its main drawback, and the reason it not incorporated into our demo system, is that it does not guarantee real-time performance. In clean scenes, relatively few regions are produced by pixel labeling and region growing, and the initial candidate road is nearly correct. Then the search proceeds quickly, goes through few iterations of generating and evaluating candidate roads, and is quite efficient. In more complex situations, such as dappled sunlight or leaves scattered on the road, it is possible to produce literally hundreds of separate regions. Searching over all those regions is not very efficient. Merging small regions into their larger neighbors helps with search time, but at the expense of distorting region boundaries and possibly corrupting the solution. This approach requires, and merits, further work before it is practical in real-time road following, and to enable it to not to rely on good predictions.

4 Demo Systems

During the course of the year, we built two demo systems to test new capabilities of vision and the rest of the Navlab navigation system. The first system was a reimplementation of the PARK I system using the WARP machine. The second system was built to test some of the new algorithms discussed above.

4.1 Warp Demo System

The PARK I system required about 10 seconds to complete one vision cycle on a Sun 3/160. This is much slower than we would eventually like. Therefore, we wanted to run the PARK I system on the prototype wire wrapped WARP machine, to get an idea of the speed we could get from the current supercomputer technology. A prototype WARP was installed on the Navlab for this purpose [3]. The WARP is an experimental high speed computer consisting of

- an array of four cell processors that can perform parallel computations,
- an interface unit to transfer data to and from the cell array,
- three 68020 processors to perform higher level data processing and sequencing, and
- a host computer that links high level programs to the WARP machine.

The WARP group provided software support to make this demo possible. Programs for the WARP machine are specified in an Ada-like language called W2 [4] [5]. The W2 compiler produces code that runs in parallel on the cells of the Warp. The warp_call() interface allows a C program running on a Sun host to call W2 functions on the parallel processing cells, and automatically handles all data transfer and format conversions.

We divided the vision program into modules that were consistent with the function of the WARP:

- Roberts' gradient operator
- Texture operator
- Color and texture classification
- Statistics collection
- Statistics class adjustment
- Hough space voting

All of these modules, except for the Hough module, were implemented by dividing the input images into column swatches. Each of the cells of the WARP array was responsible for one part of the image data. The results were then merged when they were read from the array. For the Hough space module, the Hough space accumulator was divided among the cells by column swatches. The input image was passed to all of the cells, therefore, each cell would look at each pixel in the input image and vote for all the road locations within the range of angles described by its swatch of the Hough accumulator.

Receding our algorithms for the Warp succeeded, and even on code that was not designed for parallel processing we achieved a speedup of 2.5, from 10 seconds per image to 4. We learned several practical lessons:

- Designing for parallelism in the first place would have saved us much more time. The Warp is much better at some things, such as regular computations, and not so fast at others, such as conditional branching. Efficient use of the WARP requires designing algorithms with these constraints in mind.
- Even with the computing power of the Warp, standard coding optimizations, such as unrolling loops and folding constants, are still important time savers.
- The prototype Warp did not support constructs, such as variable loop bounds, so we had to pad data to maximum sizes, thereby wasting computation.

The production version of the Warp, which we have since mounted on the Navlab, helps solve those problems.

42 Park System II Vision Module

The PARK II demo system tested some of our experimental algorithms and tested the new PC WARP. Specifically, we wanted a system that used the two camera pre-processing phase and the convolution interpretation algorithm. The vision system would then be capable of recognizing intersections in even strong shadows.

An outline of this system can be seen in Figure 3. This system is based on the PARK I system with some important changes. The pre-processing phase is replaced with the two camera system that was discussed in section 3.1. The labeling phase of the PARK I system then processes the combined images. The interpretation phase of this system is the convolution interpretation scheme discussed in section 33. The location of the road is then reported to the navigation system through the new interface phase, which is modified to transfer intersection descriptions. The polygonal model of the road or intersection is passed to the model update phase, which updates the bright and dark color models used for classifying the next image. The more detailed description of the modifications to each of the modules is given below.

Calibration. We expected to need calibration between the dark and bright cameras. By mounting the cameras as close to each other as possible, and by careful bore sighting, we were able to get nearly perfect alignment For the ranges of interest;, and at reduced resolution, typical errors between the two cameras were less than a pixel

Calibrating the geometry between a camera and the vehicle became more important for this system. In the PARK I system, vision only reported locations of straight roads, so calibration of distances along the road was not crucial. In **the** PARK II system, vision had to report intersection locations as well, which required careful calibration of all camera parameters. We developed and used a calibration scheme that uses two different views of a grid of spots to deduce camera geometry [6]*

Resolution. To identify intersections at reasonable distances, the (30 x 32) resolution of the road probability images was not high enough. The branches of the intersection were typically smoothed over by the image reduction. Therefore, we increased the image size to (60 x 64). We also developed a subroutine to digitize from both of the cameras almost simultaneously by digitizing at reduced resolution into separate portions of the frame buffer on successive camera frames. Our input color images were therefore (240 x 256) rather than (480 x 512).

Convolution Interpretation. The convolution algorithm is used for the interpretation phase of this system. The navigation system predicts the shape of the road or intersection in the field of view, or of multiple possible objects if vehicle position is imprecisely known. Each shape is matched to the road seen in the image. The object with the best matching shape is returned as the identified object, and its best fit position in the image is projected onto the ground plane and returned to the navigation system. This shape description is also passed to the model update phase.

The location of a road can be described with two parameters, namely (x, θ). To describe the location of an intersection we need three parameters (x, y, θ). Unfortunately, when the dimension of the convolution increases, the computation increases exponentially. In order to reduce the computation time for intersections, we use an on board gyrocompass to localize the θ dimension, thus reducing the intersection convolution back to two dimensions, (x, y). The navigation system reads the gyro information, properly rotates the intersection model before passing it to the interface phase. The vision module then matches the image data to the intersection model, and the properly translated model is returned to the navigation system.

Results. The system ran successfully through the three intersections on our test course. Each of the intersections had different shapes, and two of the intersections were in shaded areas of the site. The first intersection was located on a curved section of the path, and the branch exiting on the left was comprised of a different material and was narrower than the two other branches. The second intersection was Y shaped, located at the crest of a hill. The third intersection was a T intersection where the approaching branch, the upright of the T, was angled.

Two of the results of our system are shown in Figure 4, one showing the algorithm for heavily shaded road interpretation, and the other showing intersection detection. Each result is shown in a cluster of four images. The top two are the bright and dark images from the cameras. The lower left is the combined image. The result of the segmentation is shown in the lower right. The position of the detected road or intersection is drawn on top of the dark image in the upper left.

We made many runs, including runs in a variety of lighting conditions and longer runs than any of our previous systems made. The navigation system used perceived roads not only for navigation but also for map building, and used intersections as landmarks. The final maps showed all of the bends and curves in our test course, and correctly located the intersections. Along with our successes, we also discovered areas for further work.

- **Shape descriptions** It is impractical to depend on exact shape descriptions of intersections. Moreover, if the shape descriptions are significantly incorrect, not only is the calculated location somewhat off, but the statistics update phase may incorrectly update its model of road and nonroad colors, leading to misclassifications in later scenes.
- **Assumed ground plane** The ground plane assumption is unusable for the second intersection of our test site, where the intersection lies on the top of a hill. The perceived shape of the intersection changes dramatically as the vehicle pitches forward at the top of the hill.
- **Gyro problems** We had difficulty using the gyro to orient the intersections. This was the first system to attempt using the gyro on the Navlab. The device would drift in orientation to the point that dead reckoning of the vehicle location provided better orientation information than did the gyro. Therefore, when matching the intersection models to the intersections, we would often receive models incorrectly oriented from the navigation system. Future systems will have to have better heading information, or will have to search for intersections in x, y, and θ.

5 Future and Continuing Work

Our current research focuses on five areas: increasing camera dynamic range to an even greater extent, dealing with shadow fringes, using approximate shape models, getting improved ground plane models from direct 3-D sensors, and improving unsupervised classification labeling.

Because our two cameras had a fixed iris setting, large changes in illumination could cause problems. When the sun went behind a cloud, in some instances even the *bright* image was not bright enough to see into the shadows. We are investigating computer controlled irises and computer controlled digitizer gains to try to expand further the dynamic range of the vision system.

Shadow fringes in conjunction with changing illumination also provide problems. The pixels that are most often misclassified are those at the fringes of shadows, where the illumination changes from bright to dark. These mixed pixels may not belong unambiguously to any one color class. In images where there is a sharp shadow line, this causes problems in only a few pixels. In other images, however, with dappled sunlight filtering through trees, a large portion of the image is composed of shadow fringe, and many pixels may be misclassified. We are working on methods to detect shadow fringes and discard those unreliable labels.

In the current system, an exact intersection model is required by the interpretation phase of the vision system. However, exact models are not realistic in real navigational situations. They are also impractical even if the exact shape is measured. Therefore we would like to continue our research into the region searching interpretation method. A parallel effort in our project has developed an expert system image interpreter that finds roads and intersections in very difficult scenes and which has only weak shape models [7]. This expert system, written in OPS, takes up to a half hour per image. If we can extract the most powerful heuristics from this system and combine them with our experiments in region-based road finding, we can perhaps build a powerful yet still not too slow system.

We would also like to improve our ground plane assumption in the vision system. In order to construct a ground surface, we need three-dimensional information about the ground surface. Another effort in our project is using a scanning laser range finder to build 3-D terrain maps. We are looking at methods for merging this 3-D data with our color images and using the combined data to give true intersection or road shape and location.

We are also continuing our work on the unsupervised classification method. This labeling algorithm should be able to handle changing illuminations. The remaining problems are that the labeling scheme results in far too many regions to be reasonably processed in real-time. We hope to develop some better interpretation methods or region reduction algorithms.

6 Conclusions

Our road following algorithms have made significant improvements, but there remains much work before we achieve our goal of reliable, autonomous, road following. Specifically we believe the following:

1. Using two cameras with different apertures helps significantly with the problem of small dynamic range. We continue to work on problems of rapidly changing illumination, and on changes of color at shadow fringes and within shadows.
2. Convolving an intersection template mask with classified image pixels, projected onto the ground plane, works for recognizing intersections. We continue to research methods for finding intersections without exact shape models.
3. Region based road finding methods can help by using not just individual pixel probabilities by other geometry, such as edge straightness an parallel edges. Such heuristics are powerful but time consuming.
4. Unsupervised classification has promise for producing cleaner regions for region based methods. Since it does not rely on matching with known color models, unsupervised clustering correctly models the colors in each image, and produces cleaner region boundaries.
5. Other non-color factors are important, such as fusion with three-dimensional data to get accurate ground shape models and planning for parallel implementation.

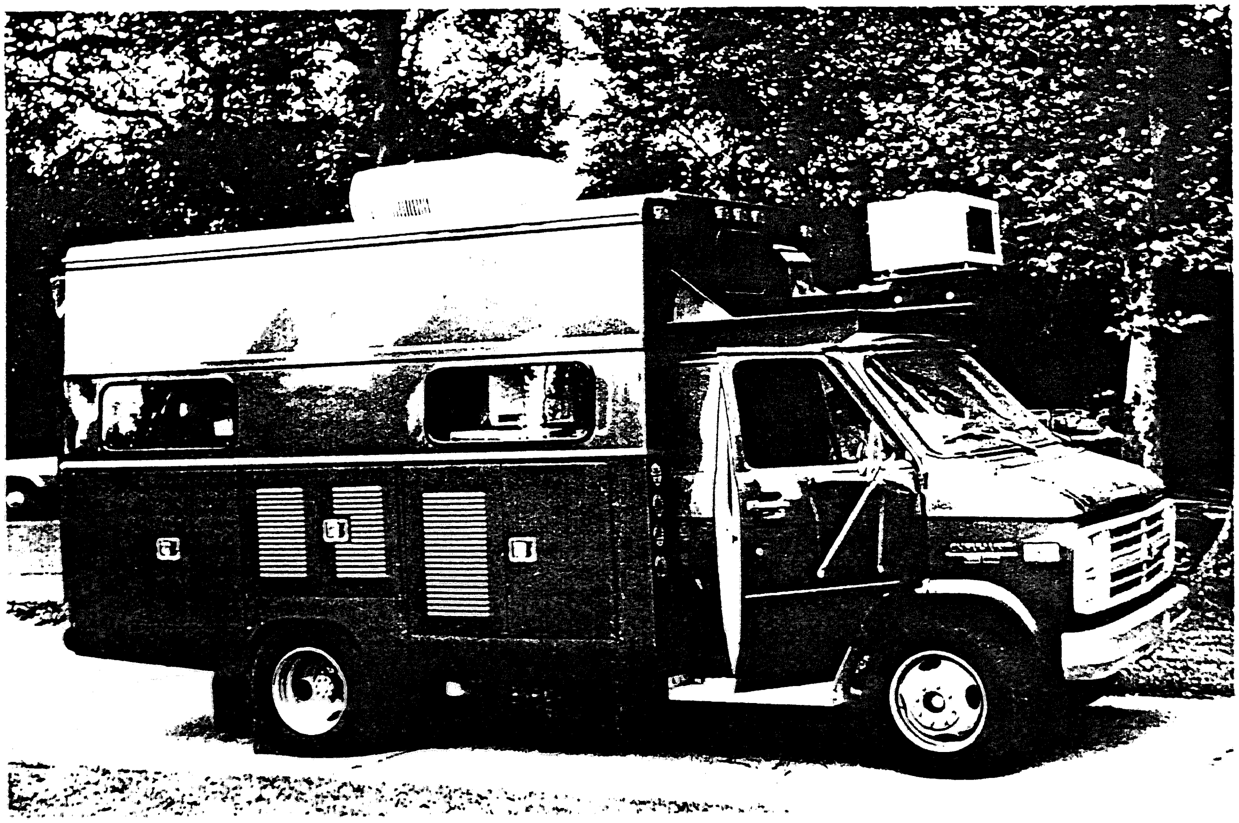


Figure 1: The Navlab Vehicle

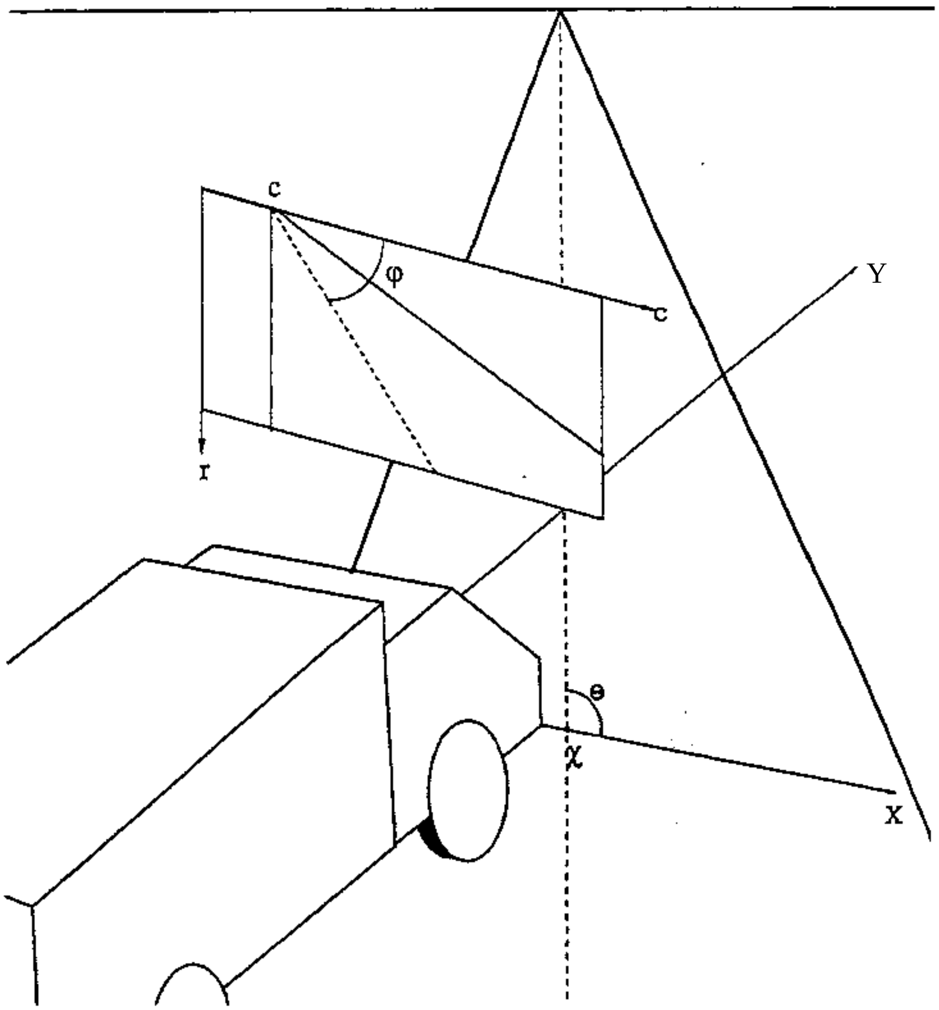


Figure 2: The PARK I System and the Vehicle Coordinate Frame

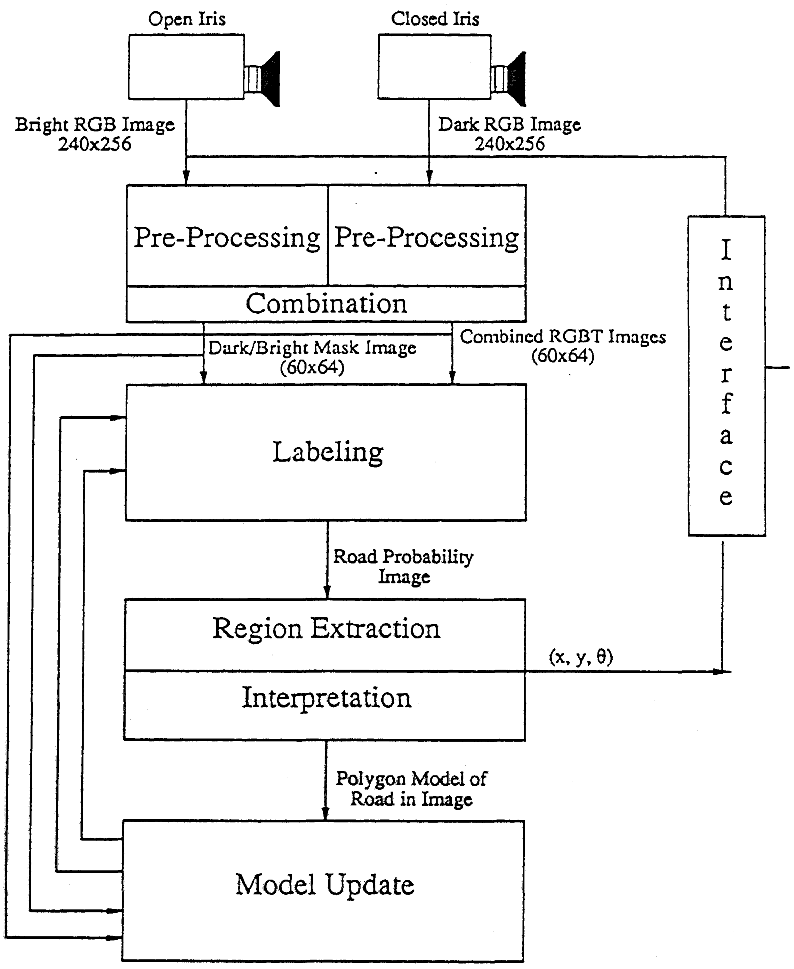

Open Iris
Closed Iris
Bright RGB Image 240x256
Dark RGB Image 240x256
Pre-Processing
Pre-Processing
Combination
Dark/Bright Mask Image (60x64)
Combined RGBT Images (60x64)
Interface
Labeling
Road Probability Image
Region Extraction
(x, y, θ)
Interpretation
Polygon Model of Road in Image
Model Update

Figure 3: The PARK II System

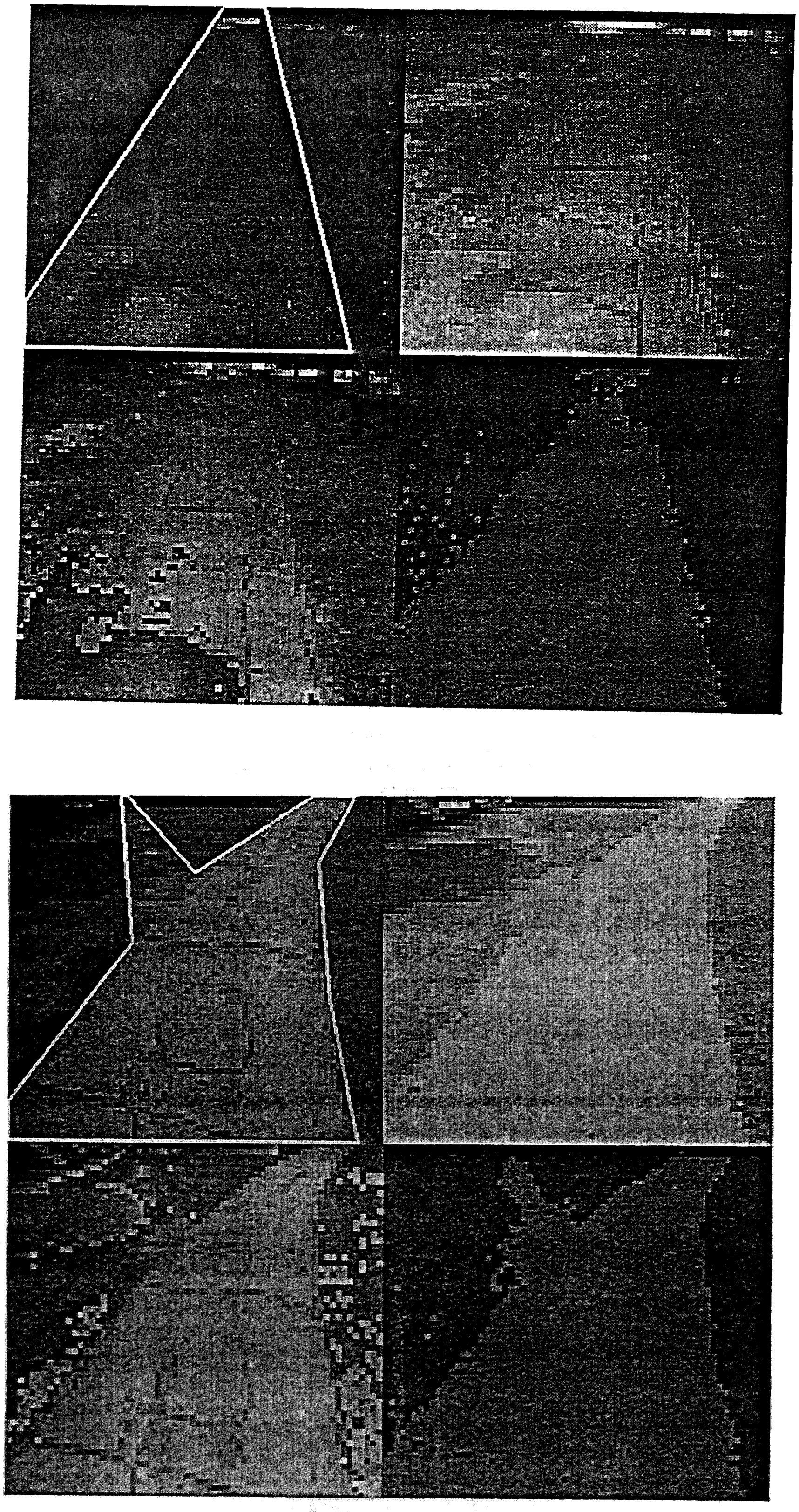


Figure 4: Results of PARK II System

References

[1] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Vision and navigation for the Carnegie-Mellon Navlab. *Annual Review of Computer Science*, 2:521–556, 1987.

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

[3] M. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Menzilcioglu, and J. Webb. The Warp computer: architecture, implementation, and performance. *IEEE Transactions on Computers*, C-36:1523–1538, December 1987.

[4] Monica Sin-Ling Lam. *A Systolic Array Optimizing Compiler*. PhD thesis, Carnegie-Mellon University, May 1987.

[5] T. Gross and M. Lam. Compilation for a high-performance systolic array. In *Proceedings of the SIGPLAN 86 Symposium on Compiler Construction*, pages 27–38, ACM SIGPLAN, June 1986.

[6] K. Gremban, C. Thorpe, and T. Kanade. Geometric camera calibration using systems of linear equations. *IEEE International Conference on Robotics and Automation*, 1988.

[7] T. Fujimori and T. Kanade. *Knowledge-Based Interpretation of Outdoor Road Scenes*. Technical Report, Department of Computer Science, Carnegie-Mellon University, 1988. In preparation.

3-D VISION FOR OUTDOOR NAVIGATION BY AN AUTONOMOUS VEHICLE

Martial Hebert Takeo Kanade

Carnegie Mellon University The Robotics Institute
5000 Forbes Avenue, Pittsburgh PA 15213

Abstract

This paper reports progress in range image analysis for autonomous navigation in outdoor environments. The goal of our work is to use range data from an ERIM laser range finder to build a three-dimensional description of the environment. We describe techniques for building both low-level description, such as obstacle maps or terrain maps, as well as higher level description using model-based object recognition. We have integrated these techniques in the NAVLAB system [10].

1. Introduction

Research in robotics has recently focused on the field of autonomous vehicles, that is mobile units that can navigate under computer control based upon sensory information. Several components are involved in the design of such a system. A high level cognitive module is in charge of making decision based on the perceived environment and the mission to be carried out. Sensory modules, such as video image analysis or range data analysis, transform the sensors' output into a compact description that can be used by the decision-making modules. Low-level control software converts decisions into actions performed by the hardware.

In this paper, we focus on one type of sensory component, the analysis of range data for an autonomous vehicle navigating in an environment with features such as trees, uneven terrain, and man-made objects. In particular, we study the use of the Environmental Research Institute of Michigan (ERIM) laser range finder to perform four tasks: obstacle detection, surface description, terrain map building, and object recognition (Fig. 1-1). Obstacle detection is the minimum capability required by an autonomous vehicle in order to navigate safely. Surface description is needed when the obstacle detection is not sufficient for safe navigation, in the case of uneven terrain for example, or when a more accurate description of the environment is needed, *e.g.* for object recognition. Terrain map building is the process by which surface descriptions from different vantage points are accumulated in a consistent map. Object recognition capabilities are required when the vehicle must recognize and locate known landmarks, *e.g.* a traffic sign, in order to carry out its mission.

This research was sponsored in part by DARPA monitored by the U.S. Army Engineer Topographic Lab under contract DACA76-85-C-0003, by DARPA monitored by the Air Force Avionics Laboratory under contract F33615-87-C-1499, and by NSF under contract DCR-8604199.

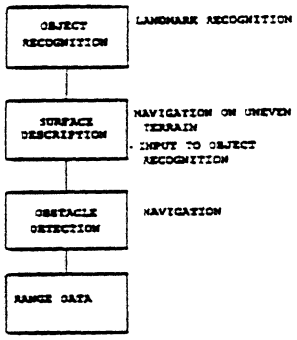


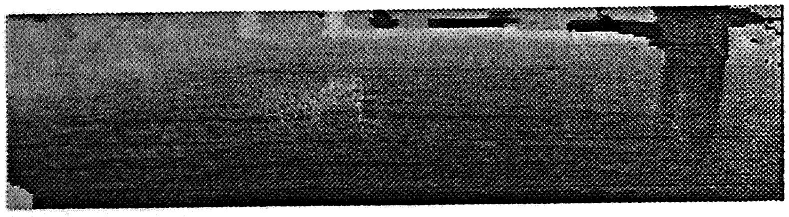
Figure 1-1: Range data processing

2. Intermediate representations of range data

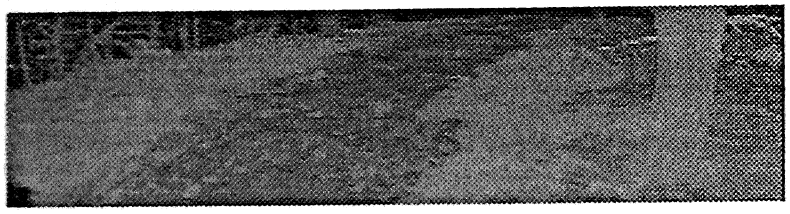
2.1. Sensor data

The ERIM sensor derives the range at each point by measuring the difference of phase between a modulated laser beam and reflection from the scene. A two-mirrors scanning mechanism directs the beam onto the scene so that an image of the scene is produced. In the ERIM-ALV version, the field of view is ±40 in the horizontal plane and 30 in the vertical plane, from 15 to 45 . The resulting range image is a 64 × 256 8-bit image. The frame rate is currently two images per second. The nominal range noise is 0.4 feet at 50 feet. The sensor also produces reflectance images in which the value of each pixel is the amount of light reflected by the target. Figure 2-1 shows an example of a range image and the corresponding reflectance image.

The ERIM sensor presents some limitations: Since only the phase difference is measured, the range values are known only *modulo* 64 feet. This causes ambiguity in the range data. The resolution degrades rapidly as the range increases. This is due to the divergence of the beam, which produces larger footprints as the distance increases, and to the scanning mechanism. Since the scanning angles are discretized using constant increments, the density of points decreases as the range increases. Points may be incorrectly measured at the edges of objects due to multiple reflections of the beam. This effect is common to all active scanning techniques and is known as the "mixed points" problem. We have found that applying a median filter to the original image eliminates most mixed points.



(a) Range image: the darker pixels are closest.



(b) Reflectance image.

Figure 2-1: An example of range and reflectance images

2.2. Vehicle centered coordinates

The raw data from the ERIM scanner represent range as a function of angles θ and ϕ of the two mirrors.

As shown in Figure 2-2, we assume that a coordinate frame $\vec{u}, \vec{v}, \vec{w}$ is attached to the scanner. We use another coordinate frame, the "vehicle" frame, $\vec{i}, \vec{j}, \vec{k}$ to express the measured points so that the resulting values are vehicle-centered and are therefore independent of the sensor configuration. We can thus derive the coordinates (x, y, z) of the point measured at pixel (row, col) with range D. If ϕ is the angle between $(\vec{u}, \vec{v})$ and the direction of the measured beam $\vec{d}$ at pixel i, j, θ is the angle between $(\vec{u}, \vec{w})$ and the direction of the measured beam $\vec{d}$ at pixel i, j, ϕ_s is the starting vertical scanning angle, $\Delta\theta$ and $\Delta\phi$ are the angular increments, and τ is the tilt angle of the scanner, that is the angle between the planes $(\vec{i}, \vec{j})$ and $(\vec{u}, \vec{v})$ as shown in Figure 2-2, then the conversion is:

$$\theta = (j - 128) \times \Delta\theta$$
$$\phi = \phi_s - i \times \Delta\theta$$
$$x = D(\cos\theta(\cos\tau\cos\phi - \sin\tau\sin\phi))$$
$$y = D\sin\theta$$
$$z = D(\cos\theta(\sin\tau\cos\phi + \cos\tau\sin\phi))$$

Figure 2-3 shows the data of the image of Figure 2-1 after conversion to vehicle coordinates as viewed in the direction of the $\vec{w}$.

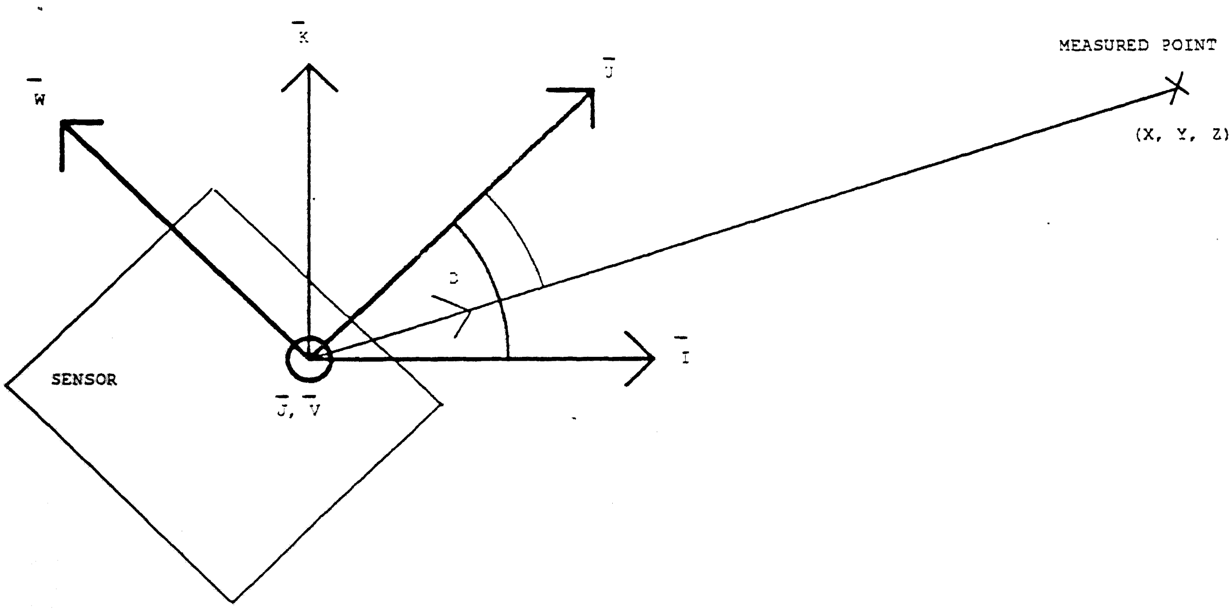


Figure 2-2: Conversion to vehicle coordinates

2.3. Bucket map

In outdoor environments, the ground plane $(\vec{i}, \vec{j})$ has a privileged role: the terrain can be modeled as a function $z = f(x,y)$, x and y being the coordinates on the ground plane. In order to take advantage of the ground plane, we used a intermediate representation, the bucket map.

A bucket map is defined by a regular grid on a reference horizontal plane. Each cell of the grid, or bucket, contains a set of measured points as shown on Figure 2-4. The points within a bucket may all be from the same image or from several consecutive images. The size of a bucket is typically 30 cms x 30 cms.

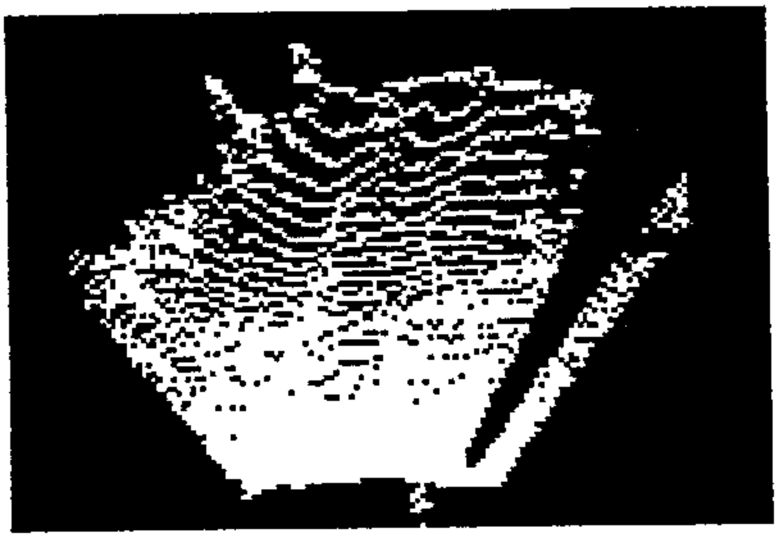


Figure 2-3: Overhead view of the data of Figure 2-1

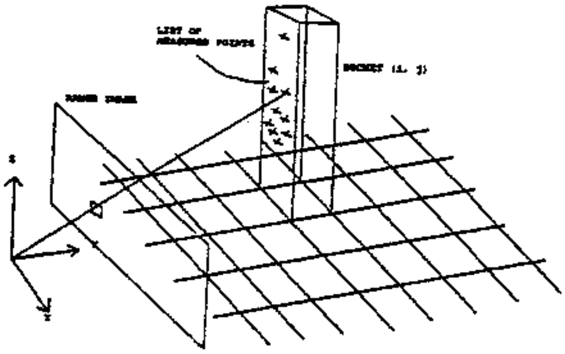


Figure 2-4: The bucket map structure

3. Obstacle detection

The first task of range data analysis for navigation, is to report the portions of the environment that are potentially hazardous. We must identify individual objects in the environment that the vehicle must avoid- Most obstacle detection algorithms combine surface discontinuities an! surface slope (6 11] to extract untravcrsable regions in the image using a vehicle model [3]. Faster algorithms use aprioxi knowledge of the terrain* *e.g.* flat ground assumption, by computing the difference between the range image and the expected kicai image [4], Since we want to be able to navigate in a variety of environments, we chose the first approach wfakh* although computationally expensive, allows us to handle uneven terrains. Specifically, we identify points in the bucket map at which the elevation exhibits a large discontinuity and points at which the surface slope is above a given threshold The first set of poixuv corresponds; to *th&* edges of the objects, the second lies within the surface of an object facing the sensor. The obstacle decoction algorithm is divided *km* four steps:

1. Detect elevation dlsecmtiiiutiies on *ib&* bucket map. The discontinuities are computed in 2x2 windows around each point;
2. CoragHUB the surface nornwi for each bucket;
3. Detect fee backets for which the surface normal whose angle with the vertical direction is greater than a given **threshold;**
4. Extract cotmectod regions from the ret of buckets detected at step 3 so that each region corresponds to an object Two buckets arc connected if itiey do not cross the line of elevation discontinuity.

The rationale for using two criteria, elevation and surface normals, is that the surface normals are meaningless at the edge of an object, and the elevation cannot be used .lionc without some knowledge of a ground-plane on which objects are known to rest

One possible undesirable result of the detection algorithm is that a small portion of the terrain might be reported as an obstacle due to the resolution of the bucket map. This error can be corrected only when a more complete object description is created. It does not, however, significantly affect the behavior of the vehicle since only small regions are involved.

For the purpose of obstacle avoidance, the detected objects are represented by polygons on the ground surface in order to be used by a path planner. Figure 3-1 shows a range image, the location of the buckets classified as parts of objects, and the polygonal representation of the obstacle map. The squares indicate the buckets in which the objects have been found. The large polygon enclosing the map is the boundary of the portion of the environment seen by the sensor.

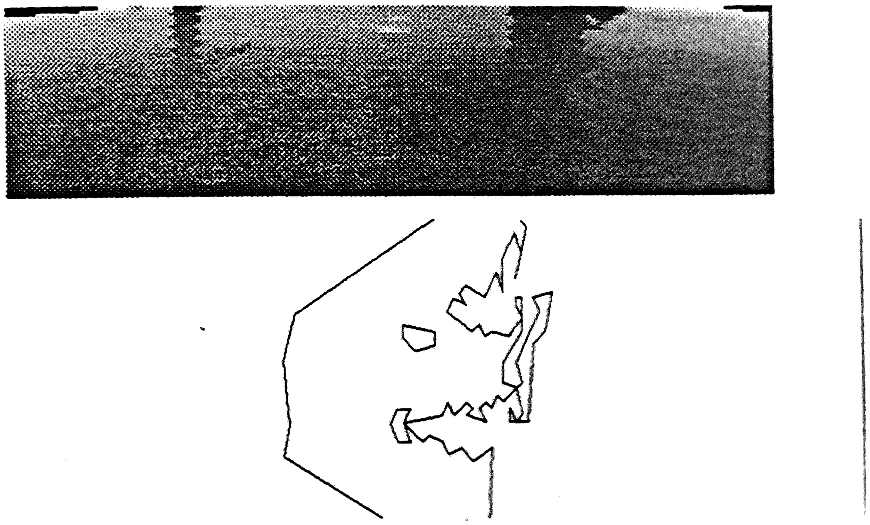


Figure 3-1: Obstacle Detection

4. Surface description

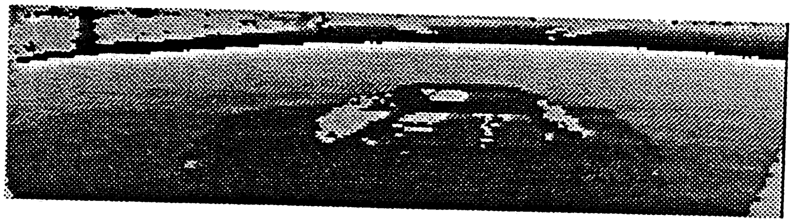
The obstacle detection algorithm is sufficient for vehicle navigation in a simple environment which includes only a smooth terrain and discrete obstacles. A typical example of such an environment occurs in road following applications. We need a more sophisticated representation in two cases:

- The surrounding terrain is uneven. In that case, part of the environment that may be hazardous or costly to navigate cannot be described as discrete objects.
- The mission requires the recognition of specific objects given apriori models. In that case, the mere knowledge of the existence and position of an object in the world is not sufficient, we need a more detailed description of its shape.

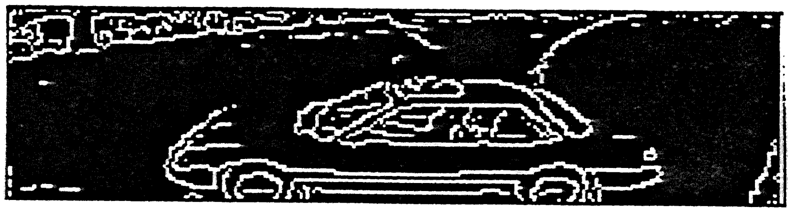
We describe surfaces by a set of connected surface patches. Each patch corresponds to a smooth portion of the surface and is approximated by a parameterized surface. In addition to the parameters and the neighbors, each region has two uncertainty factors: σ_a, and σ_d. σ_a is the variance of the angle between the measured surface normal and the surface normal of the approximating surface at each point. σ_d is the variance of the distance between the measured points and the approximating surface. Those two attributes are used in the object recognition algorithm.

The surface description is obtained by segmenting the range image into regions. Several schemes for range image segmentation have been proposed in previous work [1]. These techniques are based either on clustering in some parameter space, or region growing using smoothness criteria of the surface. We chose to combine both approaches into a single segmentation algorithm. The algorithm first attempts to find groups of points that belong to the same surface, and then uses these groups as seeds for region growing, so that 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 the least-squares sense.

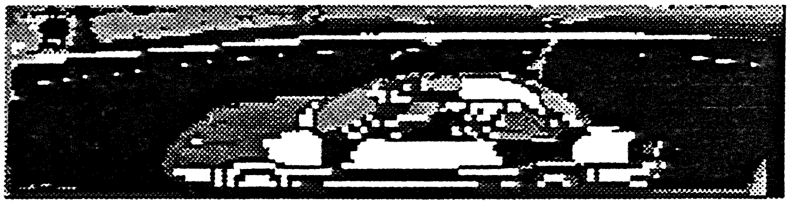
The strategy for expanding a region is to merge the best point at the boundary at each step. This strategy guarantees a near optimal segmentation. It has, however, two major drawbacks however: it may be computationally expensive, and it may lead to errors due to sensor errors on isolated points, such as mixed points. To alleviate these problems, we use a multi-resolution approach. We first apply the segmentation to a reduced image in which each pixel corresponds to a $n \times n$ window in the original image, n being the reduction factor. This first, low-resolution, step produces a conservative description of the image (Fig. 4-1.c). The low-resolution regions are then expanded using the full-resolution image (Fig. 4-1.d). No new regions are created at full resolution. Figure 4-2 shows the segmentation of an image of uneven terrain.



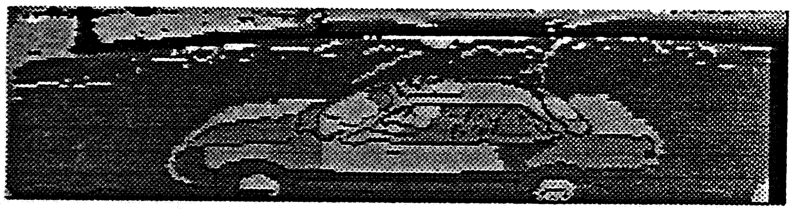
(a) Range image.



(b) Edges from reflectance image.



(c) Low-resolution segmentation ($n=2$).



(d) Final segmentation.

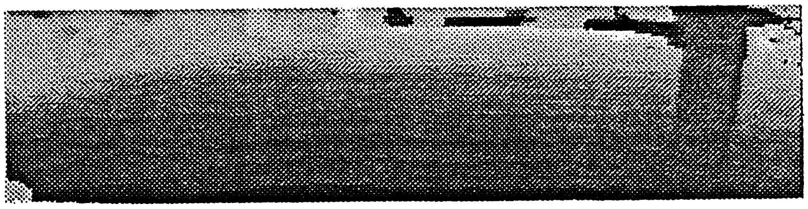
Figure 4-1: Range image segmentation

In addition to the pure region segmentation, we use the edges extracted from the reflectance image to improve the description. In the low-resolution segmentation step, pixels that correspond to a window that contains at least one edge pixels are removed. In the full-resolution step, regions are expanded so that they do not cross an edge. As a result, edge pixels are all part of the regions boundaries. Explicitly including edges improves the segmentation in two ways: First, edges that correspond to low-amplitude occluding edges separates regions that may be merged in the range image segmentation. Second, reflectance edges can delineate surface markings that are not visible in the range image. Figure 4-1.b shows an edge image obtained by applying a 10 × 10 Canny edge detector.

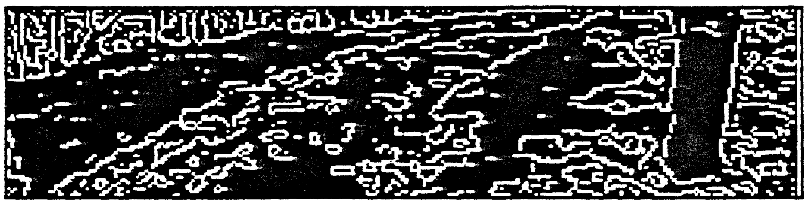
5. Terrain map building

Map building is the process of combining observations of an unknown terrain into a coherent representation. Building a terrain map serves two purposes: It allows to report a set of observations as a product of an exploration mission, and it improves the performance of an autonomous vehicle when the vehicle traverses a previously mapped region. Two types of information are kept in a terrain map: the low-level measurements that are accumulated in a bucket map as described in Section 2.3, and the terrain or object features.

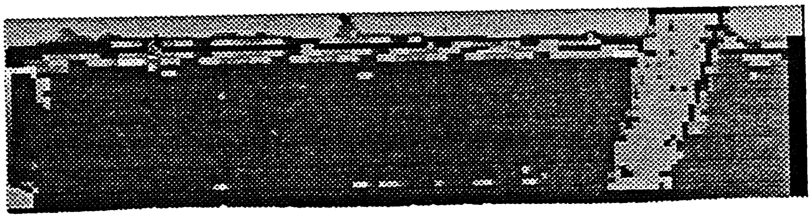
The main issue in the map building process is the matching and enhancement of a map built from measurements acquired from different vantage points. This issue presents some challenging aspects. Since we do not put any constraints on the vehicle's



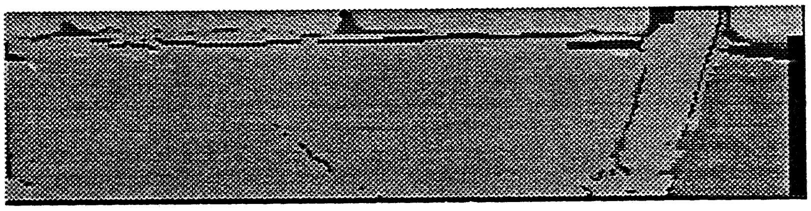
(a) Range image.



(b) Edges from reflectance image.



(c) Low-resolution segmentation ($n=2$).

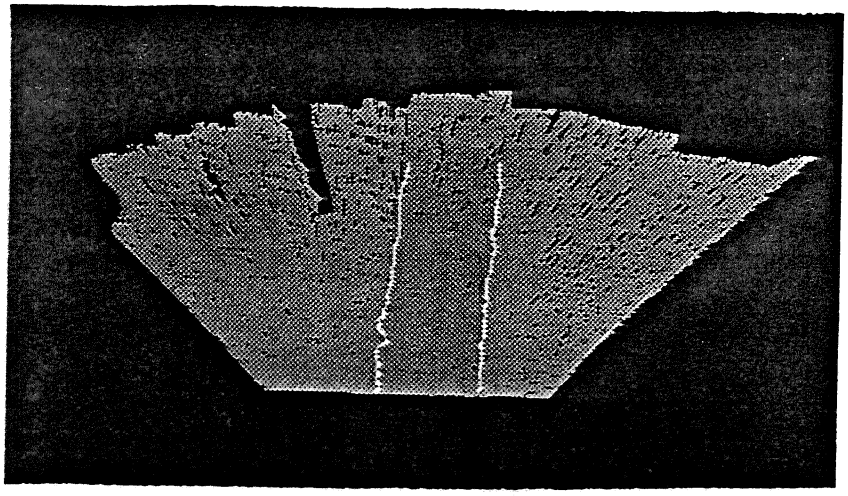


(d) Final segmentation.

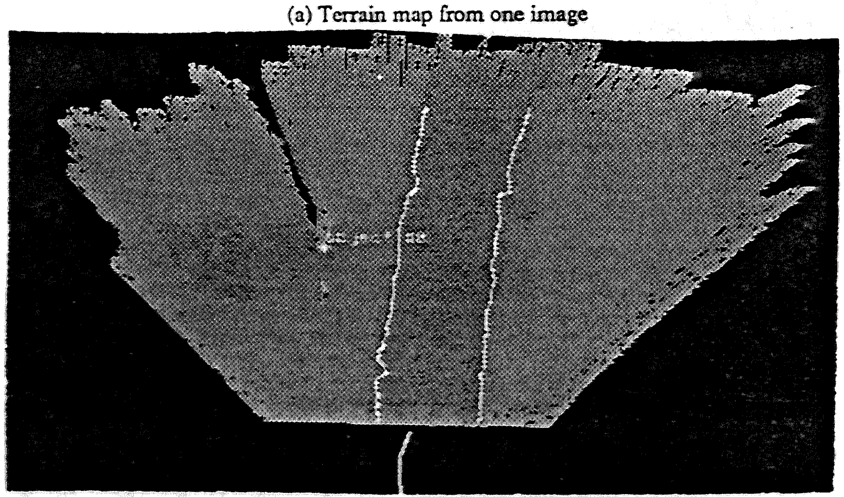
Figure 4-2: Range image segmentation

trajectories, observations of the environment may be radically different from one observation to the other. This requires the identification of common features between observations, as well as the use of uncertainty on the various vehicle position estimates. The insertion of a new frame in the map must therefore proceed in three steps: First, the current estimate of the vehicle position is used to predict matchings between the current map and the new observed features. Second, a new position estimate is computed based on the matchings and the current position estimate. Third, the map is updated by inserting the new observations. This involves the insertion of new measurements in the grid representation, the insertion of the new features in the map, and the updating of existing map features that have been matched with newly extracted features.

We have included the map building techniques in the Carnegie-Mellon NAVLAB system. A terrain map was maintained over a hundred meters while the vehicle was running autonomously under control of the road following program. The current vehicle



(a) Terrain map from one image



(b) Terrain map from four images

Figure 5-1: Terrain map building (including road features)

estimate was used as an initial estimate for the matching. Due to memory limitations, only the portion of the map within a window centered at the current vehicle was kept in the system. The features used for the matching part in that implementation are: the primitives of the surface description (as described in Section 3), the location of discrete objects (as described in Section 2), and location of the road edges extracted from the reflectance channel. The features are weighted according to their uncertainty, for example the variance σ_a in the case of planar features. The road edges are a special type of features: since road detection from reflectance is currently not reliable, the edges are used only if the contrast in the current reflectance image, *e.g.* the strength of the detected road edges, is high enough.

6. Object recognition

6.1. Recognition strategy

The goal of an object recognition algorithm is to find the most consistent interpretation of a scene given a stored list of primitives $(M_{i1},\dots,M_{in})$, the model, and a segmentation of the observed scene, $(S_{j1},\dots,S_{jn})$. The algorithm must therefore search all the possible matchings $((M_{i1},S_{j1}),\dots,(M_{in},S_{jn}))$. This search being an combinatorial problem, the main issue is to prune the search space in order to be able to process complex scenes. Many strategies have been proposed for solving the object recognition problem [1]. The most common approach is to use geometric constraints to constrain the search, assuming the objects are rigid. This approach may require an accurate geometric model which is usually not available in our application. Another approach is to generate beforehand the possible appearances of the object to recognized in order to reduce the search space by precompiling the constraints in the model [8, 5]. We use a combination of both approaches in which geometric constraints are precompiled in the model. The model has two components: a set of surface patches, M, and a set of constraints, C. The constraints encapsulate knowledge about the object's shape, such as "surfaces M_i and M_j are orthogonal". A constraint, c, associated with a set of regions $(M_{i1},\ldots,M_{in})$ can be viewed as a function that decides whether a partial matching $((M_{i1},S_{j1}),\ldots,(M_{in},S_{jn}))$ is acceptable. The number n of regions involved may be different depending on the constraint. For example the constraint on the area of a region is a unary constraint, while the orthogonality constraint is a binary constraint. The list of constraints and their implementation are discussed in the next two Sections.

The search algorithm first constructs a list of candidates for each model region, M_i, by applying the unary constraints associated with M_i to every scene region, S_j. This provides a first reduction of the search space according to unary constraints. The algorithm then explores the remaining search space, discarding the partial solutions that do not satisfy the remaining constraints. In other words, each time a new pairing, (M_i,S_j), is added to a partial solution, $((M_{i1},S_{j1}),\ldots,(M_{in},S_{jn}))$, the constraints associated with M_i are evaluated over the new set of pairings. The partial solution is not explored further if one of the constraints is not satisfied. The result of all the constraint evaluations are stored in tables, so that a constraint is never evaluated twice on the same set of pairings.

The result of the search algorithm is a small set of solutions. The last step of the recognition algorithm is to compute a score that reflects the quality of each solution. This last step is necessary since there is no way of forcing the search to produce only one solution because of near-symmetries in the model, segmentation errors, or even the presence of several instances of the object in the scene. The actual computation of the score is discussed in detail in Section 5.3. The next sections describe the details of the algorithm. We use a car as an example of object model in discussing the algorithm.

6.2. Constraints

The constraints we currently use are:

- *NX, NY, NZ*: constrains the components of the surface normal of a region. This constraint is used to implement natural limitations on the orientation of an object, such as "the roof of a car cannot be vertical".
- *Z*: constrains the vertical position of a region.
- ***AREA***: constrain the area of one region.
- ***ANGLE***: constrains the angle between two regions.
- ***NEIGHBOR***: constrains two regions to be neighbors by computing the distance between the boundaries of two regions.
- ***EXCLUDE***: Forbids two regions to be visible at the same time. This constraint is based on the notion of aspects [8, 7]. An aspect is a set of regions that can be observed from a given viewpoint. Instead of explicitly enumerating the possible aspects of an object, the ***EXCLUDE*** constraint describes them implicitly.
- ***DISTANCE***: constrains the distance between two surfaces.

The constraints are precomputed and stored in the model. Each constraint is described by the following structure:

- **number of arguments,** N: For example, the constraint *ANGLE* which constrains the angle between two regions has two arguments. The maximum number of arguments is currently three.
- **evaluation function,** F: A function that returns an interval given N regions. For example, the constraint *ANGLE* computes an interval centered around the angle between two input regions. The size of the interval is determined at run-time by the uncertainty on the parameters of the regions. In the case *ANGLE*, the interval width is given by the angular variance σ_a within the two input regions.
- **interval,** I: An interval, or set of intervals, which must intersect the computed interval to satisfy the constraint.

This representation of constraints is flexible: A new type of constraint can be easily added to a model by simply defining the appropriate evaluation function. Building a new model is easier since the constraints are not hardcoded in the recognition program.

6.3. Evaluation of the solutions

One would like to have a recognition program that generates only one solution that is reported as the recognized object in the scene. Unfortunately, the search algorithm generates many solutions that have to be evaluated in order to determine the best one. There are three reasons why the search generates multiple solutions: First, the constraints we use are very liberal so that the same model can be used for a wide range of scenes, consequently false solutions are difficult to avoid. Second, the object may have near-symmetries that lead to several equally valid interpretations. Third, the image segmentation being imperfect, an object region may be broken into several pieces in the image, thus producing several equivalent solutions.

Our approach to evaluating solutions is to first compute the position and orientation, or pose, of the object for each solution, to then generate a synthesized, or *predicted*, range image using the estimated pose, and to finally correlate the predicted image and the original range image. We derive a score from the correlation measure between the two images which is used to discard erroneous solutions, and to sort the other solutions.

The pose is calculated for each solution by minimizing the two sums:

$$\sum_{(i,j)} area_i \| R\vec{n}_i^{model} - \vec{n}_j^{scene} \|^2$$

and

$$\sum_{(i,j)} area_i \| R\vec{c}_i^{model} + \vec{t} - \vec{c}_j^{scene} \|^2$$

where R and $\vec{t}$ are the estimated rotation and translation, $\vec{n}_i^{model}$ and c_i^{model} (resp. $\vec{n}_j^{scene}$ and c_j^{scene}) are the surface normal and center of region i (resp. j) of the model (resp. scene). The summation is over all the pairs (scene region j, model region i).

In order to compute the predicted image, we have to compute the position of each point of the model in image coordinates, as well as the predicted range. The position in the predicted range image of a point $\vec{p}$ on the surface of the model is given by:

$$\phi^{predicted} = \mathrm{atan}\,(x/z)$$

$$\theta^{predicted} = \mathrm{atan}\,(y \times \cos(\phi))/x)$$

$$row = (\phi_s - \phi^{predicted})/\Delta\phi$$

$$col = (\theta^{predicted} - \theta_s)/\Delta\theta$$

$$d^{predicted} = \sqrt{x \times x + y \times y + z \times z}$$

In this equation: x, y, z are the coordinates of the transformed point $R\vec{p} + \vec{t}$, (row, col) is the predicted location is the image, and $d^{predicted}$ is the predicted range value at that location.

The correlation between predicted and actual range images is given by:

$$C = \sum_{(i,j) \in P \cap O} a_{ij} \times | d_{ij}^{predicted} - d_{ij}^{original} |$$

where a_{ij} is the area intercepted by pixel i, j, and the summation is made over the intersection $P \cap O$ of the object in the predicted image, P, and the object in the observed range image, O.

The correlation must be normalized to obtain a score S that is between 0 and 1:

$$S = area(P \cap O) / area(P \cup O)$$

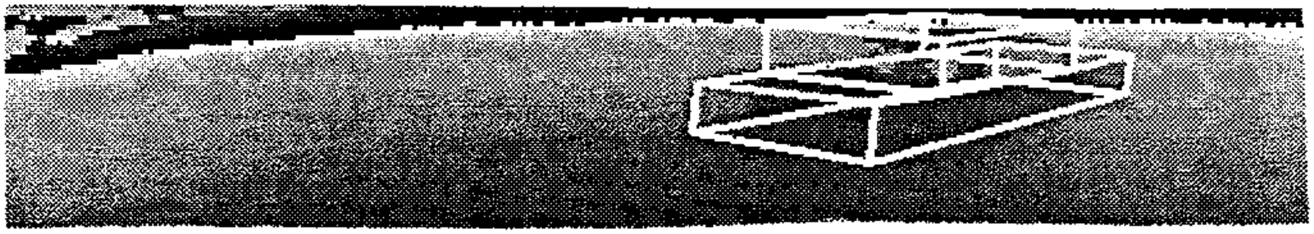
$$\times (1 - C/(K \times area(P \cap O))))$$

In this equation, K is the maximum range difference allowed between predicted and observed images (independent of the image). S is normalized by $area(P \cap O)/area(P \cup O)$ to avoid problems when $P \cap O$ is very small, in which case we would give a high score to a very poor solution. We use the score S to eliminate false solution (typically $S < 0.5$), and to sort the solutions by decreasing score. Figure 6-1 shows the solution of highest score found on one image, the top image shows the superimposition of the recognized model and the range image, the bottom image is the overhead view of the superimposition.

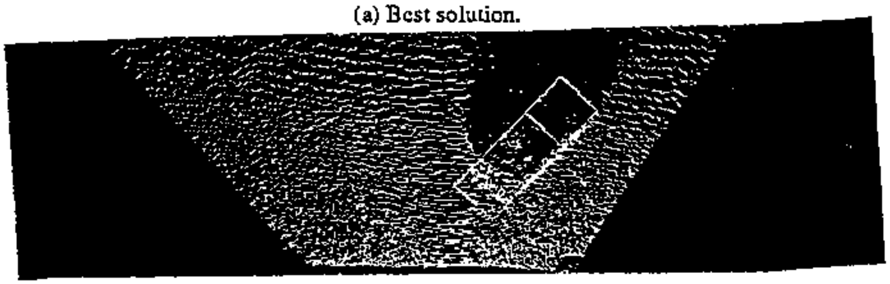
6.4. Results

We have tested the object recognition program on 23 images of two different objects, each image corresponds to a different aspect of the objects, or to a different distance between the object and the sensor. Figure 6-2 shows a sample of the set of images. The failure modes of the program are as follows:

- In two cases, the program failed to produce any interpretation. These cases occur when not enough regions have been extracted to perform the recognition.
- In two cases, the program produced a set of interpretations, but the correct solution was not in the top-score set of solutions.
- In three cases, the program produced the correct interpretation as part of the top-score set of solutions but not as the best solution. These cases occur when a near symmetry in the image cannot be disambiguated based on the shape



(a) Best solution.



(b) Overhead view.

Figure 6-1: Object recognition

information only, as a result, both the correct solution and its symmetrical have the same score.

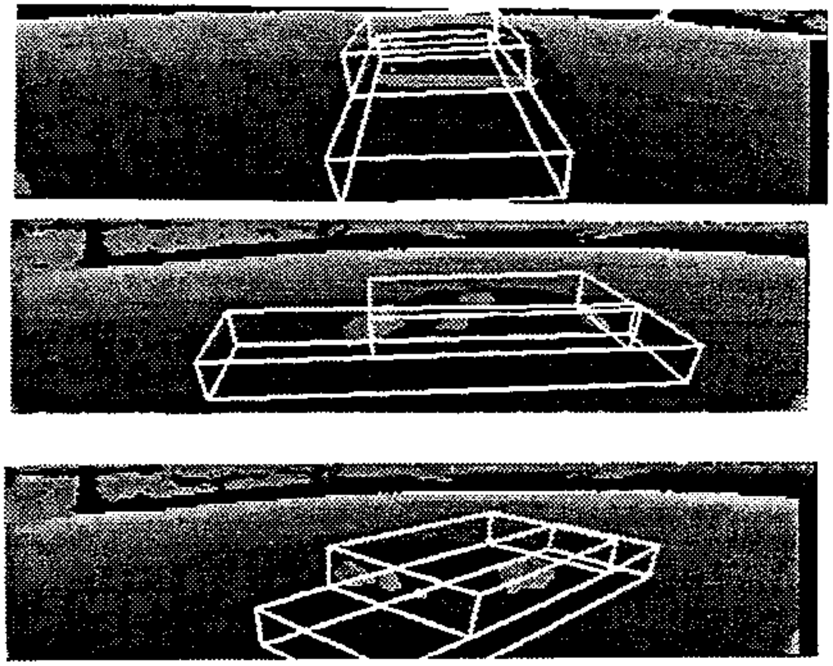


Figure 6-2: Object recognition on a sample set of images

The object recognition algorithm can be improved in several ways; The recognition of one object requires an average of 1,500 constraint evaluations, most of which are rarely used for rejecting a partial solution. One optimization is to order the constraints in the model so that the constraints most likely to prune the search are evaluated first.

The scoring procedure leads to very close scores for symmetric or ambiguous solutions. As an example, Figure 6-3 shows two symmetric solutions for which the difference between the two scores is below 1%. This can be improved by including other sources of information, such as video or reflectance images, in the scoring procedure.

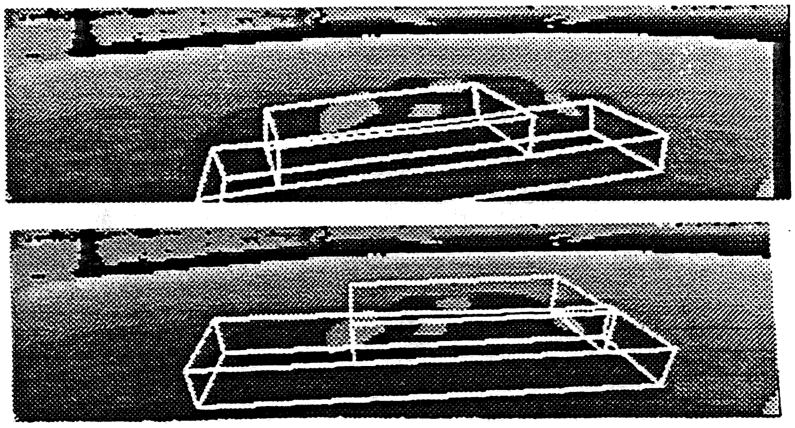


Figure 6-3: Symmetrical solutions

7. Conclusion

We have presented a set of techniques for producing 3-D environment descriptions for an autonomous vehicle. Those techniques and their output descriptions have been designed to fulfill the requirements of the major tasks of an autonomous vehicle such as obstacle avoidance, landmark recognition, and map building. We have conducted an numerous demonstrations in the CMU autonomous vehicle system, the NAVLAB, for navigation and map building applications.

We are currently pursuing further research along three lines: implementation on faster hardware, handling uncertainty, and developing a more general scheme for object recognition.

The primary bottleneck in the development of an autonomous vehicle is the computation time required by the sensory modules such as the range image analysis module. We are in the process of porting the algorithms to a fast systolic machine, the WARP [12]. The obstacle detection module has been already successfully demonstrated with a cycle on the order of one second on the WARP. The second line of work is the representation of uncertainty. Sensor measurements, vehicle position estimates, and segmentation process induce errors in the final 3-D description. These errors can be quantified and taken into account in all the range analysis algorithms. We plan to use an explicit representation of the uncertainty building surface description [9, 2]. Third, we plan to develop a more general scheme for object recognition. Third, we plan to apply the object recognition algorithm to a larger class of objects and algorithms.

References

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

[2] R. M. Bolle, D. B. Cooper.
On Optimally Combining Pieces of Information, with Application to Estimating 3-D Complex-Object Position from Range Data.
PAMI 8(5), september, 1986.

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

[4] R. T. Dunlay, D. G. Morgenthaler.
Obstacle detection and avoidance from range data.
In *Proc. SPIE Mobile Robots Conference*. Cambridge, MA, 1986.

[5] C. Goad.
Special purpose automatic programming for 3D model-based vision.
In *Proc. Image Understanding Workshop*. 1983.

[6] M. Hebert, T. Kanade.
First results on outdoor scene analysis.
In *Proc. IEEE Robotics and Automation*. San Francisco, 1985.

[7] M. Hebert, T. Kanade.
The 3D Profile method for object recognition.
In *Proc. Computer Vision and Pattern Recognition.* San Francisco, 1985.

[8] K. Ikeuchi.
Precompiling a geometrical model into an interpretation tree for object recognition in bin-picking tasks.
In *Image Understanding Workshop.* Los Angeles, 1987.

[9] L. Matthies, S. A. Shafer.
Error Modelling in Stereo Navigation.
Technical Report CMU-RI-TR-86-140, Carnegie-Mellon University, the Robotics Institute, 1986.

[10] C. Thorpe, M. Hebert, T. Kanade, S. Shafer.
Vision and navigation for the Carnegie-Mellon Navlab.
Annual Review of Computer Science, Volume 2.
Annual Reviews Inc., 1987.

[11] D. Y. Tseng, M. J. Daily, K. E. Olin, K. Reiser, F. M. Vilnrotter.
Knowledge-based vision techniques annual technical report.
Technical Report ETL-0431, U.S. Army ETL, Fort Belvoir, VA, 1986.

[12] Webb, H. and Kanade, T.
Vision on a Systolic Array Machine.
Computing Structures and Image Processing.
Academic Press, 1985.

Knowledge-Based Interpretation of Outdoor Road Scenes

Takahiro Fujimori and Takeo Kanade*
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper describes an approach to robust road scene interpretation in high-level vision. There are two key ideas. One is adjustable explicit scene models which express the global information of a scene. The other is an interpretation cycle: evaluation, modeling, and extrapolation, which is the mechanism for adjusting the explicit scene models.

First the interpreter picks up some tractable segment regions and generates initial hypotheses, and then it iterates the interpretation cycle until every region is labeled with one of the scene objects expected. In each interpretation cycle, labeled regions play a survival game. Those that are consistent get their plausibility value increased. Those that are not get their plausibility value decreased and some of them die (become unlabeled). Surviving regions propagate their labels to their adjacent similar regions. In the mean time, explicit scene models are also adjusted with the latest interpretation in each cycle. The models gradually extract global information out of the scene. When the plausibility values of the models become high, global evaluation starts by the models in addition to local evaluation. Some deceptive regions are easily killed by the global evaluation.

More than 20 road scenes, some of which are destructively shaded, are reliably interpreted by the implemented system INSIGHT-III. From the view point of high-level vision, inexhaustive region segmentation in low-level vision is also suggested and discussed.

1. Introduction

The goal of this study is to build a robust road scene interpreter. Our approach is to interpret components such as roads, grass, soil, shadows, sun spots, tree trunks, tree foliage, vertical objects, sky, etc., in a road scene explicitly. We focus our study on theoretical aspects of natural scene understanding rather than practical aspects such as how quickly a road is extracted in order to navigate an actual vehicle. In natural scene interpretation, there are three substantial problems: low-level region segmentation, interpretation, and domain knowledge organization. This paper concentrates on the second problem, *interpretation*.

We have used available tools for low-level region segmentation. The segmentation tool is histogram-based and semi-automatic. A threshold value in each histogram is interactively determined by an operator. Automatically segmented images are poor and some of them are uninterpretable. Insisting on fully-automatic segmentation could have obscured potential problems of *interpretation*. As the domain, road scenes on sidewalks in a park were chosen, so the kinds of objects in a scene are moderately limited, currently 10, which alleviates the problem of domain knowledge organization. On the other hand, complex road scenes on sidewalks were intentionally selected so as to uncover the potential problems in *interpretation*. Several scenes are severely shaded by trees. A couple of them are not usually understood even by a human without predictions. Another intention was to accumulate knowledge out of many scenes (more than 20 to date). This was also requisite.

---

*Supported by a grant from Sony Corporation.

Outdoor natural color road scenes are ill-structured. Trees and objects often shade the road badly. The edges formed by shadows and sun spots are much stronger than those of roads. Natural objects are not usually made with straight lines. Old road pavements have many cracks and spots. The color intensities of the road pavements vary significantly depending on their surface conditions, the weather, the direction of sun, the camera condition, and other circumstances. In *interpretation* of natural outdoor color scene, there are two main questions:

1. How to extract global information which depicts a scene roughly.
2. How to generate reliable initial hypotheses.

If the system has global information about the scene, this reduces the search space and local region labeling becomes unambiguous. For example, if the system knows the rough location of the road in a scene, only the regions along the rough road edges are ambiguous. More reliable road edges will be obtained by checking intensity steps crosswise in long narrow windows along the expected locations of the edges. Of course, global information is not available in advance. The interpreter itself has to extract it In our approach, explicit scene models represent global information and the interpretation cycle is a mechanism for adjusting them.

Good initial hypotheses usually lead to successful interpretation while bad initial ones often do not It is hard to change the labels of the initially labeled regions. This is natural because initially labeled regions are the seeds for the foEowing interpretation. This problem is partly beyond this paper, because it also depends on the low-level region segmentation. However, several ideas for generating reliable initial hypotheses are described in Section 6.2.1. Inexhaustive region segmentation in low-level vision is also suggested and discussed in Section 7.1.

The major contribution of this study by explicit scene models and an interpretation cycle is enumerated as follows:

- Global information of a natural scene can be extracted with them and make interpretation reliable.
- They eliminate most of the search space and interpret oversegmented and fragmented regions consistently.
- To some extent, they can cope with the problem of undersegmented regions.
- Cyclic interpretation mechanism is appropriate for intensive interaction between high-level and intermediate-level.

The following Section 2. explains pioneering work in natural scene interpretation. Section 3. illustrates the main idea of this study: explicit scene models and an interpretation cycle. In Section 4., the overview of the developed system, INSIGBT-III is presented. In Section 5., the results of 11 images are shown. Three examples of the explicit scene model behaviors follow. Section 6. goes into the detail of INSIGHT-in. Possible enhancements are explained in Section 7. Section 8. concludes this study.

2. Related Work

Draper [t]f2] las experimented with knowledge-based road scene interpretation in the "Schema System" architecture, which originates from the VISIONS system [3]. His system comprises independent object-specific interpreters, called schemata, and a central blade board for communication among them. The system interprets a road scene in a tofHtowa Hwmer by stating from the road-scene schema and thai calling subpart schemata such as roof, sky, road, and *tost* MewcMcaily. Each schema generates initial hypotheses based on color, texture, shape, area, and posiiim of segmented regions. Then it corroborates or denies them by checking positive and negative evidence, extending hypotheses, resolving conflicts, and calling subpart schemata. Information about the latest interpretation with confidence values is exchanged through the central blackboard.

Ohta [4] has built a knowledge-based outdoor color scene interpreter which recognizes four objects and two subobjects: sky, trees, buildings with subobjects (windows), and roads with subobjects (cars) in a combined bottom-up and top-down manner. In his system, each large region initially has four hypotheses: sky, a tree, a building, and a road with correctness values which are modified by evaluation. The hypothesis which gets the highest correctness value wins. It first tentatively merges small regions with one of their large neighbor regions and then calculates the four correctness values of the hypotheses on each large region by checking color, texture, shape, position of segmented regions, relations, contrast, lines, etc., in a bottom-up manner. After this, these hypotheses are evaluated. Then, the interpretation goes to rule-based top-down analysis, which tries to select the most appropriate label for each large region and find labels for smaller regions. Whenever the top-down analyzer makes a significant decision such as the position of the scene horizon, the interpretation goes back to the bottom-up process and re-evaluates the labels for large regions. In this way, the bottom-up process and top-down process work cooperatively.

McKeown [5][6] has demonstrated a rule-based system, SPAM, with approximately 500 OPS5 rules for aerial airport imagery containing terminals, parking aprons, hangars, runways, taxiways, parking lots, roads, tarmac, grassy areas, and functional areas. SPAM is a region-based bottom-up system with top-down predictions. It allows multiple initial hypotheses on each region, which are derived by examining local properties such as 2-D shape, location, orientation, texture, area, and perimeter. Then it checks consistency locally, groups local interpretations into functional areas, and generates one or a few airport models. Height information extracted by stereo images is also utilized when necessary. Recently he also developed knowledge acquisition, machine rule generation, and performance analysis tools [7] in order to ease and expedite knowledge acquisition processes [8]. Knowledge for suburban house scenes was collected in the SPAM architecture with these tools.

Brooks [9] has developed a viewpoint-insensitive 3-D model-based system called ACRONYM, which was applied to jet plane recognition in aerial photographs. ACRONYM has an intermediate level representation with ribbons (trapezoids with long height) and ellipses, to which both 3-D model and low-level edges are transformed. The interpretation takes place in this level in a top-down manner. Recently, ACRONYM was transferred to industry [10] and uncertainty management [11] has been incorporated into it.

A survey by Binford[12] discusses model-based image analysis systems including work by Nagao and Matsuyama [13], Ohta [4], Brooks [9], and Shirai [14].

3. Adjustable Explicit Scene Models and Interpretation Cycle

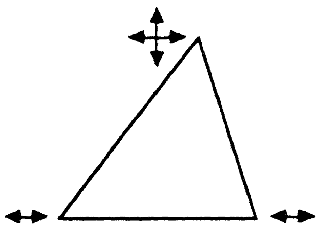
Adjustable explicit scene models and interpretation cycle are the two key ideas of INSIGHT-III for extracting global information out of a interpreted scene. The global information makes global evaluation possible and thus leads to reliable consistent interpretation. In scene interpretation, a chicken and egg problem has arisen: a scene is hardly interpreted locally and correctly without global information of the scene; the global information, on the other hand, can not be obtained without local interpretation fragments. These ideas are an approach to this problem.

At first, the interpreter does not have global information of a road scene at all,although usually there are some tractable regions which are often large, occupy a lower portion of the image, have typical colors, and can be labeled in a bottom-up manner. Once even just a few regions are labeled, they tell a little about the scene. Very rough global information of the scene can be extracted. Of course, this global information is not reliable, but better than nothing. It can be used skeptically in the following interpretation stage. Gradually, more regions are labeled and thus the extracted global information is improved. If this global information feedback is reasonable, interpretation converges to an appropriate one.

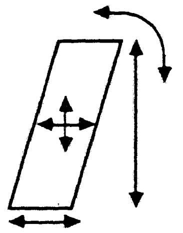
3.1. Adjustable Explicit Scene Models

Adjustable explicit scene models explicitly approximate the appearances of the objects in a road image with simple geometrical figures. Those models have four or five degrees of freedom and are adjusted to the latest interpretation in each interpretation cycle.

Figure 1 (a) depicts a triangular explicit scene model for the appearance of a road or a long narrow stripe on an assumed flat ground plane. It has four degrees of freedom: the vanishing point moves horizontally and vertically; lower ends of the right and left edges move horizontally. [1] Figure 1 (b) is a parallelogram explicit scene model [2] for a tree trunk or a vertical object protruding from the ground. Since its upper and lower bases remain horizontal, it has 5 degrees of freedom: location (2 degrees of freedom), height, width, and lean.



(a) Road Model



(b) Tree Trunk Model

Figure 1: Adjustable explicit scene models

Figure 2 illustrates how the road model is adjusted to an interpreted image. To measure the appropriateness of the road model, a discrepancy value is defined: the discrepancy for the road model is the sum of the road area outside the model + the grass/soil area inside the model. The road model is adjusted such that the discrepancy value is locally minimized. Note that the explicit scene models are not interfered by jagged region boundaries.

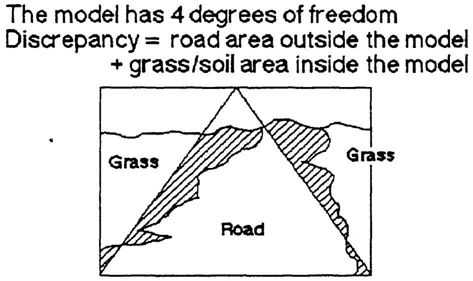
Figure 3 shows the tree trunk model adjustment. The discrepancy value for the tree trunk model is the sum of the tree trunk area outside the model + the non-tree-trunk area inside the model. The tree trunk model is also adjusted to the local minimum of the discrepancy value. Because of this, it is not affected by the trunk areas of other trees. Finding the local minimum of the discrepancy value is not very expensive. It only requires checking the pixels along the model boundaries. Section 6.5. explains the details.

In the early stages of interpretation, some regions are labeled and others are not. But the explicit scene models are adjusted by using partially labeled regions. Though the adjusted models may be less reliable, they can grasp very rough information of the scene. For example, unlabeled regions in the middle of the adjusted road model are likely to be fragments of the road, and those away from the adjusted road model are likely to be fragments of the grass or the soil. If the plausibility value of the model becomes higher for some reason, the model can classify the

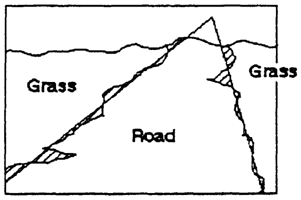
---

[1] The road model can be expressed with another combinations of parameters. For example, vanishing point, road direction, and road width, which also has 4 degrees of freedom.

[2] Though these scene models are 2 dimensional, they may be derived from 3 dimensional scene models by perspective projection.


The model has 4 degrees of freedom
Discrepancy = road area outside the model
+ grass/soil area inside the model
Grass
Grass
Road

(a) Before Adjustment



(b) Adjusted Road Model

Figure 2: Road model adjustment

unlabeled regions and even discard some contradicting labels.[3]

These explicit scene models are simple and cannot express the exact shapes or the details of roads and tree trunks, such as curves and tree branches. Complex or flexible models with more degrees of freedom may be applied afterwards if necessary. With the global information extracted by the simple models, those complex or flexible models will be adjusted quickly and reliably to the scene.

An interesting aspect of the explicit scene models is that they help detect misinterpretation. The very wide or narrow road is strange. The very high or low vanishing point is uncommon. If a tree trunk is found in the middle of the road, it implies something strange has happened. The system can easily determine unusual situations by checking the adjusted models. Then it will be able to retry the interpretation of the scene with different parameters in low level, or activate a contingency planner which may decide to decelerate the autonomous land vehicle and take an image with another camera iris or angle. The acceptable range of the adjusted models will be determined heuristically or by the interpretation result of the previous scene.

3.2. Interpretation Cycle

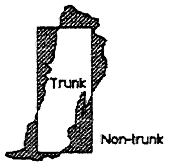
Figure 4 shows the interpretation cycle of the high-level scene interpreter. It consists of an initial hypothesis generator, an evaluator, a modeler, and an extrapolator. In general, the interpretation proceeds as follows. First, the initial hypothesis generator picks up some tractable regions by checking their properties (areas, locations, intensities, adjacent regions' intensities, shapes, straight lines, parallel lines, trapezoidal lines, etc.) and makes initial hypotheses.[4] Then, the interpretation cycle starts and interates until every region is labeled with one of the objects expected. The evaluator checks local consistencies of the current interpretation by basically looking at pairs of labels of adjacent regions. If they are consistent, their plausibility values are increased, if they are not, their plausibility values are decreased. Every region with negative plausibility value is unlabeled. When the plausibility values of the models become higher, evaluator starts checking global consistencies of the current interpretation by looking at labels and adjusted models. If a label contradicts a model, its plausibility value is

---

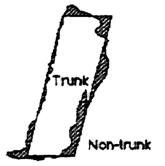
[3] This action might, in turn, require model adjustments.

[4] Initial hypothesis generator is more organized, which is described in Section 6.2.

The model has 5 degrees of freedom
Discrepancy = trunk area outside the model
+ non-trunk area inside the model



(a) Before Adjustment



(b) Adjusted Trunk Model

Figure 3: Tree trunk model adjustment

decreased drastically. Then, the modeler adjusts the explicit scene models to the current interpretation, one model after another. It generates necessary models and occasionally eliminates inappropriate models. The extrapolator picks up some moderately tractable unlabeled regions, that are adjacent to labeled regions and makes new hypotheses by examining their properties and labeled neighbor regions. Finally the control goes back to the evaluation and starts the next interpretation cycle.

Figure 5 illustrates the interpretation cycles by a simplified example, which has only a region map, color data, and four interpretation cycles. Figure 5 (a) is the region map with 10 segmented regions. Figure 5 (b) shows the colors of the regions. It includes one ambiguous region and two deceptive regions. R9 is ambiguous since its color is in-between. R6 and R10 are deceptive because their colors differ from those of the neighbor regions. The initial hypothesis generator picks up the tractable regions R1, R4, R5, and R7 and makes hypotheses: sky, grass, road, and grass, respectively. Then, the interpretation cycle starts.

In the first cycle, the evaluator slightly raises the plausibility values of R1, R4, and R5 by checking the consistency between adjacent labeled regions. The modeler then adjusts the triangular road model. The adjusted road model is far from ideal since its vanishing point is too high, but it has obtained very rough global information of the road. Then, the extrapolator is activated-R5 propagates to R8, R7 labels R2 with grass. Both R4 and R5 try to propagate to R9 since its color is between them. R6 becomes road and R10 becomes grass, erroneously. In the second cycle, many of the regions have been labeled, but R6, R9, and R10 are mislabeled. The evaluator raises the plausibility values of the labeled regions, except those mislabeled. Because the plausibility value of R9 becomes negative, R9 is unlabeled. Then the modeler adjusts the model and gets much better global information. The vanishing point is almost correct. The plausibility level of the model is now *usable*. Then, the extrapolator partially uses the model and R5 propagates to R3 and R9. In the third cycle, all regions except deceptive ones are correct. R9 survives the evaluation, but the deceptive regions are rejected by using adjusted road models. The modeler extracts the proper road location and the plausibility level of the model becomes *reliable*. In the following extrapolation, the deceptive regions are appropriately labeled with the reliable road model. In the fourth cycle, every region survives the evaluation. The road model does not change in the modeling step and the interpretation ends.

Unlike some other region based interpretation systems [4][5], INSIGHT-III gives a segmented region at most one label of the scene objects at a time. With this restriction, the explicit scene models can be adjusted uniquely. Since this restriction temporarily excludes alternative labels, the evaluator has a local and global negative evaluation

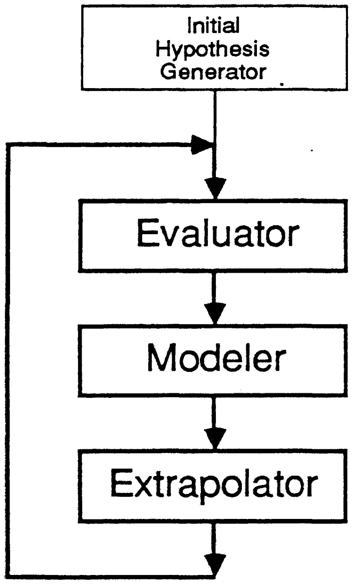


Figure 4: Interpretation Cycle

mechanism which refuses inconsistent labels and gives chances to the alternative labels.

The robustness of the interpretor resides in a balance among four modules depicted in Figure 4. None of the modules is perfect. The four modules work together to compensate each other. Some strong rules in a module may work effectively on a particular road scene, but not on other scenes. Rules should be simple and straight-forward so that the obvious regions are labeled quickly. Occasional mistakes are taken care of by the other modules.[5] For example, the initial hypothesis generator should label a few of the easiest regions but can leave slightly ambiguous regions unlabeled, since the following interpretation cycle is capable of disambiguating them. The extrapolator can make easy hypotheses since the negative evaluation is able to reject bad hypotheses. The evaluator does not have to be nervous because the modeler is obtaining global information and will settle the interpretation at the end. In the case of a very difficult scene, the interpretation does not converge. But it can be detected by monitoring the number of interpretation cycles iterated and the plausibility level of the models. Then a contingency planner may be activated.

4. System Overview

The knowledge-based road scene interpretation system, INSIGHT-III, is capable of extracting roads, grass, soil, shadows, sun spots, tree trunks, tree foliage, vertical objects, sky, and unknown without predictions except for the existence of the road and the rough correspondence between a horizontal line in an image and an actual scene horizon. It also detects pairs such as a tree trunk and its shadow, the direction of sun (right or left), and the weather (sunny or not), and uses this information in interpretation. By inverse projection (back projection), it generates a road map of the scene although its depth information is relative. Its high-level interpretor has 187 OPS83 [15] (descendant of OPS5 [16]) rules and C++ [17] functions for model adjustment. The number of rules

---

[5] Actually, the implemented system frequently changes its mind. Once it was observed that a region was relabeled four times and finally the fourth hypothesis survived.

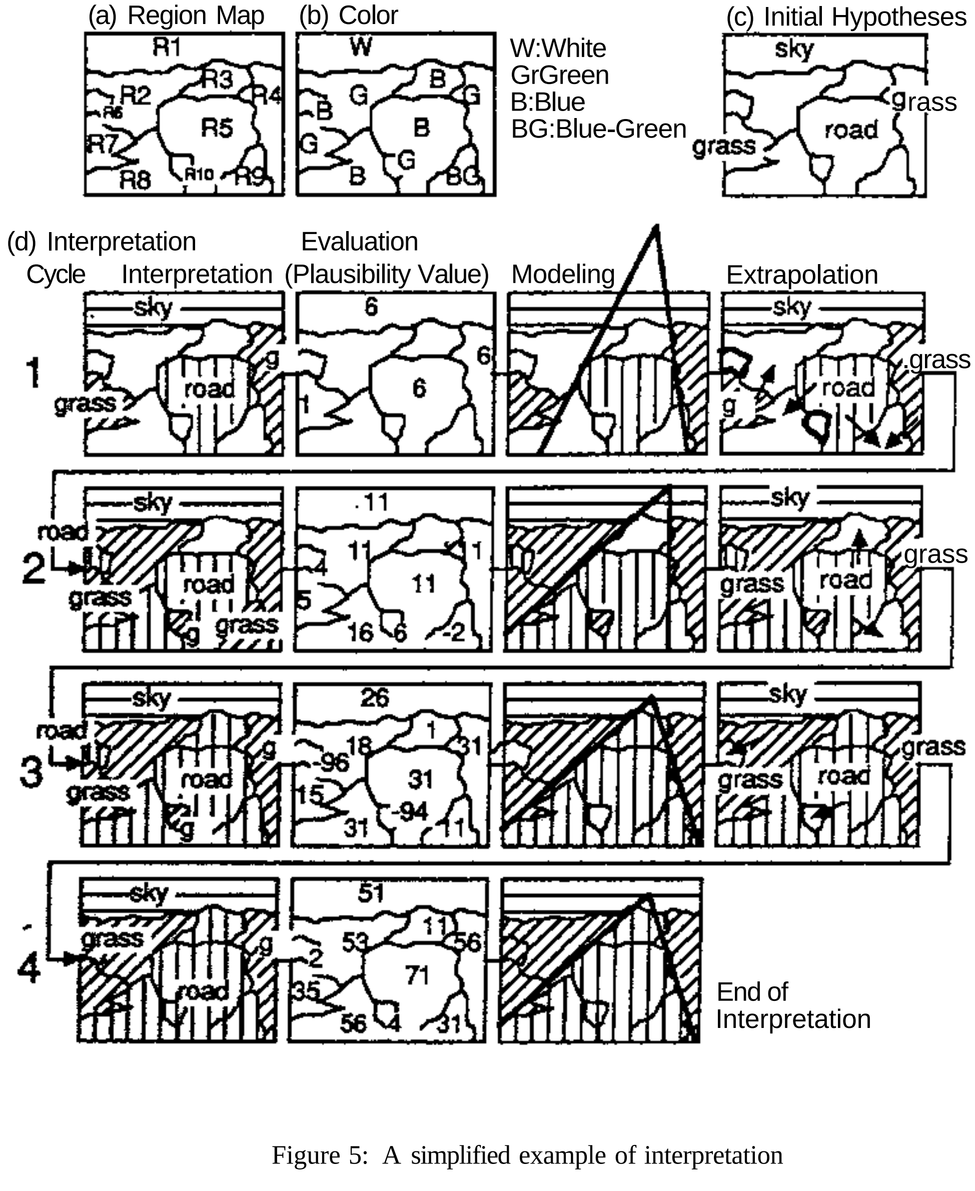

(a) Region Map
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
(b) Color
W
G
B
BG
W:White
GrGreen
B:Blue
BG:Blue-Green
(c) Initial Hypotheses
sky
grass
road
(d) Interpretation Cycle
Interpretation
Evaluation (Plausibility Value)
Modeling
Extrapolation
1
2
3
4
sky
road
grass
g
6
11
-4
5
16
-2
26
18
-96
15
31
-94
51
53
2
35
56
71
End of Interpretation

Figure 5: A simplified example of interpretation

fired is approximately 2000 to 5000 for one road scene.[6] The system runs on SUN-3/260 workstations. One road scene takes about 30 to 60 minutes to interpret, depending on the complexity of the scene. An experimental fully-automatic low-resolution version of the system interprets it in around eight minutes. 10 road scenes, some badly shaded, are interpreted by the system. In addition to these, more than 15 road scenes are interpreted by the fully-automatic low-resolution version of the system. Their high-level knowledge-based interpreters are exactly the same.

Figure 6 shows the system organization. The color median filter described in Appendix A, which considers color pixels as vectors and keeps color edges sharper than the scalar median filter, is first applied to a raw color road scene image of 512 × 480 in spatial resolution with 8 bit pixel depth. Second, the noise reduced image is semi-automatically and recursively segmented by a histogram-based region segmentation tool, PHOENIX [18] [19]. At each step PHOENIX calculates red, green, and blue histograms of a scene portion, and a human selects one histogram and determines a threshold value by looking for the deepest valley in it. Then the scene region is divided into several smaller regions by using the threshold value. This process recurs until all the regions become homogeneous or small. Third, the color edge detector [20] extended from the Canny edge detector [21] extracts edge information from the raw image. Fourth, the feature extraction program calculates the intermediate representation of each region out of the region map, the edge data, and the raw image. The intermediate representation includes the region map, mean intensities, standard deviations, texture, area, perimeter, mass center, minimum bounding rectangle (MBR), scatter matrix, boundaries, vertices, lines, and relations among regions, boundaries, vertices, and lines. Fifth, the extracted intermediate representation data are fed to the high-level knowledge-based interpreter and the road scene is interpreted. Last, the adjusted scene models proceed to the map generation and a road map is drawn by inverse projection.

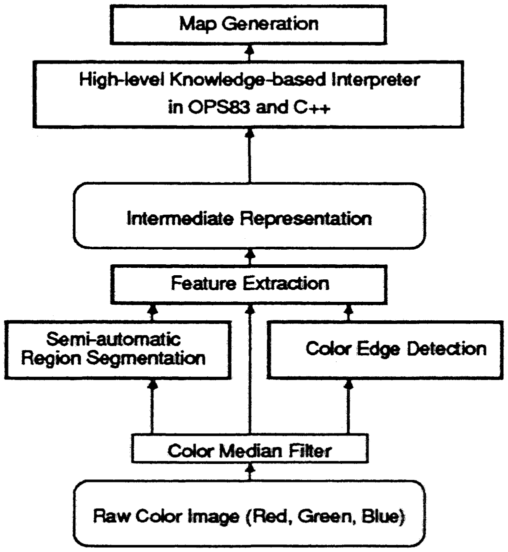

Map Generation
High-level Knowledge-based Interpreter in OPS83 and C++
Intermediate Representation
Feature Extraction
Semi-automatic Region Segmentation
Color Edge Detection
Color Median Filter
Raw Color Image (Red, Green, Blue)

Figure 6: System organization

In the case of the fully-automatic, low-resolution version of the system, only the region segmentation tool is different. Instead of PHOENIX, a region segmentation tool [22] based on the distances among pixels in color space are utilized. It generates a region map of 64 × 60 in spatial resolution by grouping similar pixels. This version will be integrated with an actual Autonomous Land Vehicle [23] [24] developed at Carnegie Mellon University.

Figure 7 illustrates the high-level knowledge-based interpreter configuration. The intermediate representation data enter the working memory of the rule-based system. The rules of the initial hypothesis generator are activated and label some tractable regions. Then, the rules of the evaluator, the modeler, and the extrapolator examine the scene interpretation data cyclically until every region is labeled. All the intermediate representation data, the temporal data, the explicit scene models, and the latest interpretation expressed as an iconic 512 × 480 array of labels reside in the working memory.

---

[6] Because of the cyclic mechanism, the number of rule firings is much larger than the number of rules. This is an uncommon characteristic of the system as opposed to ordinary expert systems.

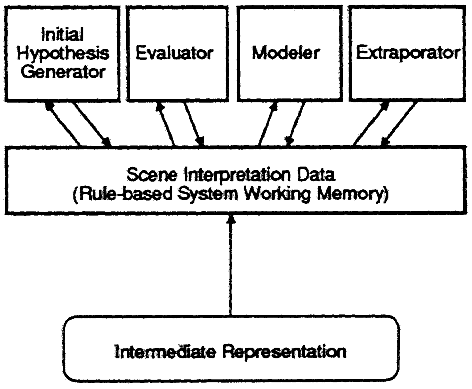

Initial Hypothesis Generator
Evaluator
Modeler
Extraporator
Scene Interpretation Data (Rule-based System Working Memory)
Intermediate Representation

Figure 7: High-level knowledge-based interpreter configuration

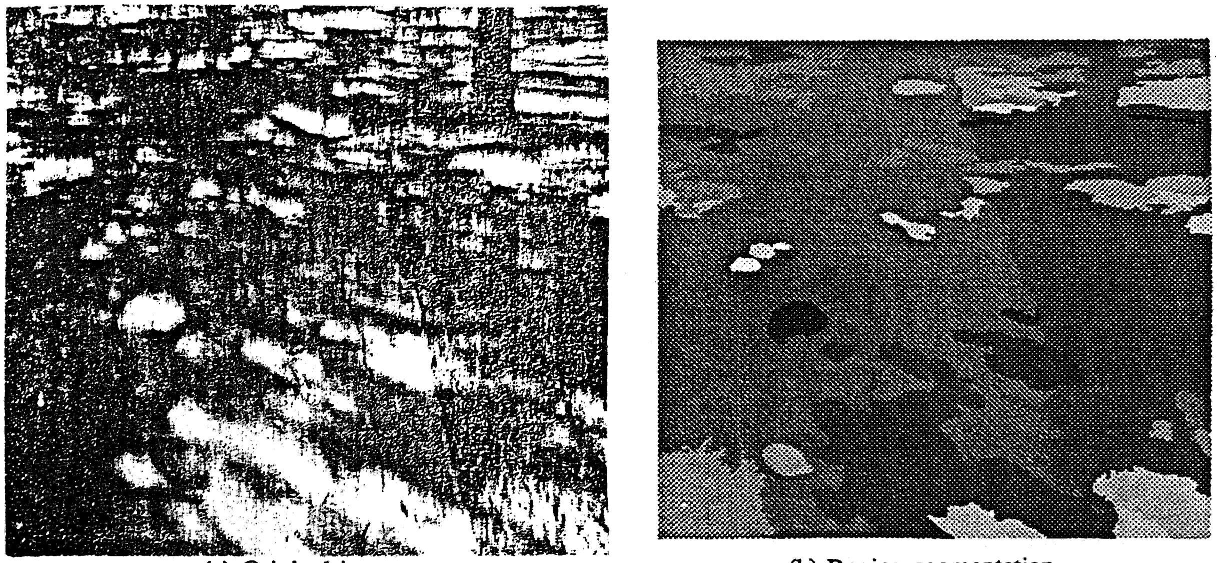
5. Results of the Road Scene Interpretation

Figures 8~18 show the results of road scene interpretation by INSIGHT-III. Figures 8~11 are the shaded roads. Figures 12~15 have trees in the scenes. In Figure 16, there are two vertical objects (two men). Figure 17 is an intersection with trees. Figures 8~18 (a) are original road scenes. Figures 8~18 (b) show their region segmentation. Regions are painted with their identification numbers. Figures 8~18 (c) tell their solution class. Regions shaded with vertical lines mean *above ground* solution class. Those shaded with horizontal lines are *on the level of ground* solution class. Figures 8~18 (d) illustrate their interpretation. Each of the 10 objects is painted with the same intensity and texture. Road regions are labeled properly, and thus the road looks flat except for the very dark shadows and sun spots. In Figures 8~18 (e), adjusted scene models are drawn on segmented regions painted with their mean intensities. They are extracting the location of the road appropriately. Generated maps are depicted in Figures 8~18 (f). Roads are expressed with pairs of parallel lines, and trees or vertical objects are drawn with circles.

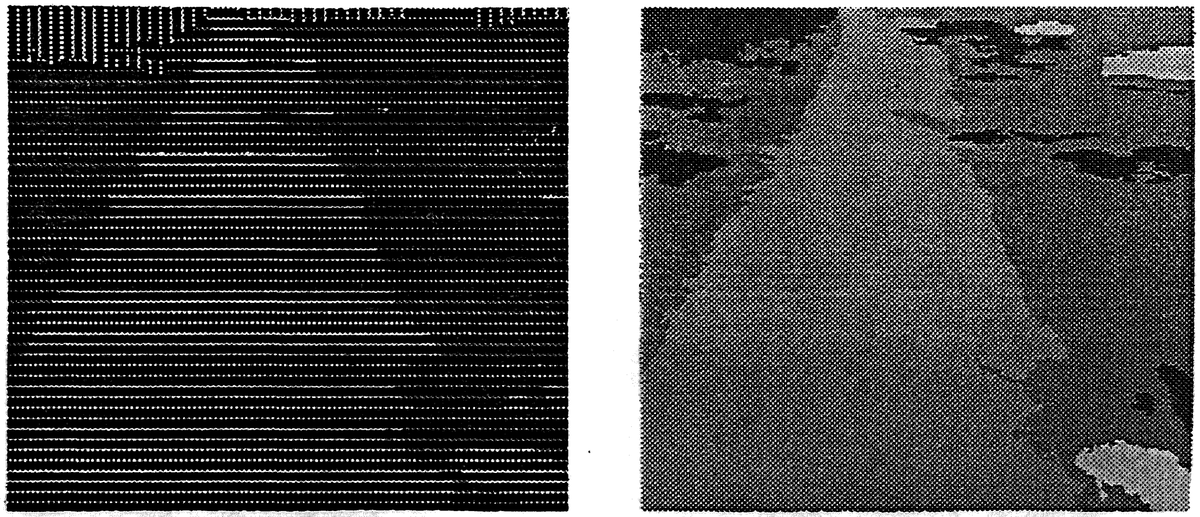
Because of different camera calibration, the images of Figure 12 and Figure 15 look blue and those of Figures 8~11 and Figures 14~15 look red. Figure 12 and Figure 13 were digitized from VCR tapes. Their original images are blurred and very noisy. Some of the vanishing points are in the scene while others are beyond the top of the image frame. Segmented regions are not always extracting proper shapes. Their boundaries are jagged. There are many deceptive and ambiguous regions whose properties are very different from typical ones.

Figure 10 and Figure 11 are very difficult scenes because of the destructive shadows. Even a human cannot understand the scene in Figure 11 without predictions. To interpret those road scenes, INSIGHT-III needs more interpretation cycles. The road models are not stable at first and take a longer time to converge.

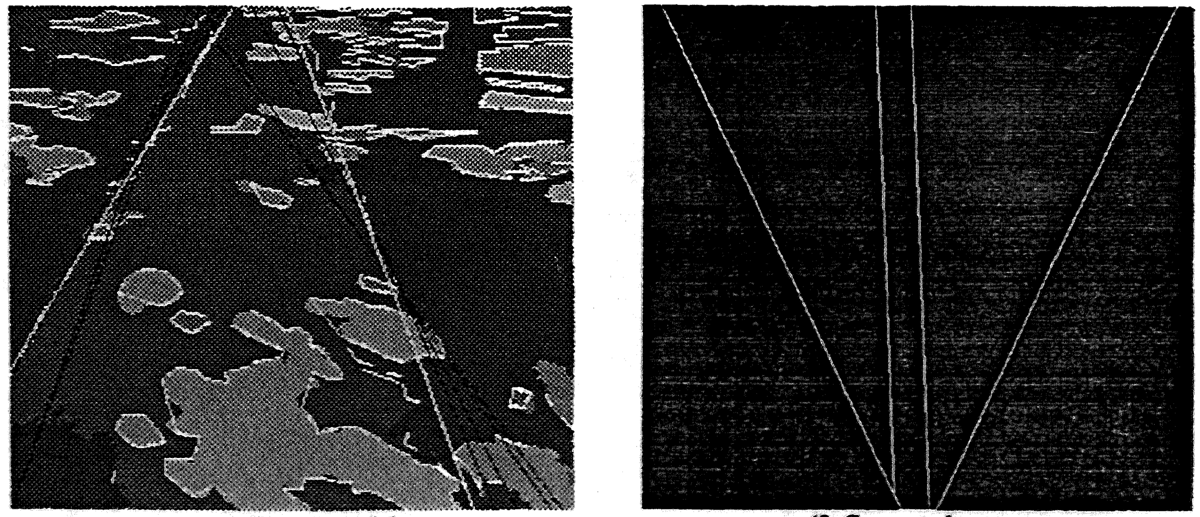
Figures 19~21 trace the behavior of the models which have been shown in Figure 9, Figure 14, and Figure 17, respectively. The initial hypothesis generator is skeptical and makes only several hypotheses as are shown in Figures 19~21 (a).



(a) Original image (b) Region segmentation

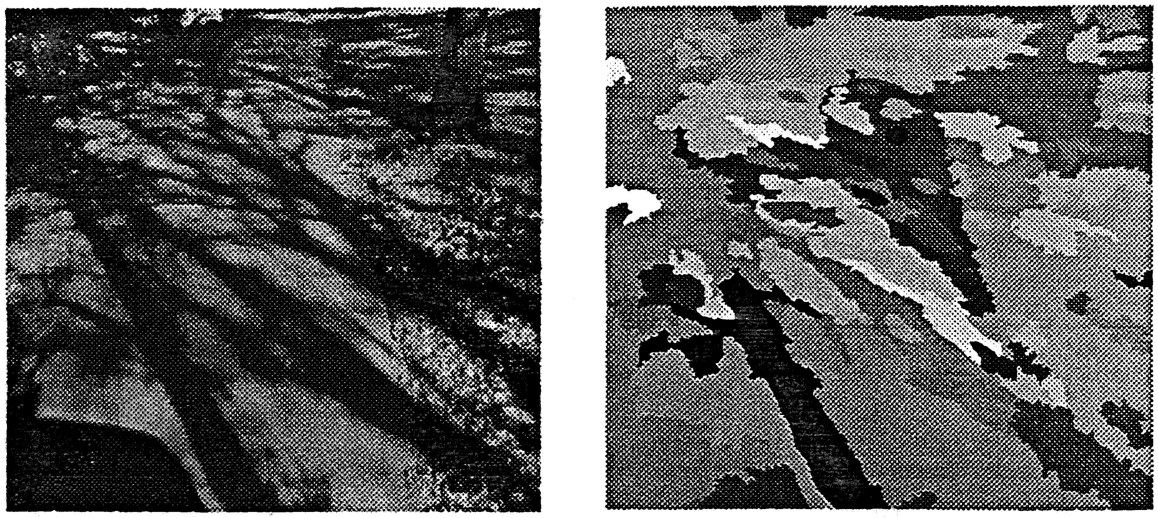


(c) Solution class (d) Interpretation



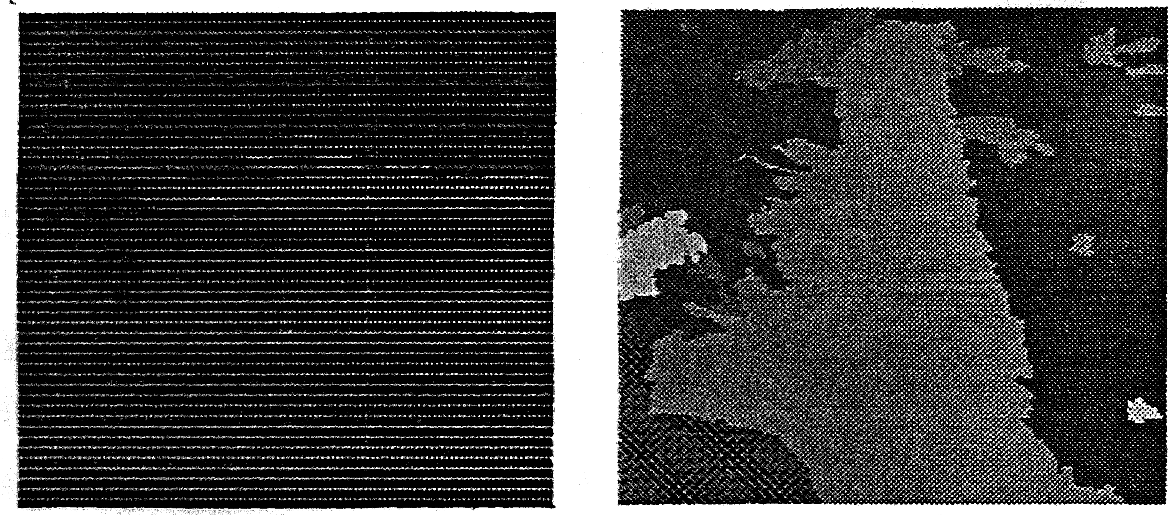
(e) Adjusted model (f) Generated map

Figure 8: Shaded road 1 (53 regions, 103 boundaries, 97 vertices, and 332 lines)



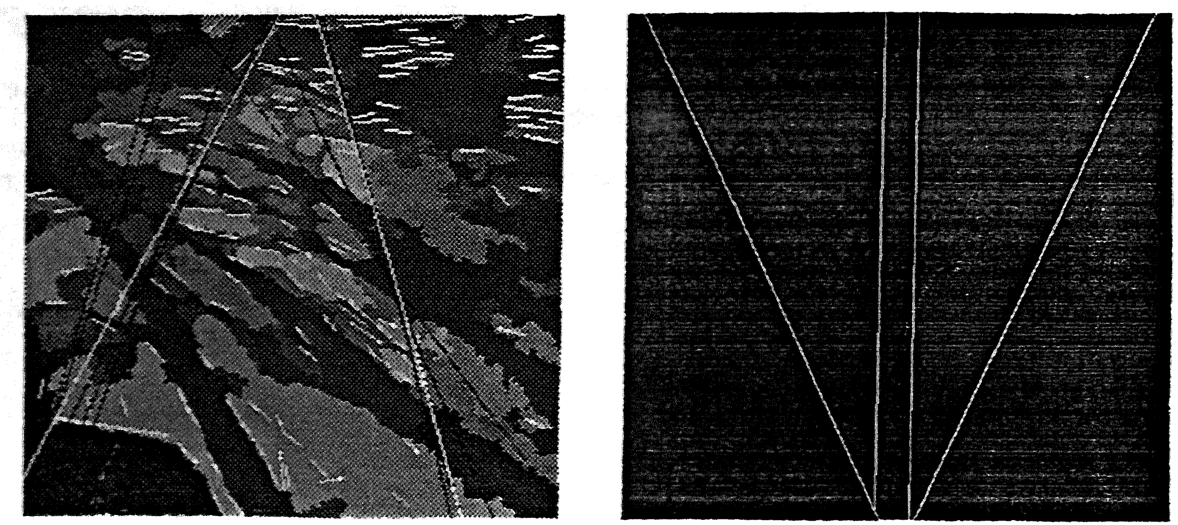
(a) Original image

(b) Region segmentation



(c) Solution class

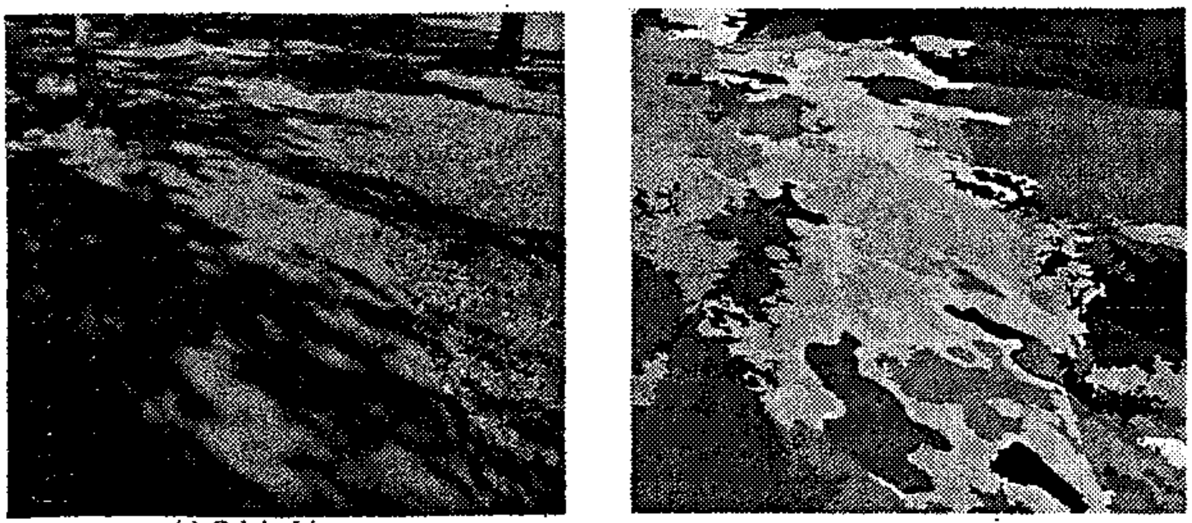
(d) Interpretation



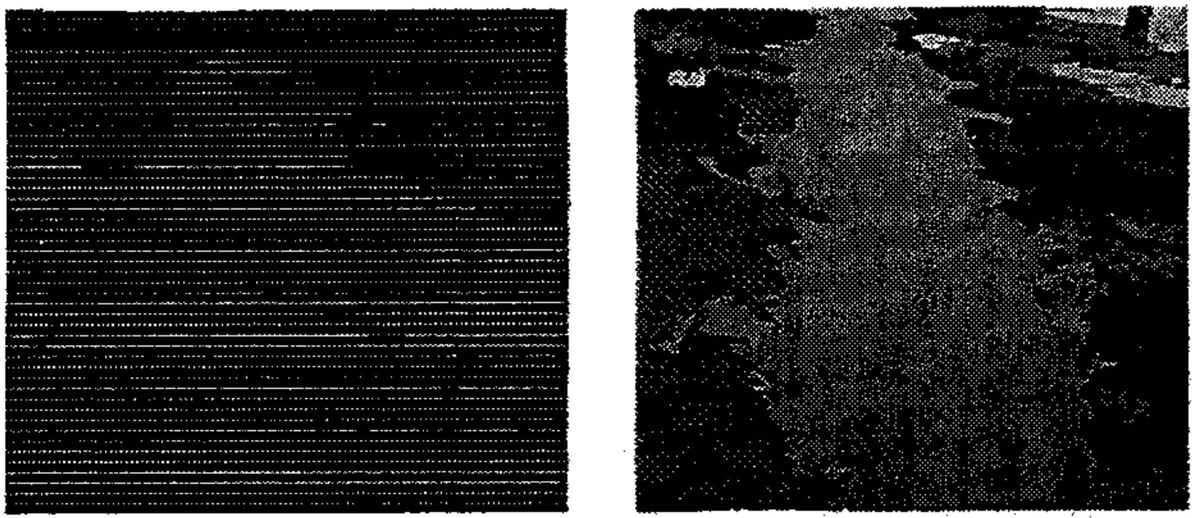
(e) Adjusted model

(f) Generated map

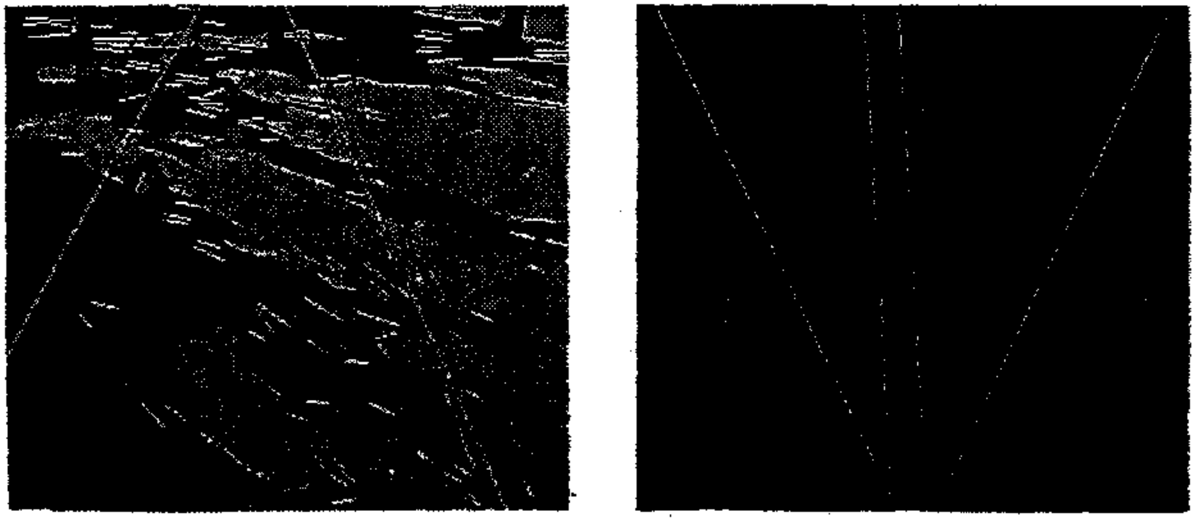
Figure 9: Shaded road 2 (122 regions, 299 boundaries, 229 vertices, and 139 lines)



(a) Original image (b) Region segmentation

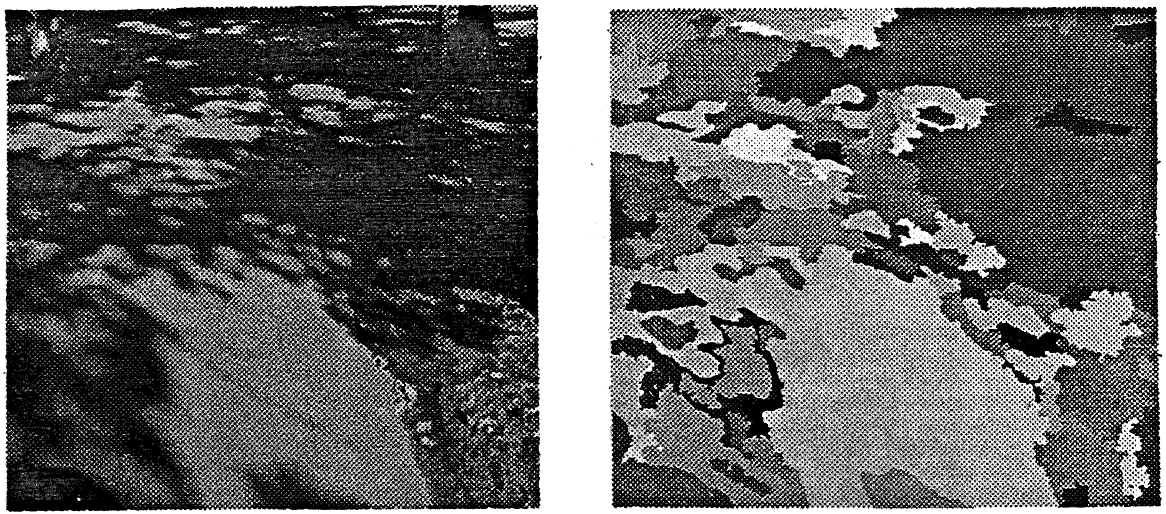


(c) Solution class (d) Interpretation

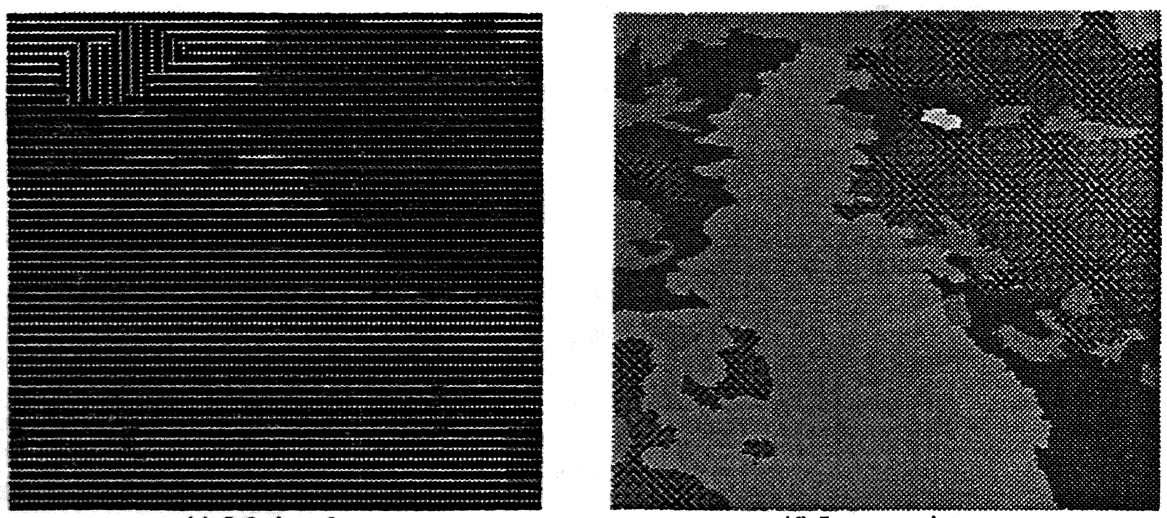


(e) Adjusted model (f) Generated map

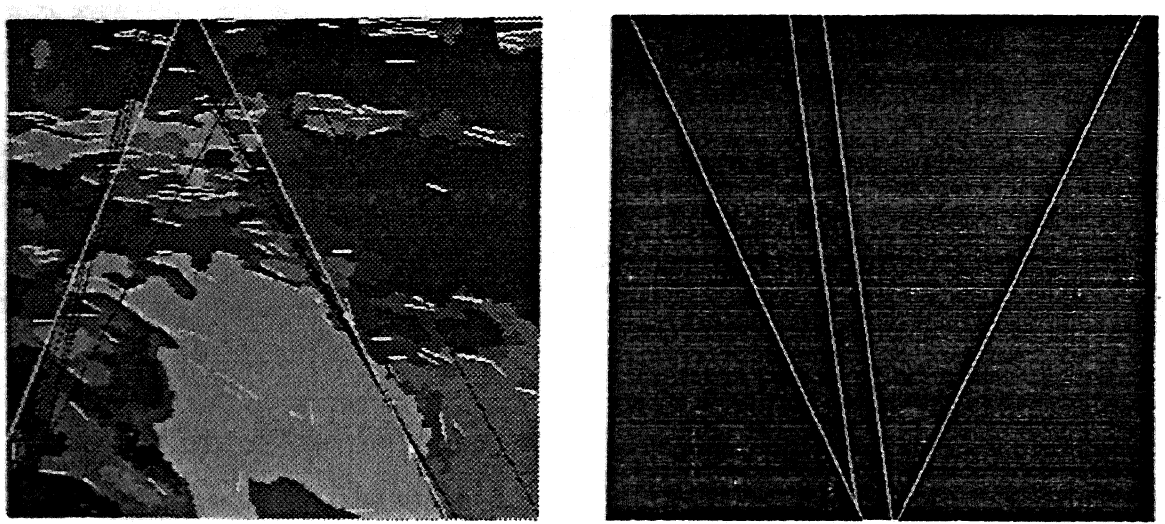
Figure 10: Shaded road 3 (113 regions, 271 boundaries, 226 vertices, and 169 lines)



(a) Original image (b) Region segmentation

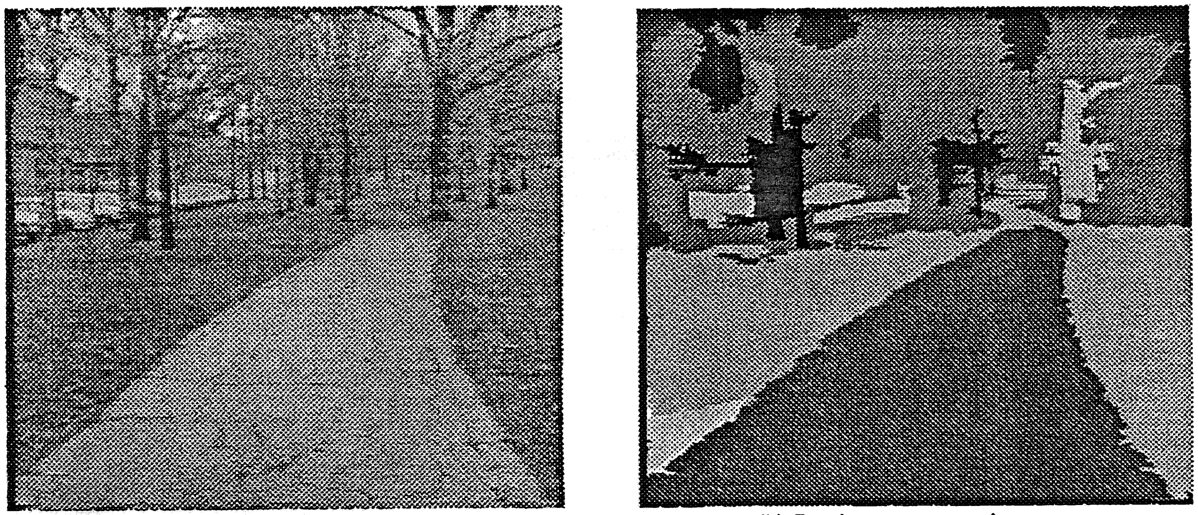


(c) Solution class (d) Interpretation

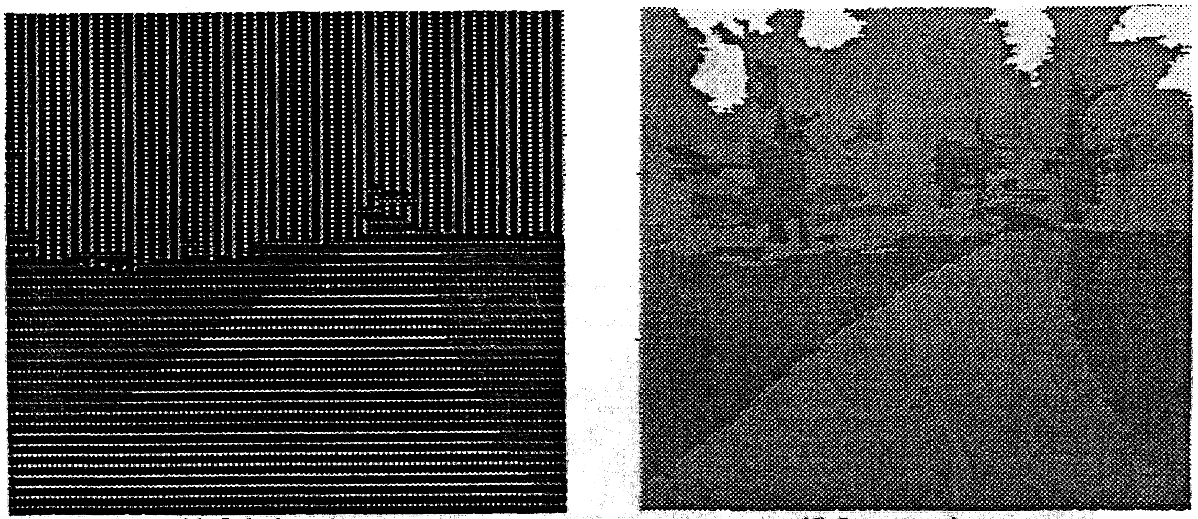


(e) Adjusted model (f) Generated map

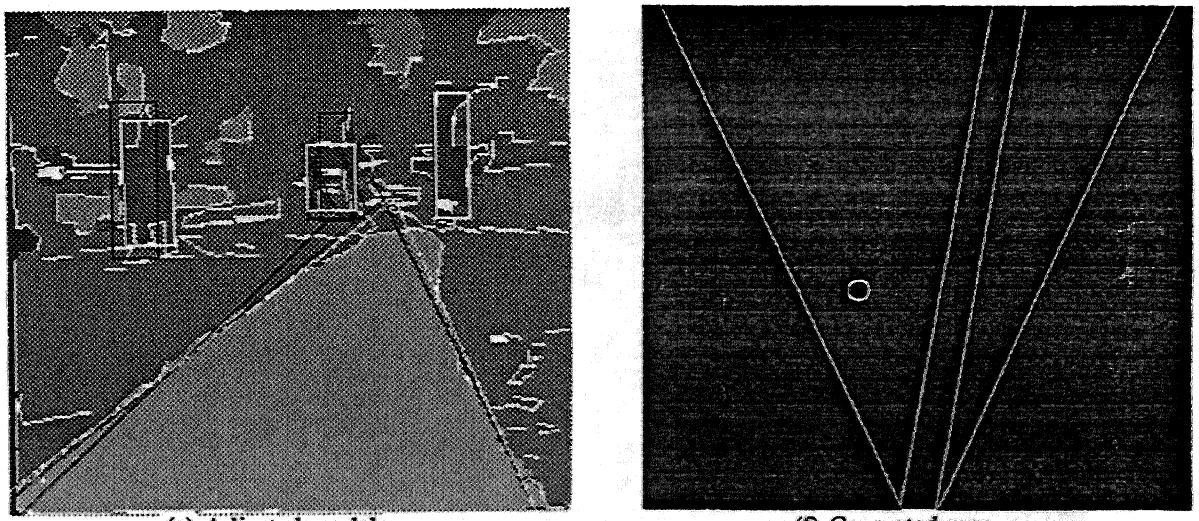
Figure 11: Shaded road 4 (121 regions, 300 boundaries, 233 vertices, and 291 lines)



(a) Original image (b) Region segmentation

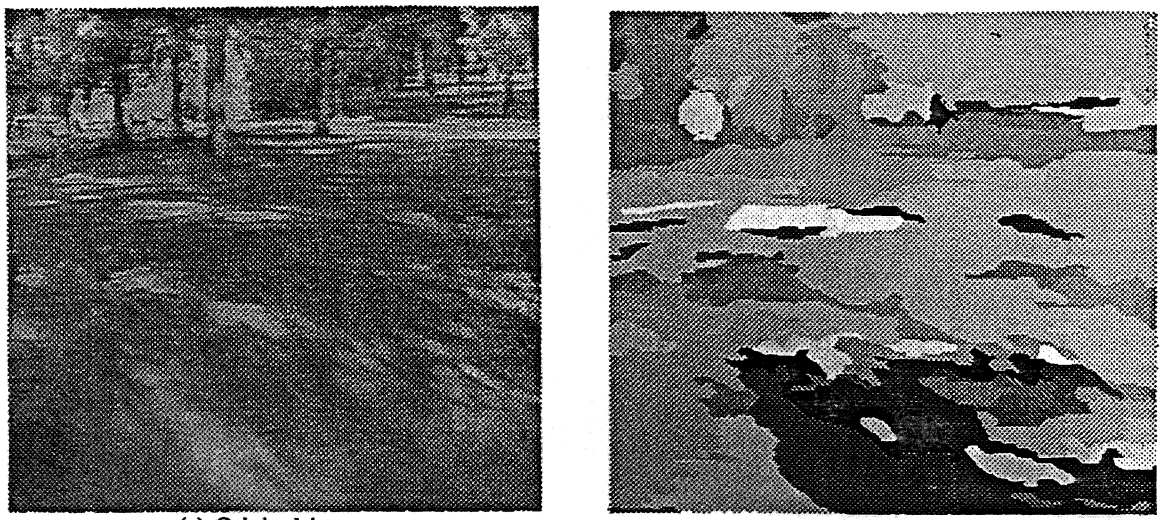


(c) Solution class (d) Interpretation



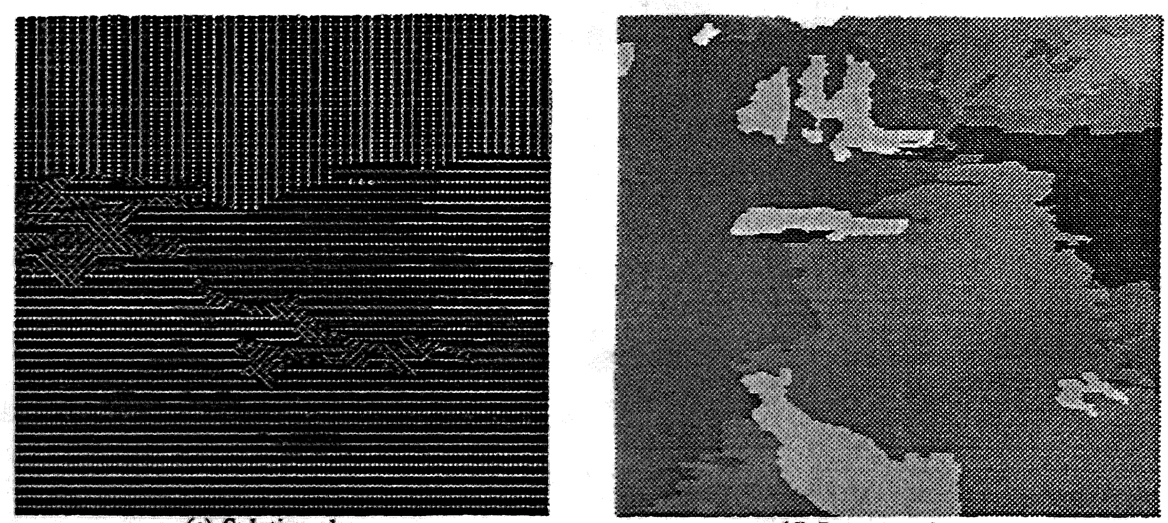
(e) Adjusted models (f) Generated map

Figure 12: Road with trees 1 (21 regions, 30 boundaries, 33 vertices, and 326 lines)



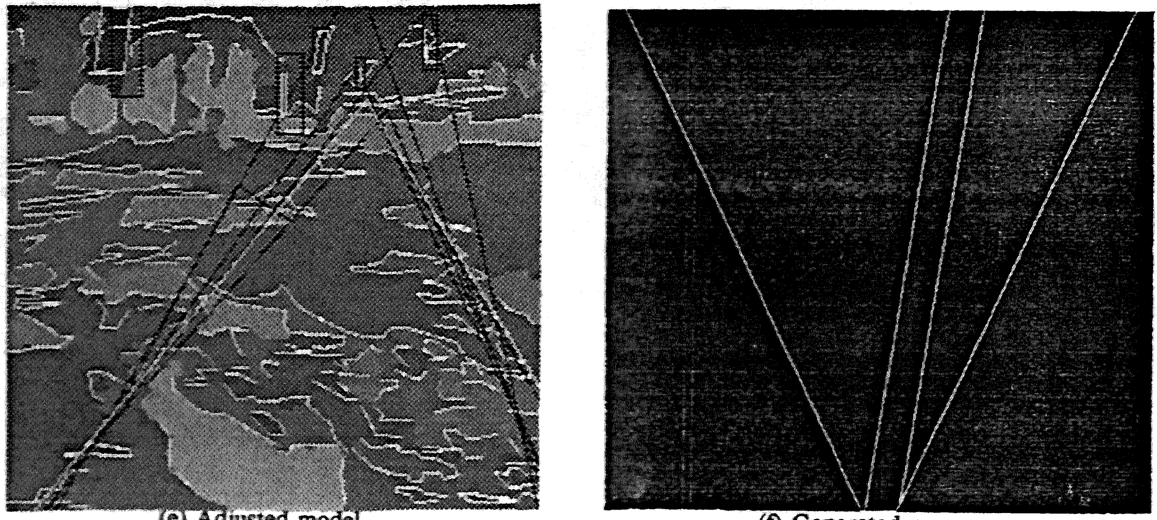
(a) Original image

(b) Region segmentation



(c) Solution class

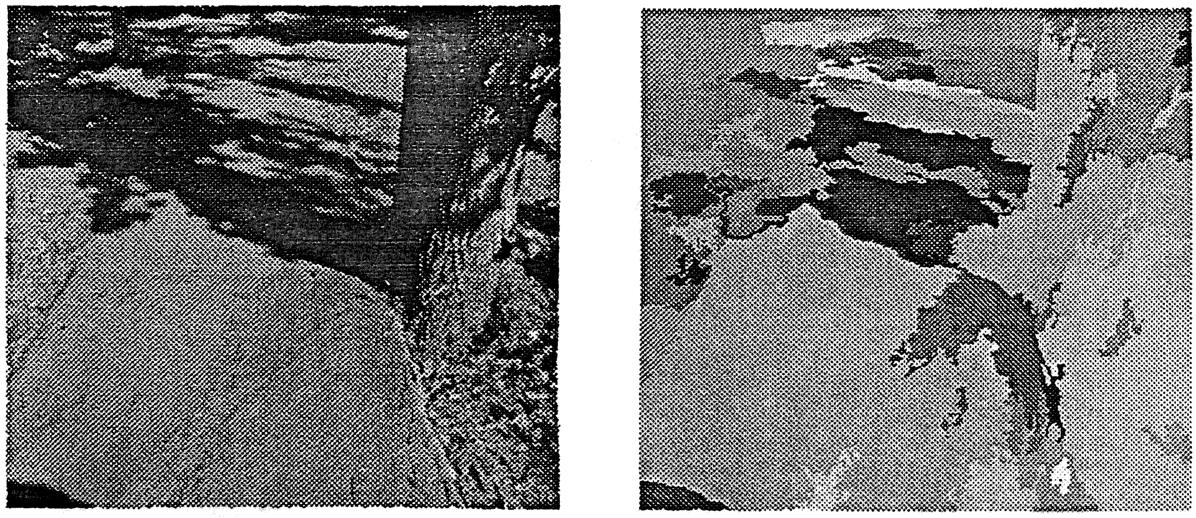
(d) Interpretation



(e) Adjusted model

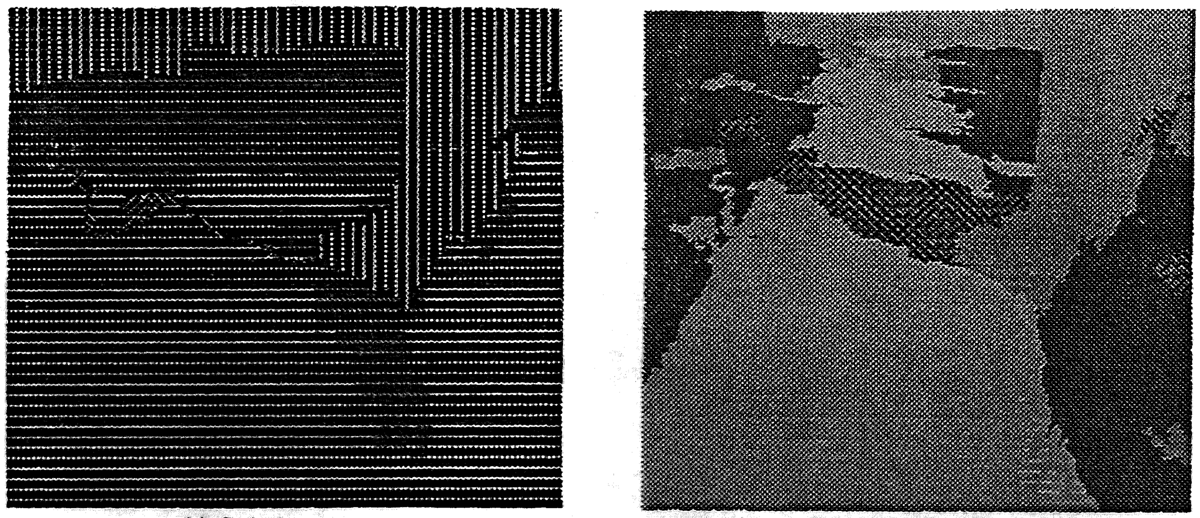
(f) Generated map

Figure 13: Road with trees 2 (95 regions, 255 boundaries, 213 vertices, and 543 lines)



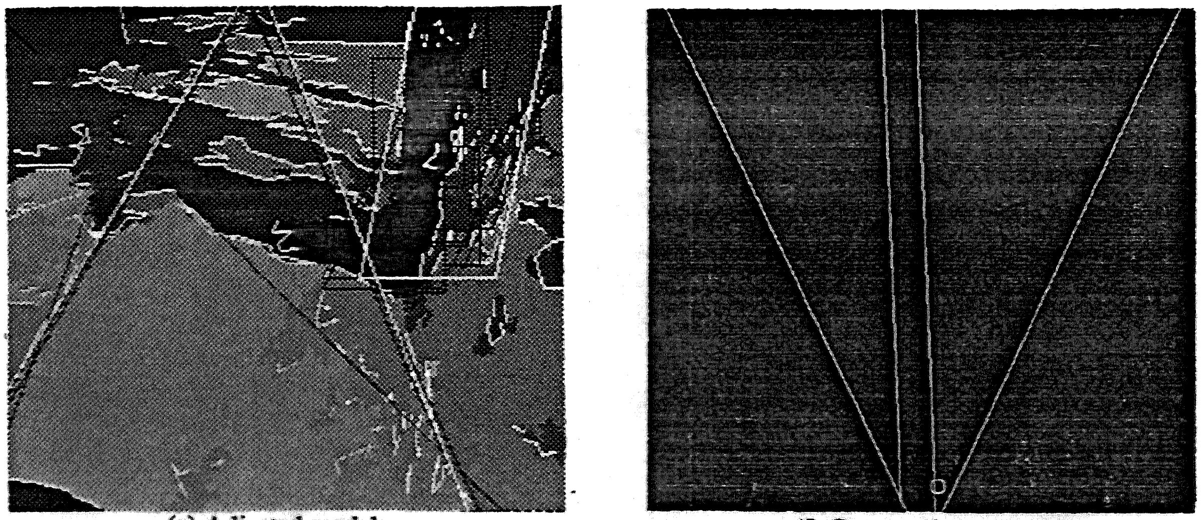
(a) Original image

(b) Region segmentation



(c) Solution class

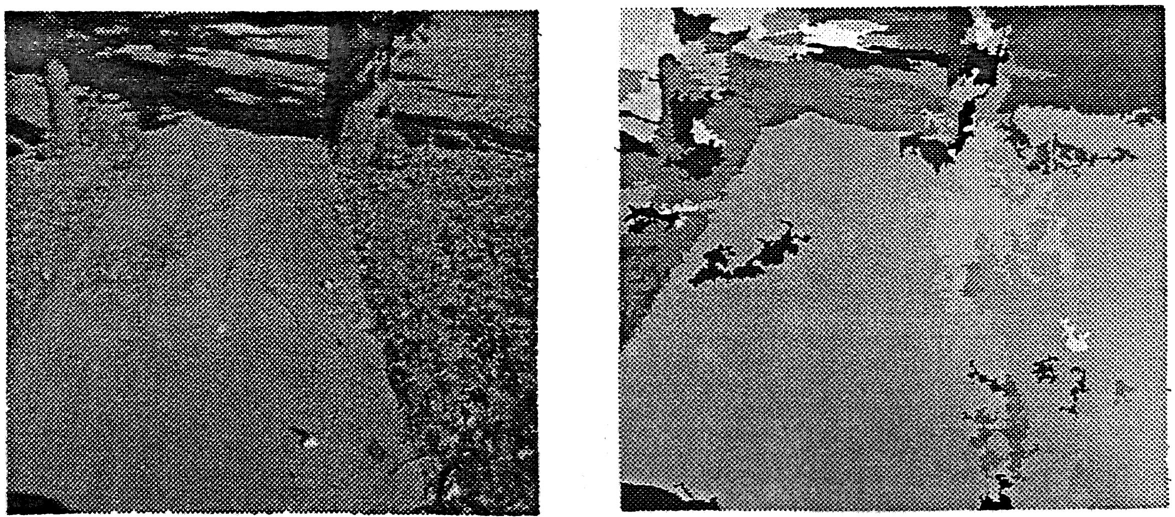
(d) Interpretation



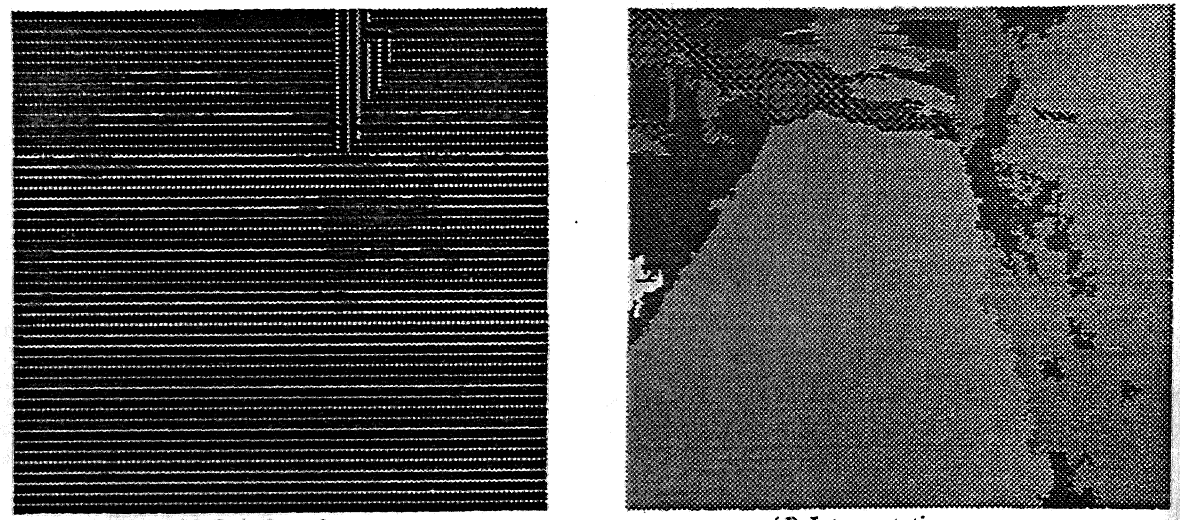
(e) Adjusted models

(f) Generated map

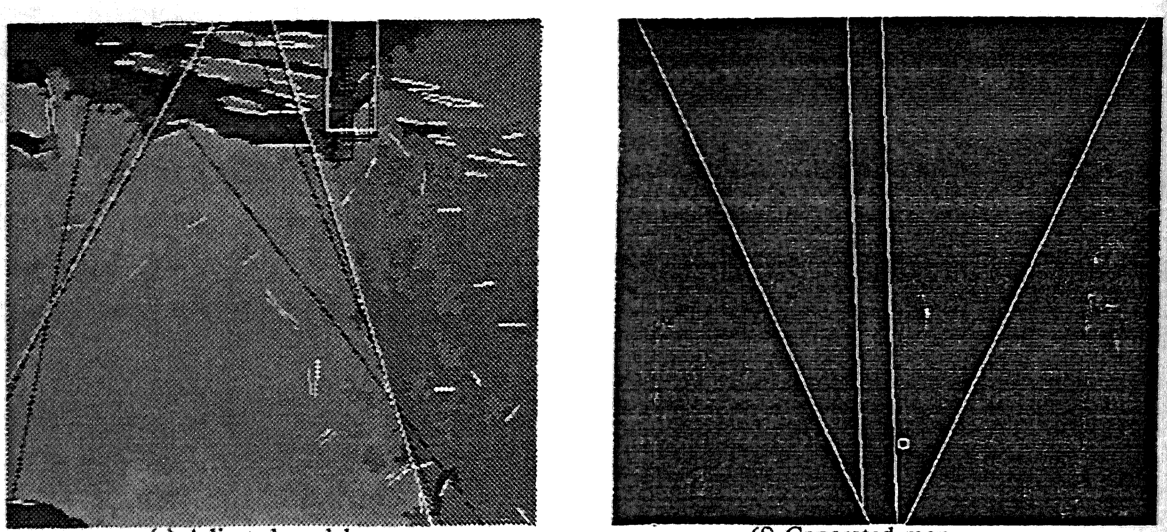
Figure 14: Road with a tree 3 (78 regions, 194 boundaries, 155 vertices, and 133 lines)



(a) Original image (b) Region segmentation

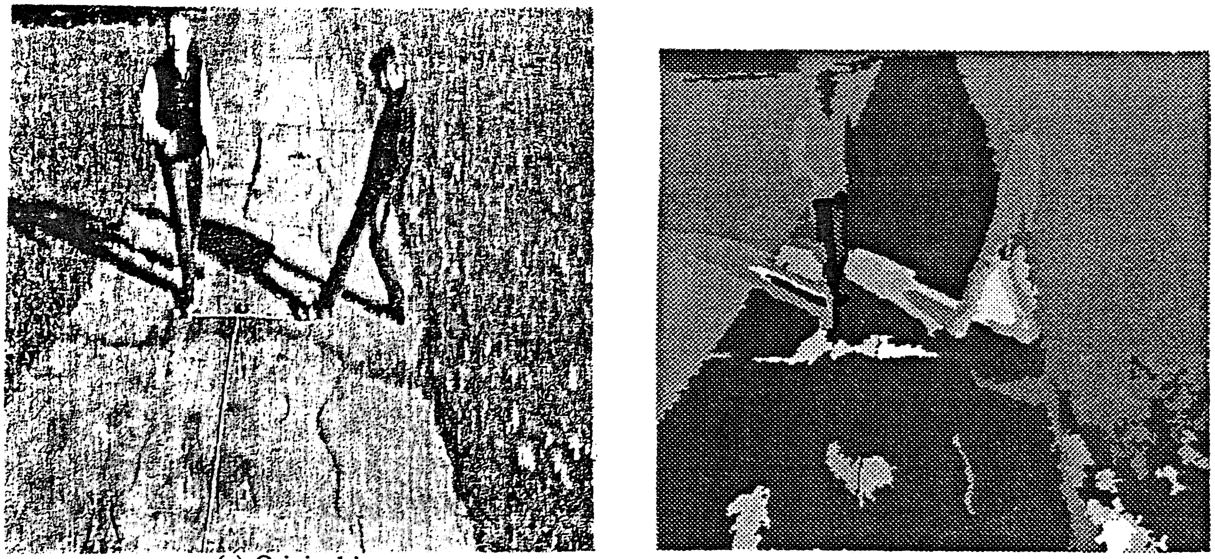


(c) Solution class (d) Interpretation



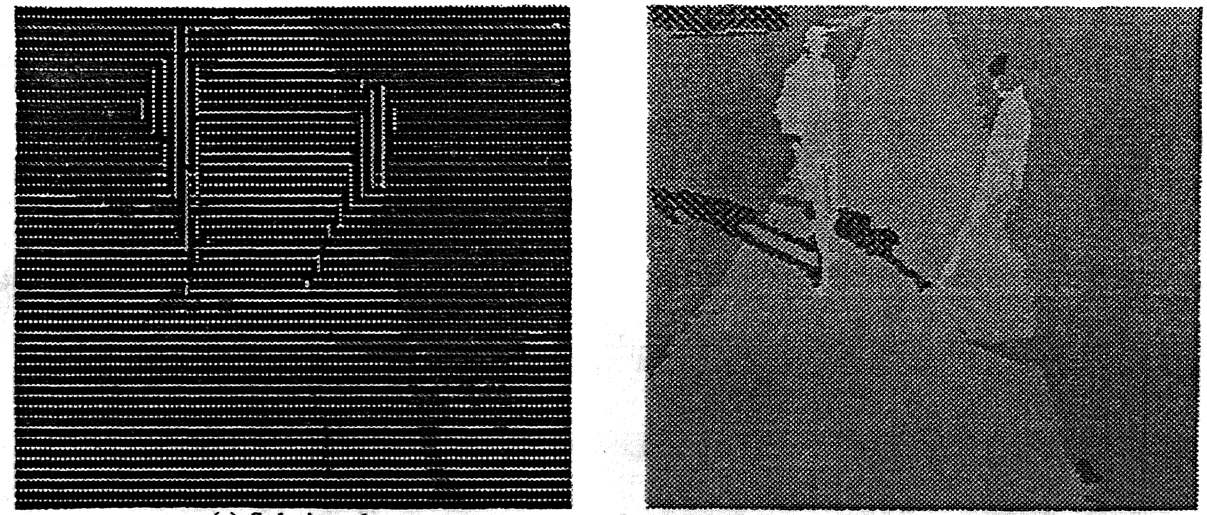
(e) Adjusted models (f) Generated map

Figure 15: Road with a tree 4 (96 regions, 234 boundaries, 187 vertices, and 96 lines)



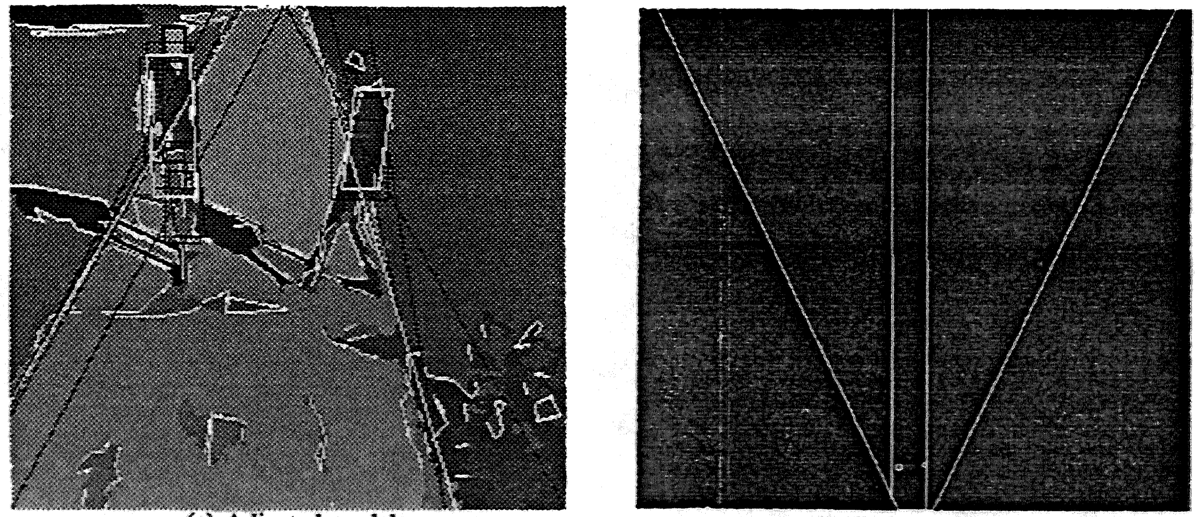
(a) Original image

(b) Region segmentation



(c) Solution class

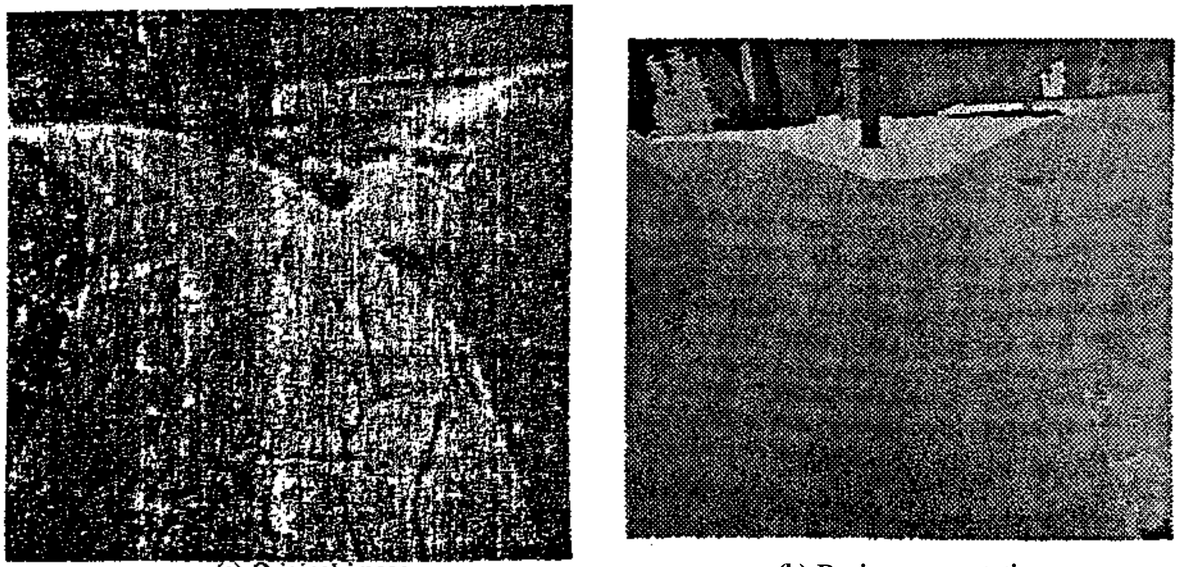
(d) Interpretation



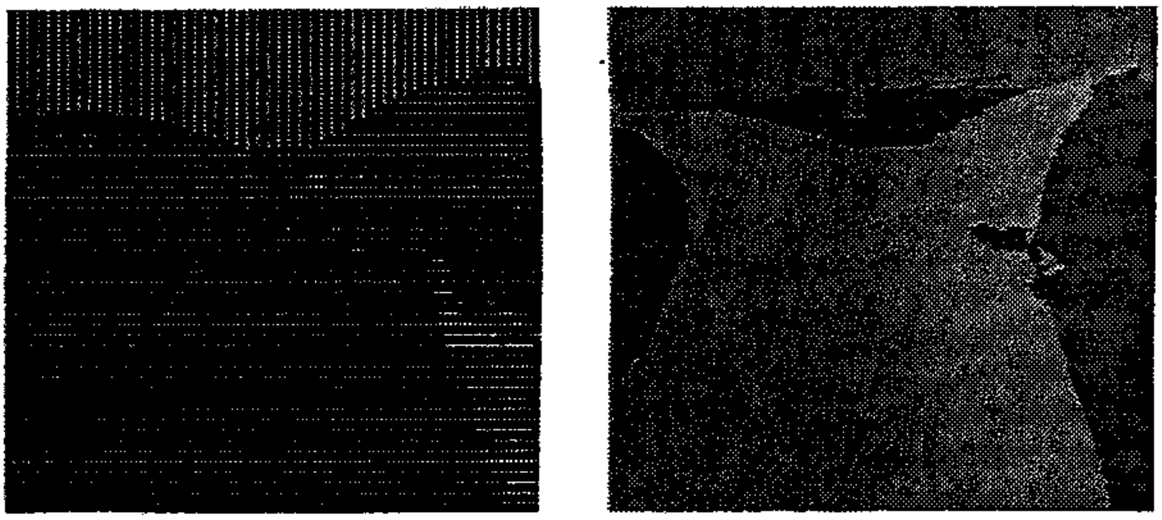
(e) Adjusted models

(f) Generated map

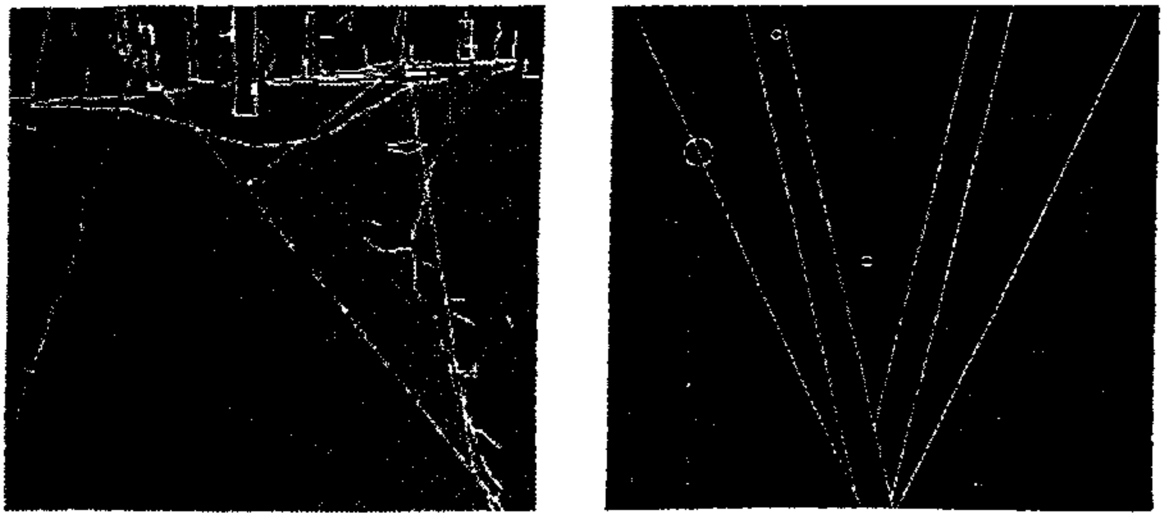
Figure 16: Road with vertical objects (54 regions, 116 boundaries, 98 vertices, and 228 lines)



(a) Original image (b) Region segmentation

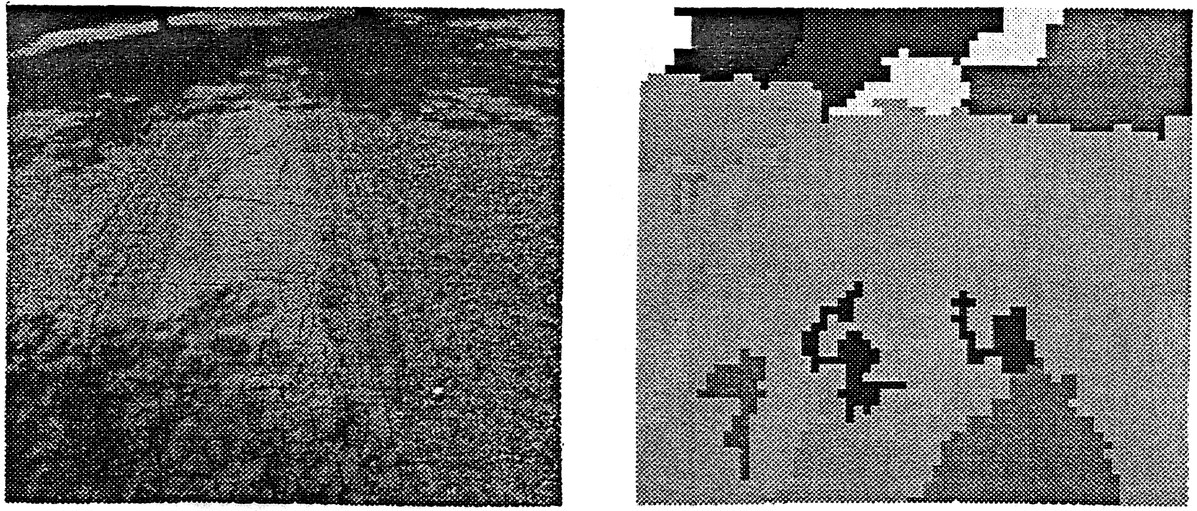


(c) Solution class (d) Interpretation

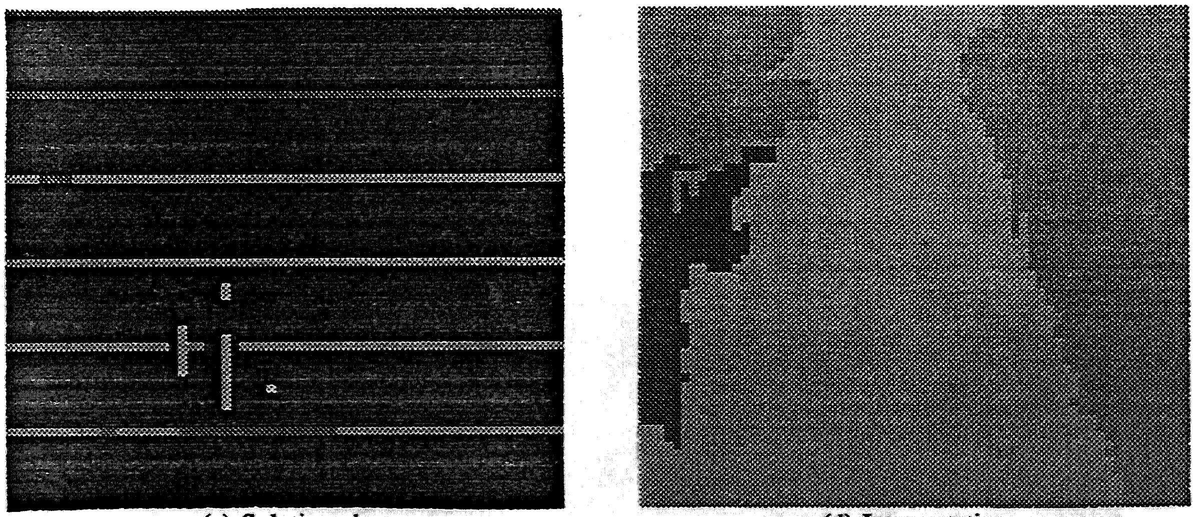


(e) Adjusted models (f) Generated map

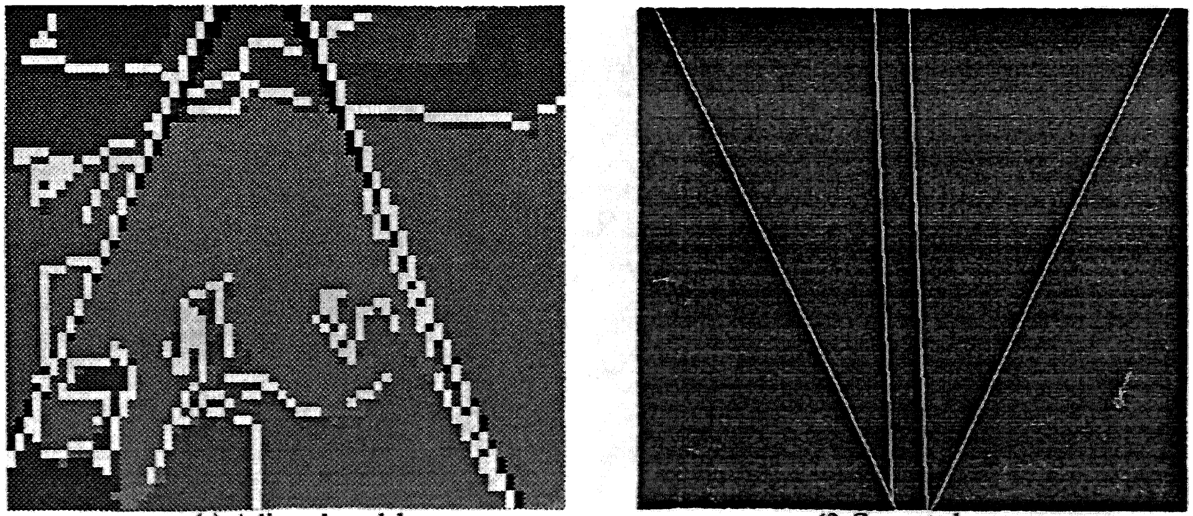
Figure 17: Intersection (39 regions, 67 boundaries, 67 vertices, and 159 lines)



(a) Original image (b) Region segmentation

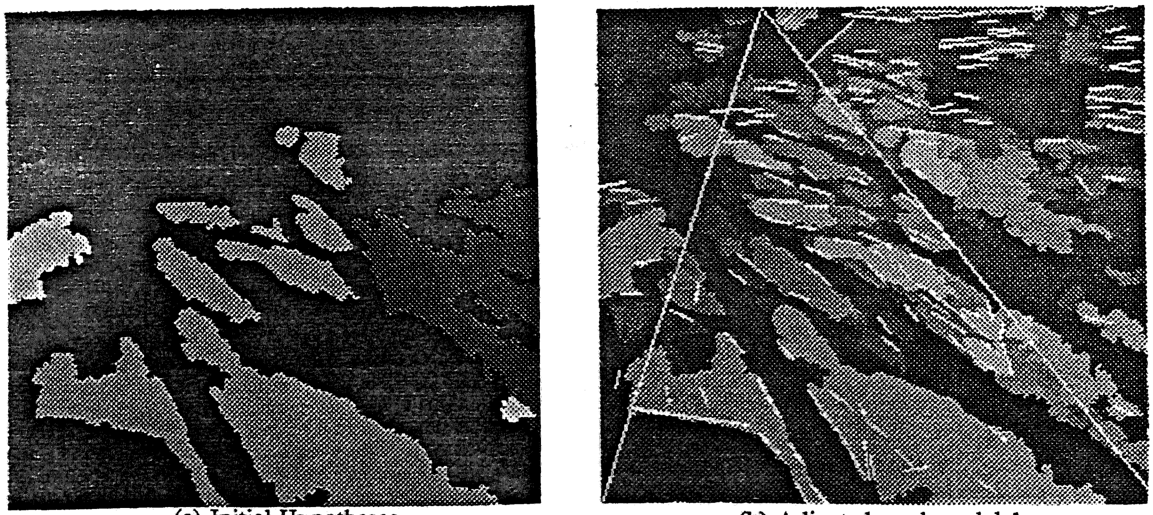


(c) Solution class (d) Interpretation

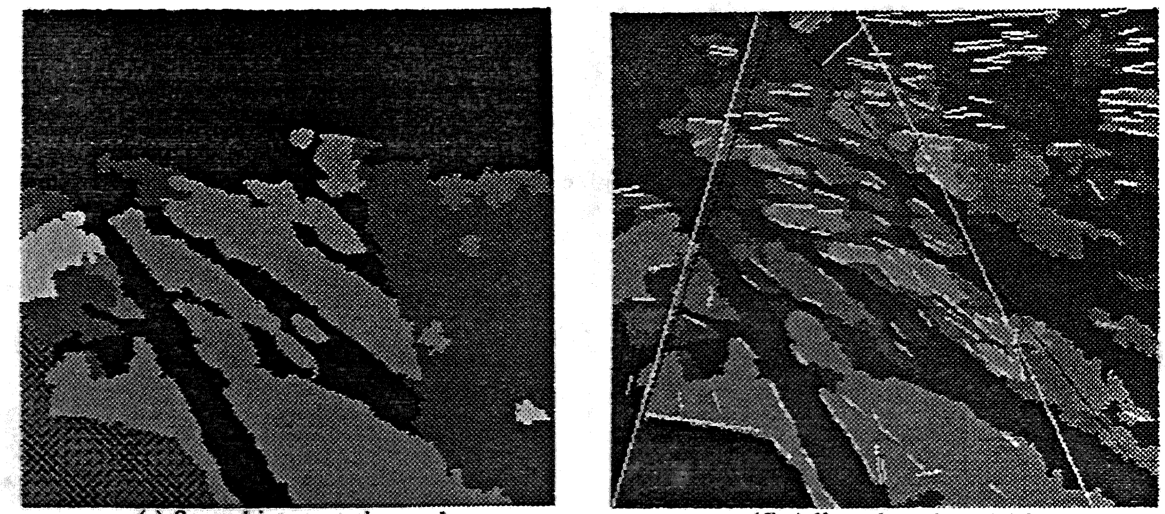


(e) Adjusted model (f) Generated map

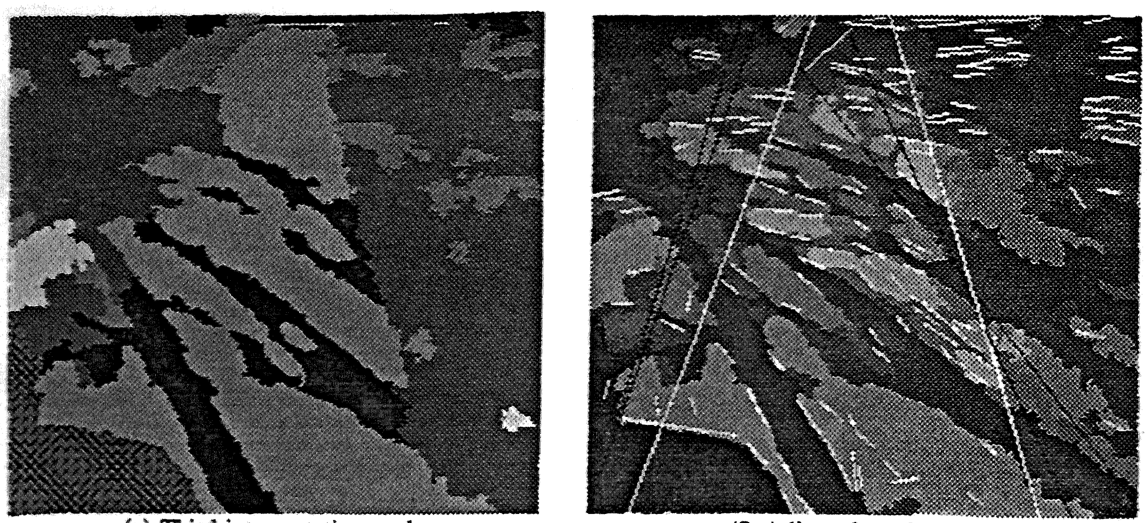
Figure 18: A full-automatic system example (18 regions, 32 boundaries, 43 vertices, and 105 lines)



(a) Initial Hypotheses (b) Adjusted road model 1

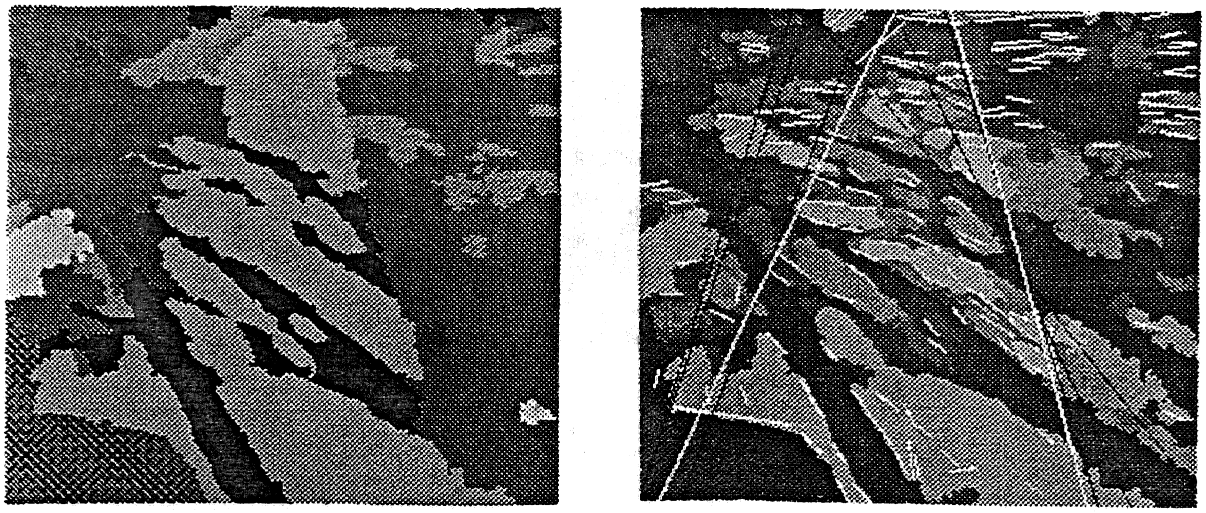


(c) Second interpretation cycle (d) Adjusted road model 2

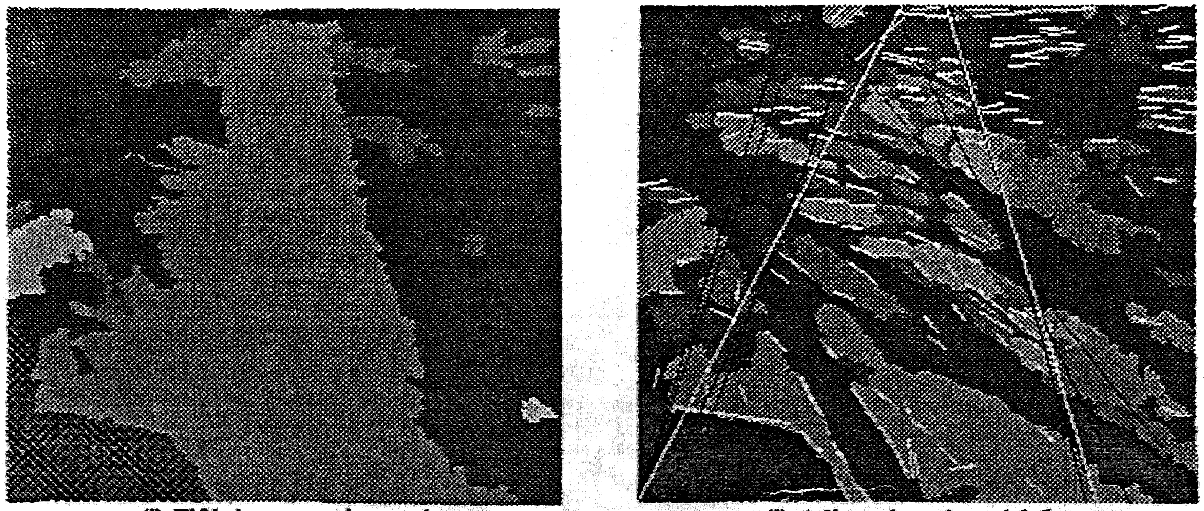


(e) Third interpretation cycle (f) Adjusted road model 3

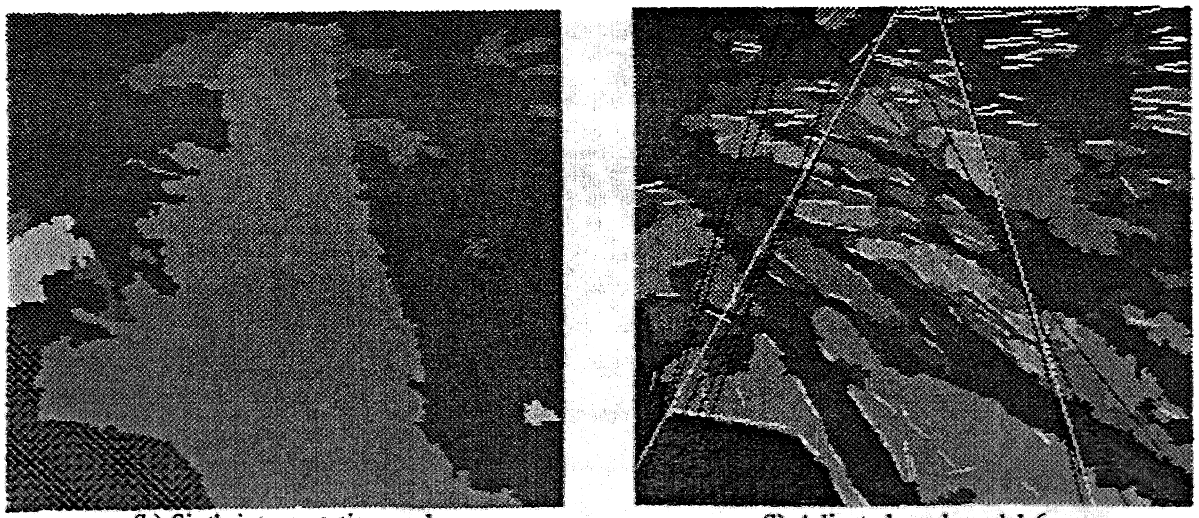
Figure 19: Behavior of a road model



(g) Fourth interpretation cycle (h) Adjusted road model 4

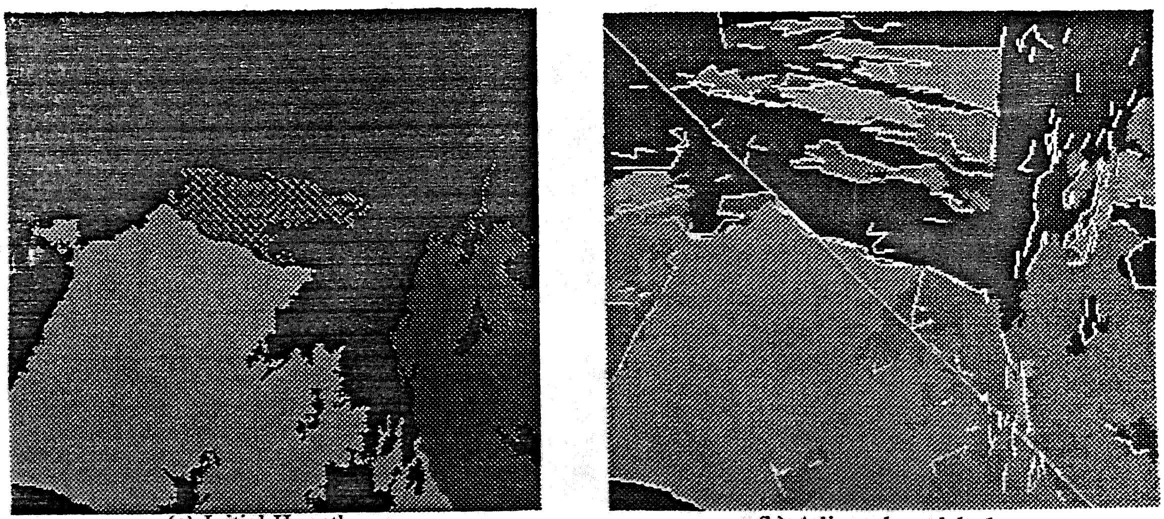


(i) Fifth interpretation cycle (j) Adjusted road model 5

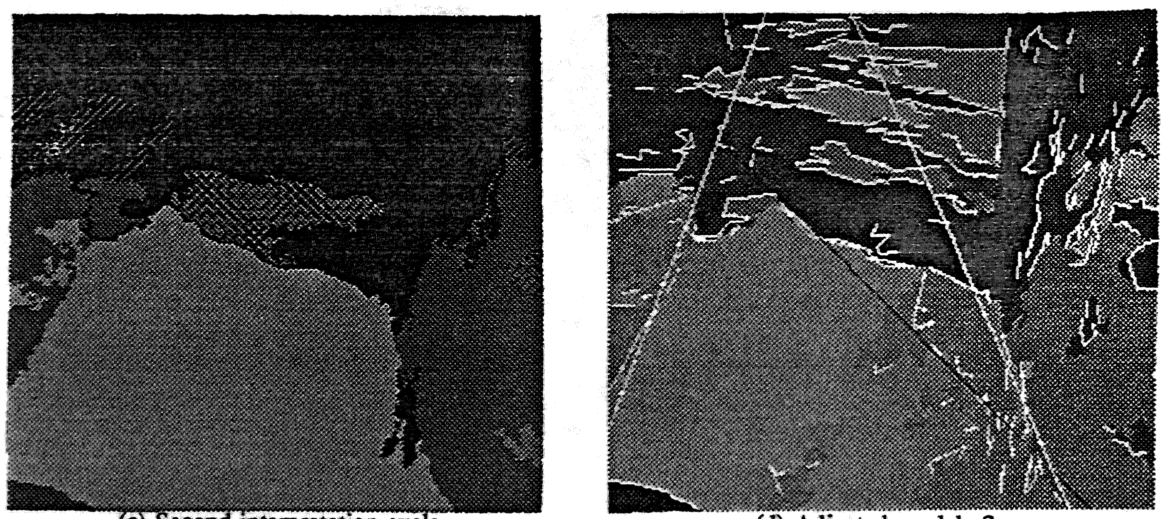


(k) Sixth interpretation cycle (l) Adjusted road model 6

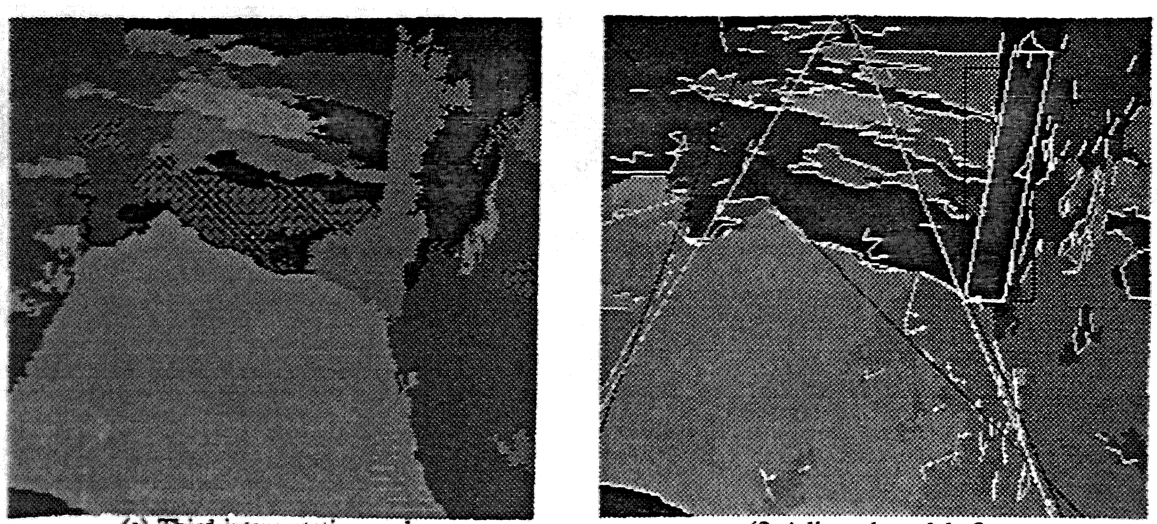
Figure 19: Behavior of a road model



(a) Initial Hypotheses (b) Adjusted models 1

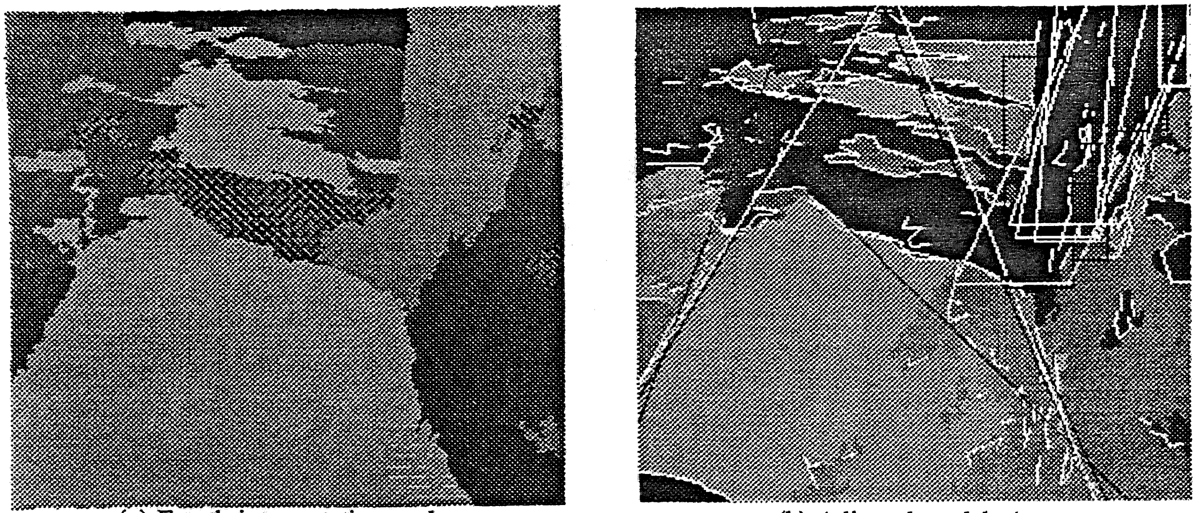


(c) Second interpretation cycle (d) Adjusted models 2

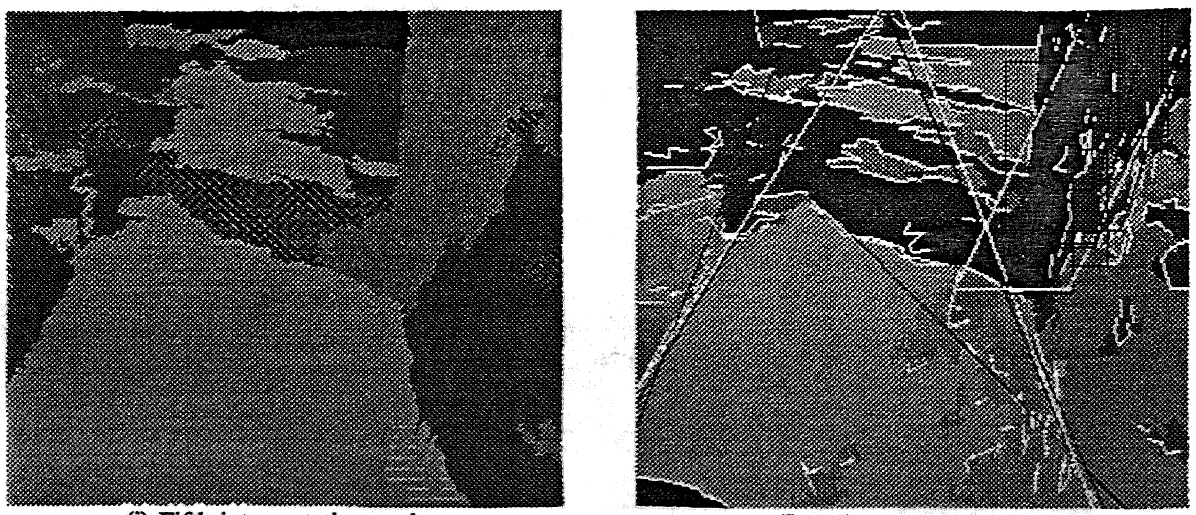


(e) Third interpretation cycle (f) Adjusted models 3

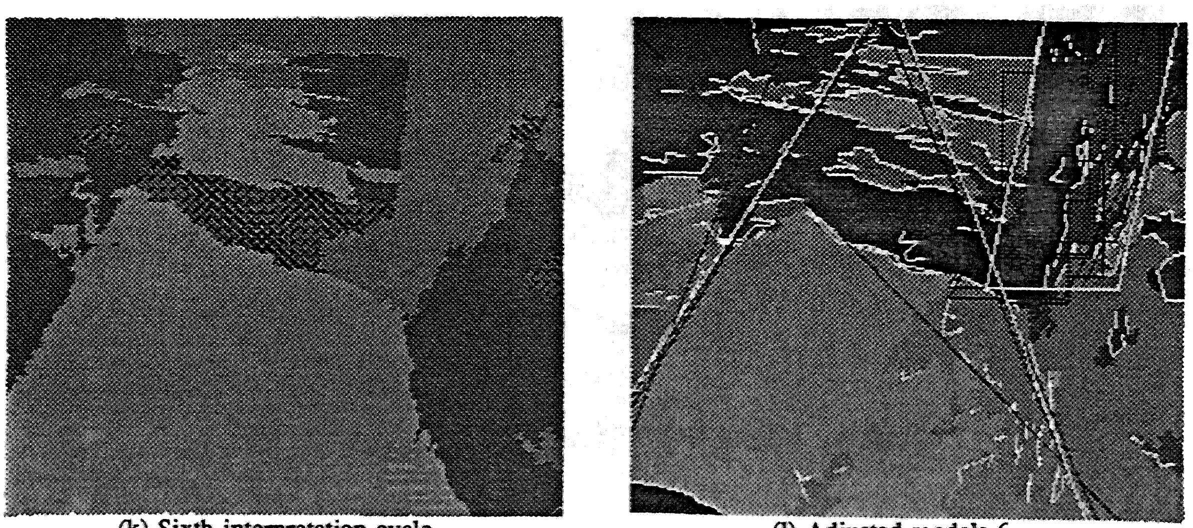
Figure 20: Behavior of scene models



(g) Fourth interpretation cycle (h) Adjusted models 4

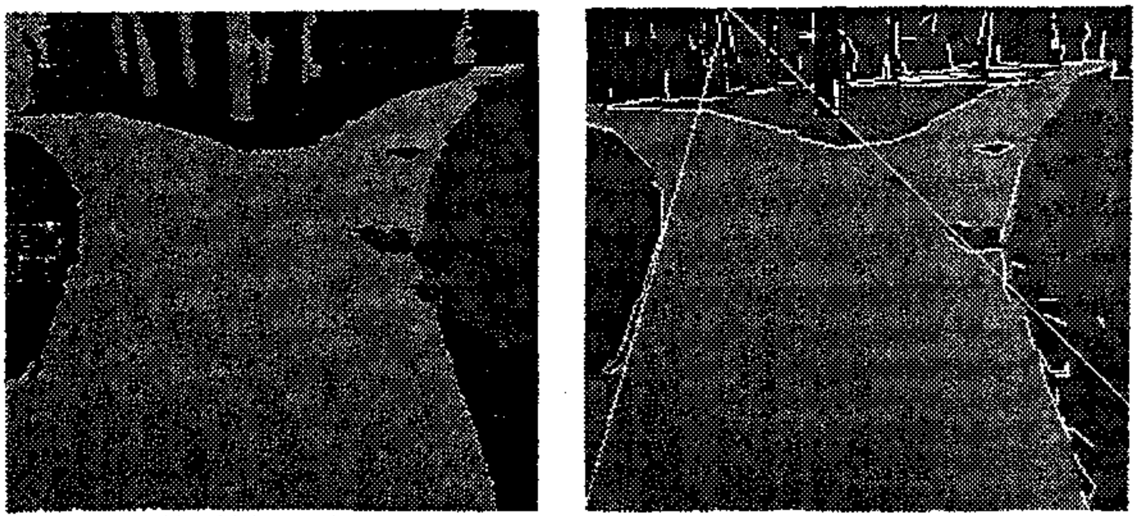


(i) Fifth interpretation cycle (j) Adjusted models 5



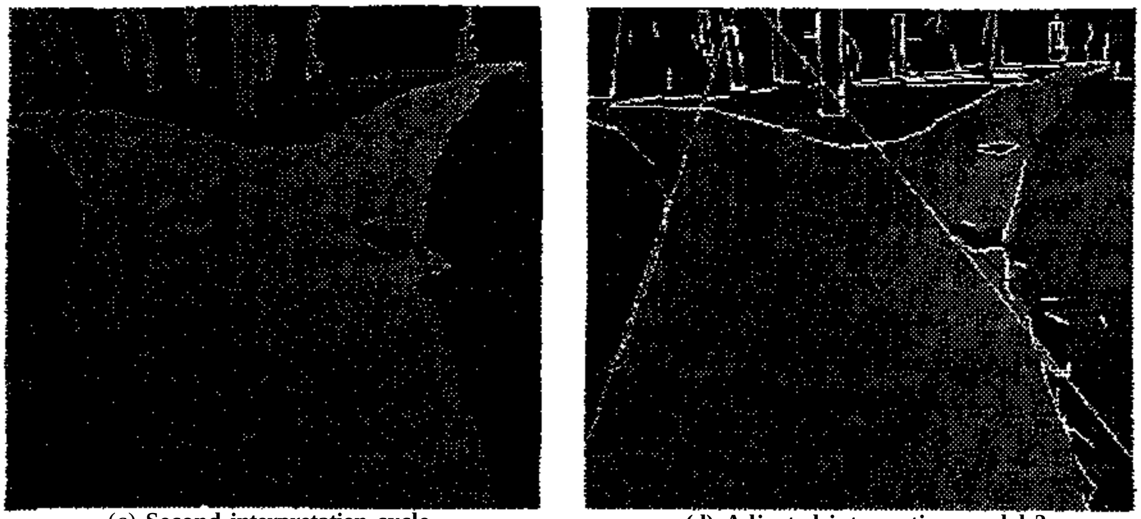
(k) Sixth interpretation cycle (l) Adjusted models 6

Figure 20: Behavior of scene models



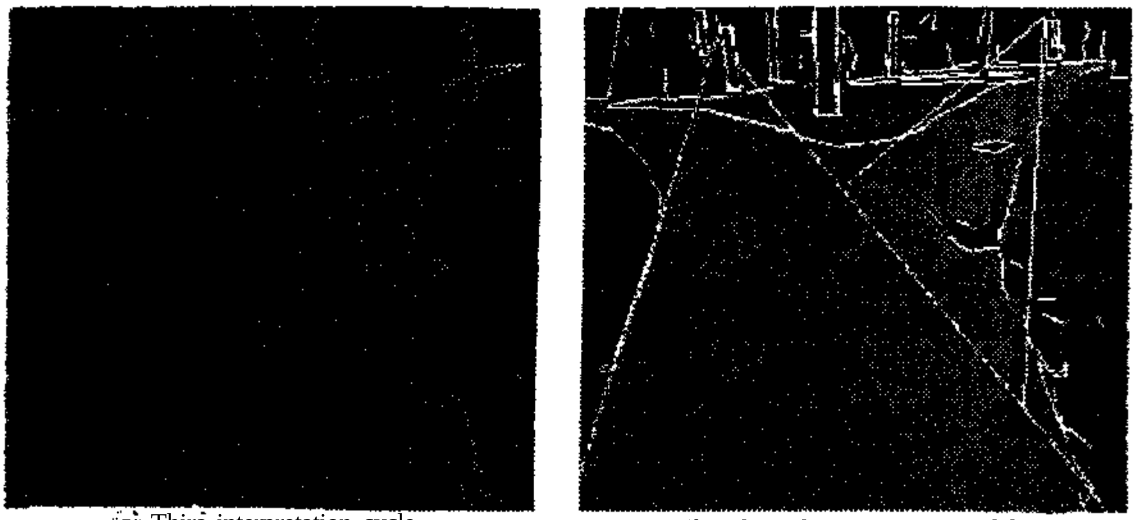
(a) Initial Hypotheses

(b) Adjusted intersection model 1



(c) **Second interpretation cycle**

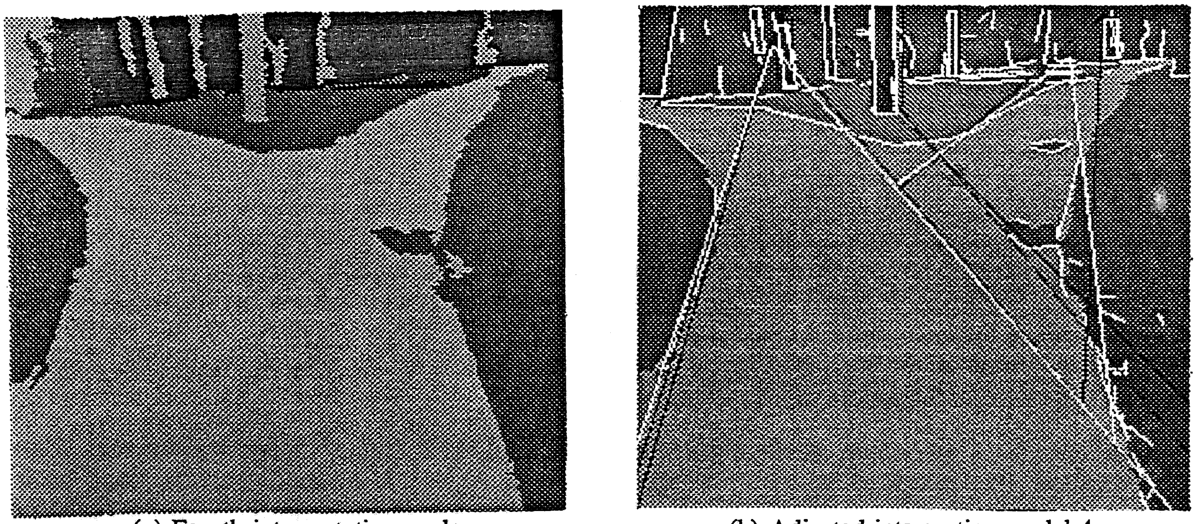
(d) Adjusted intersection model 2



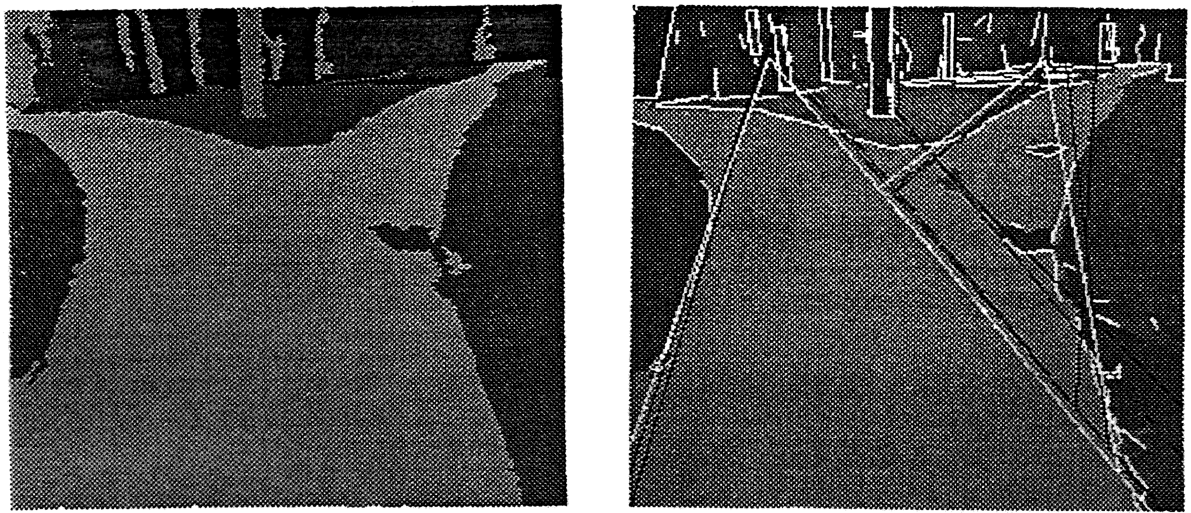
(e) Third interpretation cycle

(f) Adjusted intersection model 3

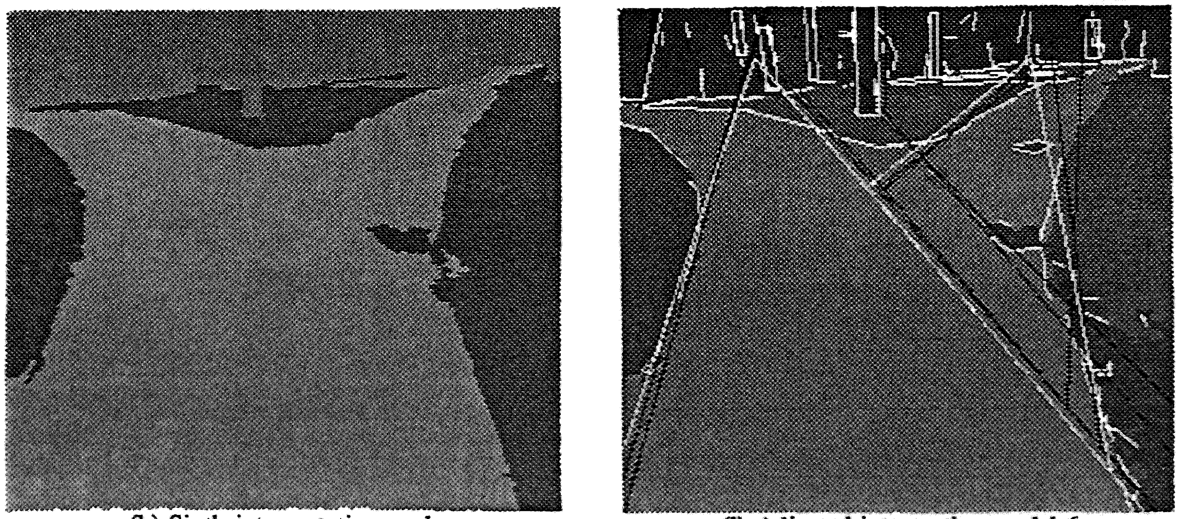
Figure 21: Behavior of a intersection model



(g) Fourth interpretation cycle (h) Adjusted intersection model 4



(i) Fifth interpretation cycle (j) Adjusted intersection model 5



(k) Sixth interpretation cycle (l) Adjusted intersection model 6

Figure 21: Behavior of a intersection model

6. The Road Scene Interpretation System in Detail

6.1. Feature Extraction and Intermediate Representation

The quality of the semi-automatic region segmentation by PHOENIX is better than fully-automatic segmentation, but still far from ideal. Especially when the image is shaded and complicated, it often generates hair-like long and narrow portions extending from the bodies along edges. Some regions are not separated and some are fragmented. The boundaries are jagged. In the feature extraction, first the region trimmer is executed on the region map to alleviate these problems. It breaks hair-like portions, cuts bridges connecting two big bodies, and then merges each small noisy fragment to the adjacent region which shares the longest boundary. After region trimming, the intermediate representation data are calculated. The data structures of the intermediate representation are similar to those of the work by Ohta [4].

In addition to the region map, regions, boundaries, vertices, and lines are the primary elements of the intermediate representation. Their structures are shown in Figure 22, Figure 23, Figure 24, and Figure 25. The relation among regions, boundaries, vertices, and lines are depicted in Figure 26. A region is enclosed by several boundaries; each one of them is terminated by two vertices. A line is linked to the nearest boundary.

```
struct region{
  region identification number

  number of neighbors
  neighbor identification numbers
  number of holes
  hole identification numbers
  number of boundaries
  boundary identification numbers

  mean intensity
     (Black&White, R, G, B, R-B)
  standard deviation
     (Black&White, R, G, B, R-B)
  texture value

  area
  perimeter
  mass center row, column
  minimum bounding rectangle (MBR)
     rowmin, rowmax, colmin, colmax
  scatter matrix
}
```

Figure 22: Region data structure

6.1.1. Region Description

Figure 27 describes a region which has various geometrical properties in addition to statistical properties.

Texture. Texture value extraction algorithm is as follows:

```
struct boundary{
   boundary identification number

   vertex identification numbers
   region identification numbers
   number of lines
   line identification numbers

   length
   contrast
   chain code length
   chain code
}
```

Figure 23: Boundary data structure

```
struct vertex{
   vertex identification number

   number of boundaries
   boundary identification numbers

   postion row, column
}
```

Figure 24: Vertex data structure

1. Derive a binary edge image from the black & white image with a 3 × Laplacian operator shown in Figure 28 (a) and a cutoff value T ($T = (standarddeviation) \times 1.4$).
2. By utilizing the 9 × 9 window shown in Figure 28 (b) on the binary picture, calculate the number of "1"s in the window. The number is the texture value for the pixel at the center of the 9×9 window.

Perimeter. Perimeter is a sum of the arc lengths of all boundaries, counting horizontal and vertical moves as one and diagonal moves, which need two steps, as $\sqrt{2}$.

Mass Center. Mass center of a region is computed as follows:

```
struct line{
   line identification number

   boundary identification number

   start row, column
   end row, column
   distance from origin
   orientation
   length
}
```

Figure 25: Line data structure

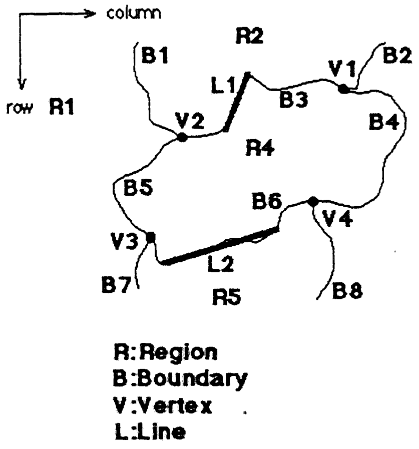

column
row
R1
R2
B1
B2
V1
L1
B3
V2
B4
R4
B5
B6
V4
V3
L2
B7
R5
B8
R:Region
B:Boundary
V:Vertex
L:Line

Figure 26: The relation description

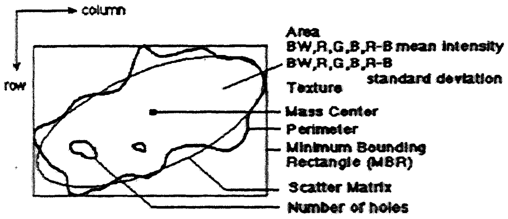

column
row
Area
BW,R,G,B,R-B mean intensity
BW,R,G,B,R-B standard deviation
Texture
Mass Center
Perimeter
Minimum Bounding Rectangle (MBR)
Scatter Matrix
Number of holes

Figure 27: The region description

$$M = \frac{1}{N}\sum_{i=1}^{N} P_i,$$

where N is the number of pixels in the region, P_i is the position vector of i-th pixel.

Minimum Bounding Rectangle. The Minimum Bounding Rectangle (MBR) is defined as a set of four numbers: rowmin, rowmax, colmin, and colmax. They are the maximum and minimum values of the pixel locations in the region.

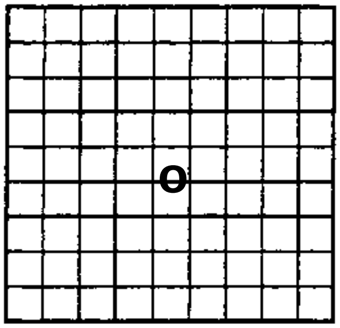
Scatter Matrix. The scatter matrix represents an elliptical area which approximates the shape of the region. The scatter matrix C of the region can be calculated as:

$$C = \frac{1}{N}\sum_{i=1}^{N}(P_i - M)(P_i - M)^t,$$

where N is the number of pixels in the region, P_i denotes the position vector of i-th pixel, and M denotes the mass center of the region.

0	1	0
1	-4	1
0	1	0

(a) Laplacian operator



(b) 9 x 9 window operator

Figure 28: A Laplacian operator and 9 x 9 window

6.1.2. Boundary Description

Figure 29 describes a boundary. Boundaries are the borders of regions in a region map. They are not related to the color edge detector.

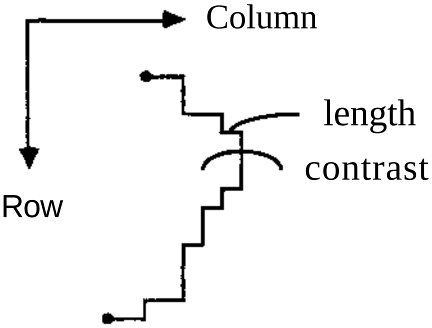


Figure 29: The boundary segment description

Length. Length is a sum of the arc lengths of the boundary, counting horizontal and vertical moves as one and diagonal moves, which need two steps, as *\PL*

Contrast Contrast is a difference of mean intmsity Black&White between the regions forming the boundary.

6.1.3. Vertex Description

Only three or four boundaries can meet at a vertex because of the square tessellation.

6.1.4. Line Description

Figure 30 describes a line. Lines are obtained by applying the Miwa line extractor [25] to the edges derived by the color edge detector [20]. The Miwa line extractor has a pair of connected arms tracking along the edges and determines a line by checking the angle formed by the arms.

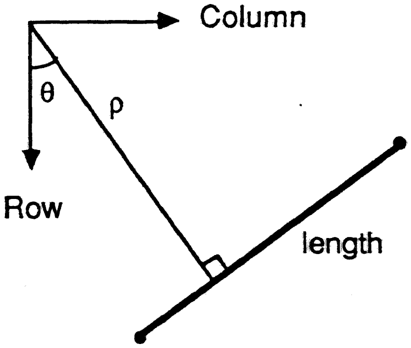


Figure 30: The line segment description

6.2. Initial Hypothesis Generation

The initial hypothesis generator picks up some tractable regions and makes initial hypotheses. It is skeptical and tries to avoid erroneous hypotheses.

Clancey's classification problem solving model [26] is applied to initial hypothesis generation. By analyzing more than 10 expert systems such as MYCIN, SACON, The Drilling Advisor, GRUNDY, and SOPHIE-III, he discovered a generic problem solving strategy for classification problem solving, which these systems have in common. Classification problem is a problem which classifies input data into known solutions. If the number of solutions is fairly limited, the model works very well. It has three steps. The first step is data abstraction: the input data are abstracted and converted to more tractable forms. The second step is heuristic match: data abstractions are projected heuristically into solution classes. Each solution class groups similar solutions. The last step is refinement: one of the known solutions in that solution class is selected by refining the data.

Figure 31 shows how the classification problem solving model is applied to the initial hypothesis generation for color road scene interpretation. The intermediate representation data are first abstracted by simple rules and the system makes data abstractions such as large area, green area, lengthwise area, parallel lines, etc. One region gets as many data abstractions as possible. Second, some regions with data abstractions are projected into solution classes: *on the level of ground* or *above ground. On the level of ground* groups roads, grass, soil, shadows, and sun spots. *Above ground* groups tree trunks, tree foliage, vertical object, and sky. Last, each region, which has fallen into a solution class, is refined and labeled with a solution selected out of the members of its solution class.

Solution classes are determined because the appearances of the objects protruding from the ground are very different from those of the objects on ground level. When tracing objects vertically, the appearance of the *above*

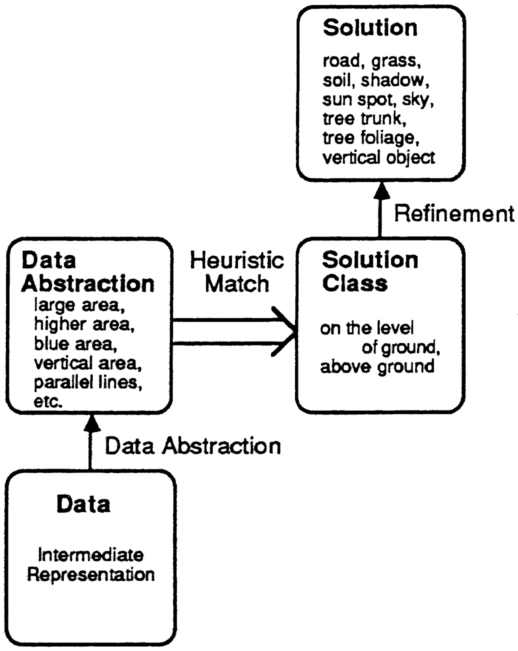

Solution
road, grass, soil, shadow, sun spot, sky, tree trunk, tree foliage, vertical object
Refinement
Data Abstraction
large area, higher area, blue area, vertical area, parallel lines, etc.
Heuristic Match
Solution Class
on the level of ground, above ground
Data Abstraction
Data
Intermediate Representation

Figure 31: Classification problem solving model applied to initial hypothesis generator

ground objects such as tree trunks does not change so much, whereas, the *on the level of ground* objects such as roads change drastically because of the perspective projection. When generating initial hypotheses, it is less dangerous to pick up one of the solution classes first and then refine it than to pick up one of the solutions directly.

6.2.1. Data Abstraction

All the data abstractions are listed in Table 1. There are six categories: area, location (in a image), color, texture, shape, and line. The color category has three subcategories: bright, normal, and dark. Data abstraction rules are simple. They check some features of regions and generate data abstractions. Color data abstractions are not very exclusive. For example, one region can be blue area and greener area at the same time.

Data abstraction is the most important component of the initial hypothesis generation. Thus, it is one of the most important components in the road scene interpreter. Though it partly depends on the low-level region segmentation as mentioned in Section 1., some robust ideas are embedded in the data abstraction rules:

Look at Neighbor Regions. Compare features with those of the neighbor regions, then more reliable data abstractions can be generated. Though global information of the scene is not available, rules can check the neighbors locally. Two actual examples are shown in Figure 32. If a region is darker than its neighbor regions, the region is really dark. If the color of a region is closest to pure green (Red=0, Green=255, Blue=0) among neighbor regions, it is guaranteed that the region is really green.

Use Color Ratio . Use the color ratio such as Red/Blue and Green/Blue. Do not use R, G, and B intensities directly except for detecting dark or bright regions. R, G, and B intensities vary significantly, depending on weather, camera iris, camera white balance, video amplifier, A/D converter, etc. R, G, and B intensities were used directly in the ancestral system INSIGHT-II, but as the number of images increased, contradicting intensities were observed. Consequently, in INSIGHT-III, the color usage was totally rebuilt by discarding direct color intensity references and

Category		Data Abstracion
Area		large area, moderately large area
Location		higher area, lower area
Color	Bright	bright area, bright red area, bright green area
	Normal	blue area, bluest area, green area, greener area, brown area
	Dark	dark area, darker area, darkest area
Texture		textural area
Shape		lengthwise area, crosswise area, vertical parallel area, trapezoidal area
Line		long line, linear lines, horizontal line, vertical line, vertical parallel lines, trapezoidal lines

Table 1: Data abstraction rules

introducing color ratios. Examples are in Figure 33. In the case of a blue area, as the strength of blue decreases, the rule checks the red property so as to distinguish blue areas from brown areas.

Detect Very Dark and Very Bright Regions. Detect the very dark and very bright regions beforehand and do not use their color information, because their colors are severely distorted owing to the limited dynamic range of the camera. The two solutions: shadows and sun spots, are in a sense for these kinds of regions, though the dark or bright regions do not necessarily have to be labeled with shadows or sun spots. Two kinds of data abstractions: *very dark* and *very bright*, are requisite for outdoor natural color scene interpretation.

6.2.2. Heuristic Match

There are 3 kinds of heuristic match rules: on the level of ground, above ground, and both, which are listed in Table 2. Most of them are for inferring *on the level of ground* objects. The reason is that two *above ground* objects, tree foliage and sky, are more ambiguous than those on the level of ground. They could be grass and road, respectively. So, they usually remain unlabeled in the initial hypothesis generation, since they can be easily disambiguated by the position of the vanishing point of the road which will be available later. One rule, which infers an *above ground* object and an *on the level of ground* object at the same time, is intended to detect pairs of a tree trunk (vertical object) and its shadow. It turned out to be a powerful rule. Some examples are in Figure 34.

6.2.3. Refinement

Each solution, grass, soil, shadow, sun spot, tree foliage, or sky has one region-based refinement rule which checks the solution class and other data abstractions. Road, tree trunk, or vertical object has a region-based refinement rule and an edge-based refinement rule, since the shapes of these objects are much more constrained than the other objects. Figure 35 explains some rules.

```
Darker Area:
  IF      black&white<64
    and  (20<=black&whJte  or  20<=red)
    and  green<=blue+32
    and  darkest  among  neighbor  regions
  THEN  The  region  is  "darker area"
```

```
Greener Area:
  IF      48<green
    and  blue+10<green
    and  closest  to  pure  green(0,255,0)
                    among  neighbor  regions
  THEN  The  region  is  "greener area"
```

Figure 32: Data abstraction rules by looking neighbor regions

```
Bluest Area:
  IF    48<blue  and  g/b<=0.90
  THEN  The  region  is  "bluest area"
```

```
Blue Area:
  IF  36<bf  ue  and
     [      g/b<=0.97
        or  (g/tx=1.05  and  r/b<=1.40)
        or  (g/b<=1.10  and  r/tx=1.25)  ]
  THEN  The  region  is  "blue area"
```

g/b=green/blue, r/b=red/biue

Figure 33: Data abstraction rules by using color ratio

- *63.* Context Control

Figure 36 illustrates the context control of the road scene interpreter. It looks slightly different from that in Figure 4, though they are basically the same. After the extrapolation, the interpretation cycle goes back to the heuristic match of the initial hypothesis generator. The reason is that a part of the task of the extrapolator is very similar to that of the heuristic match and refinement Simply, the extrapolator shares some rules with the initial hypothesis generator.

6.4. Evaluation

Each labeled region has a plausibility value. In each interpretation cycle, labeled regions are evaluated and gel their plausibility values increased or decreased- If a plausibility value becomes negative, its label is removed and gives chances to alternative labels. Each region has 2 slots for memorizing 2 previous labels, so that the removed label cannot take place again unless 2 alternative labels have been tried.

Table 3 shows the composition of the evaluation rales. Evaluation rules are categorized into local and global evaluation rules. Local and global evaluation rales have positive and negative evaluation rales. A local evaluation role picks up a pair of labeled regions at a time and etiecks their consistency. A global evaluation rate picto up

Data Abstraction	Heuristic Match
higher-darker-lengthwise, parallel area	**Above ground**
large-lower, large-blue, bluest, bright-lower, large-darkest-crosswise, trapezoidal, moderately_large-lower -green-touching _vertical_frame	**On the level of ground**
higher-darker-lengthwise and darkest -non_lengthwise are forming a shape L or┘	**Above ground and On the level of ground**

Table 2: Heuristic match rules

Above Ground:

```
IF     no solution class is generated
  and  higher area
  and  (darker area or darkest area)
  and  lengthwise area
THEN   It is "above ground".
```

On the Level of Ground Region:

```
IF     no solution class is generated
  and  large area
  and  lower area
THEN   It is "on the level of ground".
```

Figure 34: Heuristic match rule examples

one labeled region at a time and checks its consistency with one of the explicit scene models. Four of the negative global evaluation rules are dominant. When the plausibility values of explicit scene models become reliable, these four rules enforce the model by rejecting labeled regions contradicting the models.

Some local evaluation rules are in Figure 37. If a pair of labeled regions is consistent, the rule gives 5 points to each labeled regions. If a pair of labeled regions is inconsistent, the rule takes 8 points from the less plausible labeled region and 2 points from the more plausible region; if they are equally plausible, it takes 5 points from the each labeled region. When a pair of labeled regions are inconsistent, one labeled region could be good and the other one could be bad. The negative evaluation rule can not figure out locally which one is wrong. But the good labeled region must survive. However, each labeled region has a plausibility value. If the plausibility value is high, this means the region has survived a couple of interpretation cycles and got its plausibility value increased by positive evaluation rules. Thus, the more plausible labeled region should have an advantage over the less plausible labeled region.

Some global evaluation rules are in Figure 38. After a couple of interpretation cycles, the plausibility of the models becomes moderately high, and then the global evaluation starts in addition to the local evaluation. If a labeled region is consistent with a related model, the rule gives 10 points to the labeled region. If a labeled region

Soil Found:

```
IF     on the level of ground
  and brown area
  and not darkest
  and 32<MBR_rowmax
  and road model plausibility<=usable
THEN  The region is "soil".
```

Road Trapezoid Found:

```
IF     on the level of ground
  and blue area
  and trapezoidal area
  and not textural area
  and road model plausibility<=usable
THEN  The region is "road".
```

Figure 35: Refinement rule examples

Category		Positive	Negative	Total
Local		7	11	18
Global (by Models)	Ordinary	4	7	11
	Dominant	-	4	4
Total		11	22	33

Table 3: Evaluation rules

is inconsistent with a related model, the rule takes 10 points from the labeled region. When the plausibility of the models becomes reliable, four dominant negative global evaluation rules can fire. If a labeled region is inconsistent with a related model, the dominant negative global evaluation rule takes 10000 points from the labeled region, so that the label is removed.

6.5. Modeling

Initially, one road model is already generated and ready to be adjusted as depicted in Figure 2 (a). The model is adjusted to the latest interpretation in each cycle by the functions written in C++. In each model adjustment, first, the vanishing point is moved horizontally until the discrepancy reaches a local minimum; then, the lower end of the left road edge is slid horizontally until the discrepancy reaches a local minimum; then, the lower end of the right road edge is slid horizontally; last, the vanishing point is moved vertically. This road model adjustment cycle is iterated one to three times depending on the plausibility level of the road model.

Model adjustment for each degree of freedom is not very expensive. Only the pixels along the boundary need to be tested. Figure 39 explains it. In the figure, the lower end of the left edge is being adjusted. If it moves left, the area shaded with dots minus the area shaded with grids improves the discrepancy value. If it moves right, the discrepancy value deteriorates. Thus, it goes left. Then, it calculates the gain area minus the loss area for the next step. It keeps going left as long as the calculated value is positive. When the calculated value becomes negative, it stops and its position is the local minimum of the discrepancy value. The same operation is applied to the other degrees of freedom one after another until the discrepancy value settles. The required computing power

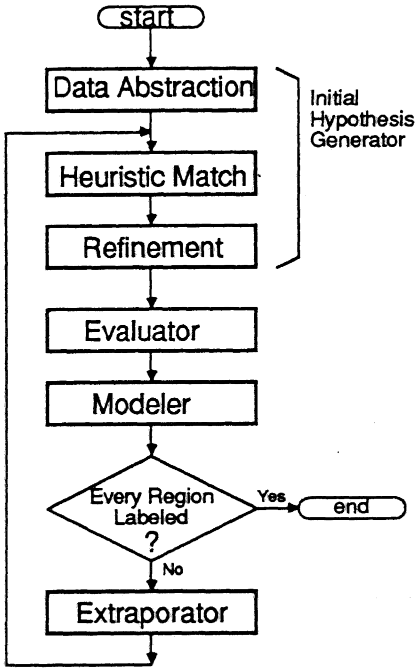


Figure 36: Context control

is proportional to the number of degrees of freedom, since only one degree of freedom is adjusted at a time.

Modeling is one of the most important components in the road scene interpreter. The explicit road scene models extract global information of the road scene, on which the evaluator and the extrapolator rely. If the global information of the models is wrong, the whole interpretation is confused and corrupted. Especially for the first one or two interpretation cycles, the models are unstable simply because there are only several labeled regions. So, for the first a few interpretation cycles, the models must be less sensitive until quite a few regions are labeled. We observed that the models were once oversensitive and adjusted very strangely. A couple of ideas are adopted in order to make the models less sensitive for the first a few interpretation cycles:

Do Not Adjust a Road Model Edge Unless Grass or Soil is Found on that Side. Grass or soil regions prevent the road model from expanding. If there is no grass or soil region found on the right or left, the lower end of the right or left road edge of the model can expand too much. So, if no grass or soil is found, the lower end of the road edge of the model on that side should not be adjusted.

Restrict the Vanishing Point and Width of the Road Model Loosely. It is very strange if the vanishing point moves far off the top edge of the image or very close to the bottom edge of the image. If the road width becomes twice as wide as expected, this is also unusual. These situations should be avoided for the first few interpretation cycles.

Though it belongs to domain knowledge organization, experimentally, a Y-intersection detection mechanism has been implemented to confirm the extendibility of the adjustable explicit scene models and the interpretation cycle. Figure 40 shows its basic idea. Since the road adjustment looks for a local minimum of the discrepancy value, a road model is adjusted to one of the roads by chance. The other road makes the discrepancy value big. So, if there is big discrepancy on either right or left, the modeler generates an additional road model in that direction. Figure 21 is an actual result. The modeler is also capable of eliminating erroneous additional road models. Elimination of an erroneous additional road model is easy, since the vanishing point of the road model becomes very close to the

```
Local Positive Evaluation:
   IF      a foliage and a tree trunk
               are adjacent
     and foliage's mass center is
               heigher than that of the trunk
   THEN  give each region 5 points.

Local Negative Evaluation:
   IF      a road and a tree trunk
               are adjacent
     and the trunk is the highest
               among adjacent trunks
     and top of the road is higher
               than top of the trunk
     and usable<=road model plausibility
   THEN  take 8 points
               from less plausible region
       and take 2 points
               from more plausible region
     when they are equally plausible
           take 5 points from each region
```

Figure 37: Local evaluation rule examples

main road model if it is inappropriate.

Adjustment of the tree trunk model is one degree of freedom after another: mass center row, mass center column, width, height, and lean. Each degree of freedom finds a local minimum of the discrepancy value and this model adjustment cycle takes place once in each interpretation cycle. Generation and elimination of the tree trunk models differ from those of the road models. Every time a fragment of a tree trunk is found, a tree trunk model is generated and adjusted independently. Thus, one tree trunk may have several tree trunk models at first. By checking the overlaps of the tree trunk models, some models are eliminated and one model remains for each tree trunk. This is shown in Figure 20 (h). If a tree trunk model becomes very small, it is also eliminated.

As the interpretation progresses, the plausibility level of the models becomes higher. Table 4 shows the five plausibility levels of the road model. It is determined by the area of the reliable regions whose plausibility value is greater than or equal to 10. At first, the plausibility level of the models is not usable since the models are not reliable. Then, it becomes carefully usable; the system uses the models but still checks the features of regions carefully. Then, it becomes usable; the system uses the models and checks just a few features of regions. Then, it becomes reliable; the system uses the models without checking the features of the models. Last, it becomes dominant; the system totally relies on the model and rejects all of the inconsistently labeled regions. It can use the object label *unknown* in this plausibility level and quits interpretation as soon as possible. Currently, only the plausibility level of the main road model is calculated and represents all the plausibilities of the models.

6.6. Extrapolation

Labeled regions, which have survived some interpretation cycles, propagate to their adjacent unlabeled regions by checking the location, shape, color, area, and so on. Compared to initial hypothesis generation rules, less strict conditions are required in extrapolation since each rule is based on one labeled region. Extrapolation rules are simple and make easy guesses. Most of the guesses will be good, but some of them might be wrong. They are judged by the evaluator. Table 5 lists the categories of the extrapolation rules.

1/3 of the extrapolation rules infer information about solution classes. Two of them remove *on the level of*

Global Positive Evaluation:

```
IF     a tree trunk is inside a tree model
  and carefully-usable<=model plausibility
  and it is dark area, darker area,
           or darkest area
  and it is not on the level of ground
THEN  give the region 10 points.
```

Global Dominant Negative Evaluation:

```
IF     a tree trunk is not inside
           any tree trunk model
  and reliable<=model plausibility
THEN  take 10000 points
           from the region
```

Figure 38: Global evaluation rule examples

The direction of the movement can be determined by checking only the labeled pixels along the boundary.

In the illustration, if the bottom of the left edge moves left, the discrepancy value is improved.

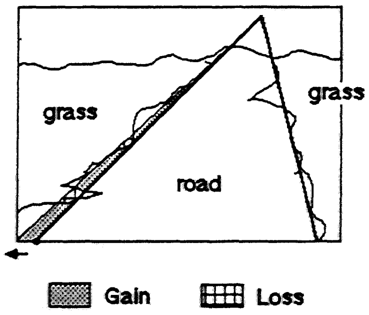
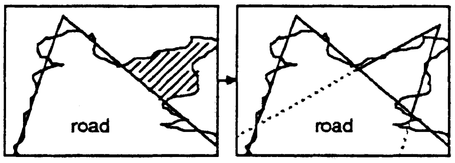


Figure 39: Model adjustment algorithm

ground solution class or *above ground* solution class. Removal of a solution class assigned is often caused by the transition of the vanishing point of the road model. The rest of the rules generate solution classes. Figure 41 shows an example.

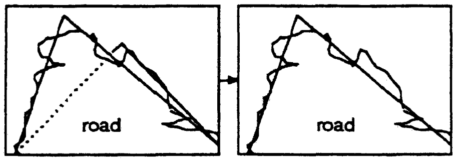
Two-thirds of the extrapolation rules infer information about solutions. Adjacency is a primary cue for extrapolation and used by most of the rules. For instance, a tree trunk expects tree foliage above it. Figure 41 has another example. A road fragment will be adjacent to another road fragment. Occasionally, a small spot such as a crack on a road or a dent in a tree trunk forms an independent small region surrounded by similar large regions. This kind of small regions are deduced if the surrounding regions take the same label. Two rules use the sun direction sun and try to propagate tree trunk regions to that direction, since the sunny side of the tree trunk is often bright and ambiguous. An actual example is shown in Figure 20 (e) (g), where the tree trunk expands to the right. Two-fifth of the rules use scene models to infer.

One weather rule determines whether it is sunny or not by simply checking the existence of sun spots or shadows. Then another rule detects the direction of the sun by finding an appropriate pair of a shadow and a vertical object touching at the bottom of the vertical object. Unless shadows are vertical in the image, it is fairly easy to detect the direction of sun and utilize it in interpretation.



If there is big discrepancy on either left or right, the system generates additional road model.

(a) Road Model Generation



If the vanishing point of the additional road model becomes very close to the main road model, the system eliminates the additional road model.

(b) Road Model Elimination

Figure 40: Intersection model

6.7. Road Map Generation

Because explicit road scene models extract the vanishing point of a road and the bottoms of trees, a road map can be generated under an assumption that the roots of trees are on the level of ground. If its view frame angle θ is known, then inverse projection becomes possible.

Figure 42 illustrates the inverse projection. The projection plane (image plane) is set on the lower base of the trapezoidal view frame. All we have to know is the location of the center of projection and the tilt angle of the projection plane. If the angle θ of the view frame is given, by extending the sides of the view frame, the location of the center of projection in the plane view is derived as depicted in Figure 42. The distance d between the projection plane and the center of projection in the plane view is calculated as

$$d = \frac{b_{lower}}{2} \cot \frac{\theta}{2},$$

where b_{lower} is the length of the lower (shorter) base of the trapezoidal view frame as in the plane view in Figure 42. Note that if we draw a perpendicular line segment from the center of projection to the projection plane, the foot of this segment lies at the center of the image. Now the remaining unknowns are the height of the center of projection and the tilt angle of the projection plane. Let us think of the plane determined by the center of projection and the vanishing point level in the image plane. This plane must be parallel to the ground plane. By tilting the projection plane until this plane becomes parallel to the ground plane, the height of the center of projection and the tilt angle are determined at the same time as drawn in the side view in Figure 42.

6.8. System Analysis

Table 6 shows the composition of rules. The initial hypothesis generator consists of 60 rules: data abstraction 27, heuristic match 11, and refinement 22. This is the largest rule module. The extrapolator has 52 rules and is the second largest rule module. The modeler has only 16 rules, because most of it is written in C++ functions. If C++ functions are taken into account, the modeler is the biggest module. In total, the interpreter comprises 187 rules to date. The source code of the interpreter has about 8300 lines, 2/3 in OPS83 and 1/3 in C++. If the source code

The road model has 5 levels of plausibility.

	Retio of reliable regions in a scene (10<region plausibility value)
1. Unreliable	-
2. Carefully-usable	1/3
3. Usable	1/2
| usually 1 interpretation cycle	
4. Reliable	1/2
| 2 interpretation cycles	
5. Dominant	-
(quit interpretation as soon as possible)	

Table 4: Plausibility levels of the model

of the feature extraction program is added, it exceeds 10000 lines. Since main body of the interpreter is written in rule-based programming language OPS83, it is still very maintainable.

There are two important modules: data abstraction and modeling. The data abstraction of initial hypothesis generation is the first knowledge base applied to the intermediate data. The very first stage always plays an influential role in a total system. It connects the intermediate level and the high level, and preserves its parallelism because rules in the data abstraction are independent Modeling is significant because adjusted scene models become influential and then dominant in the latter half of the interpretation. Once the models start converging, everything goes well. Every effort in knowledge acquisition is made such that the scene models converge. When looking at a new road image, the first task of the knowledge engineer is to mentally fit a triangular road model to the road. No mailer how much the road is shaded, if the road model can be adjusted properly, it is worth working on the image.

Negative evaluation is not troublesome as oppose to the intuition. Actually, evaluation is the smallest one among four modules as it is in Table 6 (modeler is the largest since its functions are in C++). 3 reasons are enumerated:

1. Global negative evaluation is reliable as long as models are properly adjusted.
2. Since local negative evaluation cannot avoid mistakes, the simple strategy in Section 6.4. is not much different from the optimal strategy.
3. The basic interpretation strategy, cut-and-try, of the system with interpretation cycles alleviates the evaluation task.

Four modules, initial hypothesis generation, evaluation, modeling, and extrapolation, are in a certain balance. None of the modules is perfect The four modules work together to compensate for each other and perform reasonable interpretation. So, when modifying the system, the balance needs to be taken into account The knowledge engineer has to recognize the weakest module and enhance it Another aspect of the system is that one rule could be fired more than 50 times because of the cyclic mechanism. For example, there can be more than 20 pairs of road and grass regions in a scene. A road-grass positive evaluation rule is fired on each pair of these repots in every Interpretation cycle in order to keep them alive. This characteristic is very different from other production systems.

Since the system adopted a cyclic mechanism, various phenomena such as oscillation, overshoot, divergence, etc. [illegible]*al in* control theory can potentially happen. Those phenomena were observed while developing INSIGHT-EL

Category		Number of rules
Solution Class	higher/lower than vanishing level	6
	higher than a above-ground region lower than a on-the-level-of-ground region	2
	similar to	3
	surrounded by	1
	with sun direction	2
Solution	object A expects object B	9
	similar to	6
	surrounded by	5
	with sun direction	2
	by scene model	13
Total		49

Table 5: Extrapolation rules

7. Future Work

To enhance the initial hypothesis generator, a couple of effective improvements are possible and discussed.

Intermediate representation data need to be appropriately abstracted in order to provide reliable initial hypotheses for the following interpretation stages. Conversely, it is also true that perfect data abstraction cannot be generated locally. However, at least a few significant regions, in terms of perceptual organization [27][28][29], must be properly abstracted in a bottom-up manner as initial clues, otherwise there is no way to interpret a natural color scene without predictions. Even for a human, if a scene is very dark and does not give good initial clues, it is difficult to interpret. Therefore, pursuing reliable data abstraction in a bottom-up manner is at least a necessary approach to more robust natural color scene interpretation.

7.1. Inexhaustive Region Segmentation for High-level Interpreter

In inexhaustive region segmentation, some ambiguous pixels are not necessarily segmented. It alleviates the segmentation task and is also appropriate to the high-level interpreter.

From the view point of the high-level scene interpreter, little consideration has been given to low-level region segmentation. The objective of low-level region segmentation has been roughly how well an image can be segmented in comparison with an ideal one segmented by a human. This objective automatically includes semantics, which is not very successful [30]. But an ideal region segmentation for a high-level interpreter can be different. The objective of low-level region segmentation should be to give pure low-level cues to a high-level interpreter. In the case of the system INSIGHT-III, the objective must be to give pure low-level cues to the data abstraction module of the initial hypothesis generator.

Color transition occurs near the boundary of two regions. But those transition pixels are always merged into one of the regions. We observed that the average colors of the segmented regions differ from those of the corresponding portions of the original image. This is caused by those heterogeneous pixels along the boundaries and in the portions of a region undersegmented. If heterogeneous pixels are mixed, the averaged color tends to be neutral. This color contamination weakens the color cues and makes data abstraction erroneous. Since region boundaries are not very

```
Lower than vanishing level is ground:
  IF     a region has no solution class
    and vanishing point level + 80
                              < MBR_rowmax
    and usable <= road model plausibility

  THEN  the region could be on the level
           of ground

Road expects grass:
  IF     an unlabeled region is adjacent to
            a road region
    and it is (green area or bright green area)
    and it is not lengthwise area
    and road model plausibility <= usable

  THEN  the region could be grass.
```

Figure 41: Extrapolation rule examples

important to a high-level interpreter with adjustable explicit scene models, exhaustive region segmentation is not requisite. Initial hypotheses in INSIGHT-III are often dispersed fragments of the labeled regions as in Figure 19 (a), but the scene models are adjusted appropriately. Therefore, the objective of the region segmentation should be to extract very homogeneous portions of the image for reliable data abstraction. A shaded road scene image contains many ambiguous pixels. Those ambiguous pixels along region boundaries need not be merged to the regions.

Inexhaustive region segmentation gives rise to some discussion. *First, what is the image interpretation result, labeled regions or adjusted models?* It depends on the application. For autonomous vehicle navigation, adjusted explicit models or a road map is better. If the global information extracted from inexhaustively segmented regions is sufficient, exhaustive region segmentation is no longer necessary. Conversely, some application may need labeled regions. In this case, the image can be exhaustively segmented afterwards with the global information extracted. *Second, can explicit scene models be adjusted appropriately?* Basically, there is a local minimum of the discrepancy value near the best match. Although the gaps between regions may make some weak local minima, they can be overcome by modifying the adjustment algorithm slightly. This will be much easier than unlabeling misinterpreted regions caused by color contamination. *Third, how can relations among regions be expressed?* Adjacency can be defined among unconnected regions. For example, if two regions are within a certain distance at the closest points and no other region is between them, then those regions are adjacent. *Last, won't it lose edge information?* Edges are separately extracted with color edge detector as depicted in Figure 6.

7.2. Adaptive Data Abstraction

Since the interpreter is capable of recognizing interpretation failure by checking the consistency of adjusted scene models or detecting infinite iteration of interpretation cycles, the parameters for data abstraction can be changed several times until the interpreter generates reasonably adjusted scene models. In this case, INSIGHT-III is used as a part of the bigger system with cycles. This meta-interpretation cycle works as a mechanism for recovering from fatal interpretation failure. If the number of parameters are limited, it becomes feasible. While developing INSIGHT-III, many efforts were made on differentiating road and soil regions and it was of importance for road extraction. Often these two regions are similar. Both are *on the level of ground* and have little texture. The only cue is the slight difference of the colors: soil is a little redder and darker than road, though it is not always so because of camera condition, color contamination, noise, etc. This difficulty will be alleviated by this adaptive data abstraction.

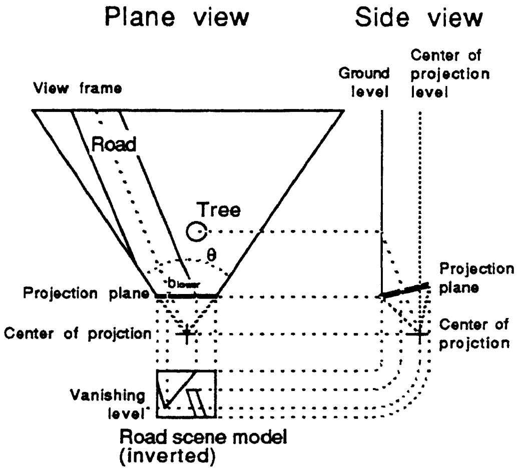

Plane view
Side view
View frame
Road
Tree
θ
b_{lower}
Projection plane
Center of projction
Vanishing level
Road scene model (inverted)
Ground level
Center of projection level
Projection plane
Center of projction

Figure 42: Inverse projection

8. Conclusion

The outdoor natural color road scene interpreter with adjustable explicit scene models and an interpretation cycle has been developed, and more than 20 natural road scenes, some destructively shaded, are reliably interpreted. By moderately simplifying the low-level region segmentation and domain knowledge organization, and intentionally selecting ill-structured scenes instead, some of the issues of robust natural color scene interpretation have been clarified:

1. Global information of a natural scene, which makes the interpretation substantially reliable, can be extracted by adjustable explicit scene models and using an interpretation cycle.
2. Negative evaluation is not troublesome. It performs well in concert with global information extraction.
3. Classification problem solving model is appropriate as an initial hypothesis generator in natural scene interpretation.
4. Color contamination caused by the heterogeneous pixels along boundaries and undersegmented portion of a region weakens initial hypothesis generation.

Acknowledgement

The writers wish to thank Thomas Chen for region feature extraction, Chuck Thorpe and Jill Crisman for fully-automatic region segmentation, Steven Shafer and Karl Kluge for PHOENIX, Carol Novak for color edge detector, Hirobumi Miwa for line extractor, and the members of the Strategic Computing Vision Laboratory at Carnegie Mellon University for their valuable comments and discussions.

		Number of rules	
Context control		22	
Data Abstraction	Initial Hypothesis Generation	27	60
Heuristic match		11	
Refinement		22	
Evaluation		33	
Modeling		16	
Extrapolation		52	
Map generation		4	
Total		187	

Table 6: The composition of rules

A The Color Median Filter

Abstract

In this appendix, the mathematical concept *median* is extended to multiple dimensions. This extended concept *vector median*, is then applied to color image noise reduction. The evaluation of this color median filter shows that it preserves the edges of a color image more than the method which smoothes each component of the image by median filter separately.

A1. Median Filter

The median filter [31] is a powerful smoothing technique that does not blur or enhance edges. It is probably the best smoothing filter, when the characteristics of the noise are unknown. This advantage comes from the fact that the median filter never calculates an average; in other words, it never introduces a new pixel. It merely replaces the value at a point by the median of the values in a neighborhood of the point.

How can a median filter be applied to a multi-band image such as a color image, whose pixels are vectors? This had been in question since median was defined only on scalar quantities. To inherit the advantage of the median filter, a new pixel should not be introduced, because the new pixel could blur the edge. If the median can be defined on vector quantities, the answer is straightforward: the median filter merely replaces the vector at a point by the *median vector* of the vectors in a neighborhood of the point. The definition is discussed in section A2.

The median filter had been used unsatisfactorily on the red, green, and blue components of the color image separately. This method introduces new pixels whose components are extracted from the different neighbor pixels. The new pixels not only blur the edges but also could change the colors disparately.

An extreme case is depicted in Figure 43. When scanning a color image with a small cross window, five pixels could be at magenta (red:255, green:0, blue:255), magenta (255, 0, 255), cyan (0, 255, 255), green (0, 255, 0), and yellow (255, 255, 0), which are shown in color space. If median filter is utilized on each component separately, it introduces white (255, 255, 255), since two pixels are at 0 and the remaining three pixels are at 255 when projected on each red, green, or blue axis. The white is far from the original pixel colors. This is worse than averaging filter that will introduce gray (153, 153, 153). If a new pixel should not be introduced, which one of the five pixels is the most appropriate as a median? Intuitively, it will be magenta, because there are two pixels there while the others are alone. This magenta pixel preserves the edge more than the white (255, 255, 255) or the gray (153, 153, 153).

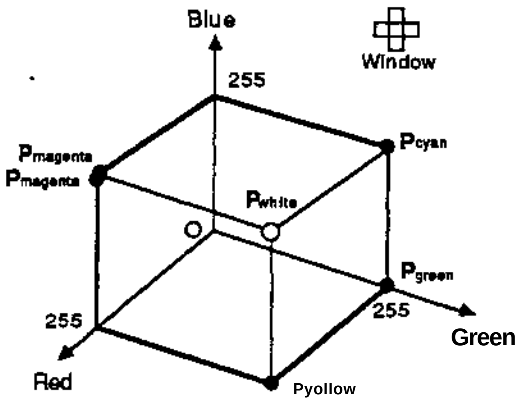


Figure 43: A new pixel introduction

If median filter is applied to each component separately, the extracted vector is not unique; it depends on the axes chosen. Figure 44 illustrates this dependency in two dimensions. The axes *X* and *Y* select the vector m while the axes X^1 and *T* draw the vector m^7. The extracted vector can be at various locations when the axes rotate. This characteristic is undesirable.

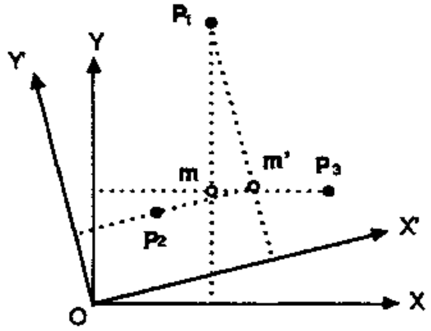


Figure 44: Dependency on the axes chosen

A2. Mathematical Extension of Median to Multiple Dimensions

The problem:
Given a set of vectors Pi, P_{2j}..., P«. Select a vector out of them which is the median vecton

Definition of vector median: Let *dy* denotes the distance between P; and $P_{y\text{f}}$ and S; = £JLi *d$*. The median vector is the F_m whose summation S_m is the smallest among *Si* $(1 \leq i \leq n)$.

A2.1. Vector Median in One Dimension

The foregoing definition of vector median is exactly the same as the scalar median in one dimension. This is depicted in Figure 45. Three points Pi, Pi, and P3 are at the same position on the left P_4 is at the distance *a* from

P_1, P_2, or P_3. P_5 is far away on the right at the distance b from P_4. The summations of the distances to all other points for P_1, P_2, P_3, P_4, and P_5 are $2a+b$, $2a+b$, $2a+b$, $3a+b$, and $3a+4b$, respectively. Hence, the median vector is P_1, P_2, or P_3. This is the same as the scalar median. It should be noticed that vector median does not necessarily select the pixel that is the closest to mean. In the Figure 45, P_1, P_2, or P_3 is not the closest point to the mean. The mean will be on the right side of P_4, since $a \ll b$.

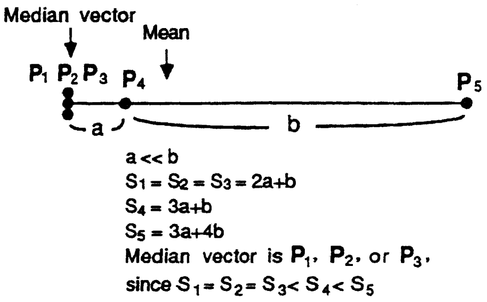

Median vector
Mean
P1 P2 P3 P4
P5
a
b
a<<b
S1 = S2 = S3 = 2a+b
S4 = 3a+b
S5 = 3a+4b
Median vector is P1, P2, or P3,
since S1 = S2 = S3 < S4 < S5

Figure 45: Vector median in one dimensional space

A2.2. Vector Median in two Dimensions

A two dimensional case is illustrated in Figure 46, where five points are at the vertices of a regular pentagon except P_1 which is slightly inward. The length of the side is a. The length of the diagonal line is b. The distances from P_1 to the closer points and to the farther points are a' and b', respectively. Intuitively, P_1 should be the median vector, because it is inward. The summations of the distances to all other points for P_1, P_2, P_3, P_4, and P_5 are $2a'+2b'$, $a+a'+2b$, $2a+b+b'$, $2a+b+b'$, and $a+a'+2b$. Since $a' < a$ and $b' < b$, S_1 is the smallest. Thus, the medina vector is P_1 which follows the intuition. It is interesting that a unique median vector is usually found among even number of vectors in multiple dimensions.

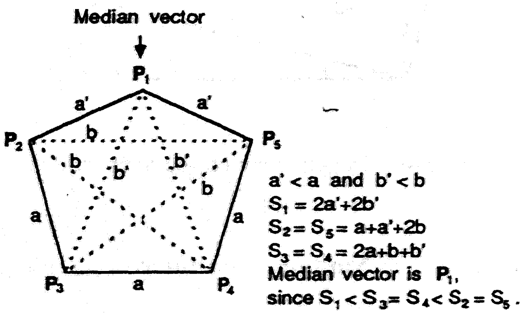

Median vector
P1
a'
a'
P2
P5
b
b
b'
b'
b
a
a
P3
a
P4
a' < a and b' < b
S1 = 2a'+2b'
S2 = S5 = a+a'+2b
S3 = S4 = 2a+b+b'
Median vector is P1,
since S1 < S3 = S4 < S2 = S5 .

Figure 46: Vector median in two dimensional space

A2.3. Vector Median in three Dimensions

The example in Figure 43 is examined in Figure 47. The summations are $S_{magenta} = 255(2\sqrt{2}+\sqrt{3}) \approx 1163$, $S_{cyan} = S_{yellow} = 255(1+3\sqrt{2}) \approx 1337$, and $S_{green} = 255(2+2\sqrt{3}) \approx 1393$. Therefore, $P_{magenta}$ is the median vector. The median vector is attracted by points in close proximity.

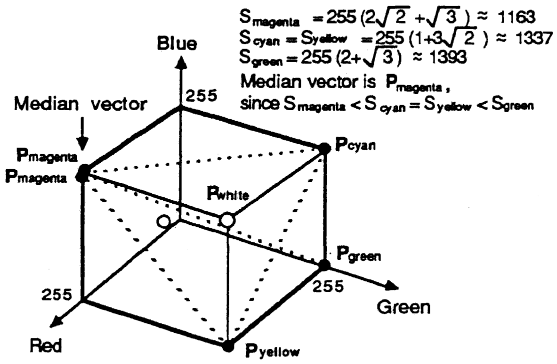

$S_{magenta} = 255(2\sqrt{2} + \sqrt{3}) \approx 1163$
$S_{cyan} = S_{yellow} = 255(1+3\sqrt{2}) \approx 1337$
$S_{green} = 255(2+\sqrt{3}) \approx 1393$
Median vector is $P_{magenta}$,
since $S_{magenta} < S_{cyan} = S_{yellow} < S_{green}$
Blue
255
Median vector
$P_{magenta}$
$P_{magenta}$
P_{cyan}
P_{white}
P_{green}
255
255
Green
Red
P_{yellow}

Figure 47: Vector median in three dimensional space

A3. Color Median Filter

Applying the vector median to color images is straightforward. The color median filter merely replaces the color pixel at a point by the median vector of the color pixels in a neighborhood of the point. In the color case, the number of pixels in the filtering window is not necessarily odd, since, in two or more dimensional space, a median vector is uniquely determined among the even number of vectors that is greater than or equal to four.

By definition, the color median filter does not introduce new color pixels. This is preferable and enhances the edge preserving advantage of the median filter. It is also independent of the color axes chosen, since it depends only on the distances among color pixels.

A4. Evaluation of Color Median Filter

Both the color median filter and the method which smoothes each component of the image separately are applied on a car image in Figure 48. The window size is 3×3. Edge profiles of the red component original, the red component after median filtering, in the row 240 are in Figure 49 and Figure 50, respectively.

At the column 220 in Figure 50, a sharp intensity valley is preserved by the color median filter, while it is corrupted by the other method which smoothes each component separately. Most of the peaks are apparently acute in the results of the color median filter. But at the column 410 in Figure 50, a fluctuation in the middle of the steep slope is smoothed more by the color median filter than by the other method, because of the influences by the green and blue components. This shows that color median filter does not merely pass high frequency but has a strong preference when picking median vector pixels.

A5. Conclusion

The color median filter naturally preserves the edges of images more than the method which smoothes each component of the image separately by scalar median filter.

Reference

[1] B.A. Draper, A.R. Hanson, and E.M. Riseman. *A Software Environment for High Level Vision.* Technical Report, University of Massachusetts, 1986.

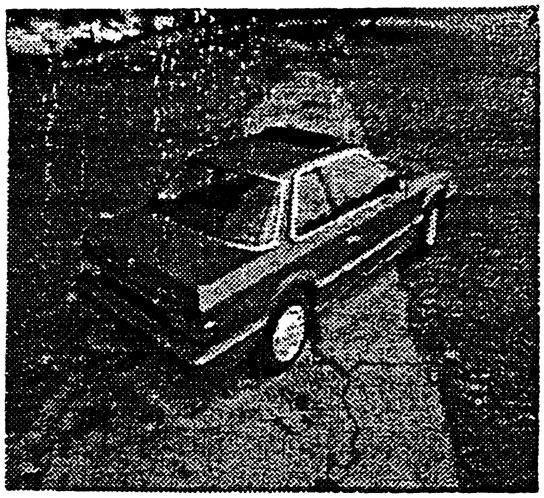


Figure 48: Car color image original

[2] B.A. Draper, R.T. Collins, J. Brolio, J. Griffith, A.R. Hanson, and E.M. Riseman. Tools and experiments in the knowledge-directed interpretation of road scenes. In *Proceedings of the DARPA Image Understanding Workshop*, pages 178–193, February 1987.

[3] A.R. Hanson and E.M. Riseman, editors. *VISIONS: A Computer System for Interpreting Scenes*, pages 303–333. Academic Press, New York, 1978.

[4] Y. Ohta. *Knowledge-based Interpretation of Outdoor Natural Color Scenes.* Pitman Publishing, 1985.

[5] D.M. McKeown, Jr., W.A. Harvey, and J. McDermott. Rule-based interpretation of aerial imagery. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 570–585, September 1985.

[6] D.M. McKeown, Jr., C.A. VcVay, and B.D. Lucas. Stereo verification in aerial image analysis. In *Optical Engineering*, pages 333–346, March 1986.

[7] D.M. McKeown, Jr. and W.A. Harvey. Automating knowledge acquisition for aerial image interpretation. In *Proceedings of the DARPA Image Understanding Workshop*, pages 205–226, February 1987.

[8] J. McDermott. Making expert systems explicit. In *IFIP Congress*, pages 1–10, Austin, Texas, 1986.

[9] R.A. Brooks. Symbolic reasonig among 3-D models and 2-D images. In *Artificial Intelligence 17*, pages 285–348, 1981.

[10] D.Y. Tseng and J.F. Bogdanowicz. Image understanding technology and its transition to military applications. In *Proceedings of the DARPA Image Understanding Workshop*, pages 310–312, February 1987.

[11] J. Pearl. *Distributed Revision of Composite Beliefs.* Technical Report CSD 860045 R-64-III, Cognitive Systems Laboratory, UCLA Computer Science Department, April 1987.

[12] T.O. Binford. Survey of model-based image analysis systems. In *The International Journal of Robotics Research*, pages 18–64, 1982.

[13] M. Nagao and T. Matsuyama. *A Structural Analysis of Complex Aerial Photographs.* Plenum, New York, 1980.

[14] Y. Shirai. *Recognition of man-made objects using edge cues.* Academic Press, New York, 1978.

[15] C.L. Forgy. *The OPS83 Report.* Technical Report CMU-CS-84-133, Department of Computer Science, Carnegie-Mellon University, May 1984.

[16] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5.* Addison-Wesley, Reading, Massachusetts, 1985.

[17] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, 1986.

[18] S.A. Shafer and T. Kanade. Recursive region segmentation by analysis of histograms. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1166–1171, Paris, May 1982.

[19] K.I. Laws. *The PHOENIX Image Segmentation System: Description and Evaluation*. Technical Report 289, SRI International, December 1982.

[20] C.L. Novak and S.A. Shafer. Color edge detection. In *Proceedings of the DARPA Image Understanding Workshop*, pages 35–37, February 1987. Section 2 of the PI-report: "Image Understanding Research at CMU" by Takeo Kanade.

[21] J. Canny. A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 679–698, November 1986.

[22] C. Thorpe and J. Chrisman. *Region segmentation based on color proximity*. Technical Report, Robitics Institute, Carnegie-Mellon University, 1988. [in preparation].

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

[24] C. Thorpe, S. Shafer, and T. Kanade. Vision and navigation for the Carnegie Mellon Navlab. In *Proceedings of the DARPA Image Understanding Workshop*, pages 143–152, February 1987.

[25] H. Miwa and T. Kanade. *Straight Line extraction*. Technical Report, Robotics Institute, Carnegie-Mellon University, February 1988. [in preparation].

[26] W.J. Clancey. Classification problem solving. In *Proceedings of the National Conference on Artificial Intelligence*, pages 49–55, Austin, Texas, 1984.

[27] D.G. Lowe and T.O. Binford. The perceptual organization of visual images: segmentation as a basis for recognition. In *Proceedings of the DARPA Image Understanding Workshop*, pages 203–209, 1983.

[28] A.P. Witkin and J.M. Tenenbaum. What is perceptual organization for? In *Proceedings of the IJCAI-83*, pages 1023–1026, August 1983.

[29] D.G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, 1985.

[30] T. Kanade. Region segmentation: signal vs semantics. In *Computer Graphics and Image Processing 13*, pages 279–297, 1980.

[31] A. Rosenfeld and A.C. Kak. *Digital Picture Processing*. Academic Press, Orlando, Florida, 1982.

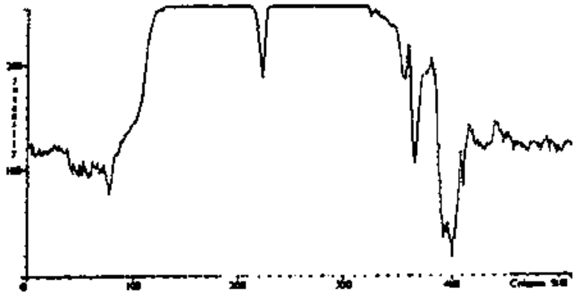
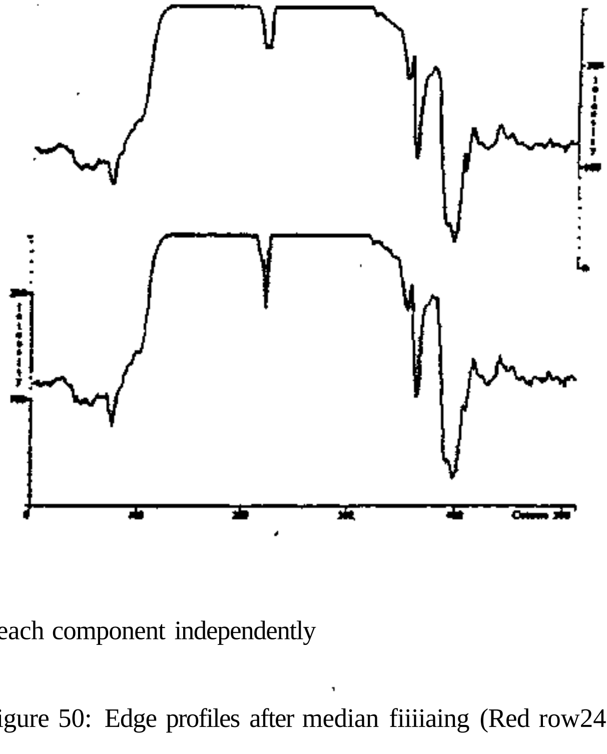


Figure 49: Original edge profile (Red row240)



Top: Median lite" used on each component independently
Bottom: Color median filter

Figure 50: Edge profiles after median fiiiiaing (Red row240)

Car Recognition for the CMU Navlab

Karl Kluge and Hideki Kuga

1. Introduction

LASSIE (*L*andmark *A*cquisition *S*ub*S*ystem *I*mproving *E*xploration) is a program which recognizes a car from several points of view. Trapezoids are formed from groups of lines extracted from a color image; instances of the car model are found by searching for sets of trapezoids which fit the constraints between parts of the car such as the windows, roof, and trunk. No information is used about the dimensions of the various parts, setting LASSIE apart from many other programs which use features extracted from images to recognize objects.

The remainder of the introduction sketches out the varied functions of object recognition in the navigation system of an autonomous vehicle and discusses the choice of domain for this work. The next section of the report discusses related work, followed by a section giving a detailed description of the LASSIE program. Results are shown for a number of images. The report closes with a discussion of directions for further work on LASSIE.

1.1. The function of object recognition in autonomous navigation

An autonomous vehicle navigating through the world may need to recognize a variety of natural and man-made objects for a number of reasons:

- to confirm or refine its internal estimate of its position in the world (*Am I really where I think I am?*)
- to make decisions about mission execution based on the presence or absence of objects in specified locations (*I should halt and look for conflicting traffic if there is an octagonal red sign at the intersection I'm approaching.*)
- to gather information for report at the end of a mission (*I should count the number of cars I see as I go along this stretch of road.*)
- to update the navigation system's internal model of the world (*Remember there was a stop sign at that intersection so I'll know that I should plan to stop and look for traffic the next time I approach it.*)
- to make its way along the path specified by its mission. Humans tend to give directions to an objective in high level qualitative terms *(go down the road until you come to the third traffic light, turn right, keep going until you see the a big church on the left, then turn left at the next street...)* rather than giving them in terms of precise distances *(go down the road 0.75 miles, turn right, go another 2.3 miles, turn left...)*.

A navigation system may select an object to serve as a landmark in response to internal goals. As an example, the navigation system might realize that its wheels were slipping and choose a stationary object in the environment in order to keep track of its position relative to the object. Alternatively, the mission may specify objects for which the system is to watch. Examples of this would be "turn right at the third stop sign," or "when you get to the big yellow marker you will be at world position (x, y, z): update your internal position estimate accordingly."

Objects to be recognized may be artificial (such as road intersections, buildings, and vehicles), or they may be natural (such as distinctive rock formations or trees or rivers). The objects may have semantic meaning (this is a car, this is a stop sign, this is a tree), or they may simply be distinctive structures detected by the navigation system's perceptual modules and tracked through time without identifying them as any particular kind of thing.

1.2. Choice of domain

Object recognition may be called upon by a broad range of capabilities built into a navigation system, as discussed in the previous section. The Sidewalk Navigation System developed for the Terregator at CMU [5] used segmentations of color and range images to recognize stairs and intersections in order to navigate around a network of campus sidewalks. In order to test other types of capabilities requiring object recognition, such as deciding which way to turn at an intersection based on the presence or absence of a specified object, it was necessary to have a program capable of finding other kinds of objects. The planning part of such capabilities can be tested and demonstrated using simple objects (such as orange traffic cones). A module capable of recognizing a more challenging type of object is more desirable, both as a much more challenging recognition problem to solve, and as more likely to bring out unexpected interactions between parts of the navigation system as the recognition module competes for system resources.

Cars seemed an attractive domain for a number of reasons. The main nontechnical reasons were that cars are fairly common objects which are easily moved and weather proof, making possible the easy collection of a variety of images to use as data. The main technical reasons were

- cars differ a great deal in size and appearance (think of the differences between the appearance of a station wagon versus a four-door sedan, versus a compact car), but the variation of appearance is much less than for a domain such as tree recognition;
- there is a great deal of structure in the edges projected into an image from a car, with many of the edges forming quadrilaterals corresponding to meaningful parts of the car;
- the significant linear features that form a car can be detected reasonably well by current line extraction techniques, but not so well that the system can assume perfect or close to perfect edge data.

1.3. The goals and state of the research

The goal of this research is to produce an object recognition program that

- is capable of extracting the significant geometric structures (in this case, trapezoids and their subclasses such as parallelograms and rectangles) formed by the edge segments extracted from an image;
- uses the relationships between those structures and constraints from an object model to find instances of the object;
- does not use more than the crudest of information about the actual dimensions of the objects (for instance discriminating station wagons from compacts based on the length of the body relative to the length of the hood, or stating that the trunk of a car projects into the image as a parallelogram, which imposes some crude constraints on the dimensions of sides of the trunk).

This approach is in contrast to many programs that perform object recognition based on a precise or parameterized model of the shape of the object(s) to be recognized. With an exact shape model, or even a parameterized model, a previously unseen car may not be recognizable. The hope is that using only the crudest kind of metrical information about the objects will allow a more generic recognition capability.

A variety of images of a particular car were used in testing the program as it evolved to its current state. The program demonstrates a fair amount of robustness in processing those images, as will be seen from the results in section 4. Recognition of different makes of cars has not been attempted, and will undoubtedly require a fair amount of modification of the model of the parts of the car, but should be possible within the framework that has been developed.

2. Related Work

In recent years much of the research in object recognition has been concerned with the recognition of objects based on data from 3-D sensors such as laser range finders. For an extensive survey of such work see Besl and Jain [1].

Brooks [2] describes ACRONYM, a system which recognized objects (represented as collections of parameterized generalized cones) using edges extracted from an intensity image. Edges were collected into "ribbons" which represented contours of generalized cones. A constraint management system (CMS) was used to verify the consistency of hypothesized matches between "ribbons" extracted from the image and object parts. A match was consistent if there was some viewpoint from which the match could have been imaged, with constraints on possible camera position accumulated based on the parameters of the generalized cones matched and the parameters of the corresponding image ribbons.

Goad [4] created a system which precompiled an exact description of the lines forming an object into a recognition program for that object. The programs would search the lines extracted from an image for matches. The object did not have to be a polyhedron, it simply had to have enough linear features for matching to take place. Matching was done via a simple backtracking search which checked sets of lines for matches based on their being consistent with some viewpoint. Rather than use a CMS as ACRONYM did, the program made assumptions which allowed it to map possible viewpoints onto a tessellation of the unit sphere. The set of cells corresponding to possible viewing directions would be restricted to reflect the new constraints on camera position as lines were added to a match.

SCERPO (Spatial Correspondence, Evidential Reasoning, and Perceptual Organization), described in Lowe [7], used estimates of the likelihood of accidental alignments to look for pairs of parallel lines and pairs of lines whose endpoints were close enough to suggest their being connected. It then looked for pairs of parallel lines connected to other pairs of lines to form trapezoids. These trapezoids were matched to parts of the object being looked for to give an initial pose estimate. The program then searched for additional lines to confirm/refute the initial match and refine the pose estimate. Like Goad's system, it required an exact model of the geometry of the lines forming the object.

Thompson and Mundy [9] describe a scheme for recognizing objects using a primitive called the "vertex pair." Approximating the projection from model to image by an affine transform, a match between a vertex pair in the model and its projection in the images completely specifies the pose of the object. Matches are found by a fairly brute-force voting scheme in the six dimensional parameter space. Once again, an exact description of the object's geometry is required.

Ohta's system [8] used color, texture, shape, and adjacency properties of regions extracted from a color image to label the regions as parts of various kinds of objects such as road, car, tree, or building. The program had no concept of the structure of the objects in a scene — a set of adjacent regions labeled "building" might be parts of one building or of several buildings.

The 3-D MOSAIC system described by Herman and Kanade [6] worked in the domain of aerial images of buildings. It extracted line segments from the scene and grouped them together into sets of segments forming faces of buildings in the scene. Monocular and binocular cues were combined with assumptions about the domain (for instance, that walls are usually vertical and roofs are usually parallel to the ground plane) to construct a wireframe representation of the scene. The system had no recognition component, but the ideas used in grouping segments into potential building faces had an influence on the grouping processes in LASSIE.

The system described by Fua and Hanson [3] works in the domain of aerial image analysis. The image is segmented into regions, and edges are extracted using the Sobel edge operator. Expectations about the geometry of "cultural regions," mainly building outlines, are used to drive the resegmentation of the image in order to extract cultural regions. These expectations are very weak and generic. The system is capable of deducing certain things about the relations between the regions it finds, for instance, that a lit region joined along an edge to a darker, non-shadow region is likely to be formed by a peaked roof.

3. The LASSIE object recognition program

3.1. Overview

The recognition process consists of the following stages:

- Extract edge points from a color image, then link them into line segments.
- Merge segments that are almost collinear and are separated by a small gap.
- Eliminate short line segments.
- Extract significant line pairs. The pairs are chosen based on their either being nearly parallel or possibly forming the two non-parallel sides of a trapezoid.
- Examine the pairs formed in the previous step to look for segments that would close the pairs off to form trapezoids.
- Examine the trapezoids found in the previous step to merge sets of overlapping trapezoids whose shapes are similar.
- Search for groups of trapezoids potentially formed by the windows, roof, and trunk of a car.
- Search, given an initial match, for other features (ellipses corresponding to wheels, lines from the back and side of the car) to confirm that match.

Apart from the final verification stage, the system is currently purely a bottom-up program, with no top-down interactions between the earlier stages.

3.2. Description of the segmentation and grouping stages

Segmentation

- A blue minus red (B - R) image is computed from the input color image.
- The Canny edge detector is run over the B - R image. The sigma chosen is 2.5.
- The 5% of image points with highest gradient magnitude are selected from the Canny output as edge points.
- A line tracker, written at CMU by Hiro Miwa, is run over the thresholded edge image. The line tracker places breakpoints in chains of edge pixels by looking for places where the edge curvature is changing rapidly at multiple scales. Gaps of one pixel in edges are jumped by the tracking program. Lines shorter than 10 pixels are discarded. The line tracker also finds ellipses in the edge data which can be used to find the location of wheels forming part of a car.

Line merging

Segmentation may break up lines in the scene because of gaps in the thresholded edge image. Also, very close parallel edge segments may arise from the same linear feature in the scene. A simple merging procedure is applied to compensate for these errors in edge extraction, illustrated in figure 3-1. Given two edge segments, S1 and S2:

- A line is fit to S1 and S2 using least squares. The endpoints of S1 and S2 are projected onto that line in order to find the endpoints of the merged segment, S3.

- S1 and S2 are merged to form S3 if the following conditions are met:
  1. the distance between the endpoints of S1 and S2 and the merged segment S3 is less than some number of pixels, and
  2. S1 and S2, when projected onto S3, either overlap or are separated by a gap smaller than a threshold set by the user.

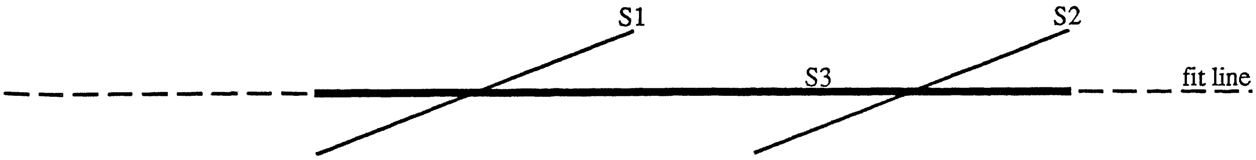


Figure 3-1: Line merging

Thresholding

The lines are thresholded on length to eliminate spurious lines created by such things as variations in the surface of the road on which the car is resting, and specular reflections of tree foliage off the surface of the car. The default threshold is 30 pixels.

Examination of line pairs

The set of lines is examined for pairs of lines which are parallel within some tolerance, or which could be the non-parallel sides of a trapezoid, i.e., the virtual lines connecting corresponding endpoints of the segments are close to parallel.

Extraction of trapezoids

Each pair of lines found in the last stage could possibly be two of the four sides of a trapezoid. To look for evidence supporting one of the sides connecting the pair to form a trapezoid

- the list of edge segments is searched for edges which meet the following criteria: they are close to parallel to the hypothesized side; their endpoints are closer than a threshold (based on the separation between the line pair) to the hypothesized side; and whose probability of being a false match to the hypothesized side is less than some threshold (using the formula given by Lowe [7]);
- the supporting segments found are projected onto the hypothesized side and the fraction of the hypothesized side covered is computed
- a line is fit to the supporting segments if a sufficient fraction of the hypothesized side is covered (see figure 3-2). If only one of the two missing sides is found by this process the other is hypothesized as present.

A number of heuristics are used to eliminate some of the trapezoids found by this process. A trapezoid is discarded if any of the following conditions are met:

- one side of the trapezoid does not have sufficient support and is hypothesized, and an edge segment crosses the hypothesized side and extends more than a threshold distance outside the trapezoid;
- the shorter side of a pair of opposite sides is less than some fraction of the length of the longer side of the pair;
- the angle between a pair of adjacent sides is greater than a threshold angle;
- a side supported by one of the original pair of lines is longer than some multiple of the length of the original segment.

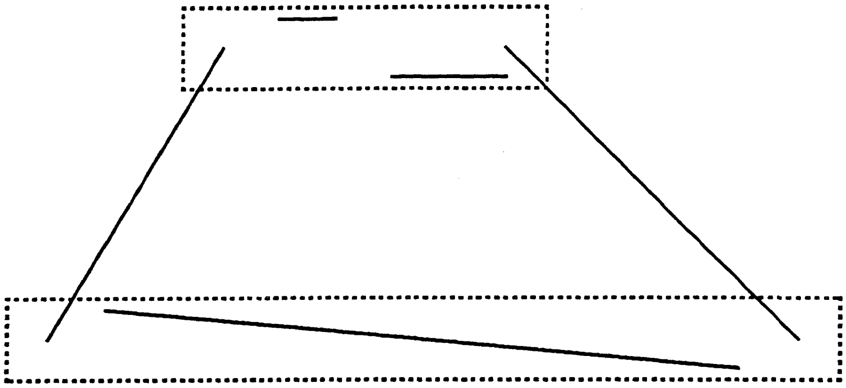
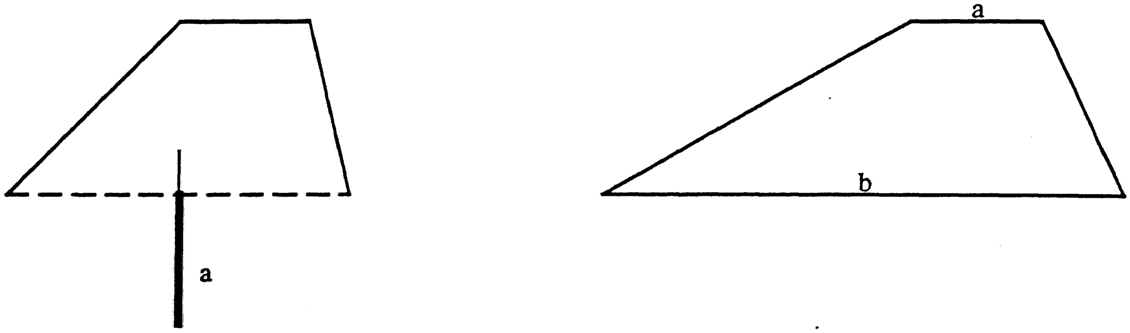


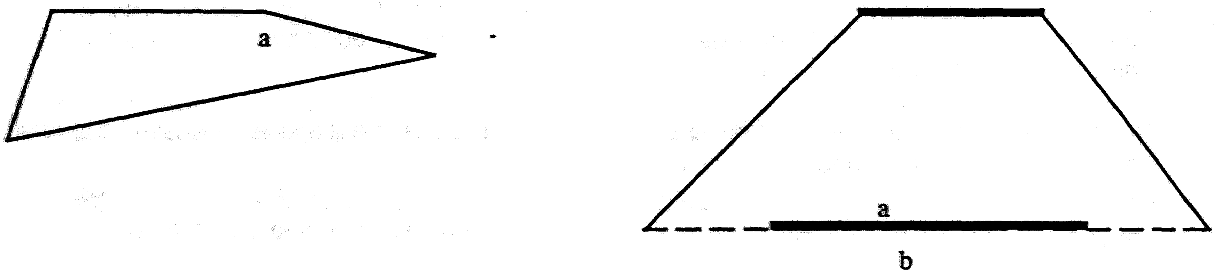
Figure 3-2: Trapezoid creation

These heuristics are illustrated in figure 3-3.



distance a extends outside > thr

(a / b) < thr



angle a > thr

(b / a) > thr

Figure 3-3: Trapezoid elimination heuristics

Trapezoid merging

The line pair finding process will find several pairs of lines from the set of edges forming a trapezoid in the image. In the simplest case, the top and bottom edges will form one pair, and the left and right edges will form another pair. This will result in the finding of two trapezoids, each using the same set of lines. These are really the same trapezoid. The duplication should be detected, and the two trapezoids merged. In addition, things such as trim around windows may create multiple pairs of lines which then give rise to trapezoids that are very similar. This stage of processing looks through the list of trapezoids returned by the previous stage and merges pairs of trapezoids which have corresponding sides which are close to each other, nearly parallel, and less that some probability of being false matches to each other (once again using Lowe's formula [7]).

3.3. Feature-fetchers and the search for initial matches

The procedures used to search for initial matches of image features to instances of cars consist of a match conflict resolver and a set of procedures called feature-fetchers (see figure 3-4). The match conflict resolver keeps track of "claims" to image features by matches. Only one match using a given image feature (the match with the highest confidence) survives, with the confidence values of competing matches for an image feature getting set to zero. Feature-fetchers are procedures which return instances of particular features. The features returned may be primitive features (such as lines, ellipses, or trapezoids) or compound features (such as a pair of trapezoids which could match the side and rear windows of a car). The model of the appearance of a car is encoded procedurally in the feature-fetchers in the form of calls to functions that evaluate constraints that should hold for a match.

A feature-fetcher may be called upon multiple times to return a given feature instance. A good example of this is the trapezoid_list feature-fetcher, which is called by all the other feature-fetchers. Each instance of a feature found by a feature-fetcher is assigned a sequential index number. That allows each feature-fetcher calling trapezoid_list to keep track of which trapezoids it has seen by storing the index of the last instance returned to it by trapezoid_list without requiring trapezoid_list to know anything about what feature-fetchers are calling it. In the case of the trapezoid_list feature-fetcher all the instances of the feature are extracted before the matching process begins. In the case of the other feature-fetchers instances are found as needed and cached. An instance can be fetched from the previously cached results if another feature-fetcher calls requesting an instance with an index number below the highest index value assigned so far. This avoids duplication of work by the feature-fetchers.

The functions of the individual feature-fetcher procedures are as follows:

- trapezoid_list returns trapezoids extracted in the earlier stages of the program
- get_windows returns a partial car match containing a pair of trapezoids which could correspond to the side and rear windows of a car instance
- add_roof gets partial matches from get_windows and looks for trapezoids from trapezoid_list which could match the roof given the trapezoids in the match of the windows
- add_trunk gets partial matches from add_roof and looks for trapezoids from trapezoid_list which could match the trunk given the window and roof trapezoids.

Feature-fetchers may employ multiple strategies to do their jobs. The add_trunk feature-fetcher is a good example. The roof and windows of the car look very similar from either the left or right, appearing roughly like a truncated pyramid. Since add_roof can't disambiguate which side of the car is being looked at, add_trunk employs two strategies to look for a trunk trapezoid to extend a partial match from add_roof. In the first strategy it assumes the car is being seen from the right and looks for the trunk to the left of the windows. When no more instances are found by the first strategy, the second strategy is invoked. This strategy assumes that the car is being seen from the left side, and looks for the trunk to the right of the windows. This involves stepping through the list of instances found by add_roof a second time, taking advantage of the caching done by the feature-fetchers.

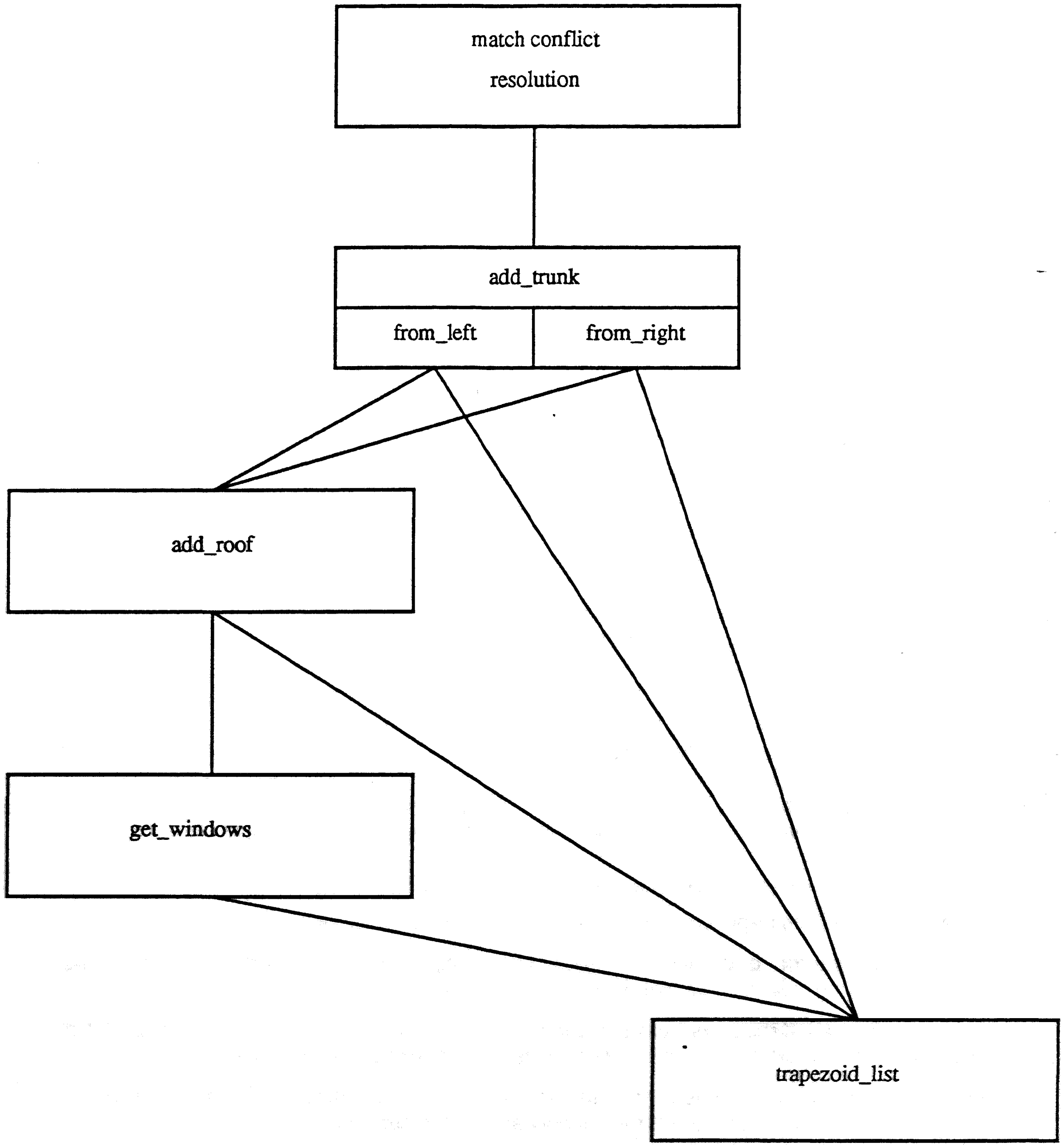

match conflict resolution
add_trunk
from_left
from_right
add_roof
get_windows
trapezoid_list

Figure 3-4: Hierarchy of search procedures

The feature-fetchers use a variety of constraints on the relationships between trapezoids and between sides of trapezoids to find matches. The routines which check if the constraints are satisfied take into account uncertainty about the objects (for instance, whether both endpoints of a segment are known for sure). Currently eight constraints are used in the model:

- **rightp**(seg1, seg2) -- Weak "right-of" constraint between segments. Satisfied if the maximum image column number of either endpoint of seg1 is greater than the maximum image column number of either endpoint of seg2. Otherwise not satisfied. This definition assumes that the camera is roughly vertical, and this constraint is used very sparingly to minimize the restrictions this places on the camera.

- leftp(seg1, seg2) - Weak "left-of" constraint between segments. Satisfied if the minimum image column number of either endpoint of seg1 is less than the minimum image column number of either endpoint of seg2. Otherwise not satisfied. Also used sparingly for the same reason as given for rightp.

- areparallef(seg1, seg2, tol) - Parallelism constraint between segments. Satisfied if segi and seg2 are parallel within the tolerance given by tol. May be satisfied if the segments aren't close to parallel, but both endpoints of either one of the segments are hypothesized. Otherwise it is not satisfied.

- angbetween(seg1, seg2, low, high) - Included angle constraint between segments. Here seg1 and seg2 are directed. Satisfied if the angle going counterclockwise from seg1 to seg2 is between the values low and high. May be satisfied if the actual direction of either segment is not known because both its endpoints are hypothesized. Otherwise not satisfied.

- trapform(trapezoid, start_side, angjypei, ang_Jype2, ang_type3, angjype4) -- Rough shape constraint for trapezoids. The ang_type variables have the value of OB (for obtuse - defined here as an angle between 70 and 180 degrees) or AC (for acute - defined here as an angle between 0 and 110 degrees). Trapform checks the angle at each vertex (with start_side used to indicate what side to consider side zero) to see if it is of the appropriate type. Trapform is satisfied if the angles at all four vertices are of the correct type. It may be satisfied if the angle at a vertex is of the wrong type but the position of the vertex is hypothesized. Otherwise it is not satisfied.

- phratio(trap1, pain, htypei, trap2, pair2, htype2, low, high) - Relative height ratio constraint between trapezoids. Trapi and trap2 are trapezoids, pain and pair2 specify which of the two pairs of opposing sides is involved from trapi and trap2. Htypei and htype2 specify the segment pair distance measure: either midpoint-to-midpoint for nonparallel lines or midpoint-to-opposite-segment for parallel or roughly parallel segments. The ratio between the height of the pair specified for trapi and the height of the pair specified for trap2 is computed. If the ratio falls between the low and high values specified the constraint is satisfied. If the ratio falls outside the specified range, but one of the segments of one of the specified pairs was only hypothesized, then the constraint may be satisfied. Otherwise the constraint is not satisfied.

- paraface(seg1, fowl, highi, seg2, low2, high2) - Constraint on the overlap between parallel or roughly parallel segments. The endpoints of seg2 are projected onto the line specified by seg1 and the fraction of the length of seg1 overlapped by the projection of seg2 is computed. Similarly the fraction of seg2 overlapped by the projection of seg1 is computed. The constraint is satisfied if the fraction of seg1 overlapped is between low1 and highi and the fraction of seg2 overlapped is between low2 and high2. The constraint may be satisfied if the overlapped fraction for a segment is out of range but the length of the segment is not know for sure (because one or both of its endpoints are hypothesized). Otherwise the constraint is not satisfied.

- onslde(trap1, edge, trap2) - Constraint on the relative position of two trapezoids. Picture walking clockwise around the sides of trapi. The constraint is satisfied if the side of trapi specified by edge is the only side such that all four vertices of trap2 are to your left as you walk along the side. Otherwise the constraint is not satisfied.

The above routines return a confidence factor between 0.0 and 1.0. Currently they return 1.0 If the constraint is satisfied; 0.1 if the constraint may be satisfied as described in the above descriptions; and 0.0 if the constraint was not satisfied. Any constraint on a possible match evaluating to 0.0 eliminates that match. The overall confidence of a match is just the sum of the values returned by the calls checking constraints from the model divided by the number of constraints.

3.4. Verification of initial matches

The verification stage goes through the list of matches which survive match conflict resolution, and checks for lines and ellipses verifying features that were not detected bottom-up. Information about expected orientation and position of lines based on a partial match is used to construct search windows to detect lines belonging to the rear and side of the car. The commitment to making no assumptions about the sizes of various parts is broken in the verification stage, which assumes that the side of the car is within some range of ratios of the height of the side window. This violation of one of the design goals can probably be eliminated by looking for the wheels and the bottom of the side of the car together. Ellipses extracted from the edge data are examined to look for wheels. The results of this stage are the final matches found by the program.

4. Results

Figure 4-1 shows the lines extracted from an image of the car in the upper left section; the lines after thresholding and merging in the upper right section; the extracted trapezoids in the middle left section; the initial match found bottom-up in the lower left section ; and the confirmed match in the lower right section. It can be seen that the line merging was able to improve the quality of the line segments, for instance by linking together the lines forming the back edge of the trunk and the line forming the top edge of the side window.

The trapezoids found include the roof, the trunk, the rear window, and bits of trim along the side. There are also a number of trapezoids found which arise from accidental alignments in the image. The hypothesized fourth sides on a number of them pass through the much stronger trapezoids found for the trunk and rear window, but do not get eliminated because no edge pierces far enough through the virtual sides.

Only one match remains after match conflict resolution. Comparing the final match with the original lines, one can see that the program has done a good job of finding the various parts of the car.

Figures 4-2 and 4-3 show the result of running the program on lines extracted from an image of the car as seen from the other side. Here two matches survive the match conflict resolution, even though there is only one car in the scene. This is a result of the match conflict resolution using features rather than image areas as the primitive things claimed by a match.

Figures 4-4 and 4-5 show the results produced by LASSIE for six additional images. The original lines extracted from the images are on the left and the verified matches are on the right. As can be seen, the program does a fairly good job of locating the parts of the car despite noise in the extracted edges.

5. Directions for future work

Analysis of trapezoid overlap to improve match conflict resolution and heuristic discarding of trapezoids. The current match conflict resolution scheme only looks at features. The trapezoid merging may not be able to merge all of the trapezoids arising from the various parts of the car. If all of the features searched for give rise to two or more trapezoids then multiple matches will survive match conflict resolution, despite there only being one car in the image. An example of this was shown in the results section in figure 4-3. Analyzing the area of overlap of the features forming the matches should allow resolution of this sort of conflict. The method will have to be designed in a way that will handle partial occlusion of one car by another.

Also, in some cases a hypothesized side which closes off a trapezoid goes through another trapezoid for which there is very strong support. There are cases where the test for lines crossing hypothesized sides fails to catch this. Looking at areas of intersection between trapezoids would help in the detection of these cases.

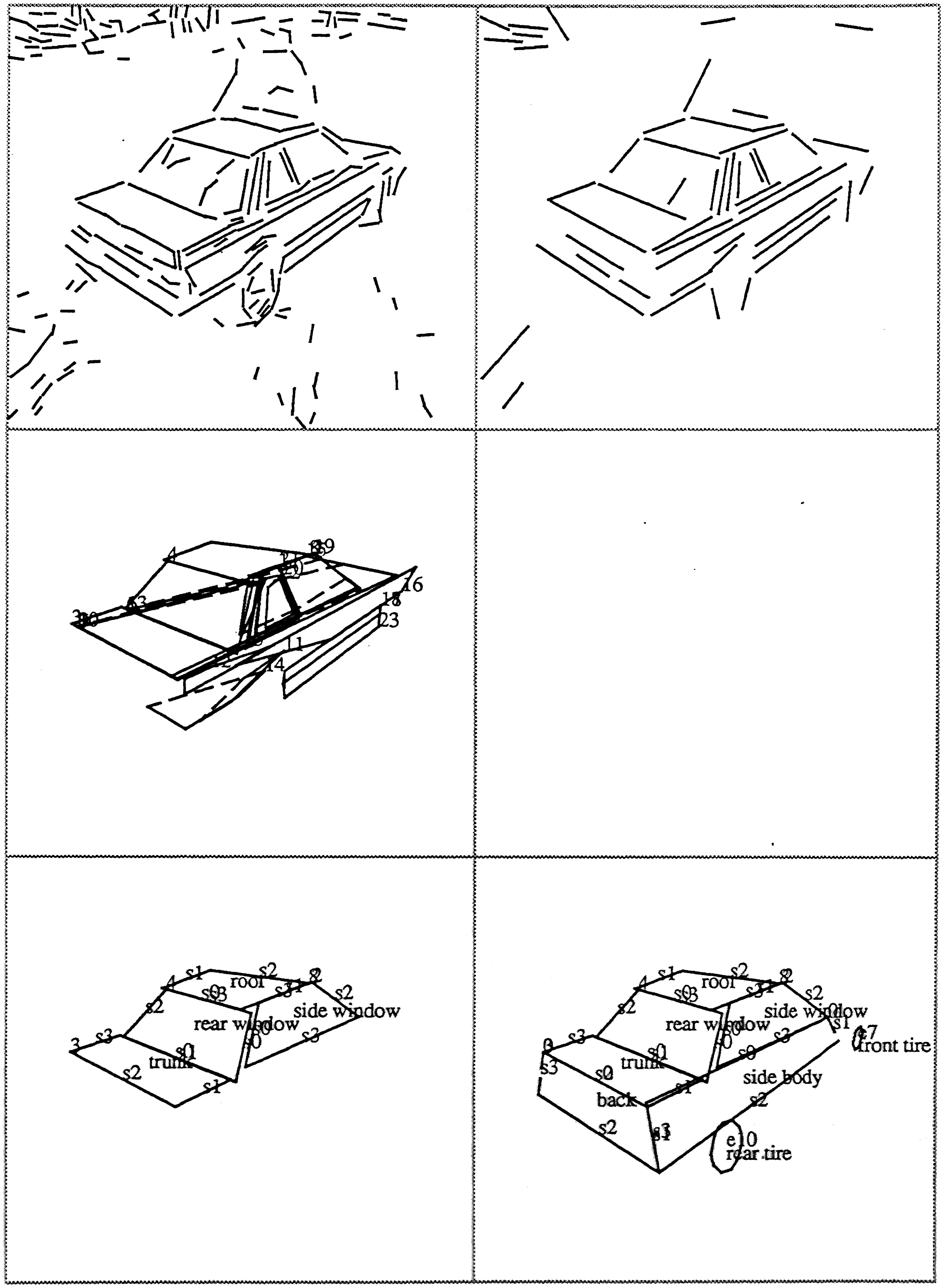

4
16
18
23
11
14
s1
s2
roof
s3
s0
rear window
side window
trunk
s1
s2
s3
3
side body
back
front tire
rear tire
e10
17

Figure 4-1: Results for the first image

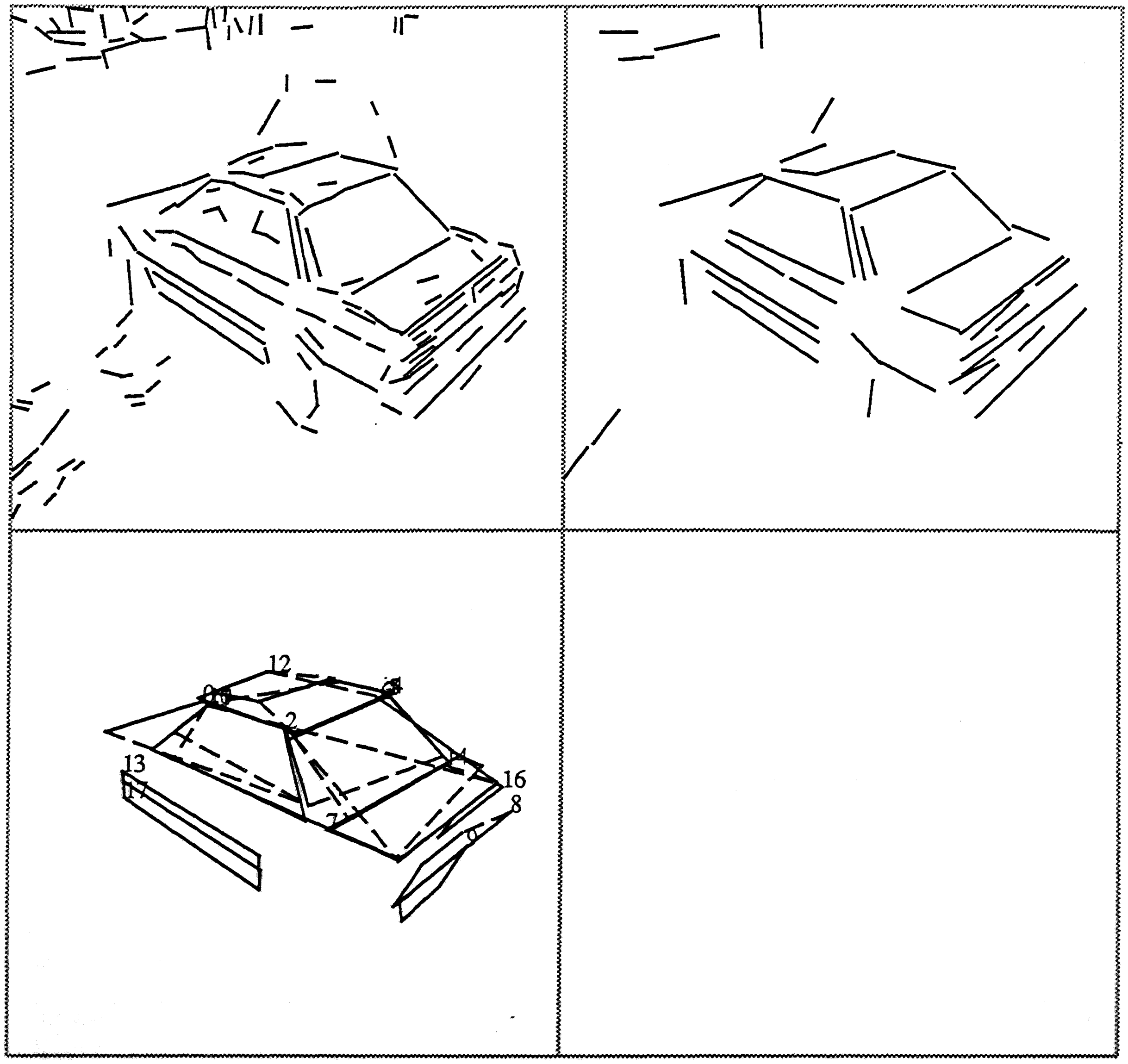


Figure 4-2: Second image: original lines, merged and thresholded lines, and extracted trapezoids

Improving the confidence factor scheme. The matches which survive conflict resolution are not always the most aesthetically pleasing ones. This is largely due to the overly simple confidence measure scheme currently used.

There are two major areas in which the confidence factor scheme could stand improvement. The first is in the granularity of the values returned by the constraint tests. Currently one of three values is returned based on whether the constraint is satisfied, unsatisfied but unsure, or definitely unsatisfied. This is too coarse. For instance the parallelism constraint could return a value proportional to how close to parallel the segments were.

The second area for improvement is the evidence combination scheme, which is completely ad hoc. Taking the product of the values returned for the various constraints in the model quickly produces values which are all uniformly small. Other techniques have not been investigated at this point.

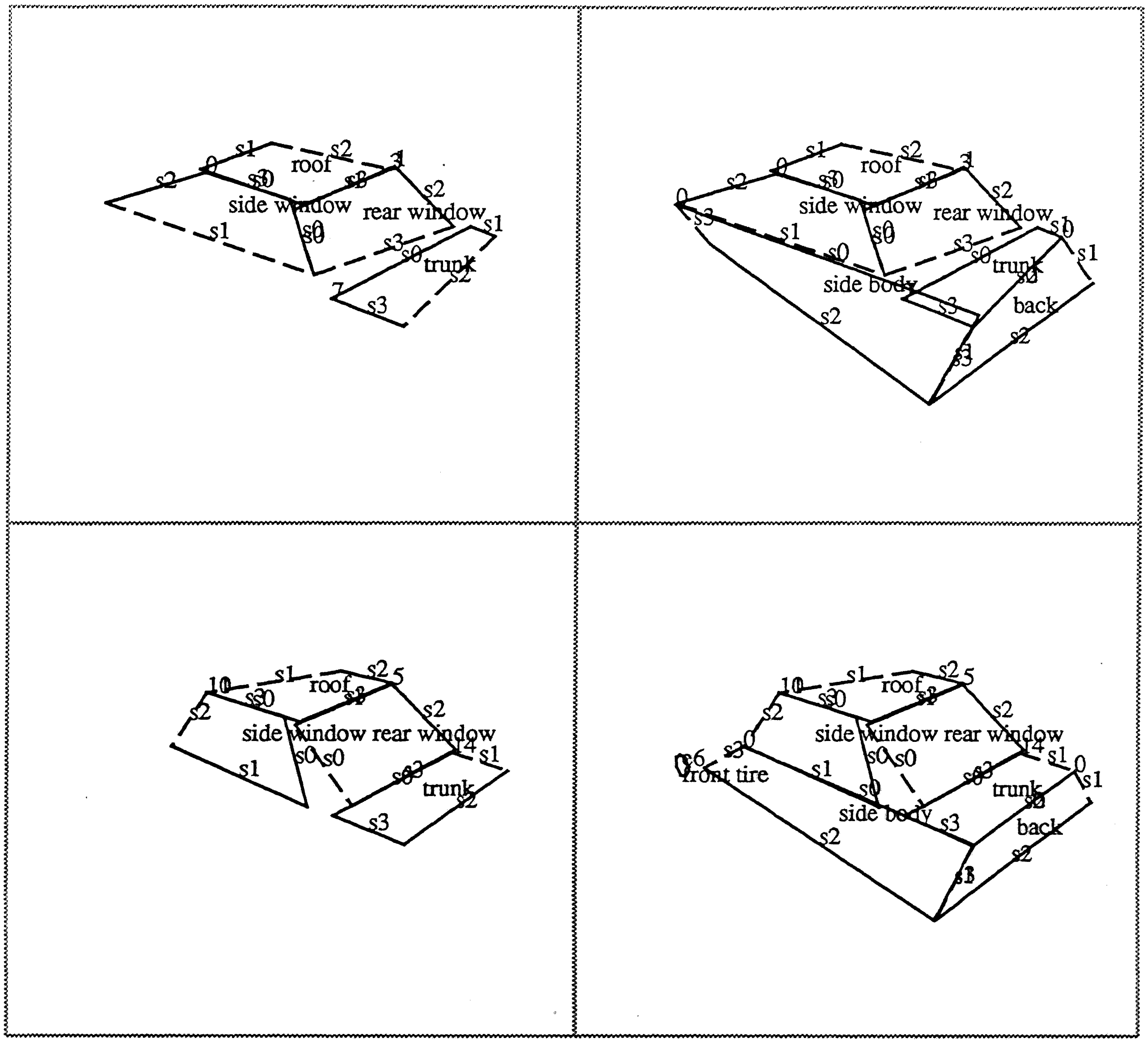


Figure 4-3: Second image: bottom-up match and match with verification

Integration of top-down search and verification into the feature-fetchers. Currently top-down verification of features not found in partial matches is done in a separate phase. Feature-fetchers should be written to try to extend a partial match by top-down search for missing features if no matching feature was found by the bottom-up feature extraction.

Making the search more flexible. Currently there is a single fixed order in which features are searched for: first find candidates for the windows, then extend by adding a roof, then extend by adding a trunk. Even with top-down search capabilities built into the feature-fetchers this is probably not sufficient. The whole idea of having feature-fetchers which cache their results was to allow more complex search strategies.

Doing something about all the thresholds. There are currently a dozen thresholds visible to the user, with another three or four buried in the code. Their default values are the result of good initial guesses followed by experimenting with how far they could be changed before the results started to

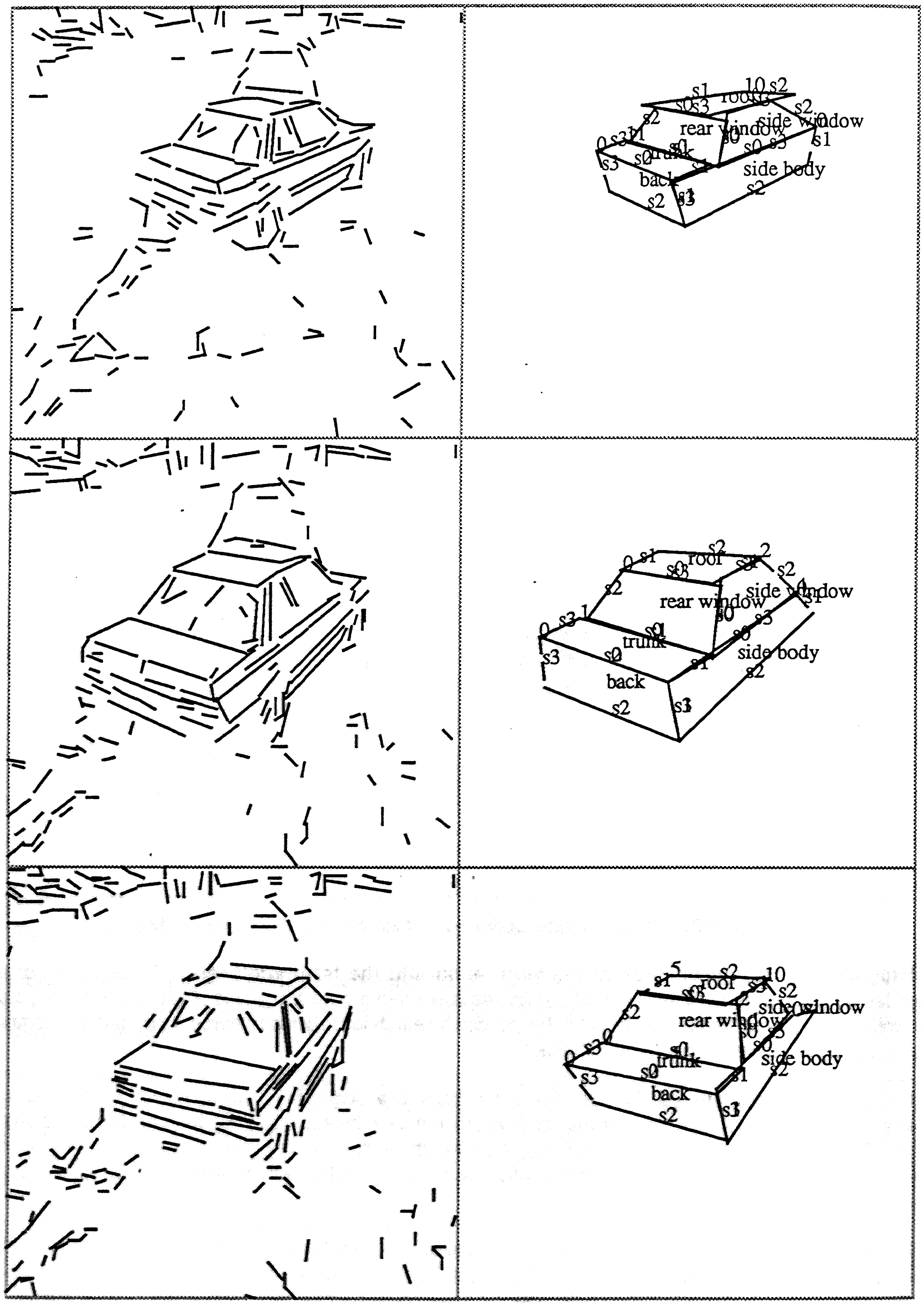

roof
rear window
side window
trunk
back
side body
s0
s1
s2
s3

Figure 4-4: Original lines and verified matches from another three images

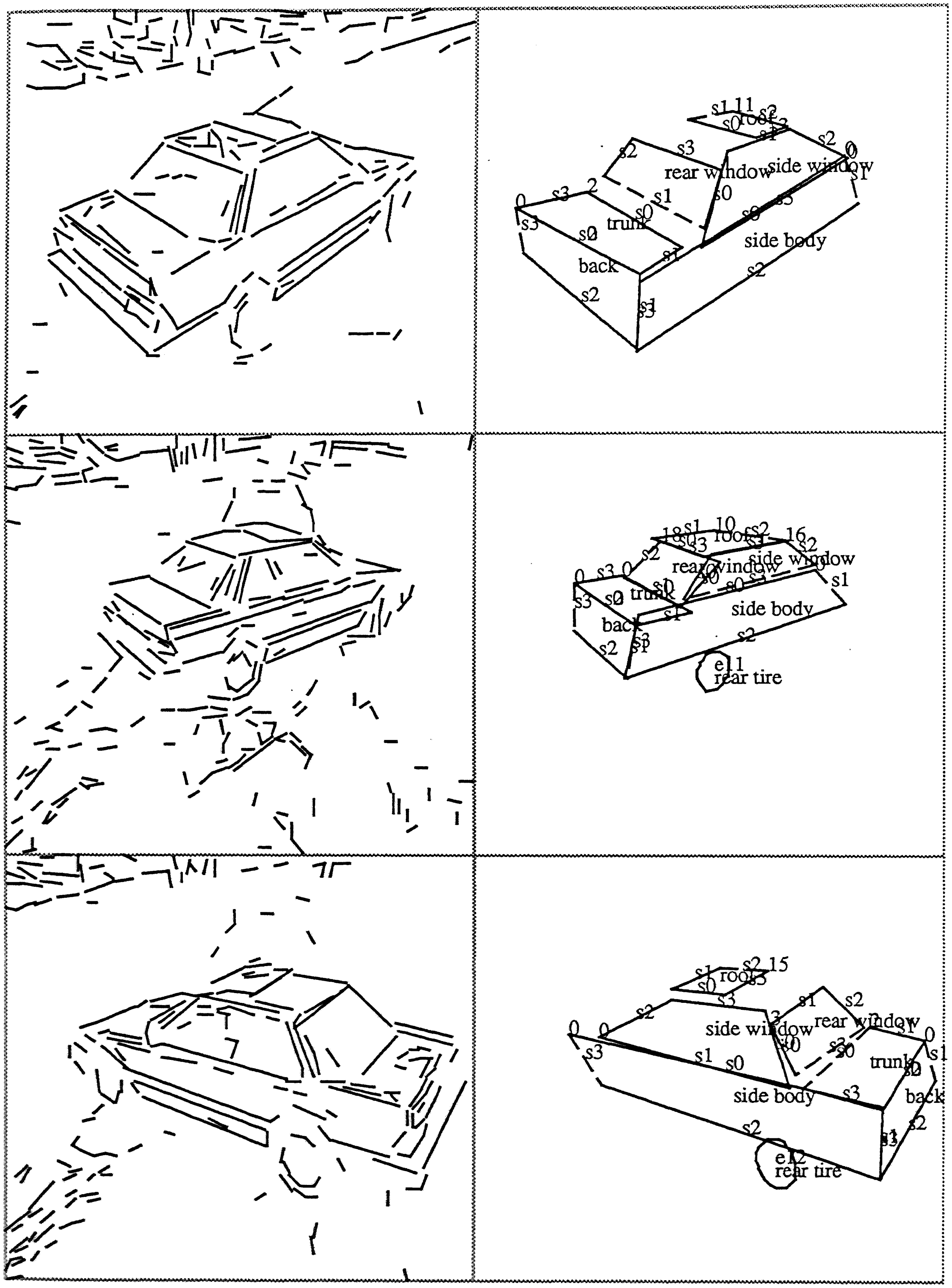

roof
rear window
side window
trunk
back
side body
s0
s1
s2
s3
roof
rear window
side window
trunk
back
side body
e11
rear tire
roof
side window
rear window
trunk
back
side body
e12
rear tire

Figure 4-5: Original lines and verified matches from the final three images

deteriorate. Setting them loosely enough to find all the features needed in the bottom-up stage of the search results in various bogus features being found %s well. Doing an initial pass with very tight thresholds to find the strongest features and letting the individual feature-fetchers loosen them as needed during top-down search would hopefully make the system less sensitive to these parameters.

Other views, other cars. Currently the system is only programmed to recognize the car as seen from two views, from behind to the right and from behind to the left. Extension to the other views of the car needs to be done. Also the program needs to be extended to recognize other makes of cars.

6. Summary

The LASSIE program has demonstrated the ability to robustly extract significant groups of lines forming trapezoids in the test images we have used. It can find instances of the test car from two views using only the geometric constraints described, without any model of the dimensions of the car other than the one exception mentioned in the section on match verification (section 3.4). It can do this despite the imperfections of edge data extracted from real outdoor images of the test car. Extensions of the work to support more complex search strategies, mixed top-down and botto.m-up searching in the feature-fetchers, and improved confidence measures can be expected to improve the performance of the program.

References

[1] Besl, Paul J., and Jain, Ramesh C.
Three-Dimensional Object Recognition.
Computing Surveys 17(1), March, 1985.

[2] Brooks, Rodney A.
Symbolic Reasoning Among 3-D Models and 2-D Images.
Artificial Intelligence 17:285-348, 1981.

[3] Fua, Pascal, and Hanson, Andrew J.
Locating Cultural Regions in Aerial Imagery Using Geometric Cues.
In *Proceedings of the DARPA Image Understanding Workshop.* December, 1985.

[4] Goad, Chris.
Special Purpose Automatic Programming for 3-D Model-Based Vision.
In *Proceedings of the ARPA Image Understanding Workshop.* June, 1983.

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

[6] Herman, Martin, and Kanade, Takeo.
The 3-D MOSAIC Scene Understanding System: Incremental Reconstruction of 3D Scenes from Complex Images.
Technical Report CMU-CS-84-102, Dept. of Computer Science, Carnegie Mellon Univ., February, 1984.

[7] Lowe, David G.
The Viewpoint Consistency Constraint.
International Journal of Computer Vision 1:57-72, 1987.

[8] Ohta, Y.
A Region-Oriented Image Analysis System by Computer.
PhD thesis, Kyoto University Dept. of Information Science, 1980.

[9] Thompson, D., and Mundy, J. L.
Three-Dimensional Model Matching from an Unconstrained Viewpoint.
In *Proc. IEEE International Conference on Robotics and Automation.* April, 1987.

Geometric Camera Calibration using Systems of Linear Equations*

Keith D. Grembai^
Martin Marietta Corporation

Charles E. Thorpe
Carnegie Mellon University

Takeo Kanade
Carnegie Mellon University

April 6, 1988

Abstract

Geometric camera calibration is the process of determining a mapping between points in world coordinates and the corresponding image locations of the points. In previous methods, calibration typically involved the iterative solution to a system of non-linear equations. We present a method for performing camera calibration that provides a complete, accurate solution, using only linear systems of equations. By using two calibration planes, a line-of-sight vector is defined for each pixel in the image. The effective focal point of a camera can be obtained by solving the system that defines the intersection point of the line-of-sight vectors. Once the focal point has been determined, a complete camera model can be obtained with a straightforward least squares procedure. This method of geometric camera calibration has the advantages of being accurate, efficient, and practical for a wide variety of applications.

TOs material ii based ispon wcxk supported by the Defense Advanced Research Projects Agency and the Army Engineering Topogmphic Lubcwatoricf under commas DACA76-M-C-0005 and DACA76-85-C-G0G3, and by the National Science Foundation wader grant DCR-S419990 and DCR-S604199. Any opinions, findings, conclusions, or recommendations expressed in this publication am those, of the autboa aid do not necessarily reflect the views of the Defense Advanced Research Projects Agency, the Amy *Bapasming* Topographic Laboratories, the National Science Foundation, or the U.S. Government.

^nalior's current mddrcts: Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213

1 Introduction

Many problems in computer vision and graphics require mapping points in space to corresponding points in an image. In computer graphics, for example, an object model is defined with respect to a world coordinate system. To generate an image, the points that lie on the visible surfaces of the object must be mapped onto the image plane; that is, 3d world points must be mapped onto 2d image points. In computer vision, the image locations of points on an object can be used to infer three-dimensional properties of the object; in this case, 2d image points must be mapped back onto the original 3d world points. In both cases, the mapping between 3d world coordinates and 2d image coordinates must be known. Geometric camera calibration is the process of determining the 2d–3d mapping between a camera and a world coordinate system.

We decompose the general problem of geometric camera calibration into two subproblems:

- The projection problem: given the location of a point in space, predict its location in the image; that is, *project* the point into the image.
- The back-projection problem: given a pixel in the image, compute the *line-of-sight* vector through the pixel; that is, *back-project* the pixel into the world.

A complete solution to the camera calibration problem entails deriving a model for the camera geometry that permits the solution of both the projection and the back-projection problems. For many applications, a complete solution is necessary. Some examples from the domain of mobile robots will help illustrate the problems.

In the CMU Navlab project [6], a robot vehicle follows roads using data from a color TV camera. In each image, the road is extracted, and the centerline and direction of the road computed in image coordinates. These parameters are then back-projected into vehicle coordinates and used to plot a course for the vehicle that stays within the road boundaries.

Turk, et al [8] describe a similar road following technique for the Autonomous Land Vehicle (ALV). Rather than parameterizing the road in terms of centerline and direction, they describe the road boundaries as a sequence of points. The line-of-sight vectors for each of the boundary points are computed by back-projection. The intersections of the line-of-sight vectors with the ground plane yield the points in the world between which the robot must steer to stay on the road. In addition, the ALV needs to know the predicted position of the road in each image. This prediction is obtained by projecting the location of the road into each image, based on the position of the road in the previous image, and the motion of the vehicle between images.

The simplest model for camera geometry is the pinhole, or perspective model. See figure 1. Light rays from in front of the camera converge at the pinhole and are projected onto the image plane at the back of the camera. To avoid dealing with a reversed image, the image plane is often considered to lie in front of the camera, at the same distance from the pinhole. The distance from the focal point to the image plane is the focal length.

A perfect lens can be modeled as a pinhole. No lens is perfect, of course, so part of the problem of geometric camera calibration is correcting for lens distortions. The most accurate and conceptually simple method of camera calibration would be to measure calibration parameters at each pixel in the image. For example, at each pixel measure the line-of-sight vector. This would produce a gigantic lookup table. Then, given a pixel in an image, simple indexing would yield the line-of-sight vector that solves the back-projection problem. To solve the projection problem, the table would be searched to find the line-of-sight vector that passes nearest the point in question.

A lookup table of calibration data for each pixel would be prohibitively expensive. The obvious compromise is to sample the image, and interpolate between data points. If the error in interpolation is less than the measurement error, no accuracy is lost. Most approaches to geometric camera calibration involve sampling the image, and solving for the parameters of the interpolation functions. The obvious differences between approaches are in the form of the interpolation functions, and the mathematical techniques used to solve for the parameters. The main intent of most calibration work has been the solution of the back-projection problem. The projection problem has occasionally been overlooked. The following paragraphs briefly describe past work.

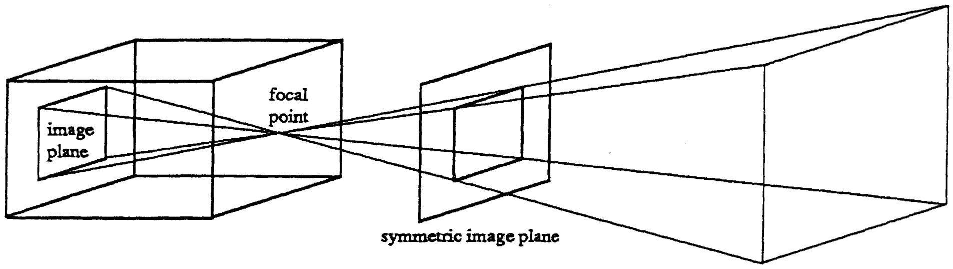


Figure 1: The Pinhole Camera Model

- Sobel [5] introduced a method for calibration that involved the solution of a large system of non-linear equations. In addition to solving for the intrinsic camera parameters, his method also solved for extrinsic camera parameters, such as camera pan and tilt. Sobel used the basic pinhole model, and solved the system using a non-linear optimization method. He did not model lens distortions, and the system depended on the user to provide initial parameters for the optimization technique.

  Tsai [7] improved on the general non-linear approach in several ways. He modeled distortions globally using fourth order functions, and presented a method for computing good initial parameters for the optimization technique. Tsai's model of lens distortions assumes that the distortions are radially symmetric.

- Yakimovsky and Cunningham [9] presented a calibration technique that also used a pinhole model for the camera. They treated some combinations of parameters as single variables in order to formulate the problem as a system of linear equations. However, in this formulation, the variables are not completely linearly independent, yet are treated as such. No lens distortions are modeled with this approach.

- Martins, Birk, and Kelley [3] reported a calibration technique that does not utilize an explicit camera model. Their two-plane calibration method consisted of measuring the calibration data for various pixels across the image. The data for other pixels is computed by interpolation. The back-projection problem is solved by computing the vector that passes through the interpolated points on each calibration plane. The interpolation can be either local or global. The two-plane method solves only the back-projection problem.

  Isaguirre, Pu, and Summers [2] extended the two-plane method to include calibration as a function of the position and orientation of the camera. They used an iterative approach based on Kalman filters to obtain the solution.

Our goal in camera calibration was to develop a single, basic calibration procedure to solve both the projection and back-projection problems for a variety of applications. Consequently, the desired procedure had to be conceptually straightforward, easily extended to obtain various degrees of accuracy, and computationally efficient. To meet these requirements, we chose to begin with the two-plane method of Martins, Birk, and Kelley. Section 2 discusses the two-plane method and the solution to the back-projection problem. This method can be made arbitrarily accurate; the only problem is that it fails to solve the projection problem. In section 3 we present a method for solving the projection problem that utilizes the calibration data from the two-plane method. The solution to the projection problem is a simple application of analytic geometry, and is completely formulated with systems of linear equations.

The calibration method presented in this paper has been implemented and tested in the Calibrated Imaging Laboratory at CMU [4]. Results are presented in section 4 that demonstrate the accuracy of this method.

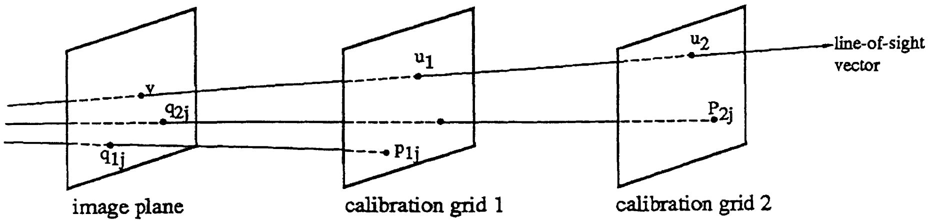


Figure 2: Two-Plane Calibration

2 The Solution to the Back-Projection Problem

Martins, Birk, and Kelly [3] first formally presented the two-plane calibration technique for solving the back-projection problem. This technique has the advantage that it provides exactly the information needed—the ray in space that defines the line of sight of a given pixel—without any explicit camera model.

Figure 2 illustrates the concept of two-plane calibration. Let P_1 and P_2 denote the calibration planes. Assume that the 3d locations of the calibration points on each plane are measured. An image of each plane is acquired, and the image location of each of the calibration points is extracted. Let the calibration points be denoted p_{ij}, and the corresponding image locations be denoted q_{ij}, where $i = 1, 2$ is the plane, and $j = 1, 2, \ldots, n$ is the point index. Thus, the image of p_{ij} is q_{ij}.

Let ρ and γ denote the row and column coordinates, respectively, of an image. Then, given a point $v = \begin{bmatrix} \rho & \gamma \end{bmatrix}^t$ in the image, the line-of-sight vector for v can be computed as follows. First, use the points p_{1j} and q_{1j} to interpolate the location of v on the first calibration plane, P_1. Call this point u_1. Then, interpolate to find the location of v on the second calibration plane. Call this point u_2. The pixel line-of-sight vector then has direction $u_1 - u_2$ and passes through the point u_1.

Various types of interpolation can be used, with different degrees of accuracy. Martins, et al report three types of interpolation: linear, quadratic, and linear spline. The two-plane method has the potential for being the most accurate of any calibration method for the solution of the back-projection problem. At the limit, this technique consists of measuring the line-of-sight vectors for each pixel in the image. As will be seen in Section 4, the number of calibration points used has a strong influence on the accuracy of the calibration.

2.1 Global Interpolation

One approach to interpolating the calibration data is to globally fit an interpolation function to the data. This function is then used for any pixel across the entire image. Global interpolation has the effect of averaging errors over all the pixels so that the resultant line-of-sight vector is exact for no pixel, but is close for all pixels. This has the advantage of reducing the sensitivity to errors or noise in measurements. On the other hand, the form of the interpolation function is an *a priori* assumption about the lens distortions, and may or may not be appropriate.

2.1.1 Linear Interpolation

Let $p_{ij} = [\ x \quad y \quad z\ ]^t$, and $q_{ij} = [\ \rho \quad \gamma \quad 1\ ]^t$. Then a linear transformation between p and q is given by

$$p_{ij} = A_i q_{ij}$$

where A_i is a 3x3 matrix. Given n measurements on each plane, we can then form the system

$$[\ p_{i1} \quad p_{i2} \quad \dots \quad p_{in}\ ] = A_i [\ q_{i1} \quad q_{i2} \quad \dots \quad q_{in}\ ]$$

or,

$$P_i = A_i Q_i$$

This system can be solved in the least squares sense by using the matrix pseudoinverse (also called the generalized matrix inverse) [1]:

$$A_i = P_i Q_i^t \left[Q_i Q_i^t \right]^{-1} .$$

Given a pixel v in the image, the direction of the line-of-sight vector through v is given by $u_1 - u_2$ where $u_1 = A_1 v$, and $u_2 = A_2 v$.

2.1.2 Quadratic Interpolation

Quadratic interpolation is similar to linear, except that second-order terms are used in the parameterization, and the matrix A is 6x6. We represent a point in space by $p = [\ x \quad y \quad z\ ]^t$, but we represent image locations by $q = [\ \rho \quad \gamma \quad \rho^2 \quad \gamma^2 \quad \rho\gamma \quad 1\ ]^t$. With these modifications, the formulation is otherwise identical. Martins, et al report that quadratic interpolation was more accurate than linear.

2.2 Local Interpolation

With no *a priori* knowledge about the lens distortions, global interpolation may be inappropriate. A better approach may be to model the distortions locally. If the calibration data is dense enough, the interpolation can be very accurate. In the paragraphs below, we discuss a technique called *linear spline interpolation*, which uses a linear function to perform interpolation over each local region.

Conceptually, this technique of interpolation consists of tesselating each calibration grid with triangles, and performing linear interpolation within each triangle. The calibration points form the vertices of the triangles. A plane is defined uniquely by three points, so no errors are introduced at the vertices. This is not the case for global interpolation techniques, in which errors are averaged over all points, including calibration points. Martins, et al achieved their best accuracy using this form of interpolation. In section 4, we report experiments which confirm this result.

In our current implementation, the grid is not tesselated in advance. Instead, for any point v in the image, each calibration grid is searched to find the three closest calibration points. The linear interpolation matrices A_i are computed using just three points each. The line-of-sight vector is then computed as in Section 2.1.1. Figure 3 illustrates the procedure.

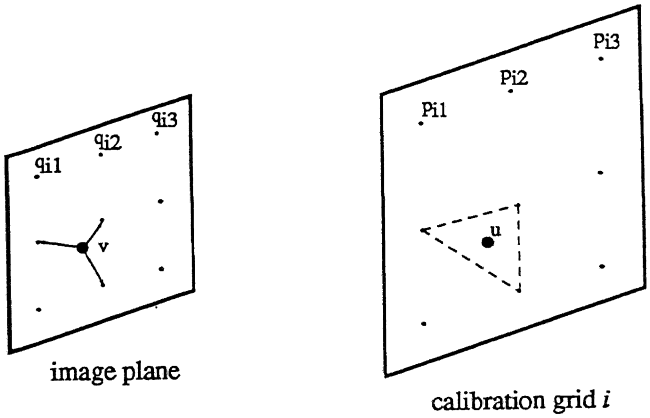

qi1
qi2
qi3
v
image plane
Pi1
Pi2
Pi3
u
calibration grid *i*

Figure 3: Linear Spline Interpolation

3 Linear Solution to the Projection Problem

The two-plane method for solution to the back-projection problem did not utilize an explicit camera model. In order to use the two-plane data to obtain a solution to the projection problem, it is necessary to have a camera model to formulate the equations. We use a model similar to that of Yakimovsky and Cunningham [9].

In figure 4 we begin with a pinhole model and define the following vectors and points:

- $P = [\ p_x \quad p_y \quad p_z\ ]^t$ == the vector from the origin to a point in space.
- $F = [\ f_x \quad f_y \quad f_z\ ]^t$ == the vector from the origin to the camera focal point.
- $R = [\ r_x \quad r_y \quad r_z\ ]^t$ == a vector that points along the direction of increasing row number. R represents the displacement vector from one pixel to the next in the row direction. The magnitude of R is the row scale factor.
- $C = [\ c_x \quad c_y \quad c_z\ ]^t$ == a vector that points along the direction of increasing column number. C represents the displacement vector from one pixel to the next in the column direction. The magnitude of C is the column scale factor.
- $[\ \rho_p \quad \gamma_p\ ]$ == the *piercing point* of the image, or the point where the optical axis pierces the image plane.

The vectors R and C define the orientation and scale of the image plane. Columns in the image plane are parallel to R, while rows are parallel to C.

The projection, $[\ \rho \quad \gamma\ ]$, of a point P onto the image plane can be computed by taking the dot product of the vector from the focal point to P, and adding the offset to the piercing point. For example, consider computing the row coordinate, ρ, of the projection. The vector $P - F$ is the vector from the focal point that passes through P. Every point along this vector will have the same location in the image. Let V be a normalized vector along $P - F$, that is, let

$$V = \frac{P - F}{\|P - F\|} \tag{1}$$

where $\|\cdot\|$ denotes the length of a vector. Let $\odot$ represent the usual vector dot product. Then $V \odot R$ represents the projection of V onto R measured in row units. Adding the row coordinate of the piercing point translates $V \odot R$ into image coordinates.

Therefore, the image location, $[\ \rho \quad \gamma\ ]$, of a point P can be computed using the equations:

$$\rho = V \odot R + \rho_p \tag{2}$$

$$\gamma = V \odot C + \gamma_p \tag{3}$$

If F, R, C, ρ_p, and γ_p are all unknown, then the resulting system is non-linear. However, the two-plane formulation of the back-projection problem yields the information needed to make solving for the focal point location a linear problem.

3.1 Focal Point Solution

In a pinhole camera, all incoming light rays pass through the focal point. Since a lens is not a perfect pinhole, we instead refer to an *effective focal point*, which is the point that is closest to all the rays. From the two-plane method for the back-projection problem, one can compute a bundle of rays that pass through the lens. The next step is to find the point in space that minimizes the distance to all the rays.

The equation for the squared distance, d^2, from a point $P = [\ x \quad y \quad z\ ]^t$ to the line through $P_1 = [\ x_1 \quad y_1 \quad z_1\ ]^t$ in direction $[\ a \quad b \quad c\ ]^t$ (where $a^2 + b^2 + c^2 = 1$) is:

$$d^2 = \begin{vmatrix} y - y_1 & z - z_1 \\ b & c \end{vmatrix}^2 + \begin{vmatrix} z - z_1 & x - x_1 \\ c & a \end{vmatrix}^2 + \begin{vmatrix} x - x_1 & y - y_1 \\ a & b \end{vmatrix}^2$$

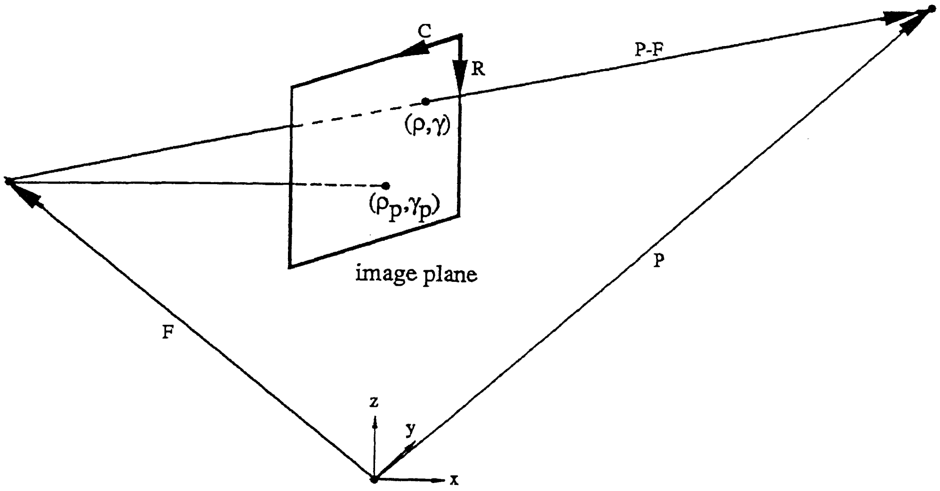


Figure 4: A Linear Model of Camera Geometry

Expansion of terms yields:

$$\begin{aligned} d^2 &= x^2(b^2+c^2)+y^2(a^2+c^2)+z^2(a^2+b^2) \\ &\quad - 2xyab - 2xzac - 2yzbc \\ &\quad + 2x(bk_3 - ck_2) + 2y(ck_1 - ak_3) + 2z(ak_2 - bk_1) \\ &\quad + k_1^2 + k_2^2 + k_3^2 \end{aligned}$$

where:

$$\begin{aligned} k_1 &= z_1 b - y_1 c \\ k_2 &= x_1 c - z_1 a \\ k_3 &= y_1 a - x_1 b \end{aligned}$$

To find the effective focal point, we need to minimize $D = \sum d^2$. Differentiating D with respect to x, y, and z yields:

$$\begin{aligned} \partial D/\partial x &= \sum 2x(b^2+c^2) - \sum 2yab - \sum 2zac + \sum 2(bk_3 - ck_2) \\ \partial D/\partial y &= \sum 2y(a^2+c^2) - \sum 2xab - \sum 2zbc + \sum 2(ck_1 - ak_3) \\ \partial D/\partial z &= \sum 2z(a^2+b^2) - \sum 2xac - \sum 2ybc + \sum 2(ak_2 - bk_1) \end{aligned}$$

The sums are taken over all the line-of-sight vectors (a, b, c, k_1, k_2, k_3 are functions of the vectors).

Now, by setting the derivatives of D to zero to find the minima, and putting the equations in matrix form, we obtain:

$$h = Af$$

where:

$$h = \begin{bmatrix} \sum(ck_2 - bk_3) \\ \sum(ak_3 - ck_1) \\ \sum(bk_1 - ak_2) \end{bmatrix}$$

$$A = \begin{bmatrix} \sum(b^2+c^2) & -\sum ab & -\sum ac \\ -\sum ab & \sum(a^2+c^2) & -\sum bc \\ -\sum ac & -\sum bc & \sum(a^2+b^2) \end{bmatrix}$$

$$f = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

So the solution we seek, f, the effective focal point of the camera, is simply:

$$f = A^{-1}h$$

3.2 Computation of the Camera Base Vectors

Equations (2) and (3) relate the position of a point in space to a corresponding image location. These equations can be written as a linear system:

$$\begin{bmatrix} \rho & \gamma \end{bmatrix} = \begin{bmatrix} v_x & v_y & v_z & 1.0 \end{bmatrix} \begin{bmatrix} r_x & c_x \\ r_y & c_y \\ r_z & c_z \\ \rho_p & \gamma_p \end{bmatrix}$$

Given N points in space, we have $2N$ equations to solve for the 8 unknowns in R, C, ρ_p, and γ_p:

$$\begin{bmatrix} \rho_1 & \gamma_1 \\ \rho_2 & \gamma_2 \\ \vdots & \vdots \\ \rho_n & \gamma_n \end{bmatrix} = \begin{bmatrix} v_{x1} & v_{y1} & v_{z1} & 1.0 \\ v_{x2} & v_{y2} & v_{z2} & 1.0 \\ \vdots & \vdots & \vdots & \vdots \\ v_{xn} & v_{yn} & v_{zn} & 1.0 \end{bmatrix} \begin{bmatrix} r_x & c_x \\ r_y & c_y \\ r_z & c_z \\ \rho_p & \gamma_p \end{bmatrix}$$

or,

$$B = WX$$

So, using the pseudoinverse to obtain a least squares solution, we have:

$$X = \left[W^tW\right]^{-1} WB$$

X contains the values for R, C, ρ_p, and γ_p.

3.3 The Local Projection Problem

In section 2 we presented several ways of modeling camera geometry for the back-projection problem. The *linear interpolation* technique (section 2.1.1), which involved fitting a first-order transformation to all the calibration data, is a global modeling technique. The *linear spline interpolation technique* (section 2.2) is a *local* modeling technique, since at each pixel, only the calibration data in a local region around the pixel is used to compute the interpolation function. The results of Martins, et al [3], and our laboratory results (see section 4) both indicate that a local modeling technique yields superior accuracy.

The solution to the projection problem presented in sections 3.1 and 3.2 is a global modeling technique. All the calibration data is used to compute the model parameters, and the results are used to solve the projection problem for any point in space. In direct analogy to the linear spline technique used in back-projection, a technique can be derived for using local information to improve the accuracy of solution to the projection problem.

Our local projection technique involves finding a linear model for local regions of the image. The technique involves two steps. In the first step, the global solution is used to obtain an estimated image location for a point in space. That estimated image location is used to find the four nearest calibration points on each calibration plane. These points are used to compute a local linear solution to the projection problem. The local linear solutions could also be precomputed, and the global solution would then be used simply to index the correct local solution.

4 Experimental Results

Measurements and tests were conducted within the Calibrated Imaging Laboratory (CIL) at CMU (Shafer [4]). The CIL is a facility that provides a precision imaging capability. The purpose of the CIL is to provide researchers with accurate knowledge about ground truth so that computer vision theories can be tested under controlled scientific conditions. Of particular interest for this study, the CIL provides facilities to accurately measure point locations, and to accurately position and orient cameras.

Position measurement of points in the CIL is performed with the use of theodolites (surveyor's transits), which are basically telescopes with crosshairs for sighting, mounted on accurate pan/tilt mechanisms. Objects to be measured are placed at one end of an optical bench; the theodolites are fixed to the other end, separated by a little more than 1 meter. To measure the position of a point, the crosshairs of each theodolite are placed over the point, and the horizontal and vertical displacements read off. Trigonometric equations then yield the position of the point in a Cartesian coordinate system defined with respect to the theodolites. As currently configured, the theodolites can determine point locations to less than 0.1 mm.

4.1 Test Scenario

The laboratory tests described below were designed to provide answers to the following questions:

1. What accuracies can be expected from off the shelf cameras and lenses?
2. How does increasing the number of calibration points affect the accuracy of calibration?
3. What is the expected accuracy for the projection problem?

Tests were performed using a calibrated grid. The grid consisted of horizontal and vertical lines 1mm in width, spaced 12.7 mm apart. The intersections of the lines on the grid were used as calibration points. A special intersection detector was implemented to extract the intersections from digital images with subpixel precision. Each time the grid was moved, new measurements were taken, an image digitized, and the intersection detector applied. The result was a data file in which each calibration point was associated with its 3d position and its image location.

A complete test consisted of data from three different grid locations. Due to the size of the laboratory, focal length of the lens, and depth of field of the lens, the grid was typically placed at distances ranging from 0.50m to 0.56m from the camera. Data from two of the grid locations was used to compute calibration parameters. These parameters were then tested using data from the third grid location. The third grid will often be referred to as the test grid. In each of the tests reported here, the focus of the camera was kept fixed. The camera used was a Sony CCD, model AVC-D1, with the standard 16mm lens.

A total of 300 calibration points were used on each grid. Rather than measure the location of each point individually, the location of each point was computed based on the measured locations of the center point and the four corners. Consequently, the accuracy of the data depended not only on the accuracy of the theodolites, but also upon factors such as the planarity of the grid, and the precision of the grid lines. In preparing for each test, the overall accuracy of the calibration data was estimated. For several points at each grid location, the 3d locations were measured using the theodolites. The measured locations were then compared with the computed locations. Differences of up to 0.2 mm were recorded, with typical differences being between 0.1 and 0.2 mm. The accuracy of the calibration method is limited by the accuracy of the calibration data, so the best accuracy achievable in this scenario is between 0.1 and 0.2 mm.

The effects of density of calibration points on calibration accuracy was tested by varying the number of calibration points used. This was easily implemented by simply skipping over some of the rows and columns in the grid. In each case, the calibration points were uniformly distributed over the image. Data is reported for the following distributions of points: 3x3, 5x7, 7x10, 15x20.

In all the tests reported below, grid 0 refers to the grid location farthest from the camera, while grid 2 refers to the grid location closest to the camera. In all cases, the number of points was varied to compute the calibration parameters, but all 300 calibration points on the test grid were used in testing.

array size	calibration grids	test grid	error (mm)	
			global	local
3x3	0, 1	2	1.921	0.731
	0, 2	1	0.388	0.296
	1, 2	0	0.561	0.317
5x7	0, 1	2	1.854	0.740
	0, 2	1	0.366	0.166
	1, 2	0	0.551	0.235
7x10	0, 1	2	1.810	0.696
	0, 2	1	0.350	0.169
	1, 2	0	0.509	0.185
15x20	0, 1	2	1.813	0.666
	0, 2	1	0.366	0.147
	1, 2	0	0.534	0.201

Table 1: Calibration Accuracy of the Back-Projection Problem

4.2 Back-Projection Results

To test the accuracy of the back-projection problem, the image location of each of the calibration points on the third grid was used to compute a line-of-sight vector. The intersection of this vector with the plane of the test grid was computed, and the distance between the intersection and the actual position was used as the error measure. In the results reported below, the errors are averages taken over all the calibration points.

Table 1 presents the results obtained for the back-projection problem. The first error column contains the results for global linear interpolation. For this method, the density of the calibration grid makes little or no difference to the accuracy of the result. This was expected; since the calibration points are uniformly distributed across the grid, additional points do not provide additional information for a linear fit. The best accuracy is achieved when the test grid is positioned between the other two grids used for calibration.

The second error column in table 1 presents the results obtained for back-projection problem using local linear spline interpolation. This time there is a general trend for greater accuracy with more calibration points. This reflects the fact that the linear spline method interpolates over local regions, and can more accurately approximate effects such as barrel distortion. There are instances observable in the table which seem to contradict the general trend; these are most likely due to noise in the measurements or in the process of point extraction. Over a number of trials, the general trend has been consistent.

The results in table 1 agree with the results obtained by Martins, et al. To summarize, the local linear spline interpolation procedure, with as few as 12 calibration points, is more accurate than global linear interpolation. In addition, the use of more calibration points improves the accuracy of the local linear spline method. The accuracies we achieved in our tests were at the level of the accuracies of our measurements.

4.2.1 Projection Results

The accuracy of the projection problem was tested with a procedure similar to that used in the back-projection problem. The 3d location of each calibration point on the test grid was projected into the image plane, and the difference (in pixels) between the projected location and the measured location was used as the error measure.

The test results for the projection problem are reported below in table 2. The first error column gives the error, in pixels, of the accuracy using global interpolation. The average error reported in all cases was less than two pixels, which is good enough for many applications. The results indicate that the standard lenses for our cameras are reasonably good, and can be approximated well with a pinhole model.

The second error column in table 2 reports the errors recorded using the local solution to the projection problem.

array size	calibration grids	test grid	error (pixels)	
			global	local
3x3	0, 1	2	1.58	1.70*
	0, 2	1	0.89	0.65
	1, 2	0	0.89	0.82
5x7	0, 1	2	1.32	1.29
	0, 2	1	0.95	0.36
	1, 2	0	0.98	0.46
7x10	0, 1	2	1.20	1.01
	0, 2	1	0.91	0.34
	1, 2	0	0.88	0.40
15x20	0, 1	2	1.15	1.38*
	0, 2	1	0.92	0.92
	1, 2	0	0.91	0.36

Table 2: Accuracy of the Projection Problem

A comparison of the two columns in table 2 shows an improvement resulting from using local information. In general, the results from the local solution to the projection problem are a factor of 2 improved over the global solution. The entries followed by a * are examples where the global result was better than the local result; this may be an effect of errors in the measurement process. The general conclusion that can be drawn is that local models of camera geometry provide more accurate results than global models—for simple interpolation functions.

4.2.2 Conclusions

In section 4.1, we enumerated three questions which were to be answered by the tests reported above. We now proceed to answer each of these questions in turn.

1. *What accuracies can be expected from off the shelf cameras and lenses?*

   Tables 1 and 2 of test results show the accuracy achievable with a standard commercial CCD, using the standard lens supplied with the camera. With a simple global interpolation scheme, accuracies as good as 1 part in 1400 (0.3 mm over 530 mm) can be obtained. With a more sophisticated local linear spline interpolation, the accuracies can be increased to 1 part in 3500.

2. *How does increasing the number of calibration points affect the accuracy of calibration?*

   We have shown that the maximam accuracy for global linear interpolation can be achieved with a small number of calibration points, provided that the points are uniformly distributed over the image. Further increasing the number of calibration points has no effect on the accuracy. With a local linear spline interpolation, adding calibration points clearly improves the accuracy of the back-projection problem, until the limiting accuracy of the calibration data is reached.

3. *What is the expected accuracy for the projection problem?*

   Using either a local or global solution, the projection problem can be solved to within two pixels; results as good as 0.34 pixels were reported. For many applications, solution of the projection problem need not be extremely accurate. In many instances, the projected pixel location is only needed to find the center of a region within which an operation will be performed. For these applications, accuracy of one to two pixels is adequate.

   It is important to note that the local interpolation outperformed the global interpolation. While the differences were not great in our tests, the lenses we used were fairly linear. If extremely wide angle lenses are used, the distortions may be large, and the ability to locally interpolate will be much more important.

5 Discussion

We have presented a calibration method that we believe meets many of the requirements of a basic calibration technique that can be used for a variety of applications. Our method is based on the two-plane method of Martins, Birk, and Kelley [3], but is extended to include a solution to the projection problem. We believe that the method presented here has many advantages, described in the following paragraphs:

- Completeness.

  The original two-plane method of calibration only provided a solution to the back-projection problem. While this is sufficient for many applications, a solution to the projection problem is also necessary for applications such as mobile robots. We have extended the two-plane method by providing a solution to the projection problem.

- Accuracy.

  The two-plane calibration method can be made arbitrarily accurate. As reported in section 4, increasing the number of calibration points results in increasing accuracy. If no improvement results from adding more points, then the accuracy of the calibration data must be improved.

  The projection problem exhibits much of the same behavior as the back-projection problem. Of particular interest is the observation that local modeling of camera geometry improves the accuracy of the projection problem, as well as the back-projection problem. The accuracies observed in our tests were typically less than one pixel.

- Simplicity.

  The two-plane model is conceptually very straightforward and easy to implement. The use of the line-of-sight vectors to solve for the parameters of a linear camera model arises intuitively from the geometry of the model. The method of solution for the camera model involves solving only linear equations, so no sophisticated optimization techniques are involved.

- Efficiency.

  Solution of either the back-projection or projection problems require only a few matrix multiplies and matrix inversions on small matrices. The operations are guaranteed to produce a unique answer within a fixed time. While a relatively large amount of data must be stored for this calibration method compared to other methods, the total amount is still insignificant.

- Practicality.

  Because the method provides a complete solution to the geometric calibration problem, the method can be used for any application. The accuracy can be arbitrarily increased (or decreased) to meet the requirements for a given application. The only change in the method is to store the data from more calibration points. The mathematics remains the same, and no special equipment is required beyond that needed to obtain precise locations for the calibration points.

In addition to the benefits of the method we presented, some general observations should be made:

- Without the benefit of *a priori* knowledge of the form of lens distortions, local modeling of distortions seems to perform better than global modeling. A global model is an attempt to fit the data into a predetermined form and average the error across the entire image. The accuracy of the interpolation is limited by how well the chosen model reflects reality. Conceivably, a different function could be required for different types of lenses to reflect different models of distortion.

  Modeling distortion locally makes no assumption about the forms of the lens distortions. The local model can be made arbitrarily accurate by simply sampling at more pixels. We have shown that relatively few points are needed to achieve the level of accuracy that the measurement devices provide. Moreover, local modeling is more accurate for solving both the projection and back-projection problems.

- Nearly all of the data reported showed that the best accuracy was obtained when the test grid was placed between the two grids used for calibration. This is a specific instance of the general fact that interpolation is more accurate than extrapolation. In calibrating a real robotic system, the calibration data should ideally be obtained so as to bound the region of interest as much as possible.

Geometric camera calibration may depend on a variety of factors. For example, the focal distance, the aperture setting, presence or absence of a filter, or even the operating temperature of a camera may all affect the calibration parameters. We are currently making measurements and conducting tests to determine the sensitivity of calibration parameters to many of these factors.

The results reported in this paper were obtained using data from the Calibrated Imaging Laboratory at CMU. Our next application of this method will be to calibrate and register three cameras and a laser range finder mounted on the CMU Navlab.

6 References

References

[1] A. Ben-Israel, and T. N. E. Greville, *Generalized Inverses: Theory and Applications*, John Wiley & Sons, Inc., New York (1974)

[2] A. Isaguirre, P. Pu, and J. Summers, 'A New Development in Camera Calibration: Calibrating a Pair of Mobile Cameras,' *Proc. IEEE Conference on Robotics and Automation*, pp. 74-79 (1985).

[3] H. A. Martins, J. R. Birk, and R. B. Kelley, 'Camera Models Based on Data from Two Calibration Planes,' *Computer Graphics and Image Processing*, 17:173-180 (1981).

[4] S. A. Shafer, 'The Calibrated Imaging Lab Under Construction at CMU,' *Proc. 1985 DARPA Image Understanding Workshop.*

[5] I. Sobel, 'On Calibrating Computer Controlled Cameras for Perceiving 3D Scenes,' *Artificial Intelligence*, 5:185–198 (1974).

[6] C. E. Thorpe, M. Hebert, T. Kanade, and S. Shafer, 'Vision and Navigation for the Carnegie Mellon Navlab,' *Annual Review of Computer Science*, Volume 2, pp. 521–556 (1987), Annual Reviews, Inc.

[7] R. Y. Tsai, 'An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision,' *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 364–374 (1986).

[8] M. A. Turk, D. G. Morgenthaler, K. D. Gremban, and M. Marra, 'Video Road-Following for the Autonomous Land Vehicle,' *Proc. IEEE Conference on Robotics and Automation*, pp. 273–280 (1987).

[9] Y. Yakimovsky, and R. Cunningham, 'A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras,' *Computer Graphics and Image Processing* 7:195–210 (1978).