

AD-A212 420

Final Report on Research in Computer Vision for Autonomous Systems

DTIC
ELECTE
SEP 1 3 1989
S D
D

Avi Kak
Mark Yoder
Keith Andress
Steve Blask
Tom Underwood
Bishara Shamee

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

Robot Vision Lab

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Project Period: January 1986 through December 1988

89 9 13 124

DISTRIBUTION STATEMENTS
(DoD Directive 5230.24)

Please check appropriate block:

☐ 1. _____ Copies are being forwarded.
Indicate whether Statement A, B, C, D, E, F, or X applies.

DISTRIBUTION STATEMENT A:

☒ APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED.

DISTRIBUTION STATEMENT B:

☐ DISTRIBUTION LIMITED TO U.S. GOVERNMENT AGENCIES ONLY; (Indicate Reason and Date). OTHER REQUESTS FOR THIS DOCUMENT MUST BE REFERRED TO (Indicate Controlling DoD Office).

DISTRIBUTION STATEMENT C:

☐ DISTRIBUTION LIMITED TO U.S. GOVERNMENT AGENCIES AND THEIR CONTRACTORS; (Indicate Reason and Date). OTHER REQUESTS FOR THIS DOCUMENT MUST BE REFERRED TO (Indicate Controlling DoD Office).

DISTRIBUTION STATEMENT D:

☐ DISTRIBUTION LIMITED TO DOD AND DOD CONTRACTORS ONLY; (Indicate Reason and Date). OTHER REQUESTS MUST BE REFERRED TO (Indicate Controlling DoD Office).

DISTRIBUTION STATEMENT E:

☐ DISTRIBUTION LIMITED TO DOD COMPONENTS ONLY; (Indicate Reason and Date). OTHER REQUESTS MUST BE REFERRED TO (Indicate Controlling DoD Office).

DISTRIBUTION STATEMENT F:

☐ FURTHER DISSEMINATION ONLY AS DIRECTED BY (Indicate Controlling DoD Office and Date) or HIGHER DOD AUTHORITY.

DISTRIBUTION STATEMENT X:

☐ DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND PRIVATE INDIVIDUALS OR ENTERPRISES ELIGIBLE TO OBTAIN EXPORT-CONTROLLED TECHNICAL DATA IN ACCORDANCE WITH REGULATIONS IMPLEMENTING 10 U.S.C. 140c (Indicate Date of Determination). OTHER REQUESTS MUST BE REFERRED TO (Indicate Controlling DoD Office).

For further information on withholding of export-controlled, unclassified technical data as referred to in Distribution Statement X, see DoD Directive 5230.25, Withholding of Unclassified Technical Data From Public Disclosure, issued 6 Nov 1984.

☐ 2. This document was previously forwarded to DTIC on _____ (date) and the AD number is _____.

☐ 3. In accordance with the provisions of Department of Defense instructions, the document requested is not supplied because:

☐ It is TOP SECRET.

☐ It is excepted in accordance with DoD instructions pertaining to communications and electronic intelligence.

☐ It is a registered publication.

☐ It is a contract or grant proposal, or an order.

☐ Will be published at a later date. (Enter approximate date, if known).

☐ Other (Give reason).

TIMOTHY J. WILLIAMS
Print or Type Name

Timothy J Williams 31 May 89
Authorized Signature Date

703-664-6066
Telephone Number

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Purdue Research Foundation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Center for Night Vision and Elecro-Optics
6c. ADDRESS (City, State, and ZIP Code) Hovde Hall West Lafayette, IN 47907	7b. ADDRESS (City, State, and ZIP Code) Fort Belvoir, Virginia 22060
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Center for Night Vision and Electro-Optics 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER ~~DAAK20-85-C-0293~~ DAALØ1-85-C-Ø456
8c. ADDRESS (City, State, and ZIP Code) Fort Belvoir, Virginia 22060	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Research in Computer Vision for Autonomous Systems

12. PERSONAL AUTHOR(S)
Kak, Avi; Yoder, Mark; Andress, Keith; Blask, Steve; Underwood, Tom; Shamee, Bishara

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Progress	FROM June 88 TO Sept.88	15-Sept.-1988	163

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report addresses FLIR processing, LADAR processing and electronic terrain board modeling. In our discussion on FLIR processing, we have analyzed the issues of classifiability of FLIR features, computationally efficient algorithms for target segmentation, metrics, etc. The discussion on LADAR includes a comparison of a number of different approaches to the segmentation of target surfaces from range images, extraction of silhouettes at different ranges, and reasoning strategies for the recognition of targets and estimation of their aspects. Regarding electronic terrain board modeling, we have shown how the readily available wire-frame data for strategic targets can be converted into volumetric models utilizing the concepts of constructive solid geometry; we then show how from the resulting volumetric models it is possible to generate synthetic range images that are very similar to real LADAR images. We also show how sensor noise can be added to these synthetic images to make them even more realistic.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Avi Kak	22b. TELEPHONE (Include Area Code) 317-494-3551 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

Final Report

on

Research in Computer Vision for Autonomous Systems

Contract Number

~~DAAK20-85-C-0293~~

DAAL01-85-C-0456

Submitted to

Tim Williams
Center for Night Vision and Electro-Optics
Fort Belvoir, VA 22060-5677

Prepared by

Avi Kak
Mark Yoder
Keith Andress
Steve Biask
Tom Underwood
Bishara Shamee

Robot Vision Lab
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Project Period: January 1986 through December 1988

ACKNOWLEDGMENT

Our collective thinking on the current state of ATR methodologies and their limitations has been much influenced by ongoing dialogue with Dr. Lynn Garn.

The research and development work presented in this report would not have been possible without the technical support provided by Matt Carroll and Jeff Lewis.

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1



TABLE OF CONTENTS

Page

1. INTRODUCTION 1-1

2. FLIR PROCESSING 2-1

2.1. CLASSIFIABILITY OF FLIR FEATURES AND METRICS 2-1

2.1.1. Information Content of FLIR Features for Interclass Separation 2-1

2.1.2. Algorithm and Image Metrics 2-37

2.2. TWO FLIR SEGMENTERS 2-55

2.2.1. An EGT Based Segmenter for FLIR Data 2-55

2.2.2. FLIR Segmentation by Tree Traversal 2-73

2.2.3. Future Work 2-92

2.3. THE USE OF HIGH LEVEL REASONING TO IMPROVE THE CLASSIFICATION OF FLIR DATA 2-92

2.3.1. Data Structures for Symbolic Reasoning 2-93

2.3.2. Pixel Level Hierarchical Data Structures 2-99

2.3.3. Using A Global Map to Improve Edge Labeling 2-104

3. LASER RADAR RANGE DATA PROCESSING 3-1

3.1. DATA DESCRIPTIONS 3-1

3.1.1. Description of the 1986 A.P. Hill Laser Radar Data 3-1

3.1.2. Description of the 1987 A.P. Hill Laser Radar Data 3-1

3.2. EVALUATION OF LADAR IMAGES 3-38

3.2.1. Preliminary Results of Measuring Classifiability vs. Range in LADAR 3-38

3.2.2. Further Experimentation to Determine Classifiability vs. Range 3-40

3.2.3. Optimal Sampling of the Feature Space 3-45

3.2.4. Target Detection Using LADAR Data 3-63

3.3. LOW LEVEL PROCESSING OF LADAR DATA 3-102

3.3.1. Low Level LADAR Range Data Processing 3-106

3.3.2. New Low Level Processing Scheme 3-120

3.3.3. A Study of Five Laser Radar Range Data Segmenters 3-169

3.3.4. Results of Classifying the 1986 A.P. Hill Laser Range Data 3-193

3.4. HIGH LEVEL LADAR PROCESSING 3-203
3.4.1. Production Systems for Target Recognition 3-203
3.4.2. A Multi-Resolution Data Structure for Model-Based Geometric Reasoning 3-231

4. ELECTRONIC TERRAIN BOARD MODELING 4-1
4.1. ETBM via PADL 4-1
4.1.1. Geometric Models of Targets 4-1
4.1.2. Modeling Clutter 4-16
4.1.3. Noise Degradation of Synthetic Range Imagery 4-24
4.2. ETBM via TWIN 4-45
4.2.1. The TWIN Solid Modeling Package 4-52
4.2.2. The Electronic Field Test 4-57
4.2.3. Conversion of BRL Objects to TWIN Objects 4-70
4.2.4. Conclusions 4-74

5. REFERENCES 7-1

APPENDIX A A-1
APPENDIX B B-1
APPENDIX C C-1
APPENDIX D D-1
APPENDIX D E-1
APPENDIX F F-1

1. INTRODUCTION

This is the final report describing CNVEO sponsored research at Purdue Robot Vision Lab. This research program had three major goals: 1) Evaluation of the currently used image processing and pattern classification procedures for FLIR data; 2) Development of algorithms for LADAR imagery; and 3) Development of techniques for Electronic Terrain Board Modeling. We believe that these three goals are all vital to the advancement of ATR science, in the sense that their synergy will be reflected in most breakthroughs of the future.

As far as FLIR processing is concerned, our work centered on measuring the information content of FLIR features from the standpoint of interclass separability, and an evaluation of image and algorithm metrics. We also strived to improve the different aspects of FLIR processing. Since image segmentation is a critical step in this processing, we developed several different approaches to the problem. Our motivation was simply to see which techniques might be best suited for FLIR.

We believe that laser and millimeter radar range data, when available, would serve as an important adjunct to FLIR information for target classification. Our hope is that a synergistic integration of our current LADAR work with the developments made (and to be made) in FLIR would probably lead to the most effective procedures for ATR. It is entirely possible that in the ATR systems of the future, FLIR would play the role of focusing the attention of a processor at potential targets, a laser-based system could then examine the geometrical attributes of spatial data in the neighborhoods of these points to confirm or disconfirm the initial guesses and to also classify the detected targets.

As is well known, during the past few years many algorithms have been developed by us (and many other researchers in the country) for analyzing range maps using geometrical approaches. These algorithms attempt to first extract from a 3-D scene the constituent surfaces of objects, and then inferences about the objects are generated by reasoning over these surfaces and their relationships. Although it is entirely possible that the available resolution today would not permit the application of such geometrical approaches for target identification, we nevertheless think that such processing schemes should not be entirely ignored for LADAR imagery. There is always the chance that even today such schemes could pay off at close ranges, and then there is always the possibility that future LADAR systems would have much higher resolution – making geometrical approaches the method of choice.

To augment our development of new algorithms for LADAR, we also worked on Electronic Terrain Board Modeling. The ETBM work is based on the following rationale: Although it is desirable to use actual sensor data for the development and testing of ATR algorithms, unfortunately such data is not always readily available – owing to the fact that field tests are expensive and often postponed, not to mention the physical impossibility of carrying out experimentation on all conceivable configurations of targets, sensors, environmental conditions and terrain make-up. Anyone with a **validated** ETBM system should be able to run "electronic"

field tests and generate realistic data without the time and cost of a conventional field test.

The use of ETBM for modeling also provided another payoff: the development of a multi-resolution data structure for model-based geometric reasoning. This technique provides a natural means of degrading the more complex geometric models of targets (which we hope will be applicable in future high-resolution LADAR processing) to simpler silhouette-like models (which are applicable to the present resolution) by utilizing the relationship between discernible target detail and the range of the target.

In 1986, the first year of the effort, we largely fulfilled the requirements of the first main goal by creating the software for testing the classifiability of FLIR features. The software was based on Parzen estimation techniques. In 1987 and 1988, we concurrently carried out investigations into LADAR processing and ETBM. In 1987, we showed how by a combination of solid modeling for targets, fractal representation for topography, and productions for terrain features, we could construct synthetic images for the purpose of testing LADAR algorithms. Another highlight of 1987 was our initial attempt at the development of LADAR algorithms for recognizing objects on the basis of their geometrical attributes. Being preliminary in nature, the algorithms assumed that the sensor was located at a particular vantage point with respect to the target. Our current work makes such assumptions unnecessary and the targets can be in any pose in relation to the sensors. We also gained a better understanding of how the modeling work in ETBM can aid in the evaluation of LADAR algorithms. The fact that we are using target and terrain modeling for algorithm evaluation proves an important point we have made all along that to score future breakthroughs in the ATR science there must occur concomitant developments in LADAR, ETBM and FLIR.

We now describe our major accomplishments as they relate to our three main goals.

1.1 FLIR PROCESSING

The following are our achievements in working with FLIR imagery. Our goals in this area were to evaluate the current state of the art in FLIR data processing, to improve FLIR segmentation techniques, and to utilize high level reasoning to improve the classification of FLIR data.

1.1.1 Evaluation of FLIR Processing

Information Content of FLIR Features for Interclass Separation

We set out to measure information content of FLIR features from the standpoint of interclass separability by testing the following conjecture:

In a typical single static FLIR frame there does not exist enough information for

target classification. [It is important to note that excluded from our consideration are close range images or images taken under ideal environmental conditions.]

The software that we developed for this purpose includes modules for Parzen estimation techniques that are used for obtaining lower and upper bounds on classification errors. We believe that the upper bound thus obtained is tight and a good measure of the classification information contained in a given set of features. For comparison, we also have a module that for a given set of features computes the Bhattacharya distance between two classes as a measure of interclass separability. In order to conduct a full scale study on FLIR features, we compiled a superset of the FLIR features used by NVL contractors.

All the software that was developed in the first trimester for measuring classifiability of FLIR features was applied to NVL simulated terrain board data. Unfortunately, this data set could not be used to either prove or disprove our non-classifiability conjecture because of inadequate viewpoint spread in the acquired imagery. In the azimuthal plane, the viewpoints were 45° apart (which is too large a spread for our study); moreover, there was no variation with the elevation angle. Also, many of the images were taken from exactly the same viewpoint.

We also ran our classifiability software on the BRITT data. Since for some of the target types we did not have enough images for a single range, we had to group together images taken at different ranges to build a large enough sample size for statistically meaningful conclusions. On hand-picked target images of good quality and using the bounded-rectangle method for segmentation, our computed lower bound on classification error was 43% using gray scale features. (As we have explained, it is not possible to use all the features at the same time, because with the resulting high-dimensionality of the feature space we are also required to have a correspondingly large sample size. So for any classification study, the dimensionality of the selected feature set is limited by how much data is available for training the classifier.)

It is possible that we could reduce the 43% classification error estimate if we used a superior segmentation strategy, such as those derived from wire frame models. The error estimate would surely go up if we did not limit ourselves to good quality hand-picked images. At the time, it was too early to tell whether these results proved or disproved our non-classifiability conjecture. All these preliminary results are presented in Section 2.1.1.4.

We improved our software for testing the classifiability of FLIR features by incorporating in the software an advanced Parzen estimator for computing bounds on classification error. This advanced estimator, based on Prof. Fukunaga's recent work, makes decision thresholds also a function of the class covariances, as opposed to only their prior probabilities, which is the case with the more traditional Parzen estimators. Other upgrades to this software consisted of our adding more features to the set we had reported on before, and the use of a superior segmentation algorithm. **Using these changes on the same BRITT data resulted in the classification error for grey scale features to change from a lower bound of 2.2% to a lower bound of 12.7%.** These results are reported in Section 2.1.1.4.

Algorithm and Image Metrics

We set out to investigate algorithm and image metrics for ATR characterization. The conclusion that we have arrived at with regard to image metrics is that the independence of metrics is probably a necessary, but definitely not a sufficient, criterion for their selection. As stated in Section 2.1.2, we believe that the dependence of a metric on different variables must be discovered by theoretically analyzing algorithms for their performance, as opposed to by heuristic specification. The point being made here is that if, say, our goal is target detection and we wish to characterize the complexity of an image with regard to target detection, we should theoretically analyze the algorithm used for the purpose and thus discover the form of the metric. Such a metric would be both task oriented and algorithm dependent, which is how it should be. There can be no absolute measures of image complexity even for specific processes such as target detection. Since it is possible to use different algorithms for detection, the measure of complexity must take into account the nature of the algorithm.

While in the first trimester we showed that TIR^2 was not a reliable metric for target detection, in the second trimester we have established **that $TBIR^2$ also suffers from serious deficiencies when it comes to measuring image complexity with regard to target segmentation.** These results are presented in Section 2.1.2.

As a first step in our attempts to come up with new metrics, we have in Section 2.1.2.2 proposed a method that might be able to measure the complexity of images from the standpoint of segmentation using thresholding. Although it has fared better than the $TBIR^2$ metric in assessing the difficulty of segmentation on the images we have tested it on, we are not yet ready to give it the label of a new metric as many questions that it has raised remain unanswered.

1.1.2 Two FLIR Segmenters

EGT

Section 2.2.1 presents results obtained with our new edge guided segmenter for FLIR data. This segmenter is much simpler than the Hughes segmenter, yet its performance is comparable, at least on the images that both were tested on. Our segmentation algorithm is based on the fact that in the traditional histogram based procedures the hardest problem is the selection of a good threshold. Our contention is that in the vicinity of the valley where a good threshold might be placed to separate the target from the background, the shape of the histogram is distorted by the boundary pixels. That is because the boundary pixels have gray levels that are intermediate between the target and the background. Therefore, in our new segmentation algorithm, we delete the contributions made to the histogram by boundary pixels.

Tree Traversal

Our long term aim is still to integrate with LADAR the FLIR algorithms we have developed and reported on previously. However, such an integration will only come about after we have stabilized the set of algorithms for LADAR processing. In the meantime, we have continued to improve the different aspects of FLIR processing. Since image segmentation is a critical step in this processing, we implemented a totally different approach to this problem – segmentation by tree traversal. Our motivation was simply to compare three or four different approaches to segmentation and to see what techniques might be best suited for FLIR. This work is reported in Section 2.2.2.

1.1.3 The Use of High Level Reasoning to Improve the Classification of FLIR Data

Because of the emphasis that we placed on the feature classifiability study, our progress on the application of hierarchical vision techniques to FLIR images has been less than what was originally anticipated. We have conceived of methods to carry out hierarchical reasoning, but haven't fully implemented any particular strategies.

Data Structures for Symbolic Reasoning

We took an important first step toward the eventual development of more sophisticated algorithms for ATR. This consisted of devising procedures for converting numerical pixel level information in an image into a symbolic map. In Section 2.3.1, we have shown symbolic data structures that will be used for associating pixels with the lowest level symbolic features, such as lines, edges and blobs. **It is important that efficient data structures be put into place; since if that is not done, simple questions like what edges a particular pixel might belong to can lead to exhaustive and grossly inefficient searches in an image.** An important side benefit of a good data structure is that it can reduce some types of elementary symbolic reasoning to simple operations such as a table look-up.

Pixel Level Hierarchical Data Structures

In hierarchical vision, we conducted some studies on the loss of classifiable information as we go up a pyramid representation of a FLIR image. The pyramid representation was obtained by simple 4×4 averaging. **To our surprise, we were for the most part unable to see any appreciable change in classification error as we moved up the pyramid. To us this means that at this time there is probably a mismatch between the resolution implied by the matrix sizes used for FLIR images and the intrinsic FLIR sensor resolution.** This material

appears in Section 2.3.2.

Global Map to Improve Edge Labeling

Data Structures for symbolic reasoning have been proposed. Under development in the Robot Vision Lab is a general purpose software tool for integrating map knowledge with images. This system, called PSEIKI (a Production System Environment for Integrating Knowledge with Images), is briefly described in Section 2.3.3. The symbolic reasoning structures previously developed will be used in conjunction with this system for applying spatial reasoning techniques to FLIR images.

1.2 Laser Radar Range Data Processing

Data Descriptions

Our second major area of accomplishment was getting a handle on the various aspects of the A. P. Hill LADAR data. We now understand the nature of the noise in the AM and the FM part of the data and, since the AM noise has much lower variance than the FM noise, we can now separate from the high-noise composite data supplied to us a version whose noise-properties are substantially the same as that of the AM part alone.

In Section 3.3, we have reported on our preliminary processing of the A. P. Hill data. This data, which is a composite of absolute range information through the FM channel and relative range information obtained through the AM channel, has some peculiar noise characteristics since the noise variances of the two channels are very different. While the variance of the FM channel is around 9 meters – this happens to be close to half of the ambiguity interval of 18.75 meters – the variance of the AM channel is only about a meter. The composite data therefore suffers from the worst of the two variances. Section 3.3 shows how, for the purpose of target recognition, it might be possible to extract from the composite data a range map whose noise properties are as good as that of the AM channel.* Another purpose of that section is to show that in this trimester definite progress was made by us in gaining a full understanding of the A. P. Hill data.

1.2.2 Segmentation of LADAR Images

* Since Section 3.3 was first written, the AM-only data has been made available to us, therefore lessening the need to extract the AM data from the composite data.

Classifiability vs. Range Experiments

Our preliminary work on the classification of LADAR imagery was extended to include the effect of range. The rationale for the study was that a most important characteristic of any LADAR algorithm is the nature of degradation of its performance with increasing range. Since only noiseless synthetic data was used in the preliminary work, the classification accuracies we obtained were unrealistically high. Noisy synthetic data – which is a more accurate representation of the real world case – was later processed through the same software to obtain more meaningful results. The range dependence of classification accuracy is reported in Section 3.2.2.

Optimal Sampling of the Feature Space

Since at large distances from a LADAR sensor it is unlikely that geometrical features, such as relationships between different surfaces, would be discernible, target recognition would have to depend upon silhouette information. A silhouette based recognition strategy is made complicated by the fact that silhouette features vary considerably over the range of all possible viewpoints. To get around this difficulty, the usual practice is to represent the space of all viewpoints by a small set of distinguished viewpoints such that each viewpoint in the small set corresponds to a Gaussian distribution for the silhouette parameters of interest and that the Gaussian distributions for all the viewpoints are as different as possible. This selection of distinguished viewpoints has hitherto been done by a human on the basis of his intuitive understanding of the dependence of silhouette features on viewpoints.

We made a first attempt at automatic selection of these distinguished viewpoints by first computing the first order probability density associated with a silhouette feature of interest. This density was computed using a large number of silhouettes uniformly sampled around the object. We then sought a small number of silhouettes that would allow us to compute the same density function with minimum error. An alternative to this method is to use clustering in the silhouette space. For that purpose, we conducted a survey of the various automatic clustering procedures that are available. This brief survey is included in Section 3.2.3.7. **An immediate practical usefulness of this work is that it would allow us to greatly reduce the number of training images needed to train a silhouette based classifier.** This work is reported in Section 3.2.3.

Target Detection Experiments

We developed an algorithm for target detection from single LADAR lines. This detection algorithm, described in Section 3.2.4, is more sophisticated than, and subsumes, simple schemes

that base decisions on the presence of constant range lines between two range discontinuity points. We say our algorithm subsumes simple approaches because, with appropriate settings in the software, the algorithm can be made to base detection decisions on mere presence of constant range lines or, for that matter, even sloping range lines for targets whose flat surfaces are at angles other than 90° with respect to the angle of look. Before the detection algorithm can be used, it must be trained on sample data; the statistical similarity between the backgrounds shown to the detector in the training phase and the background in the test phase then becomes an important determinant of detector performance. In Section 3.2.4, we have also shown that the detector performance can be improved by the enforcement of constraints like the minimum number of detected pixels that must be contiguous for a target to be declared present.

We then generalized the single-line LADAR target detection algorithm to the case of multi-line input. This work, discussed in Section 3.2.4, is aimed at examining the premise that it should be possible to improve the detector performance by combining data from different LADAR scan lines. We have examined two schemes for the multi-line case: In the first scheme if a decision is based on L adjacent lines and M pixels from each line, we simply treat the detection problem as testing a binary hypothesis in an MxL dimensional space. In the second scheme, the L lines for the multi-line case are considered to constitute L independent detectors, the subdetectors for each line working in exactly in the same manner as the single line detector described in Section 3.2.4. Using both these methods our conclusion is that, from the standpoint of enhancing target detection and minimizing false alarms, it is better, if possible, to have all the MxL samples in a single line, as opposed to being distributed over L lines. We have demonstrated that detectors utilizing a few LADAR lines may not be practical because of excessively large false alarm rates associated with them.

1.2.3 Low Level Processing of LADAR

In Section 3.4.1, we will describe in detail the low level processing that is required before any geometrical reasoning strategies can be invoked.

Edge Detection & Surface Labeling

In the first trimester of 1988, we discovered that the traditional approach to LADAR segmentation, which starts with the extraction of jump and curvature edges, is too sensitive to the high variance noise and the large number of dropouts that characterize real LADAR data. Therefore, in the second trimester of 1988 we focussed on the development of a new segmentation algorithm, which, we are happy to report, is indeed better. To compare the performance of the new segmentation algorithm with the old, we had to develop criteria for judging the quality of a segmentation.

Region Growing Approach

To carry out geometric reasoning over LADAR data, one must first extract the individual surfaces of the visible part of the target. In our previous work, this was done with the help of edge detection algorithms, the edges being mostly range jump discontinuities, roof-type edges, and curvature maximas. We started out with edge detection for low level processing because that is a common thing to do in industrial 3D robot vision and we felt that we should first try the already proven approaches. Our experience with LADAR data has shown that edge detection may not be the best approach for LADAR especially when such data is characterized by high variance noise and frequent dropouts. We therefore implemented a region growing approach to identifying the target surfaces. We report our new LADAR segmentation procedure in Section 3.3.2. It owes its superior performance in part to the fact that, prior to the computation of surface normals, we fit 2-D B-splines to the range map. As a result, we obtain bicubic approximations to object surfaces that are guaranteed to be continuous in first- and second-order derivatives. This leads to a great deal of noise suppression and results in smooth range maps and high quality segmentations. In Section 3.3.2.3, we show the results on synthetic data. Results on the real A. P. Hill LADAR data are shown in Section 3.3.2.4. In Section 3.3.2.5, we then present objective measures for testing the quality of a segmentation and report on an algorithm evaluation experiment in Section 3.3.2.6.

Study of Five LADAR Segmenters

In Section 3.3.3 we have presented a comparison of the following five different segmentation algorithms for LADAR imagery:

1. Planar-Patch Fitting Error
2. Variance-Based using 3x3 and 5x5 Windows
3. Rockwell Algorithm
4. Nettleton method using 3x3 and 5x5 Windows
5. Variance-Less-One using 3x3 Window

The last algorithm represents a heuristic fix for the problems caused by noise spikes in the other algorithms. Our comparison illustrates the sensitivity of each algorithm to the thresholds selected for segmenting out the object from the background. These silhouette-based algorithms are applicable to the present low-resolution imagery and images with distant targets, where geometric approaches are not appropriate due to lack of sufficient detail.

Classification Results

Section 3.3.4 reports on some preliminary work on the classification of targets in LADAR imagery. Since the silhouettes of the segmented outputs from LADAR data seem to be of good quality in many cases, in our initial classification work we have chosen to use features identical to those extracted from FLIR silhouettes. Although the classification results thus appear to be promising – the accuracy achieved was over 90% on one set of data – we want to impress upon the reader that ultimately LADAR classification must exploit the geometrical information contained in range maps. What we are trying to say is that given objects in arbitrary orientations with respect to the sensor, a classification strategy would work best if it is based on matching **in a relational sense** the geometrical features extracted from a LADAR image with the geometrical features extracted from object models, such matching being graph-theoretic in nature.

1.2.4 High Level Processing of LADAR

One of our major accomplishments was the development of a set of algorithms for recognizing targets in LADAR imagery on the basis of geometrical features. We are, of course, aware of the fact that with the sensor resolution available at this time, geometrical features will not be discernible for targets farther away than, say, a kilometer. However, as with most technologies, we can expect the sensor resolution to improve over the next few years, especially since there appear to be no fundamental reasons to preclude that. Therefore, the algorithms we are developing, although applicable currently to close-range targets, are really aimed more for the future. The initial set of algorithms discussed in this report is only meant to be an educational exercise to help us formulate our ideas on how one should reason over geometrical features; therefore, these algorithms only analyze range images from a single viewpoint.

Production Systems for Target Recognition

In Section 3.4.1, we have reported on some novel reasoning strategies for drawing inferences about a target using geometrical features derived from LADAR data. In one of these novel strategies, the system performs default reasoning, which can best be explained in the following manner: Let's say that from a given viewpoint a LADAR sensor is able to see N surfaces. However, because of data acquisition and processing limitations, the target recognition program is only able to discern M surfaces, where $M < N$. If the geometrical characteristics of these M surfaces and their spatial inter-relationships are the same as those of some M of the N surfaces expected to be seen on the target, we want our computer program to declare the target present, albeit with a reduced probability. As we have shown, the computation required for this kind of a recognition strategy is vastly simplified if we assume defaults for the missing object surfaces. We will show how these defaults are automatically generated in two different

computational paradigms, one based on Prolog and the other on OPS.

Multi-Resolution Data Structure for Model-Based Geometric Reasoning

Availability of LADAR data has opened the door to recognition of objects by geometric reasoning. However, one must first contend with the issue of a target being in any of an infinity of possible poses with respect to the sensor. Researchers have advanced the notion of aspect graphs to deal with this difficulty. The nodes of an aspect graph represent the clustering of all viewpoints into a small number on the basis of topological equivalences. The idea behind the use of aspect graphs is that given a target at an unknown orientation with respect to the sensor, we should first determine the aspect graph node to which the target data corresponds; we should then invoke node-specific strategies for a more precise determination of the orientation with respect to the sensor. In the context of LADAR, since not all geometrical features are equally visible from different ranges, we must use a hierarchy of aspect graphs instead of using a single aspect graph for a target. This has led to the notion of a multi-resolution aspect graph reported in Section 3.4.2 By using the TWIN solid modeling system, we are now able to generate multi-resolution aspect graphs for targets.

Another major accomplishment concerns the problem of how to incorporate the diminished sensor resolution into a target model as the model is moved away from the sensor. This is a hard problem, not generally amenable to analytical solution in its full three dimensional form. We have shown results from a scheme that deleted object surfaces on the basis of their visible areas as the object was moved away. Later we felt that since much recognition is driven by edge information, and since the ability of a sensor to discern an edge is a function of the dihedral angle at the edge, and, further, since the measurement of a dihedral angle suffers as the object is moved away, we needed to capture this effect in our model degradation process. An edge-deletion based scheme for graceful degradation of a target model as the model is moved away from the sensor is reported in Section 3.4.2, where we have also shown results on a couple of different targets. As a target is moved farther away from a sensor, its edges are removed selectively on the basis of one or both of edge-length and dihedral angle.

1.3 Electronic Terrain Board Modeling

One of our most significant accomplishments was the development of the software for Electronic Terrain Board Modeling. We believe that ultimately any such software will have to have the following features: 1) geometric models of targets, 2) models of background and foreground clutter, 3) models of environmental conditions, and 4) validation routines for testing the integrity of the simulated data. We have succeeded in developing the software for all four of these items. We are able to convert wire-frame models into solid models using PADL

descriptions. The PADL solid descriptions are then used to generate synthetic range maps. We model the background terrain in 3-D by using fractals. Finally, a production system is used for modeling clutter such as trees. One advantage of our procedure for generating trees is that every tree can be different, which is unlike some of the other terrain modeling systems. Finally, to the resulting synthetic 3-D imagery we add noise with the same drop-off statistics as the real LADAR data. All this work is reported in Section 4.1.

Using the PADL based system, we were able to create ground-truth-images of the 1987 A.P. Hill field test. This was done by using the target location information in the headers of the field test images as input to our ETBM. The ETBM placed the target models as the real targets were placed in the field test and then generated the corresponding image. These images proved to be very useful in helping us understand the location and relative position of the targets in the real images. Another advance was that we acquired the BRL solid-modeler. We attempted to integrate it with the original ETBM software. It appeared at the time that the integration of the BRL software with the rest of our system was a necessary pre-requisite before we could show more sophisticated synthetic imagery. The PADL solid modeling system we had used was unable to handle a large number of surfaces that we need for our ETBM modeling. We had hoped this difficulty would be alleviated by the BRL software.

In Section 4.2, we show that with the TWIN solid modeler we were able to generate more complex ETBM imagery. We also discuss the conversion of BRL models to TWIN and talk about the difficulties we ran into in this regard; these difficulties owe their origins to the tolerance problem in solid modeling.

2. FLIR PROCESSING

In this chapter, we will discuss our work on FLIR. Much of work in FLIR has been motivated by our concerns about the low information content of features extracted from FLIR data, especially when targets and terrain are more than a kilometer from the sensor and when the atmospheric conditions are less than ideal. Notwithstanding this concern, it remains that FLIR being passive is an excellent sensor for monitoring battlefield activity. The overall goal for the research community therefore is how to best exploit this potential of FLIR with the available discriminatory power of the current sensors. In the work we have reported in this chapter, we will focus on the classifiability aspects of FLIR features, on the segmentation procedures that appear to work well without being excessively demanding on computational resources, and on data structures that facilitate high level reasoning in such images.

2.1. CLASSIFIABILITY OF FLIR FEATURES AND METRICS

In 1987 we examined a large amount of FLIR data taken under different conditions. Since even the best of data did not inspire confidence in us, we decided to set up a conjecture about the information content of FLIR features; we of course maintained an open mind about the truthfulness or falsity of the conjecture. As the following section demonstrates. the conjecture was tested by using techniques based on Parzen estimation theory, which is capable of yielding, in a non-parametric manner, lower and upper bounds on the classification error. Although, on account of dimensionality issues we were not able to test the conjecture in the full feature space, we did examine the appropriate subspaces and arrived at the conclusion that was not sufficient information contained in the features used by most contractors at this time to warrant hardware implementations of FLIR-based ATR.

2.1.1. Information Content of Flir Features for Interclass Separation

As mentioned in the Introduction, our basic aim here is to test the following conjecture

> In a typical single static FLIR image, there does not exist enough information for accurate target classification.

The conditions on the conjecture are that the single frame should be typical of FLIR imagery (likely to be recorded under actual conditions) and definitely excluded are close-range FLIR images and images recorded under ideal environmental conditions.

We state this conjecture because typical FLIR imagery is characterized by low resolution; another reason for our conjecture is the strong dependence of FLIR target signatures on environmental conditions. These factors cause us to question the viability of statistical classification methods applied to features extracted from single frames.

By processing single static frames, we mean that no context-based, time-sequential, expectation-driven or knowledge-based processing is performed.

To test the conjecture we are compiling a superset of features that are used by the NVL contractors and have developed a software package that for any given set of features computes upper and lower bounds on their interclass separability. This is done via the computation of *probability of classification error* through Parzen and advanced Parzen estimation techniques. Obviously, how much classifiability information is contained in a set of features can be readily determined by calculating for a given set of features the upper and lower bounds on the *probability of classification error*.

As is well known, a common technique used for target classification of FLIR images is to segment the image into regions and extract various features (such as mean gray value, region height, and region width) for each region and based on these features statistically classify the region as one of the targets. By using this procedure on known data, one can determine the extent of classifiability information contained in a set of features. The basic steps of our classification study are:

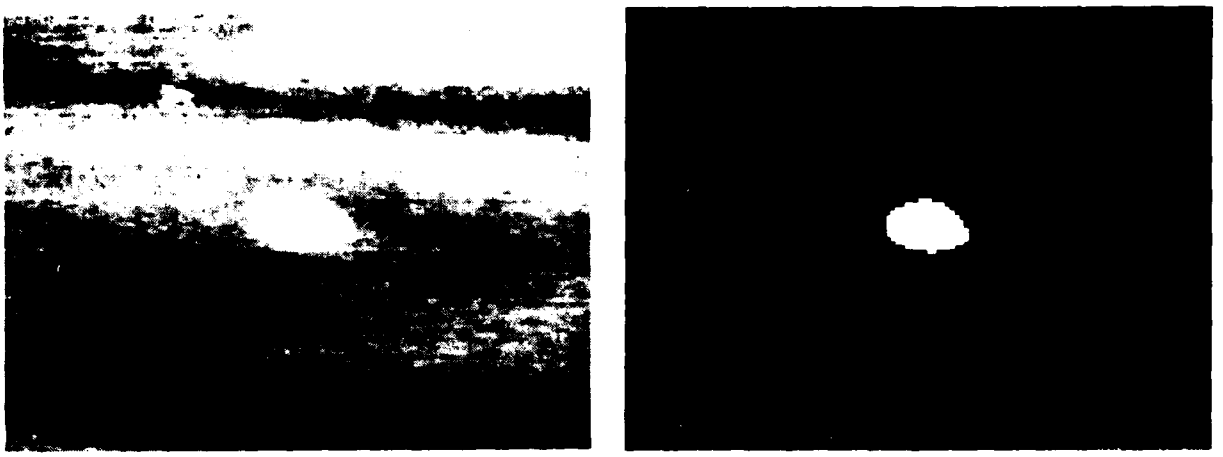
1. Segment FLIR images so that each region contains a single target.
2. Extract a superset of the features that have been revealed to us by NVL contractors.
3. Use the Parzen and advanced Parzen estimates to compute the underlying density functions for the different classes.
4. The extent of overlap between the density functions corresponding to different classes is a measure of interclass separability. Compute upper and lower bounds on interclass separability.

The next section discusses the automatic segmenter we used. Section 2.1.1.2 presents in detail the features used in the experiments. A brief review of the Bayesian decision process and Bayesian error estimation via the Parzen and advanced Parzen density estimates follow in Section 2.1.1.3. Preliminary results on interclass separability based on the Eglin Turntable Data, NVL terrain board data, and the BRITT data are given in Section 2.1.1.4. Finally, future experiments are discussed in Section 2.1.1.5.

2.1.1.1. Segmentation

In order to obtain better target silhouettes for more accurate feature calculation and therefore more meaningful classification results, we implemented the likelihood segmenter described in the Bandwidth Reduction and Intelligent Target Tracking (BRITT) Phase One Final Report by Hughes Aircraft Company's Electro-Optical & Data Systems Group [Hughes84]. We present here (see Figure 2.1.1) the target silhouettes produced by the segmenter for images with various qualities, but due to the proprietary nature of the Hughes report we will not discuss the segmentation algorithm itself.

The input images we used are from the BRITT Target Recognizer Classifier Training data set. The best segmentation output was produced for images similar to those in Figure 2.1.1 (a),



(a) britt040 : type=APC, range=5km, aspect=45deg
small "hot" target



(b) britt238 : type=APC, range=3.5km, aspect=270deg
small "cool" target

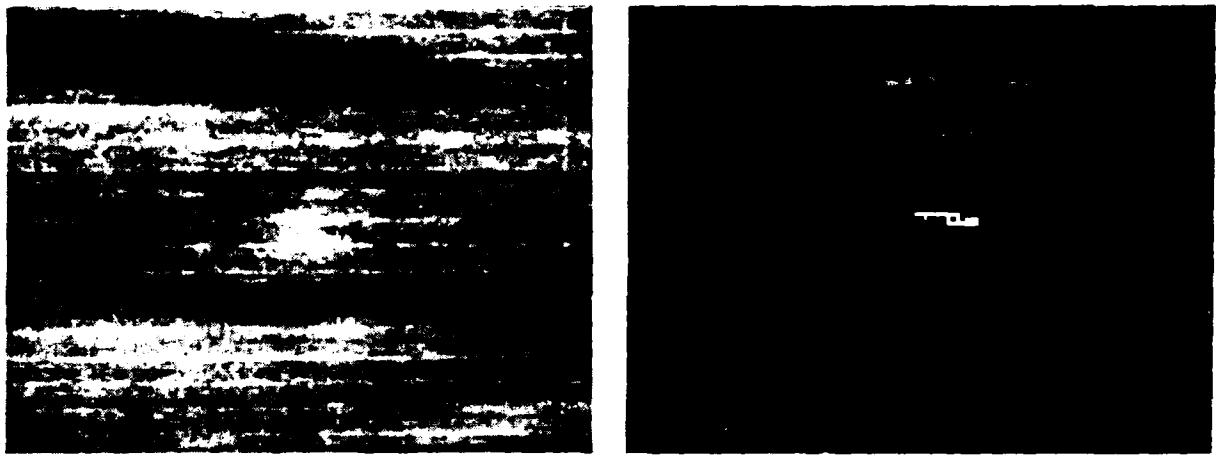


(c) britt137 : type=TRUCK, range=2.5km, aspect=90deg
large "hot" target

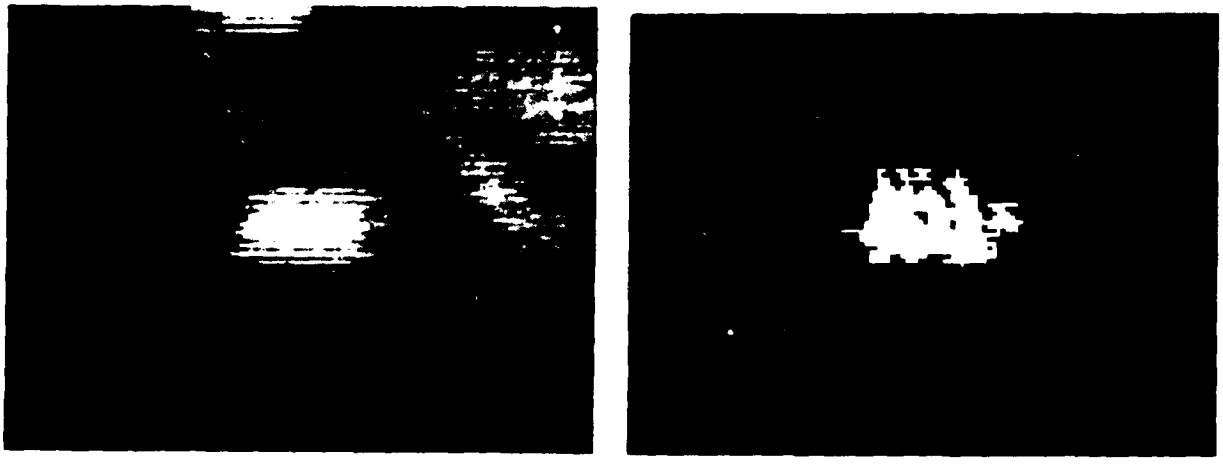
Figure 2.1.1 Sample segmenter output.



(d) britt029 : type=TRUCK, range=2.5km, aspect=180deg
large "cool" target



(e) britt003 : type=APC, range=5km, aspect=180deg
no visible target where there should be one



(f) britt515 : type=APC, range=5km, aspect=270deg
noisy target

Figure 2.1.1 Continued.



(g) britt347 : type=TANK, range=2.5km, aspect=45deg
noisy background



(h) britt277 : type=TANK, range=2.5km, aspect=270deg
target with extreme hot & cold spots

Figure 2.1.1 Continued.

(b), and (c).

2.1.1.2. Feature Extraction

Feature extraction is a critical step in ATR because the selection of features greatly influences the ability to classify. We are currently using a superset of features used by other NVL contractors.

The following description of the features assumes that $G(x,y)$ is the grey level value of the image containing the target, and T is the target region in the xy plane.

1. Mean grey value of the target

$$\bar{g} = \frac{1}{|T|} \sum_{(x,y)\epsilon T} G(x,y)$$

2. Standard deviation of the target grey value

$$\sigma = \sqrt{\frac{1}{|T|} \sum_{(x,y)\epsilon T} G^2(x,y) - \bar{g}^2}$$

3. Target height

$$h_T = \max(x_i) - \min(x_i) \qquad \textit{for all } x_i \epsilon T$$

4. Target width

$$w_T = \max(y_i) - \min(y_i) \qquad \textit{for all } y_i \epsilon T$$

5. Minimum grey value of the target

$$m = \min\{G(x,y)\} \qquad \textit{for all } (x,y)\epsilon T$$

6. Maximum grey value of the target

$$M = \max\{G(x,y)\} \qquad \textit{for all } (x,y)\epsilon T$$

7. Area of the target

$$A = |\{(x,y): (x,y)\,\epsilon T\}|$$

8. Second and third order moment invariants
9. Maitra's beta functions (six of them)
10. Height to width ratio of the target

$$\rho_{hw} = \frac{h_T}{w_T}$$

11. Perimeter to width ratio of the target

$$\rho_{pw} = \frac{perimeter}{w_T}$$

12. Rectangularity measure

$$\eta_{02}\eta_{20} - \eta_{11}^2$$

where η_{02}, η_{20}, η_{11} are normalized central moments.

13. Square of width to height

$$(w_T/h_T)^2$$

14. Normalized contrast

$$(\bar{g} - \bar{b})/\sigma$$

where $\bar{b}$ is the background average.

15. Range

16. Depression angle

$$\alpha = \sin^{-1}(elevation \ / \ range)$$

17. Square of the perimeter over the area

$$(perimeter)^2 \ / \ A$$

18. Square of the height over the area

$$h_T^2/A$$

19. Height times range squared

$$(h_T * Range)^2$$

20. Area times Range squared

$$A * Range^2$$

21. The sign of the normalized contrast

$$sign(x) \equiv \begin{cases} +1 \text{ if } (\bar{g}-\bar{b})/\sigma \geq 0 \\ -1 \text{ if } (\bar{g}-\bar{b})/\sigma < 0 \end{cases}$$

We rederived all of the features used by Martin Marietta to check for correctness. Detailed derivations of the features are presented in Appendix C. These derivations uncovered an inconsistency in one of Hu's invariants. This error was traced back to a typographical error in Hu's

original paper [Hu62]. Further study showed that this error had been reported in [Ma79]. The feature in error was η_{pq} which was given as:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{p+q}{2}}}$$

but should have been:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{p+q}{2}+1}}$$

Unfortunately η_{pq} appears in all of Hu's invariant moments, so none of the invariant moments computed by Martin Marietta could have been correct.

Maitra's invariants (derived in Appendix C) are invariant under scale, rotation, translations, and *illumination*. These invariants were used in [MM84], however when ϕ_5 is negative, β_4 is undefined. We have found that the following new definition for β_4 is more stable:

$$\beta_4 = \frac{\phi_5}{\phi_4^2}$$

2.1.1.3. Classification of FLIR Imagery

The classification problem, as it applies to FLIR imagery can be stated as follows: Once an object is detected in an image, it is usually desired to determine what type of object it is. To do this, a vector of features, X, is extracted from the object (e.g. mean grey value, various moments, etc.). The vector can then be used to estimate the probability that the object belongs to any given class. Assuming that the object can only be from one of two classes, ω_1 or ω_2, the following decision rule is applied. If the probability that the object is from class 1 is greater then the probability that the object is from class 2, the object is assigned class 1 membership; otherwise, it is assigned to class 2. This can be stated mathematically as

$$X \rightarrow \begin{cases} \omega_1 & P(X \in \omega_1 \mid X) \geq P(X \in \omega_2 \mid X) \\ \omega_2 & P(X \in \omega_2 \mid X) > P(X \in \omega_1 \mid X) \end{cases}$$

where ω_i stands for class i, and $X \rightarrow \omega_i$ indicates that X is classified to class i. The decision rule (for the one dimensional case) is shown graphically in Figure 2.1.2.

Bayes' theorem shows a way to find the *a-posteriori* probability of $P(X \in \omega_i \mid X)$ given the *a-priori* conditional probability $P(X \mid X \in \omega_i)$. Bayes' theorem can be expressed as

$$P(X \in \omega_i \mid X) = \frac{P(X \mid X \in \omega_i) \cdot P(X \in \omega_i)}{p(X)}$$

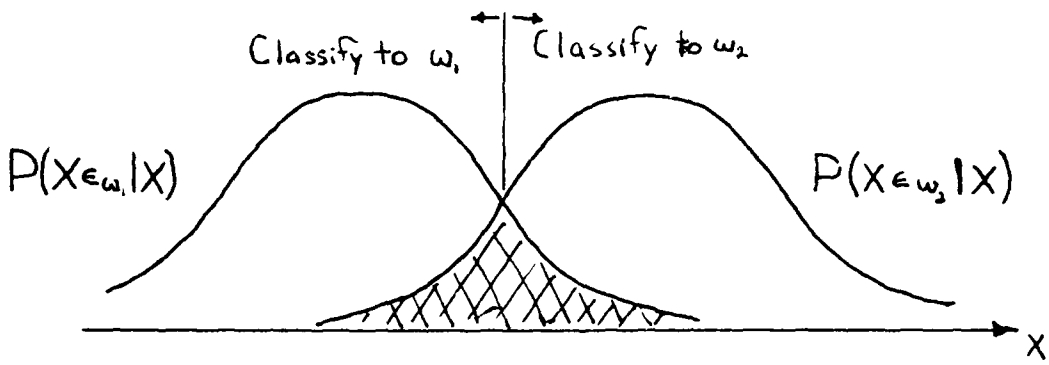


Figure 2.1.2 Classification Decision Rule

If we define the likelihood ratio as

$$l(X) = \frac{P(X \mid X \in \omega_1)}{P(X \mid X \in \omega_2)}$$

the above decision rule can be expressed as

$$X \rightarrow \begin{cases} \omega_1 & -\ln(l(X)) < t \\ \omega_2 & -\ln(l(X)) > t \end{cases}$$

where the decision threshold, t, is defined as

$$t = -ln\frac{P(x \in \omega_1)}{P(x \in \omega_2)} \tag{2.2.1}$$

It can be shown that the probability of error for the Bayesian decision rule is the area of the shaded portion of the graph in Figure 2.1.2. The decision rule can be extended to m classes, $\omega_1, \omega_2, ..., \omega_m$ which is expressed as:

$$X \rightarrow \left\{ \omega_i \;\middle|\; \ln(P(X \in \omega_i \mid X)) \geq \ln(P(X \in \omega_j \mid X)) \right\} \quad j = 1,2,...,m \tag{2.1.2}$$

if t = 0.

2.1.1.3.1. The Parzen Density Estimate

If the distribution of the features, X, is known or can be determined parametrically, the problem of finding the classification error is easy. Unfortunately, in this case we must estimate the density function. Given N samples, $X_1, X_2, ..., X_N$, from a density function, the Parzen estimate of a density function can be defined as

$$\hat{p}(X) = \frac{1}{N}\sum_{i=1}^{N}(1/h^n)k((X-X_i)/h)$$

where $k(\cdot)$ is called the *kernel* of the estimate and should be a nonnegative Borel measurable function satisfying $\int k(X)dX = 1$. An example of a one dimensional Parzen estimate is shown in Figure 2.1.3. Each sample X_i is shown with its corresponding kernel. These kernels are summed to give the estimate $\hat{p}(X)$. It should be noted that the only parameters needed for the Parzen density estimate are the form and size of the kernel function.

The Parzen density can be used to estimate the probability of error in the following manner. If N_j is the number of samples of class j and N is the total number of samples, $P(X \in \omega_j)$ can be approximated with N_j/N. If X_{ij} is the ith sample from the jth class, we can let

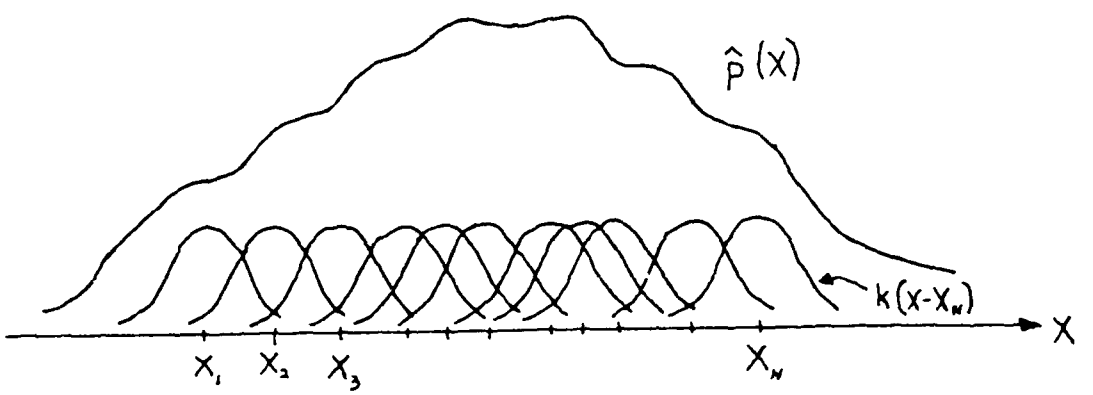


Figure 2.1.3 The Parzen Density Estimate

$$\begin{aligned}\hat{p}_j(X) &= P(X \in \omega_j \mid X) \\ &= \frac{P(X \mid X \in \omega_i) \cdot P(X \in \omega_i)}{p(X)} \\ &= \frac{1}{p(X)} \cdot \frac{P(X \in \omega_j)}{N_j} \sum_{i=1}^{N_j} (1/h^n) k_j((X - X_{ij})/h) \\ &\approx \frac{1}{p(X)} \cdot \frac{N_j}{N} \cdot \frac{1}{N_j} \sum_{i=1}^{N_j} (1/h^n) k_j((X - X_{ij})/h)\end{aligned}$$

removing terms that will appear in all classes yields $\hat{\rho}_j(X)$, which is defined by

$$\hat{\rho}_j(X) = \sum_{i=1}^{N_j} k_j((X - X_{ij})/h) \quad j = 1,2,...,m \tag{2.1.3}$$

If we substitute Equation (2.1.3) into Equation (2.1.2) we can classify

$$X \rightarrow \left\{ \omega_l \mid \hat{\rho}_l(X) \geq \hat{\rho}_i(X) \right\} \quad i = 1,2,...,m$$

If all samples, X_{ij}, are classified in this manner and we let N_{error} be the count of misclassified samples, the probability of error can be approximated by

$$P_{error} = \frac{N_{error}}{N}$$

It should be noted that the contribution of the sample itself is taken into account when the probability that it belongs to its *true* class is being computed; this corresponds to the case where the classifier is designed and tested using the same data set. This produces the so called *resubstitution error* [Fu72]. It can be shown that this gives a lower bound on the true probability of error.

To get an upper bound on the probability of error, one can use the *leaving one out* method. Basically, in this method we ignore the effect a sample has on the density estimate of its true class, and then the probability of error is computed in the same manner as before. When we leave out a sample, the equation for estimating the sample's class density becomes

$$\hat{\rho}_j(X) = \frac{N_j}{N_j - 1} \left\{ \sum_{i=1}^{N_j} k_j((X - X_{ij})/h) - k_j(0) \right\} \quad \textit{where } X \in \omega_j$$

2.1.1.3.2. Estimation of Classification Error in FLIR Data

In our classification experiments, our long term aim is to find the error when classifying between four objects: a tank, a truck, a jeep, and clutter. The Parzen estimation procedure will be used to find the classification error of a Bayesian scheme using commonly used features from FLIR images (real and simulated). Each class of objects will be split into two clusters (subclasses): front/rear view and side view. This will be done in an attempt to assure the classes have a Gaussian distribution. If it is determined that resulting clusters are still not Gaussianly distributed, it may be necessary to create three clusters per class: front, rear and side views.

Unfortunately, to produce statistically meaningful results, the number of samples per cluster must be at least an order of magnitude greater then the dimensionality of the data [KaLa83]. This is needed to accurately calculate the sample covariance matrices. Because we are going to have two or three clusters per class, if we use approximately ten features we need to have 200-300 samples of each object to accurately determine the Bayesian probability of error.

As was mentioned in the previous section, the only parameters needed to use the Parzen estimate are the kernel size and shape. We will use a Gaussian kernel of variable size. The Gaussian kernel is used frequently in Parzen density estimation, and can be shown to be optimal for classification error estimation if the classes have Gaussian densities. The Gaussian kernel can be expressed as

$$k_j(X-Y) = \frac{1}{\pi^{\frac{n}{2}} \cdot \sqrt{|\Sigma_j|}} \exp\left[- \frac{(X-Y)^T \Sigma_j^{-1} (X-Y)}{h^2} \right]$$

where n is the dimensionality (i.e. the number of features), h is the kernel size, Σ_j is the covariance matrix of class j, and $(X-Y)^T$ is $X-Y$ transposed. Removing terms that will appear in all the kernel functions yields the final kernel function

$$k_j(X-Y) = \frac{1}{\sqrt{|\Sigma_j|}} \exp\left[- \frac{(X-Y)^T \Sigma_j^{-1} (X-Y)}{h^2} \right]$$

The kernel size parameter, h, will be varied to give the lowest leaving-one-out error; this gives the optimal kernel size.

2.1.1.3.3. Advanced Parzen Error Estimation Techniques

The Parzen error estimation technique described up to this point has generally been considered state-of-the-art. However, recent work at Purdue has been performed focusing on improving the results produced by the Parzen error estimation procedure [FuHu87]. The majority of this work has been centered on changing the decision threshold from the value defined in

Equation 2.2.1. The incentive behind this work is the realization that, under certain conditions, the expected value of the estimated density with respect to X is

$$E\{\hat{p}_i\} = p_i(X) * (1/h^n)k(X/h) \tag{2.1.4}$$

where * represents convolution in R^n. This equation is valid if the covariance of the kernel function is set equal to the covariance of the data samples, Σ_i. If this is done, the covariance of the scaled kernel, $(1/h^n)k(X/h)$, is given by $h^2\Sigma_i$. As can be seen by Equation 2.1.4, the estimated density is a smoothed version of the true density function, and as h becomes small, the estimated density approaches the true density. However, although the bias of the estimate decreases for small h, the variance increases rapidly. Therefore, the choice of the sample scaling factor is critical.

The bias of the estimate can be solved for explicitly if the true density is assumed to be Gaussian. When $p_i(X)$ and $(1/h^n)k_i(X/h)$ are normal densities with covariances Σ_i and $h^2\Sigma_i$, the convolution produces another normal density with covariance $(1+h^2)\Sigma_i$. As h increases, the variance of the estimate decreases. Thus, a new estimate can be formed

$$ln(p(X)) \approx (1+h^2)ln(\hat{p}(X)) + \frac{h^2}{2}ln(|\Sigma|)$$

Using this new estimate, the decision threshold in Equation 2.2.1 can be changed to

$$t = -\frac{1}{1+h^2}\ln\frac{P(X\in\omega_1)}{P(X\in\omega_1)} - \frac{h^2}{2(1+h^2)}\ln\frac{|\Sigma_1|}{|\Sigma_2|}$$

It should be noticed that if all classes have the same number of samples and the determinant of the covariances of all classes are equal, then this expression is equal to the one shown in Equation 2.2.1.

2.1.1.4. Preliminary Classification Results

This section details the results of all the classification experiments performed to date. One experiment was reported in the first report, and three more classification experiments were reported in the second. In the time since the second report, one further experiment has been performed. All experiments used the Parzen classification error estimation procedure described in Section 2.1.1.3. The source of data used in these experiments was:

1) Gaussian data with known interclass overlap
2) Eglin turntable data.
3) Simulated FLIR from the NVL terrain board.
4) A subset of the targets from the BRITT database

The following explains each of the experiments.

2.1.1.4.1. Experiment 1 – Gaussian Data

The experiment on the first set of data was conducted to test the performance and demonstrate the statistical validity of the Parzen error estimation technique. In this experiment, the procedure was performed on Gaussian data with known Bayesian error. This test of the Parzen technique was performed for two reasons. First, it was desired to see qualitatively how tight the upper and lower bounds produced by the original and advanced Parzen error estimates were to the known error. Second, it was also desired to compare the results with the upper bound given by the Bhattacharrya [Fu72] distance and a parametric classifier for Gaussian data. The Bhattacharrya distance is a special case of the Chernoff bound [Ch62] and is commonly used to estimate the upper bound of the error probability. It was desired to compare the two methods to see if the Parzen estimate would yield a tighter upper bound then the one provided by the Bhattacharrya distance. The parametric technique for classifying Gaussian data is known as a quadratic classifier; this classifier which should give the best results possible for Gaussian data. Thus, the error bounds produced by the quadratic classifier are a good standard by which to measure the non-parametric techniques' performance.

Two separate sets of data were used in this experiment. The first data set tested performance when the means of the distributions of the classes differed. The second data set tested performance when the distributions of the data from the two classes had different covariance matrices. See Tables 2.1.1 and 2.1.2 for the parameters of the data for the two experiments.

2.1.1.4.1. Results

Ten trials were run on each data set, with results being reported for each trial. The ten trials were also averaged and the means and variances have also been reported. The experimental results of both the individual and averaged trials for the two sets of data are shown in Tables 2.1.1 through 2.1.4. As can be seen from the tables, the upper bound sometimes falls under the true error and the lower bound is sometimes slightly larger then the true error. This is due to the statistical nature of the data and occurs in all types of error estimation schemes. On the average, the bounds provided by the Parzen techniques prove to be very tight. This can be seen when the results of the Parzen and Bhattacharrya upper bounds are compared; in most cases, the Bhattacharrya bound reports almost twice the amount of actual error, while the output of the Parzen techniques closely matches the quadratic output.

2.1.1.4.2. Experiment 2 – Eglin Turntable Data

For the purpose of the first report, preliminary results were obtained simply to demonstrate that the software for computing interclass separability is in place. These preliminary results were obtained by using the Eglin turntable data. The data consists of 40 FLIR images of one target (a tank) rotating on a turntable. The images were segmented by hand (to reduce the chance of segmentation error) and the features discussed in [MM84] were extracted for each of

Table 2.1.1 Error Bounds of 10% gaussian data using Parzen error estimates.

true error: 10%				
Trial	Original		Advanced	
	Upper	Lower	Upper	Lower
0	9.5	5.5	8.0	7.5
1	9.0	9.0	8.0	7.0
2	11.0	10.5	10.0	9.5
3	11.5	11.0	11.5	11.0
4	6.5	6.0	8.0	8.0
5	8.0	7.0	9.0	8.5
6	9.0	9.0	9.0	8.5
7	8.5	8.5	8.0	7.0
8	11.0	10.5	12.5	12.0
9	9.5	9.0	9.5	9.0
avg - mean	10.3	9.5	9.4	8.8
avg - s.d.	2.2	1.8	1.5	1.7

True error: 10 %
Dimension: 8
Samples per Class: 100
Covariances: $\Sigma_1 = I$, $\Sigma_2 = I$
means:

$$M_1 = \begin{bmatrix} 1.28 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad M_2 = \begin{bmatrix} -1.28 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Table 2.1.2 Error Bounds of 9% gaussian data using Parzen error estimates.

true error: 9%				
Trial	Original		Advanced	
	Upper	Lower	Upper	Lower
0	7.0	4.5	3.0	2.5
1	15.0	7.0	11.5	9.0
2	9.0	5.0	5.5	4.5
3	10.0	5.5	7.0	6.5
4	15.0	9.0	10.0	7.0
5	12.0	10.5	9.5	7.0
6	9.5	4.5	7.5	6.0
7	11.0	6.0	8.0	6.0
8	10.5	5.5	6.5	5.0
9	11.0	7.5	8.0	7.0
avg - mean	11.2	6.1	7.8	6.1
avg - s.d.	2.6	1.4	2.4	1.9

True error: 9 %
Dimension: 8
Samples per Class: 100
Covariances: $\Sigma_1 = I$, $\Sigma_2 = 4I$
means:

$$M_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad M_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Table 2.1.3 Error Bounds of 10% gaussian data using Quadratic and Bhattacharrya estimates.

true error: 10%			
Trial	Quadratic		Bhattacharrya
	Upper	Lower	Distance
0	8.5	7.5	18.3
1	9.5	7.5	19.9
2	12.0	9.5	20.4
3	13.5	11.0	18.0
4	8.5	8.0	19.7
5	10.0	8.5	19.8
6	9.5	8.5	18.9
7	8.5	7.5	19.2
8	13.0	12.0	20.9
9	10.5	9.0	20.2
avg - mean	10.3	8.9	19.5
avg - s.d.	1.9	1.5	0.9

Table 2.1.4 Error Bounds of 9% gaussian data using Quadratic and Bhattacharrya estimates.

true error: 9%			
Trial	Quadratic		Bhattacharrya
	Upper	Lower	Distance
0	3.5	2.0	14.5
1	15.5	12.0	20.5
2	8.0	5.0	19.7
3	8.0	7.0	17.1
4	11.0	8.5	18.9
5	10.0	7.0	17.6
6	8.5	8.0	19.4
7	10.0	7.0	19.4
8	8.0	5.5	17.7
9	8.0	7.0	17.9
avg - mean	9.1	6.9	18.3
avg - s.d.	3.0	2.6	1.7

the images. These features were examined and two features were selected to be used in each of the classification experiments. The image database contains only 40 images of *one* target, so the classification experiment was designed to classify the front and back views from the side view of the tank. Only two features in each experiment were used because there were less than 20 images per class and at least 10 samples per feature are needed for each class [KaLa83].

Figure 2.1.4 shows three sample views from the Eglin turntable data, and the corresponding binary segmented images. Table 2.1.5 shows the values of the features extracted from images in Figure 2.1.4. The moments presented in Table 2.1.5 were computed with the correct invariants, as discussed in Section 2.1.1.2.

The results of six classification experiments are in presented in Table 2.1.6. The classification errors varied from 0 to 25% for a lower bound, and 0 to 37.5% for an upper bound. Due to the very limited size of the input data, the only real conclusions that can be drawn from this classification data is that we are able to run classification experiments.

2.1.1.4.3. Experiment 3 – Simulated FLIR

The data set for the third experiment was generated from the simulated FLIR obtained from the Night Vision Lab's terrain board. The terrain board data included three types of targets: APCs, tanks and trucks. In the database, each type of target was rotated through 360 degrees, in 45 degree increments. There were eleven images of each target for each target orientation, however, the target images for any orientation were nearly identical. Each image in the data set included three targets, one of each type of target. Because of the relatively large size and high definition of the vehicles in this database, target detection and identification was extremely easy.

2.1.1.4.3. Target Classes

Two experiments were performed on the terrain board data. The first experiment used the three classes of targets mentioned above. The second experiment was performed to determine if the classifiability of the targets would improve if subclasses (clusters) were formed based on target orientation. In this experiment, each class of targets was split into two clusters, one for the front/rear view and another for the side views.

2.1.1.4.3. Features

In this experiment, three different feature sets were used; they fall in the following categories

1) segmentation features
2) grey scale features

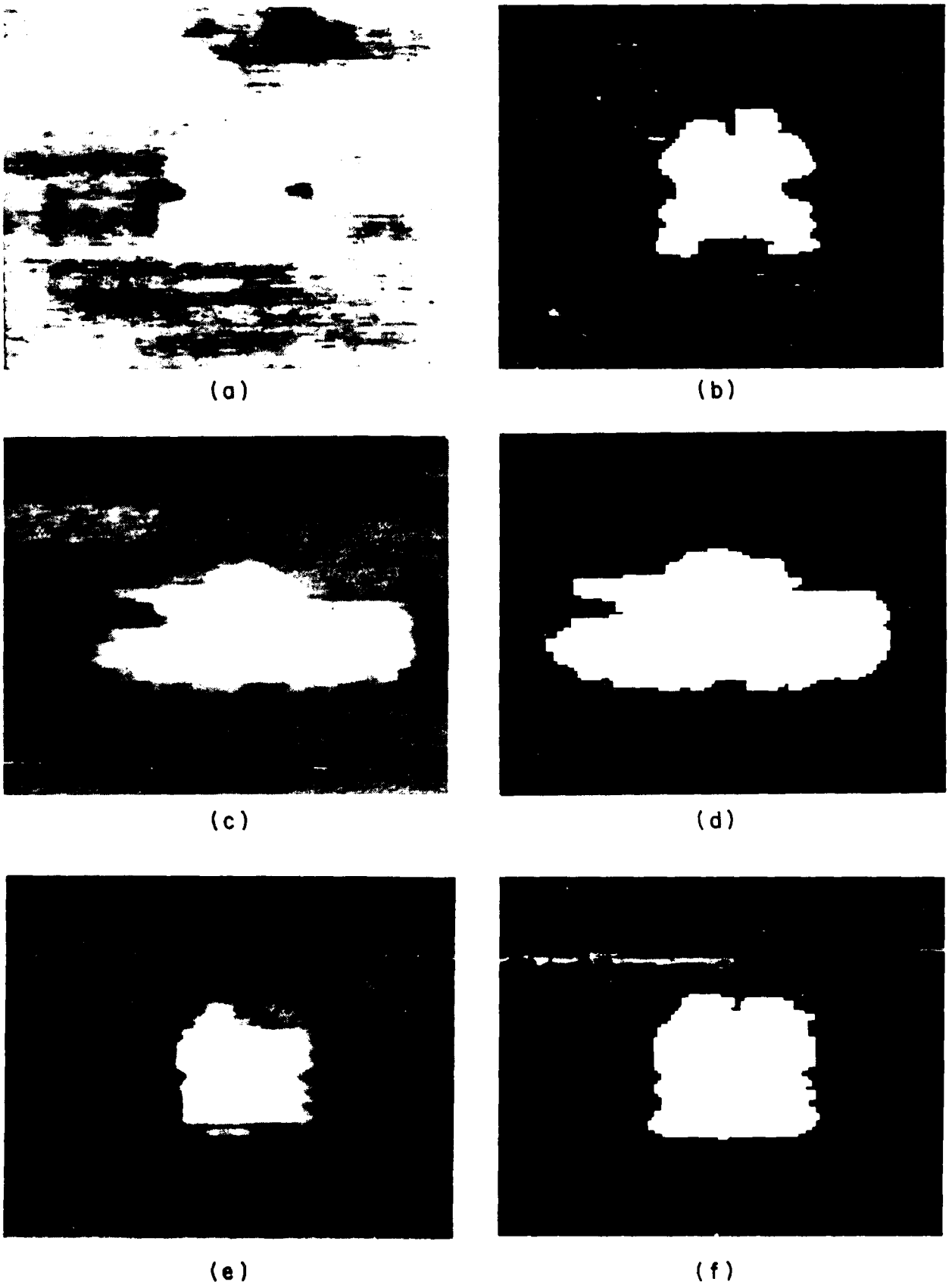


Figure 2.1.4 Sample images from Eglin turntable data. (a) Front view of tank (FILE01). (b) Segmented front view. (c) Side view of tank (FILE15). (d) Segmented side view. (e) Back view of tank (FILE27). (f) Segmented back view.

Table 2.1.5. Features extracted from images in Figure 2.1.4.

feature	*front view* FILE01	*side view* FILE15	*back view* FILE27
height	45	43	44
width	47	98	49
area	1511	2996	1800
height/width	0.957447	0.438776	0.897959
max	140	215	210
min	59	67	62
mean	102.860359	114.354805	124.765556
variance	250.677017	845.803650	1963.733765
sigma	15.832783	29.082705	44.314037
moments			
m00	1.5542e+05	3.4261e+05	2.2458e+05
m10	9.8136e+06	2.3239e+07	1.5123e+07
m01	1.0338e+07	2.4963e+07	1.5022e+07
m20	6.3890e+08	1.6087e+09	1.0442e+09
m11	6.5201e+08	1.6958e+09	1.0108e+09
m02	7.0811e+08	2.0093e+09	1.0347e+09
m30	4.2751e+10	1.1344e+11	7.3732e+10
m21	4.2411e+10	1.1747e+11	6.9740e+10
m12	4.4663e+10	1.3701e+11	6.9576e+10
m03	4.9827e+10	1.7383e+11	7.3235e+10
central moments			
u00	1.5542e+05	3.4261e+05	2.2458e+05
u20	1.9255e+07	3.2411e+07	2.5790e+07
u11	-7.4096e+05	2.6444e+06	-8.5466e+05
u02	2.0482e+07	1.9047e+08	2.9887e+07
u30	-2.1256e+07	-7.3833e+07	-6.1042e+07
u21	8.0478e+06	-9.4706e+07	5.7593e+06
u12	5.0408e+07	3.3233e+08	9.4191e+06
u03	1.7111e+06	-3.2683e+08	2.2330e+07
normalized moments			
n20	7.9712e-04	2.7613e-04	5.1136e-04
n11	-3.0674e-05	2.2530e-05	-1.6946e-05
n02	8.4790e-04	1.6227e-03	5.9258e-04
n30	-2.2320e-06	-1.0746e-06	-2.5540e-06
n21	8.4507e-07	-1.3784e-06	2.4096e-07
n12	5.2932e-06	4.8370e-06	3.9409e-07
n03	1.7968e-07	-4.7570e-06	9.3428e-07

feature	*front view* FILE01	*side view* FILE15	*back view* FILE27
Hu's invariants			
p1	3.9737e+07	2.2288e+08	5.5678e+07
p2	3.7010e+12	2.5010e+16	1.9705e+13
p3	3.0252e+16	1.1485e+18	8.0000e+15
p4	9.4508e+14	2.4451e+17	3.4540e+15
p5	-2.2993e+30	1 2865e+35	3.5068e+29
p6	-1.7689e+21	1.6371e+25	-2.7280e+21
p7	4.5000e+30	-1.5427e+34	1.8153e+31
Hu's normalized invariants			
f1	1.6450e-03	1.8988e-03	1.1039e-03
f2	6.3426e-09	1.8152e-06	7.7465e-09
f3	3.3358e-10	2.4330e-10	1.4004e-11
f4	1.0421e-11	5.1799e-11	6.0463e-12
f5	-2.7955e-22	5.7737e-21	1.0746e-24
f6	-8.0745e-16	2.9547e-14	-9.4682e-17
f7	5.4712e-22	-6.9235e-22	5.5626e-23
beta's (Maitra's)			
b1	2.3438e-03	5.0347e-01	6.3564e-03
b2	3.1971e+01	7.0591e-02	1.6376e+00
b3	3.1240e-02	2.1290e-01	4.3175e-01
b4 (new)	-2.5743e+00	2.1519e+00	2.9394e-02
b5	-4.7102e-02	3.0041e-01	-1.4185e-02
b6	-1.9571e+00	-1.1991e-01	5.1766e+01

Table 2.1.6. Classification error estimates based on Eglin turntable data.

Experiment	*Features*	*h*	*Lower Bound Error*	*Upper Bound Error*
1	Height Width	2.7500	5.00%	5.00%
2	Area, Height/Width ratio	3.5000	5.00%	5.00%
3	max grey level min grey level	0.1000	17.50%	37.50%
4	mean grey level sigma of grey level	0.1500	2.50%	5.00%
5	beta1 beta2	1.0000	0.00%	0.00%
6	beta6	0.2500	25.00%	35.00%

3) Maitra's beta functions

The first set of features depends only on the target segmentation; that is, the grey-scale values of the target do not affect them at all. These features include parameters such as the target's height, width and area. Statistical parameters of the target's grey-scale values comprised the second feature set. The third feature set consisted of the Maitra's beta functions. Table 2.1.7 shows the features used in each feature set.

Table 2.1.7 Features used for classification.

Features Used	
feature set 1 (segmentation)	target width target height target area height / width
feature set 2 (grey scale)	min grey level max grey level mean grey level variance grey level sigma grey level
feature set 3 (beta functions)	Maitra's beta functions (1,2,4,5,6)

2.1.1.4.3. Segmentation

Hand thresholding was used to segment the targets. Only one segmentation was used for all eleven target images of any given orientation. This was done because all target images of any orientation were almost identical. **It should be noted that because the targets in each orientation were segmented identically, there were not enough independent samples using the segmentation features to give statistically meaningful results.** Furthermore, in the experiment where the classes were split into clusters, no results for the segmentation features were obtained because the covariance matrices became singular because the images were nearly identical.

2.1.1.4.3. Results

The upper and lower bounds for both experiments are shown in Table 2.1.8. Feature set one is not statistically valid because the covariance matrices became singular. **These results show that FLIR targets are reliably classifiable if they are well defined and are of a relatively large size.** The results also show that the bounds given by the Parzen error estimation technique remain fairly tight when applied to real world data. This experiment also confirms one of the Parzen technique's major shortcomings, its need for a large number of samples.

Table 2.1.8 Results of classifiability experiments on simulated FLIR.

feature	with clusters		no clusters	
set	upper	lower	upper	lower
1	--.-*	--.-*	0.0*	0.0*
2	22.4%	4.2%	22.1%	3.3%
3	13.6%	0.3%	13.9%	0.3%

* not statistically valid

One surprising result is found when comparing the results of the non-clustered and clustered experiments. It was originally thought that by splitting the classes into clusters would improve classifiability. However, the results of this experiment show that no improvement is made when splitting the classes into the front/rear and side views. Since the classes were split solely on the hueristic argument that the statistics of the classes should change the most between the two views, the lack of improvement of classifiability may be due to inappropriate clustering. A statistical clustering technique may be used to verify if inappropriate clustering was the cause of this lack of improvement.

2.1.1.4.4. Experiment 4 – BRITT Data

The targets from the BRITT database consists of tanks, trucks, APCs and jeeps. The database contains images where the targets range in quality from highly distinguishable to virtually invisible. All images were collected at a range of either 2.5, 3.5, or 5 kilometers with the height of the FLIR sensor varied from 100 feet to 200 feet to give a constant angle of declination.

2.1.1.4.4. Target Classes

The experiment was run with three classes: tanks, trucks and APCs. The database was manually searched and 50 of the most visible targets from each class were selected. Unfortunately there were not enough targets to enable an entire class to consist of a single type of vehicle at a fixed range; because of this, two target classes consisted of multiple target types at varying ranges. Table 2.1.9 shows the make-up of the different classes, and Figures 2.1.5, 2.1.6, and 2.1.7 are the actual targets.

Table 2.1.9 Target types of classes.

Class compositions			
Class	Number	Type	Distance
Tanks	50	M551	2.5km
APCs	25	M113	2.5km
	25	M114	2.5km
Trucks	18	M35	2.5km
	18	M35	3.5km
	14	M35	5km

2.1.1.4.4. Features

The features used were identical to the ones used on the simulated FLIR in experiment 2.

2.1.1.4.4. Segmentation

The targets were segmented by hand. The method used consisted of enclosing each target in a **bounding rectangle** as shown in Figures 2.1.5, 2.1.6, and 2.1.7. This was done because of the highly varying grey level value of any given target ruled out simple thresholding as a segmentation technique. It is hoped that either a wire frame segmentation technique or a segmentation method currently employed by industry can be used in the future.

2.1.1.4.4. Results

Table 2.1.10 reports the upper and lower classification error bounds for the different feature sets. The confusion matrices for the different feature sets are given Tables 2.1.11 - 2.1.16. The results show that statistical classifier performance is extremely poor. There are a number of reasons for this. First, the targets are hard to classify; a human operator is hard

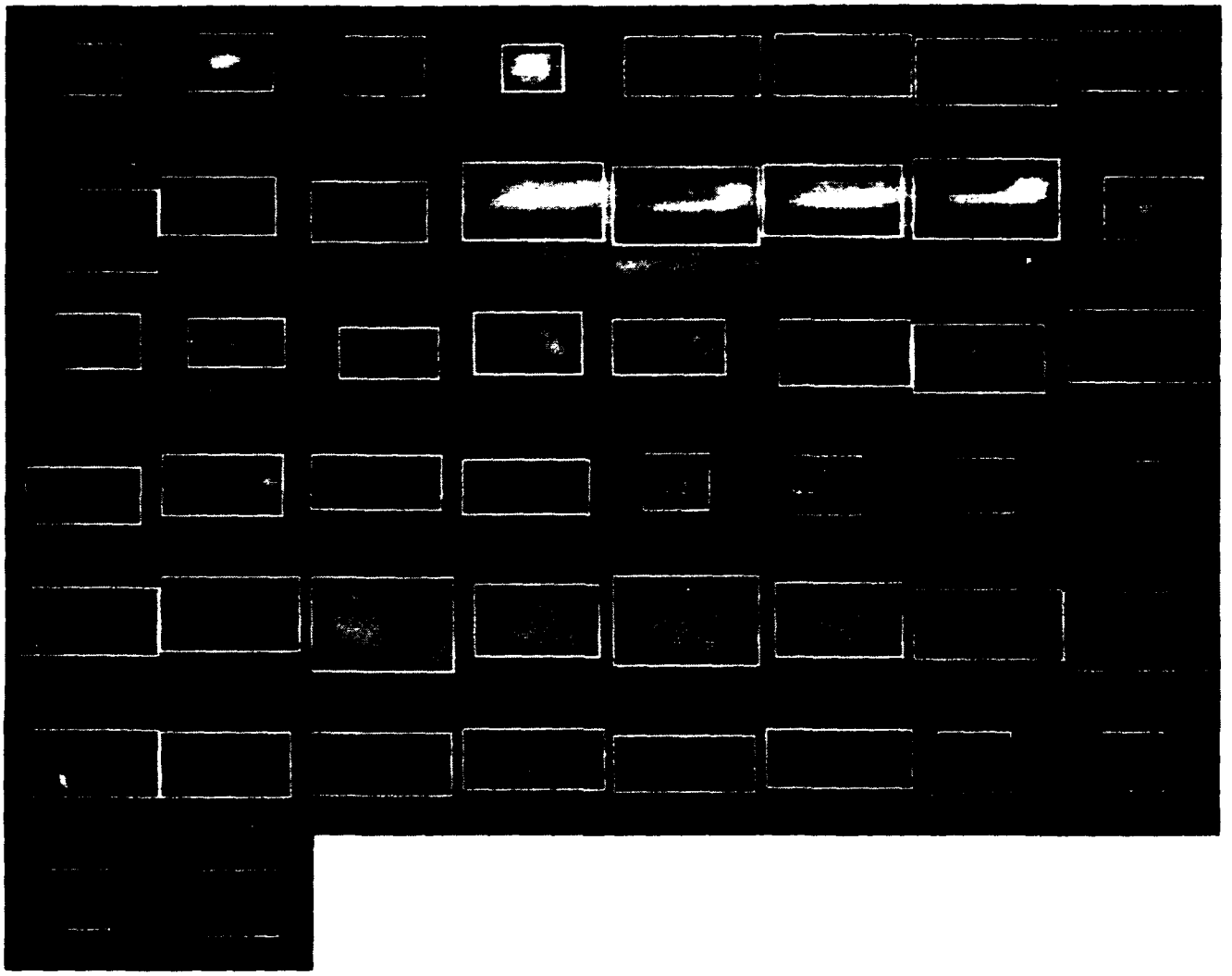


Figure 2.1.5 Fifty images in *Tanks* class with bounding rectangle segmentation shown.

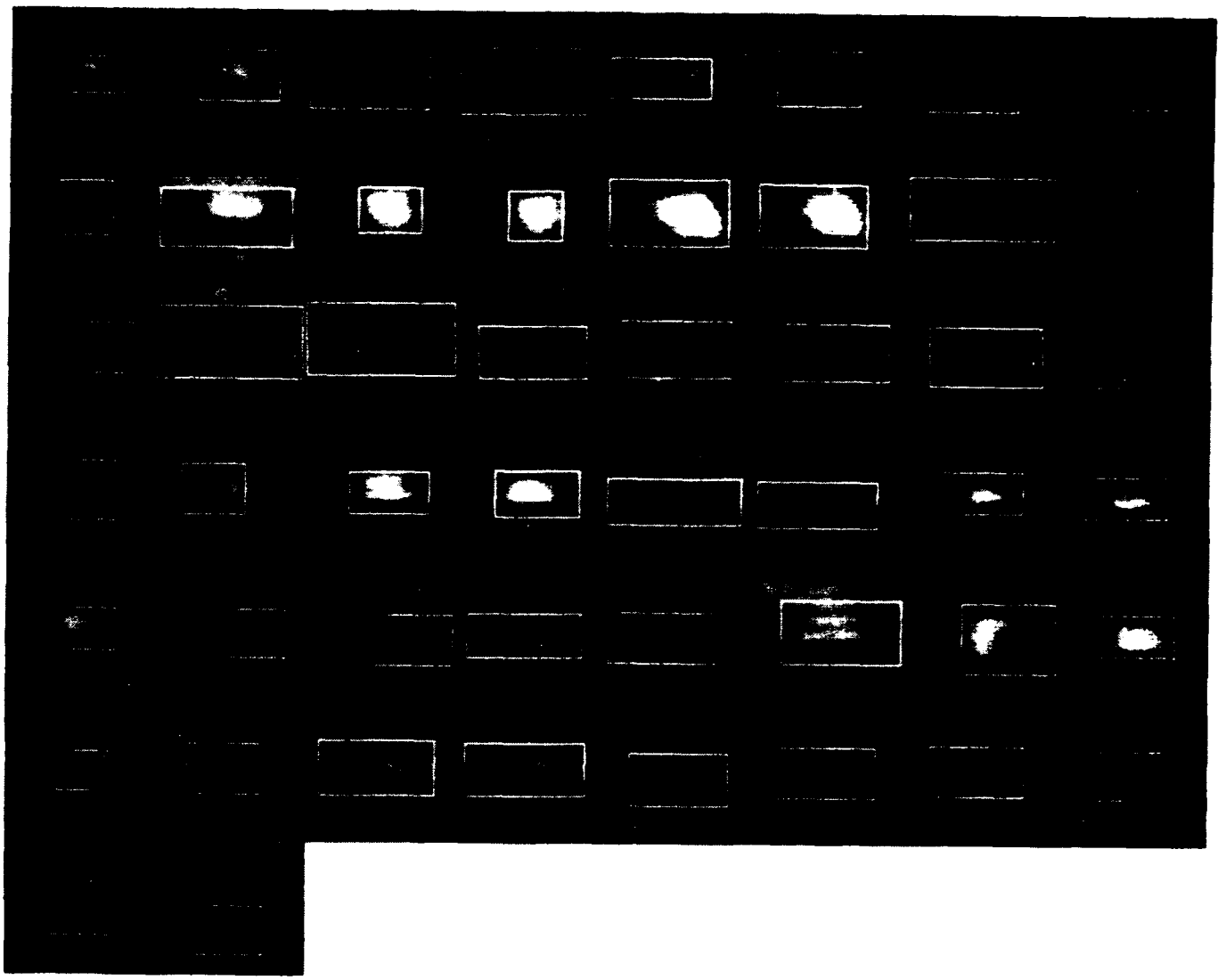


Figure 2.1.6 Fifty images in *APCs* class with bounding rectangle segmentation shown.

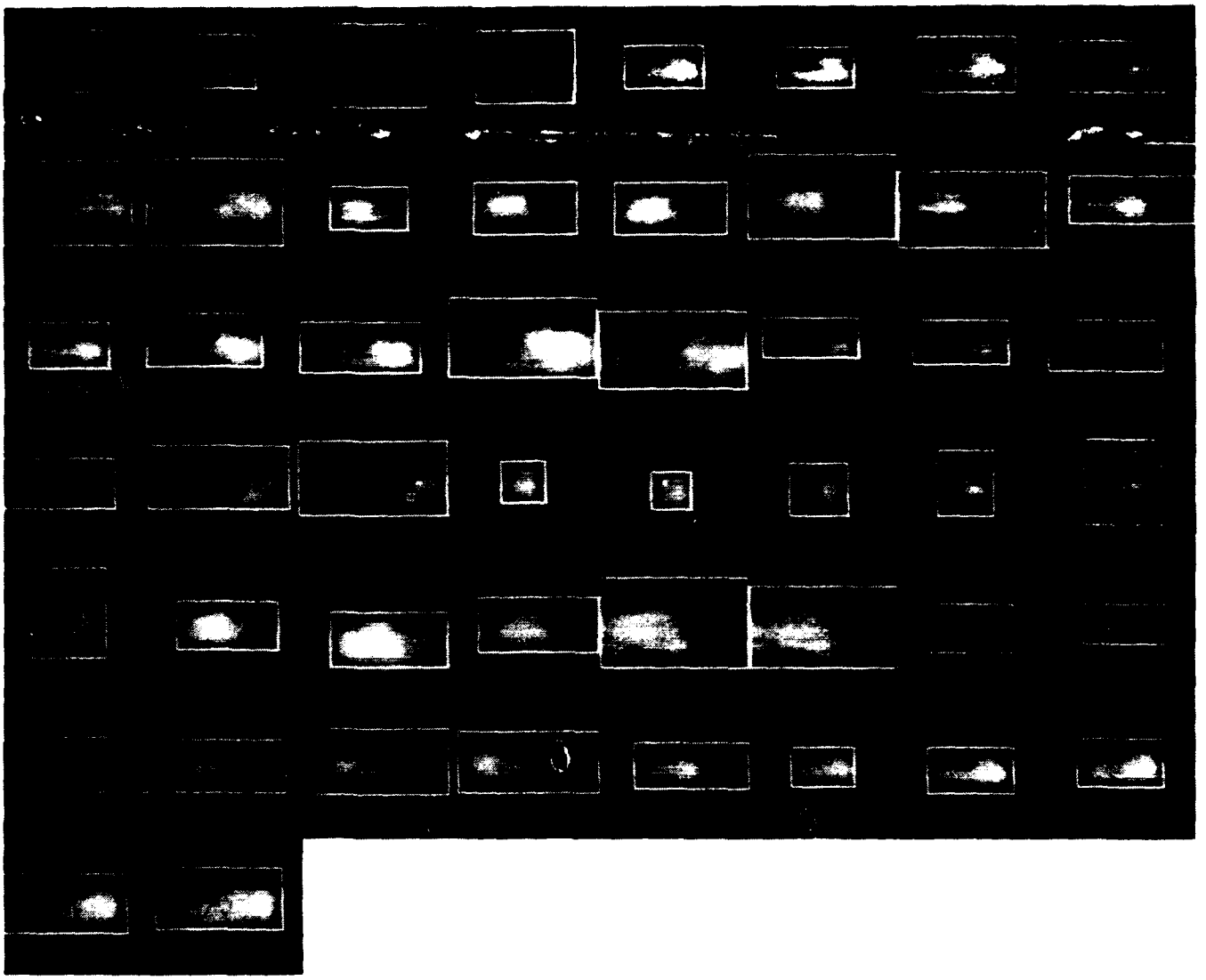


Figure 2.1.7 Fifty images in *Trucks* class with bounding rectangle segmentation shown.

pressed to classify the targets in many cases. Secondly, it is thought that the segmentation method used severely reduced classification ability. The next section shows that because the segmentation features and beta functions are highly shape dependent, the classification performance improves when a more sophisticated segmentation technique is used. Lastly, it is hard to determine classifier performance when using the beta function feature set because of the width of the error bounds. In the next section we show that much tighter bounds can be obtained by using an advanced Parzen error estimation technique.

Table 2.1.10 Results of classifiability experiments on BRITT Data.

feature set	upper bound	lower bound
segmentation	51.3%	42.0%
grey scale	46.7%	42.7%
beta functions	63.6%	11.1%

Table 2.1.11 Confusion matrix for *leave-one-out* error of **segmentation** features (51.3% error).

true	classified as		
class	APCs	tanks	trucks
APCs	34	14	2
tanks	21	28	1
trucks	30	9	11

Table 2.1.12 Confusion matrix for *resubstitution* error of **segmentation** features (42.0% error).

true class	classified as		
	APCs	tanks	trucks
APCs	42	8	0
tanks	18	32	0
trucks	29	8	13

Table 2.1.13 Confusion matrix for *leave-one-out* error of **grey scale** features (33.3% error).

true class	classified as		
	APCs	tanks	trucks
APCs	30	15	5
tanks	10	35	5
trucks	3	12	35

Table 2.1.14 Confusion matrix for *resubstitution* error of **grey scale** features (8.7% error).

true class	classified as		
	APCs	tanks	trucks
APCs	40	8	2
tanks	1	47	2
trucks	0	0	50

Table 2.1.15 Confusion matrix for *leave-one-out* error of **Beta** functions (52.7% error).

true class	classified as		
	APCs	tanks	trucks
APCs	36	7	7
tanks	30	18	2
trucks	20	13	17

Table 2.1.16 Confusion matrix for *resubstitution* error of **Beta** functions (8.0% error).

true class	classified as		
	APCs	tanks	trucks
APCs	42	2	6
tanks	0	50	0
trucks	0	4	46

2.1.1.4.5. Experiment 5 – BRITT Data – Improved Techniques

The fifth experiment was run to determine if any improvement in classifiability could be gained by using a sophisticated technique to segment real world targets. To enable comparison, the targets used were the same as those of experiment 4. The experiment was also run to test the performance of the advanced Parzen error estimation scheme on data with unknown distributions.

2.1.1.4.5. Target Classes

The experiment was run with three classes: tanks, trucks and APCs. The classes were composed of the same hand picked targets as experiment 4 and are shown in Table 2.1.9.

2.1.1.4.5. Features

Three feature sets used were identical to the ones used on the simulated FLIR and the first BRITT experiment in experiments 2 and 3 respectively. However, two new feature sets were also used; Table 2.1.17 shows the features used in these new sets.

Table 2.1.17 Features used for classification.

Features Used	
feature set 4	rectangularity perimeter^2 / area height^2 / area height^2 * range^2
feature set 5	normalized contrast depression angle area * range^2 (width / height)^2

2.1.1.4.5. Segmentation

The targets in this experiment were segmented using the Hughes segmenter referenced in Section 2.1.1.1. The parameters of the segmenter used in this experiment were set by hand to produce optimal results. Because the BRITT database includes target locations, no detection scheme was needed to locate the targets. Figures 2.1.8, 2.1.9, 2.1.10 show the segmented targets.

2.1.1.4.5. Results

Table 2.1.18 reports the upper and lower classification error bounds for the different feature sets using the original Parzen error estimation procedure. The results show the expected improvement in classifiability over the results for experiment 3. In most cases, the upper bound using the new segmentation techniques is near the lower bound produced by the bounding rectangle method. This increase in performance is due to the improved segmentation scheme. The results of applying the advanced Parzen technique to the data is shown in Table 2.1.19. The results show that the error bounds have been tightened in three of the five feature sets. Although the tightening affect is not as drastic as had been hoped for, the improvement shown is not insignificant. Because of the improvement in results shown and the theoretical argument presented earlier, the advanced Parzen error estimation technique will be used in all future

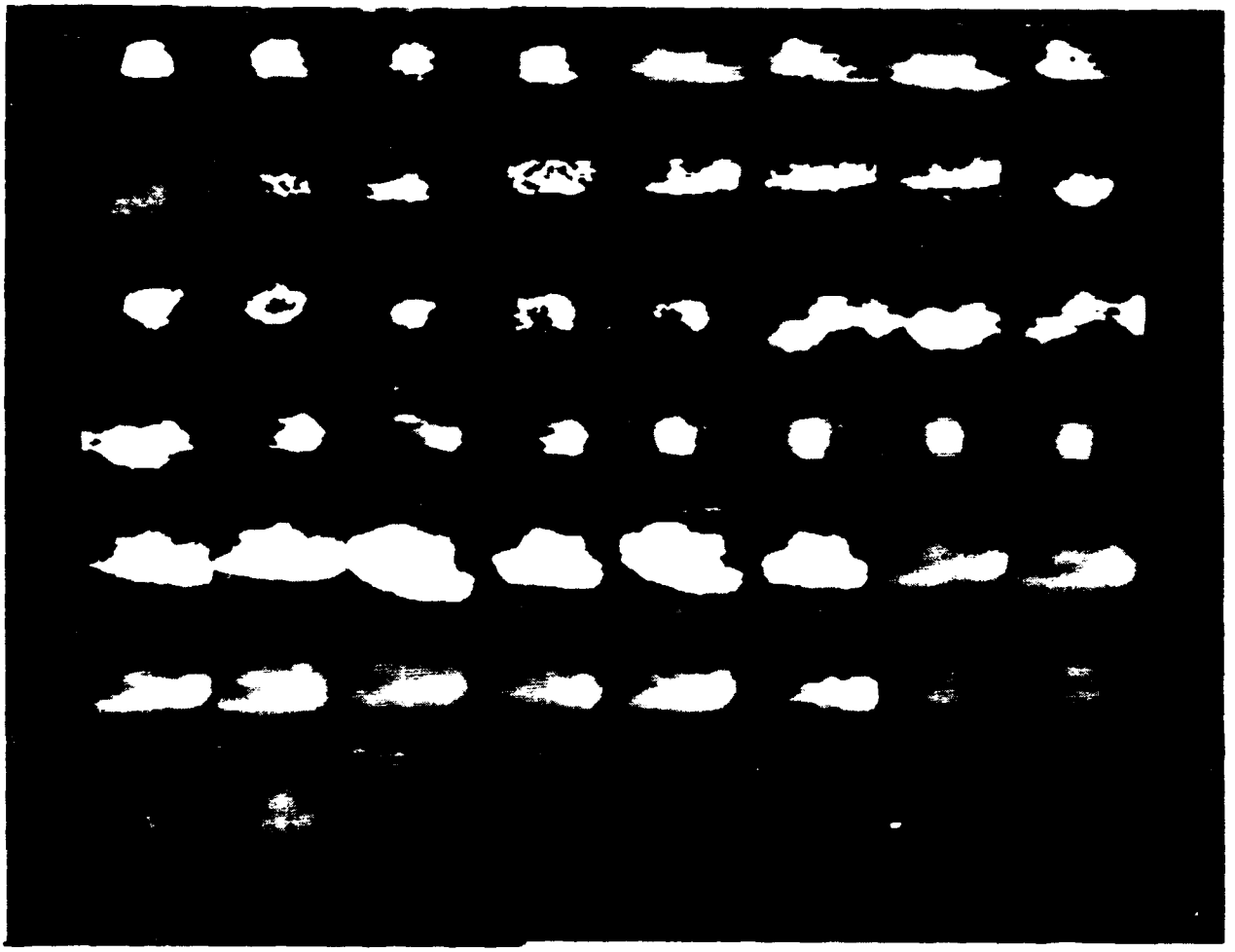


Figure 2.1.8 Fifty images in *Tanks* class after automatic segmentation.

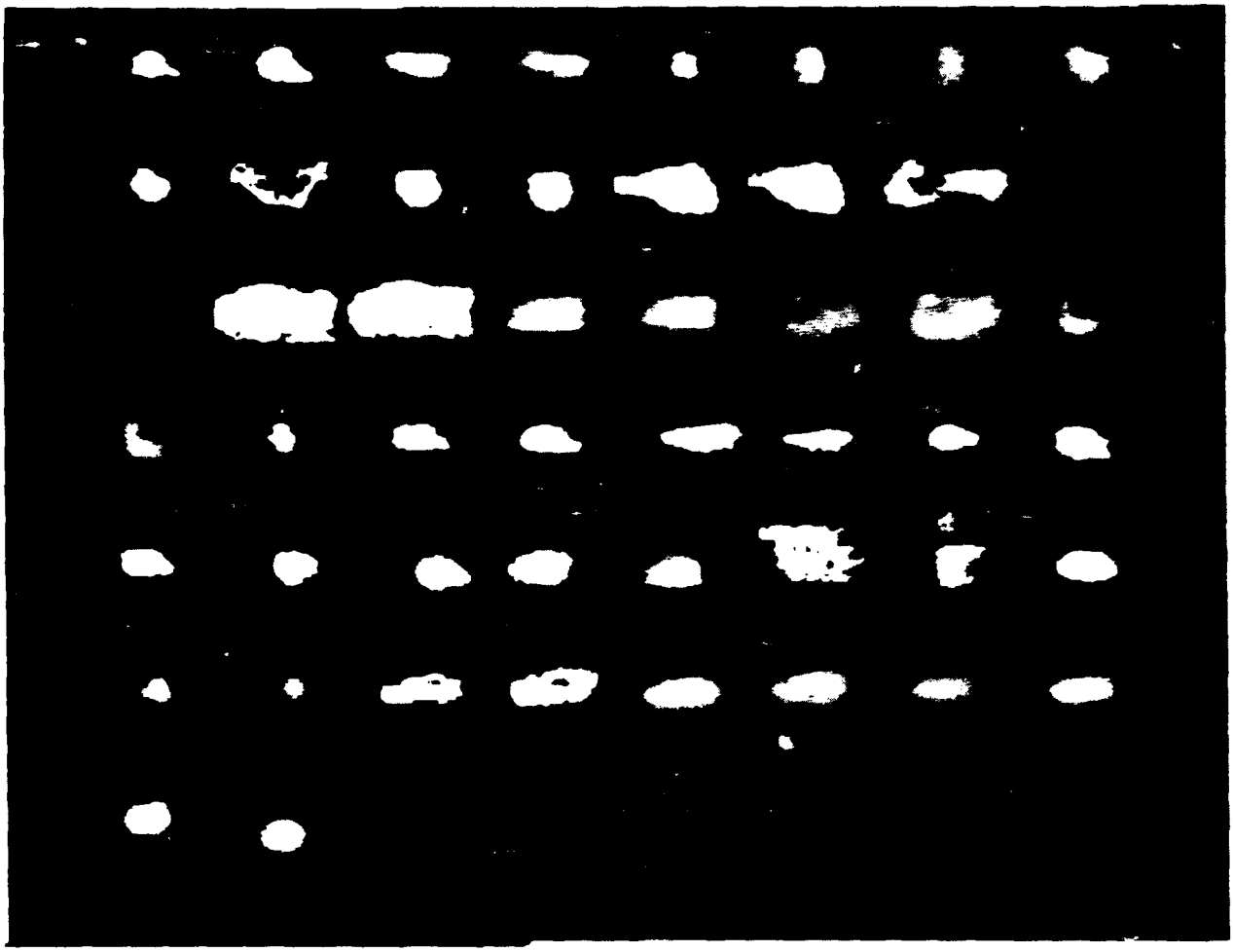


Figure 2.1.9 Fifty images in *APCs* class after automatic segmentation.

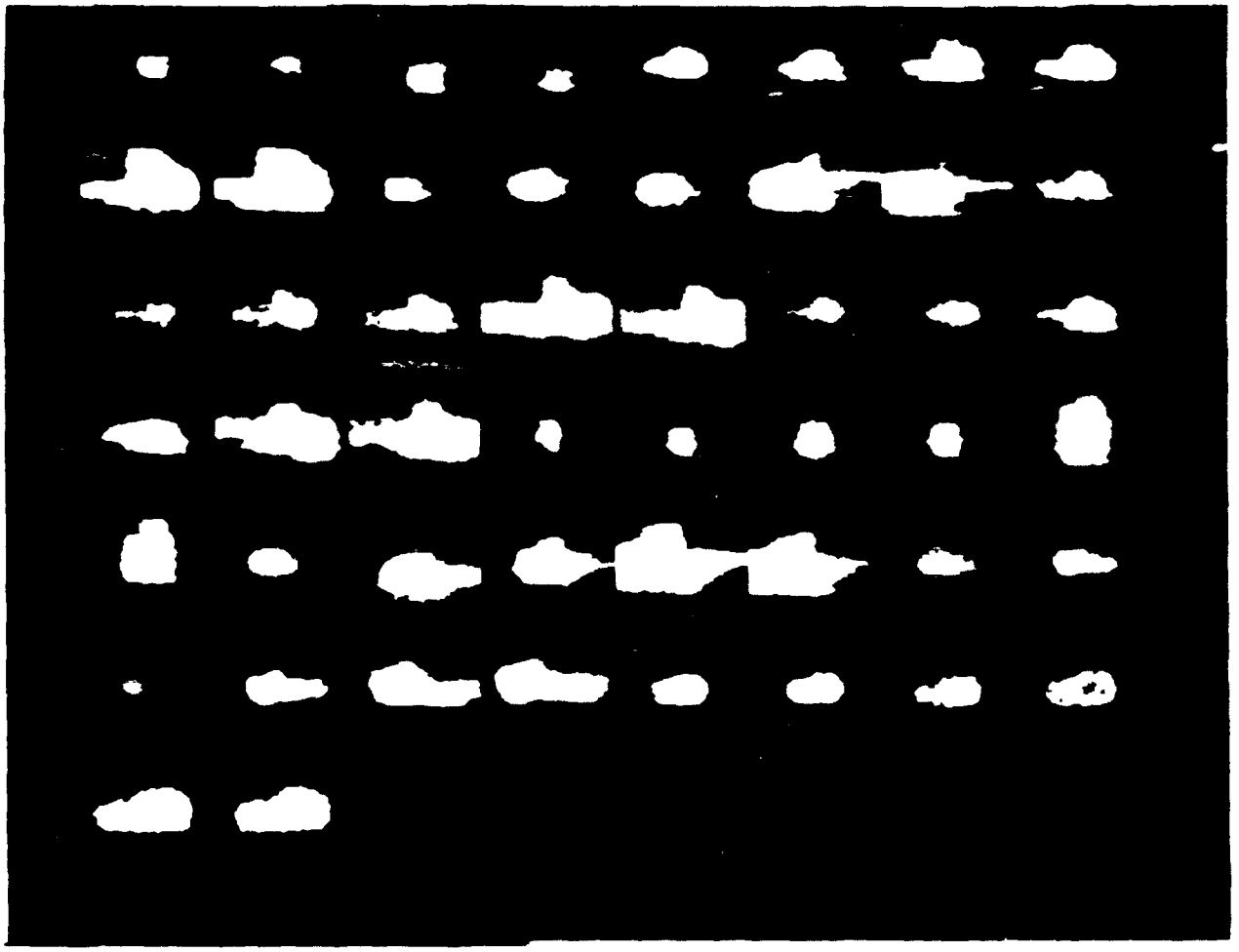


Figure 2.1.10 Fifty images in *Trucks* class after automatic segmentation.

experiments. Some people at the Night Vision Labs have also expressed an interest in seeing the results from a parametric error estimation scheme. Table 2.1.20 shows the error bound found by a parametric (quadratic) error estimator. This table points out the major flaw of parametric error estimates. Although they do not require as many samples to produce results and they do produce valid upper bounds, the upper bounds that they produce are often unnecessarily high and their lower bounds are valid only if the samples are drawn from a distribution of the expected form (in this case Gaussian).

Table 2.1.18 Results of classifiability experiments on BRITT Data, original Parzen estimate.

feature set	upper bound	lower bound
segmentation	43.3%	6.7%
grey scale	34.7%	2.7%
beta functions	47.3%	30.0%
feature set 4	47.3%	21.3%
feature set 5	36.7%	19.3%

Table 2.1.19 Results of classifiability experiments on BRITT Data, advanced Parzen estimate.

feature set	upper bound	lower bound
segmentation	43.3%	10.7%
grey scale	34.0%	12.7%
beta functions	47.3%	28.0%
feature set 4	48.0%	20.7%
feature set 5	37.3%	24.0%

Table 2.1.20 Results of classifiability experiments on BRITT Data, quadratic estimate.

feature set	upper bound	lower bound
segmentation	53.3%	49.3%
grey scale	38.7%	30.7%
beta functions	54.7%	52.0%
feature set 4	62.0%	60.0%
feature set 5	42.0%	38.7%

2.1.1.5. Future Work

The preceding experiments demonstrate that we have the tools available to find the classifiability of a set of targets represented as grey-scale images. Our future work in this area will be aimed at using these techniques to find the error bounds with a larger number of features per sample vector. It is believed that experiments with large sample vectors will show that classifier performance on FLIR targets is acceptable if the targets are highly defined and enough features are used for classification. As was mentioned previously in this report, the techniques presented here will require a minimum of 200 samples per class to prove our conjecture with large feature vectors. Once acceptable results have been obtained, we plan to find the classifiability of FLIR targets from data typical of the type drawn from the real world. Targets will be drawn at random from the database with no regard for target definition and a **clutter** class will be introduced. The motivation for the introduction of the clutter class is the fact that any target detection scheme will allow a significant number of false detections to occur if the probability that a target is missed is minimized. We plan to simulate these false target detections with the introduction of the clutter class. We hope to use this final experiment to prove our conjecture that there is not enough information in a single FLIR frame to accurately classify a random sampling of typical FLIR targets and clutter.

2.1.2. Algorithm and Image Metrics

Image metrics are supposed to provide us with an independent set of variables for image and algorithm characterization. The approach is to partition all ATRs (automatic target recognizers) into three functional areas: detection, segmentation, and classification, as shown in Figure 2.1.11. The image metrics are then used as sets of independent variables which will independently characterize each of the functional areas.

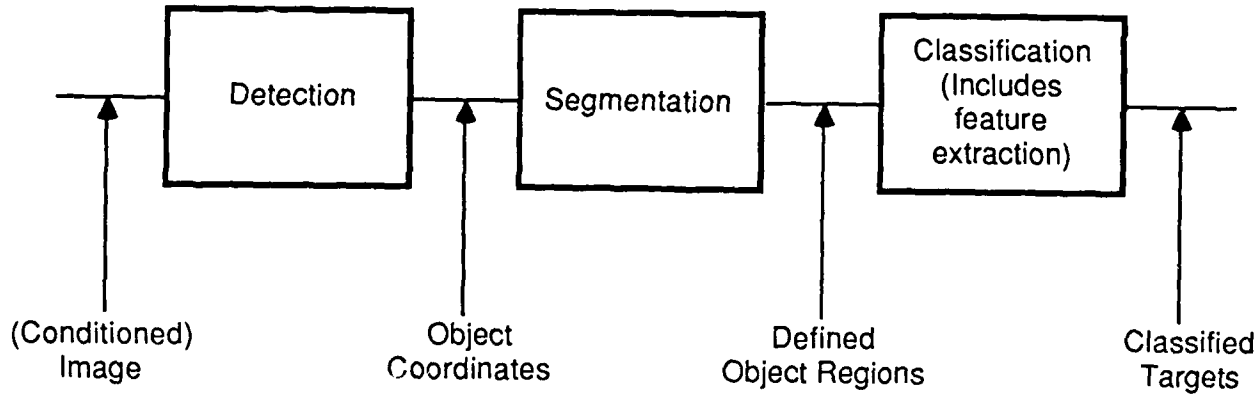


Figure 2.1.11 ERIM's Generic ATR Process.

Section 2.1.2.1 discusses some of the metrics proposed by ERIM and Section 2.1.2.2 proposed a new segmentation metric for threshold based segmenters.

2.1.2.1. Evaluation of ERIM's Metrics

ERIM has proposed many metrics for image and algorithm characterization. The following sections examine some of these metrics and show that there are many times when these metric give meaningless results. The next subsection discusses the characterization of target detection and gives examples of where images with the same TIR^2's (and therefore the same complexity) have very different P_d's, showing that there are cases where TIR^2 does not measure complexity with respect to the probability of detection.

Section 2.1.2.1.2 presents the results of our study of $TBIR^2$ as a metric for segmentation. We show that images with the same $TBIR^2$ can have different segmentation accuracies. Therefore image complexity with regard to segmentation cannot be measured by this metric.

2.1.2.1.1. Characterization of Target Detection

According to ERIM formulation, the performance of a detection algorithm can be evaluated by plotting P_d against the following "independent" variables:

1. Target-Interference Ratio Squared (TIR^2)
2. Resolution Cells on Object (RN_0)
3. Expected Resolution Cells on Object (RE_0)
4. Edge Strength Ratio (ESR)

Our basic criticism of this characterization methodology is based on the conviction:

> It can be misleading to examine the performance of an algorithm against a collection of variables **individually**, particularly when it can be shown from elementary theoretical analysis that the performance might in fact be dependent upon some combination of these variables.

What we are trying to say is that when detection algorithms are theoretically analyzed, one can demonstrate that the P_d must be a function of the product of the average contrast difference (between the target and background) and the effective size of the target, which is given by RN_0.* Therefore, examining separately the dependence of P_d on TIR^2 and RN_0 can be

*This statement is based on the following elementary result from the classical decision theory [Trees68]. Suppose we want to detect a deterministic signal s(t) that is corrupted by additive random noise n(t); the observed signal being denoted by r(t). Optimum detection is obtained by conducting the following likelihood ratio test

$$\frac{2\sqrt{E}}{N_0}\int_0^T r(t)s(t)dt \underset{H_0}{\overset{H_1}{>\!<}} \ln\eta + \frac{E}{N_0}$$

misleading.

Put another way, suppose we confine our attention to performance as measured by P_d vs TIR^2. The probability of detection, P_d, is the probability that an actual object of interest is detected at least once, measured over all the actual objects of interest. TIR^2 is defined as:

$$TIR^2 = \frac{(\bar{x}_o - \bar{x}_b)^2}{\sigma_b^2}$$

It is possible to construct two different examples of targets with identical TIR^2 measures but responding very differently to the process of detection. Suppose we have an image with the grey scale values of the background and the target pixels distributed as show in Figure 2.1.12. TIR^2 would predict a certain level of complexity for that image. Suppose we had a second image in which σ_b and σ_o were the same as the first image, but the difference in the grey level of the background and the target was greater (as shown on the bottom of Figure 2.1.12). TIR^2 would predict that the second image is less complex. This is a correct prediction because there is less overlap between the distributions, and therefore less possibility for error.

Figure 2.1.13 shows a similar example. this time the difference in means is the same, but σ_b has decreased. TIR^2 again predicts a less complex image, which is correct since there is less overlap.

The major flaw in the TIR^2 measure is that σ_o, the variance of the object, does not appear in the TIR^2 equation. Therefore all the distributions in Figure 2.1.14 have the **same** TIR^2. However the complexity will be very different for each because the overlap in the grey scale values varies greatly from one distribution to another.

That this is indeed so was verified by the following experiments, one involving uncorrelated additive noise and the other correlated noise; the latter we believe is more representative of what happens in practice. First we show the case of uncorrelated noise.

In Figure 2.1.15, we have shown a sequence of synthesized target images. These images are constructed by first adding an elliptical "target" to a uniform background and then adding uncorrelated random Gaussian random noise to the composite.

The following procedure, which we believe is close to what the NVL contractors are using, was implemented for target detection in these images. We first construct a histogram of the brightness values, the value corresponding to the most prominent valley in the histogram is used for segmentation by thresholding. The segmented output is then integrated and compared against 80 percent of the expected area of the actual target to determine whether the target is

---

where H_1 is the hypothesis that the signal is present and H_0 that it is absent. E is the total energy in the signal, N_0 the noise variance, and T the total observation time. For our application, the left hand side would be proportional to the product of TIR^2 and RN_0 for simple constant gray scale targets. Similar conclusions can be drawn from the more advanced detection theory in [Trees71] where the signal s(t) is allowed to be a random process.

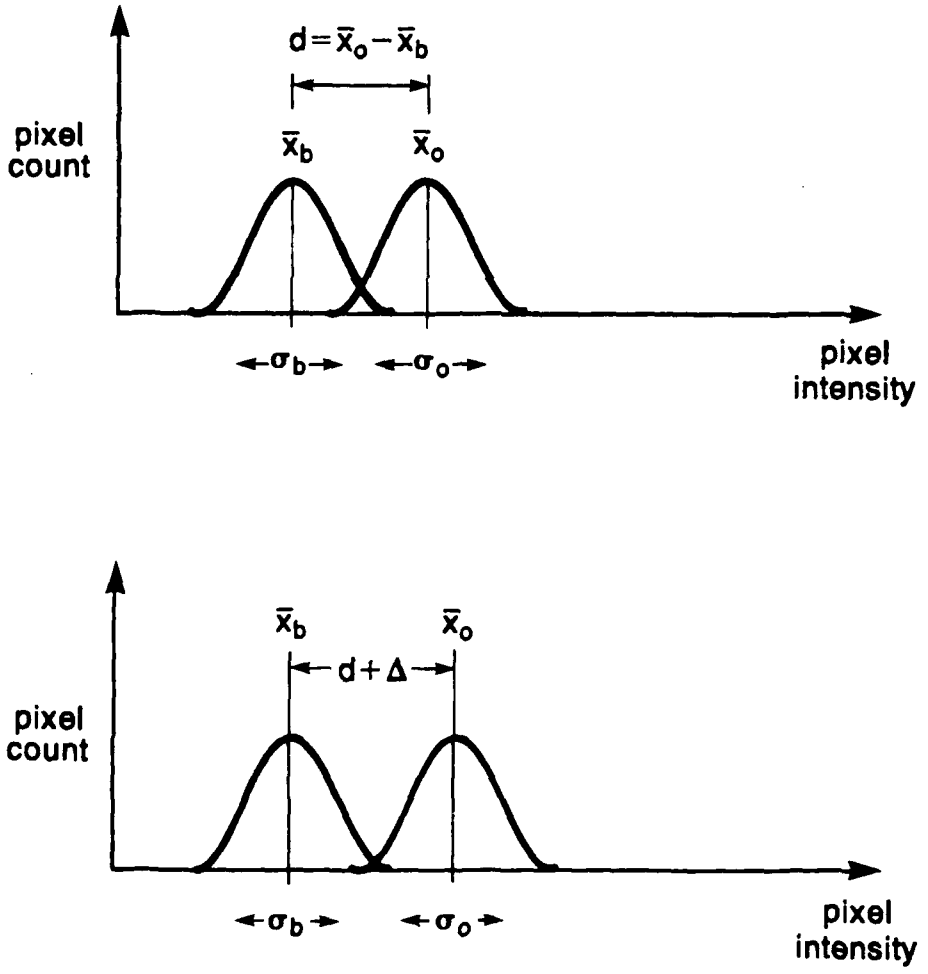


Figure 2.1.12 Change in TIR^2 with change in $(\bar{x}_o - \bar{x}_b)^2$.

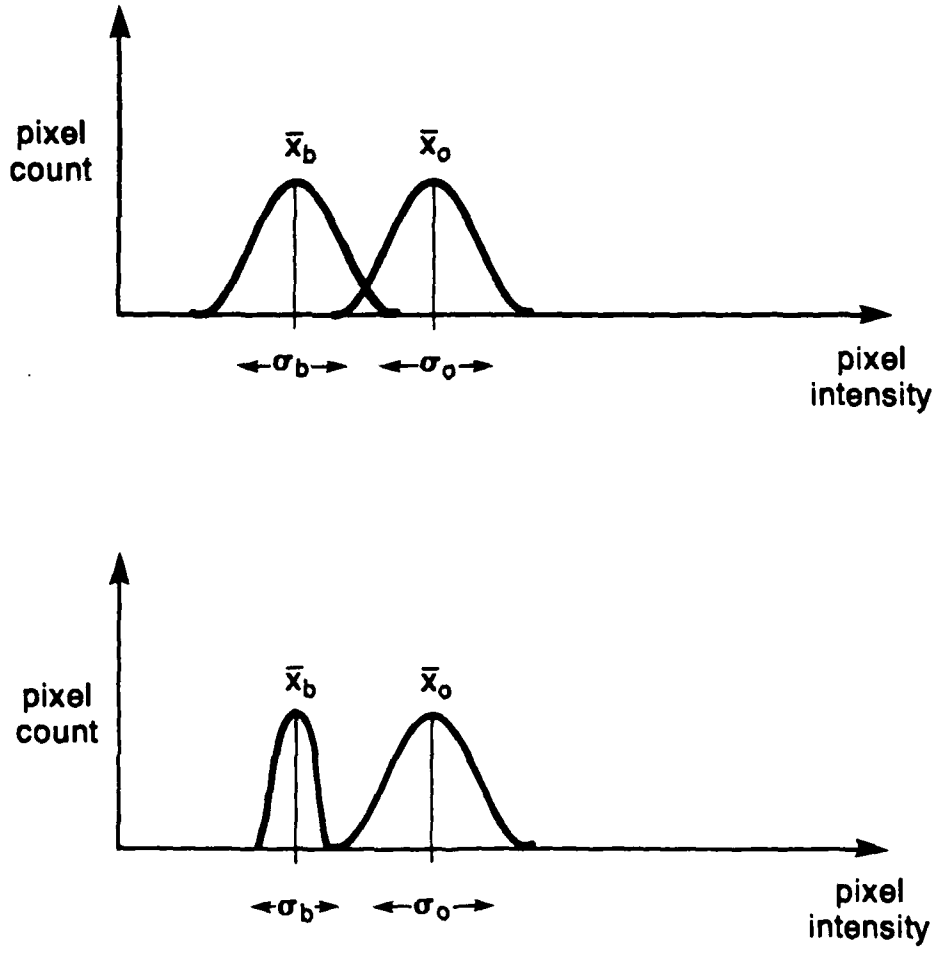


Figure 2.1.13 Change in TIR^2 with change in σ_b.

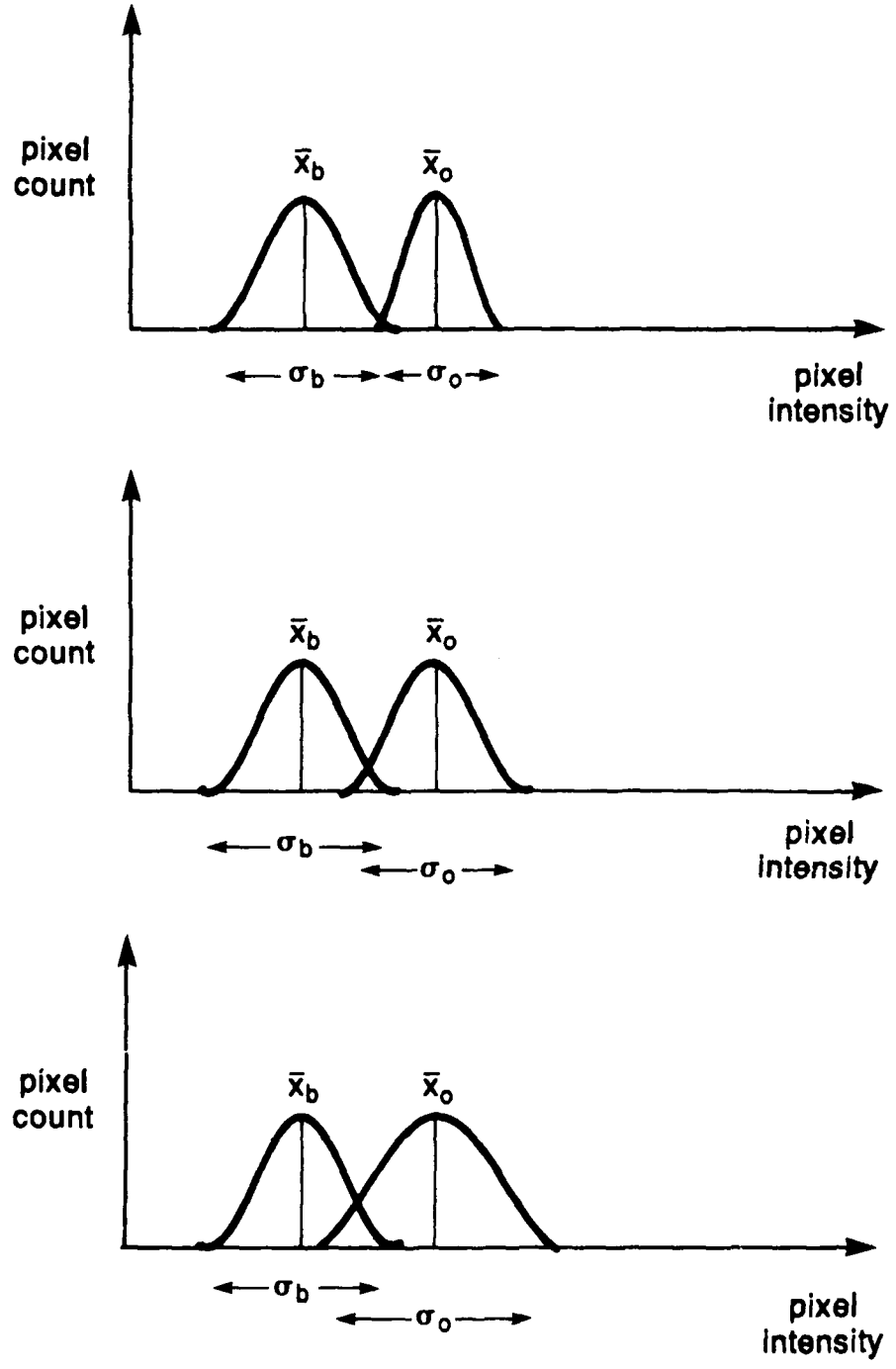


Figure 2.1.14 Distributions for images with same TIR^2 and different complexities.

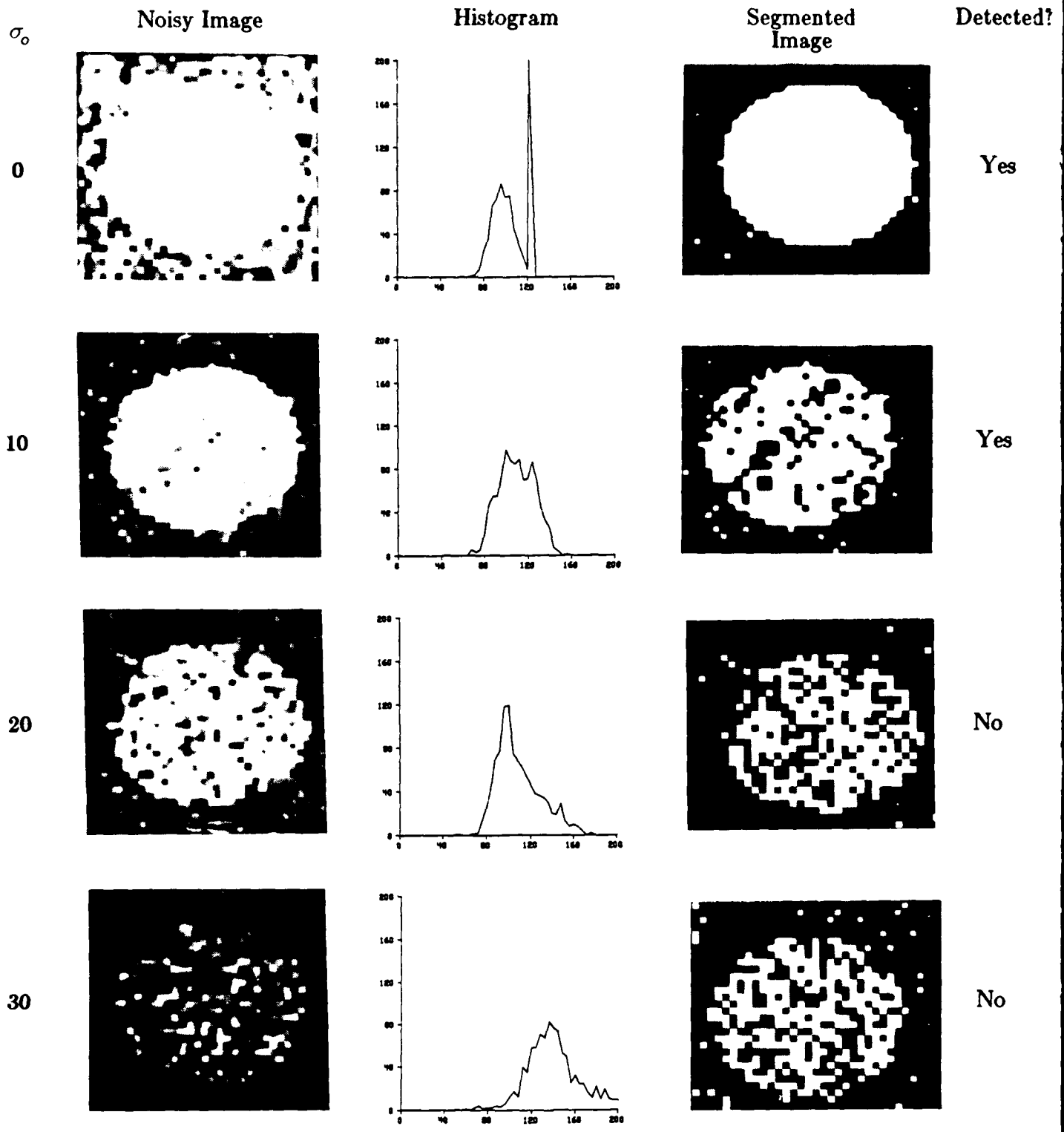


Figure 2.1.15 $\bar{x}$=100, σ_b=10, and $\bar{x}$=125 for each of these images so that the value of TIR^2 for all the four simulated target images in the left hand column is the same and is equal to 6.25. Each image consists of an ellipse embedded in a uniform background. To the composite thus obtained is added uncorrelated noise.

found.* In the middle column of Figure 2.1.15, we have shown the histograms for the gray levels, in the right column are illustrated segmentation outputs. **As is evident from the figure, the detection process completely fails for some of the targets, although they all have the same TIR^2 measures.**

We will now demonstrate similar results with correlated noise. First a few words about how noise patterns with controlled correlations were generated. Suppose $\nu(x,y)$ represents an array of uncorrelated random numbers, which are easily generated by a standard random number generator. If we now construct a new array by using the following recursion

$$\gamma(x,y) = a\gamma(x-1,y) + b\gamma(x,y-1) + \nu(x,y) - ab\gamma(x-1,y-1)$$

we can show [RosKak82] that the resulting noise pattern has the following correlation function

$$R(i,j) = e^{-a|i|-b|j|}$$

This assumes that the noise pattern is zero-mean and of unit variance. This pattern can be multiplied by an appropriate scaling factor to obtain a desired signal-to-noise ratio when the pattern is added to the synthesized target image. In the experiments reported here, we have used a=b=.4 .

In Figure 2.1.16, we have shown results for the case of correlated noise that are similar to those in Figure 2.1.15.

2.1.2.1.2. Characterization of Target Segmentation

The next step in the ATR process, after a target is detected, is to segment the target. ERIM proposes characterizing segmentation with the following *independent* variables.

1. Resolution Cells on Object (RN_0)
2. Expected Resolution Cells on Object (RE_0)
3. Target-Background Interference Ratio Squared ($TBIR^2$)
4. Edge Strength Ratio (ESR)

According to ERIM, the performance of a segmentation algorithm can be measured by plotting segmentation accuracy, A_s, versus these variables. The segmentation accuracy is defined to be the ratio of two factors: the first factor is equal to the intersection of the segmented region for an object and the true image region corresponding to that object; and the second factor is the union of these two regions. The basic idea in the ERIM methodology can be summed up as follows: Let's say we have two target images I_1 and I_2, with the value of $TBIR^2$ for I_2 larger than what

*If approximate range to the target is known, which means if the size of the target one is looking for is approximately known, it is unlikely that one would accept less than about 80 percent of that area from the segmenter in order to declare the target as present.

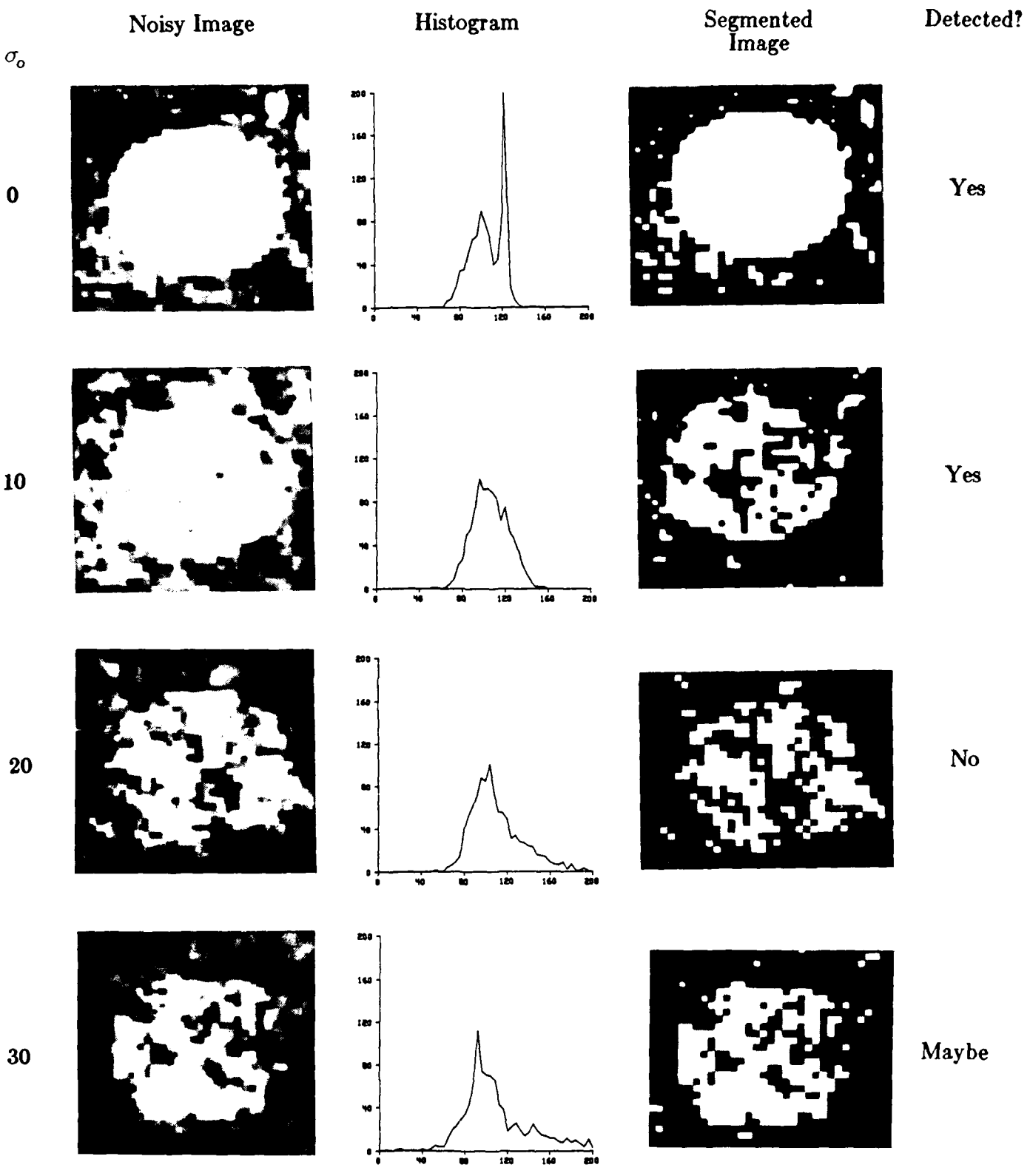


Figure 2.1.16 Same as Figure 2.1.15, except that the noise added to the image is now correlated.

it is for I_1. Then, according to ERIM, the segmentation accuracy for I_2 must be poorer than it would be for I_1. Equivalently, if we have two images of the same $TBIR^2$ value, then it should be possible to segment them both with the same accuracy.

In this report, we will show it is possible to easily construct simulated examples of two different target images of the same $TBIR^2$ value, these two images yielding very different segmentation accuracies. This was done by taking a theoretical look at $TBIR^2$ to see where it might have problems, and then construct examples which exploit the problems.

$TBIR^2$ is defined as:

$$TBIR^2 = \frac{(\bar{x}_o - \bar{x}_b)^2}{\sigma_o \sigma_b}$$

Take for example the histogram in Figure 2.1.17a, which shows a distribution for the background and the object. The $TBIR^2$ measure would predict that an image with such a distribution would be as complex as another image with a *smaller* difference in means $(\bar{x}_o - \bar{x}_b)$ if:

1. The object had a smaller variance as shown in Figure 2.1.17b,
2. The background had a smaller variance as shown in Figure 2.1.17c, or
3. Both the object and the backgrounds has smaller variances as shown in Figure 2.1.17d.

This seems like a reasonable relationship to have for Gaussian distributions, as illustrated in Figure 2.1.17, because the changes always result in distribution in which a given pixel is as likely to be correctly associated with the proper class (i.e. background or object).

One problem with using distributions to characterize an image is that spatial information is lost. Many segmentation errors are caused by targets which do not have uniform intensities. The pixels in the target whose grey scale values are close to the mean of the background can cause the segmenter to split the target into two or more segments.

To illustrate this point consider an synthetic elliptical target with a non-uniform intensity. Figure 2.1.18 show a 3D plot of such a target, plotting the grey value on the z-axis. Figure 2.1.19 shows four noisy targets which were created by adding uncorrelated Gaussian noise to the target and the background so that the images have the same $TBIR^2$. The noise statistics for the targets are shown in Table 2.1.21. The segmentation procedure used is the same as used in Figure 2.1.15 of Section 2.1.2.1.

The thresholded images show that for $\bar{x}_o - \bar{x}_b$ equal to 20 and 30, the image is split into two segments. For $\bar{x}_o - \bar{x}_b$ equal to 40 and 50, the image is left intact. Figure 2.1.19 therefore shows that two images with the same $TBIR^2$ can have very different segmentation accuracies.

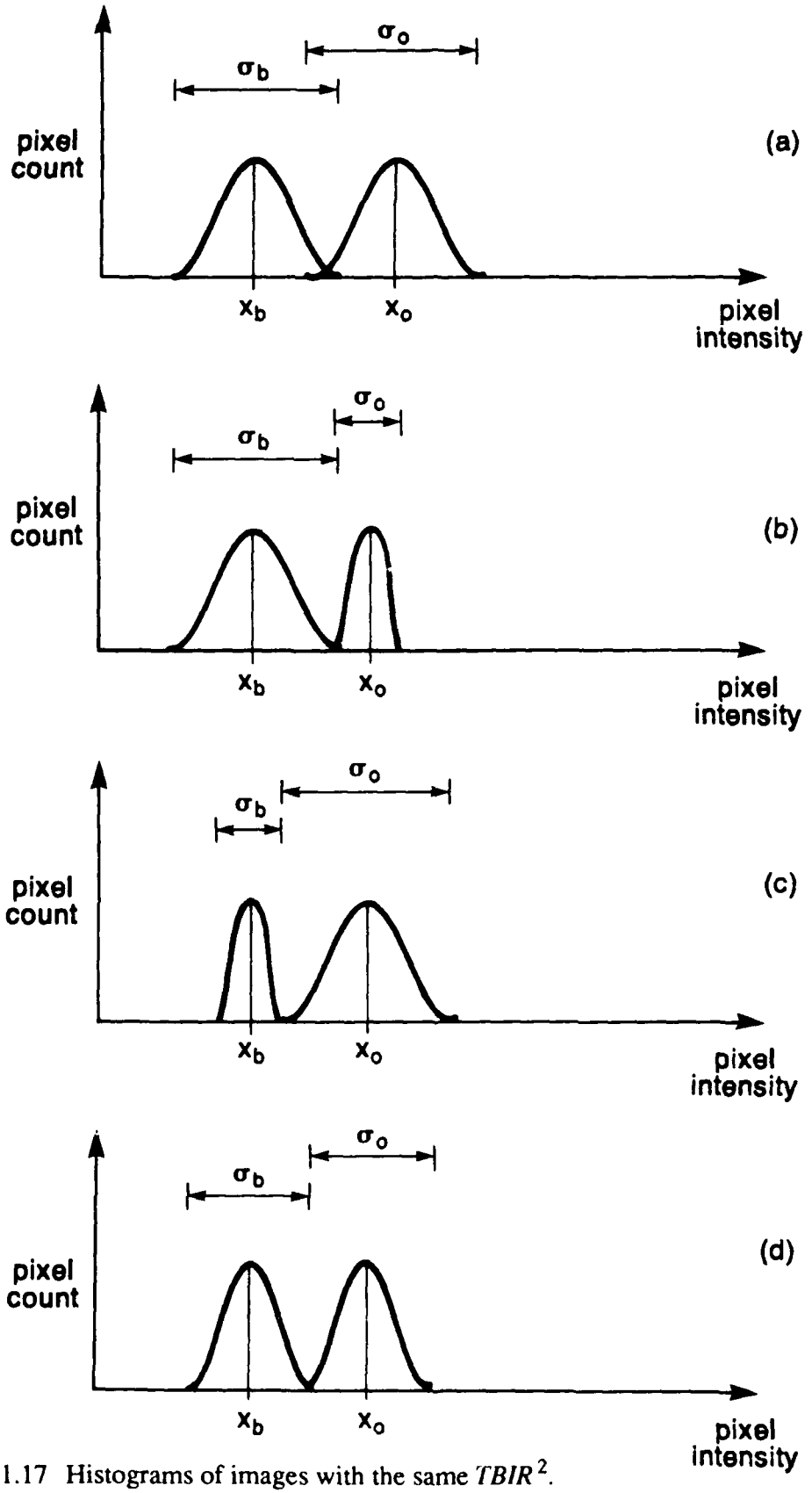


Figure 2.1.17 Histograms of images with the same $TBIR^2$.

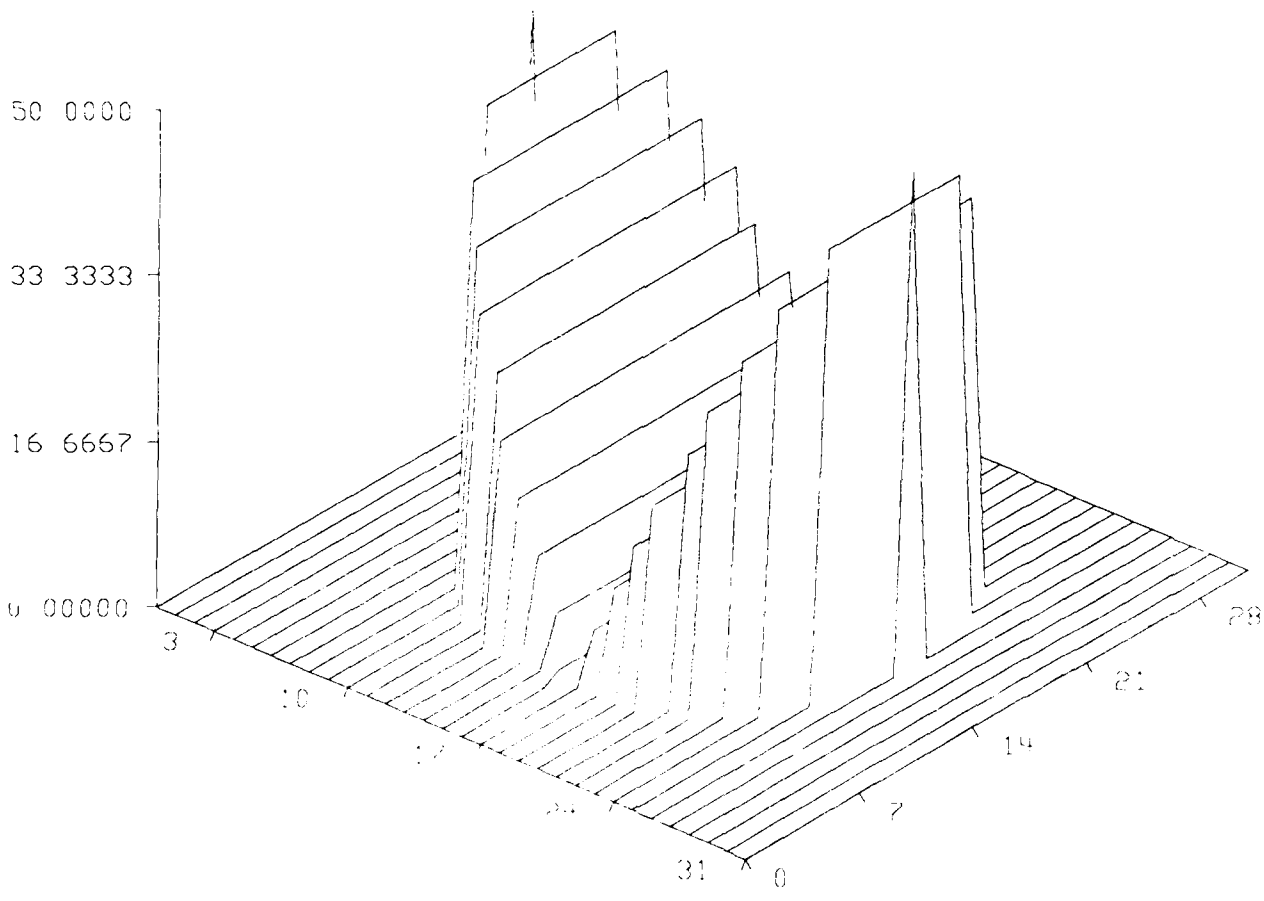


Figure 2.1.18 A 3D plot of a synthetic elliptical target with the grey values plotted on the z-axis.

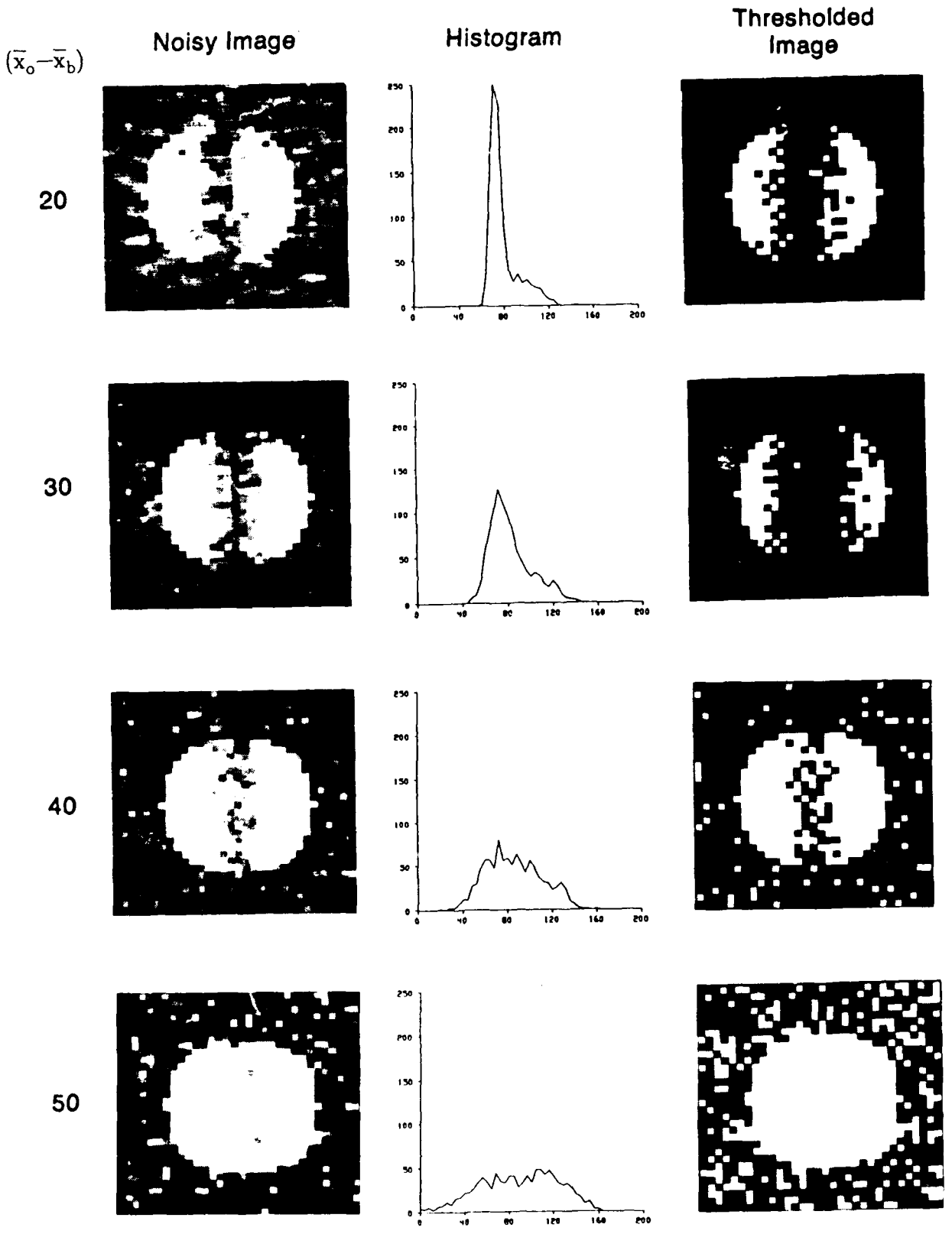


Figure 2.1.19 Four synthetic targets with the same $TBIR^2$ and different segmentation accuracies. See Table 2.1.21 for target and background statistics.

Table 2.1.21 Target and background statistics for Figure 2.1.19.

$\bar{x}_o - \bar{x}_b$	Target		Background		$TBIR^2$
	$\bar{x}_o$	σ_o	$\bar{x}_b$	σ_b	
≈20	94.277	14.9373	74.3395	4.2310	6.29000
≈30	104.738	15.0986	74.4031	9.5723	6.36703
≈40	114.246	14.6105	74.5912	16.9540	6.34826
≈50	125.025	14.3981	74.2249	28.4308	6.30434

2.1.2.1.3. Conclusions

The conclusions to be drawn from these studies is that simple metrics such as TIR^2 and $TBIR^2$ may measure the complexity of an image for the purpose of detection or segmentation **some of the time.** However, it is easy to construct examples which show they **completely fail to measure image complexity**. Therefore there are many cases where TIR^2 and TBI say nothing at all about the process of detection or segmentation.

2.1.2.2. Preliminary Work in the Formulation of New Metrics

Metrics have been devised for the systematic characterization of the algorithms in the ATR process. The approach to characterization has been to divide the ATR process into three steps, detection, segmentation, and classification. Although different metrics for each step have be proposed, the previous sections have shown that these metrics don't work for all images. We propose that each step of the ATR process should be broken down into less general algorithms and metrics be devised for each of the algorithms. Experience gained in designing metrics for the less general algorithms could then provide insight on how to design a metric for the more general algorithms (if such a metric can be found). The following subsection proposes a new metric for characterizing a threshold based segmenter.

2.1.2.2.1. A Metric for Characterizing Threshold Based Segmenters

The most critical step in a threshold based segmenter is the selection of the threshold value (or values if it is a more sophisticated segmenter). Our first attempt at designing a threshold based segmenter assumes that the complexity of an image (with respect to segmentation) is related to the number of threshold values which will give a certain segmentation accuracy*. Some images are easy to segment with a threshold based segmenter because there are many threshold values which will give a good segmentation accuracy. Other images will have few, if any, threshold values which will give good accuracy and are therefore difficult to segment. The segmentation threshold (ST) metric proposed here measures the segmentation complexity of an image with respect to a threshold based segmenter by measuring the number of threshold values which will result in a certain segmentation accuracy.

The ST metric works like this: The image to be characterized is thresholded at a value x by assigning all pixels below x to the background and all pixels equal to and above x to the target. The largest region is found and all other regions are deleted. (A region is defined to consist of a group of pixels that are four-connected to each other.) The segmentation accuracy (A_S) of the remaining region is computed relative to a ground truth image. The above process is repeated for all possible threshold values. Finally A_S vs. the threshold value is plotted. Figure

*This of course assumes that the images being characterized have the same number of quantization levels. For now, this is a valid assumption since the data we have been given have all used eight bits per pixel.

2.1.20 shows the A_S vs. threshold plots in the senter column for the same images as in Figure 2.1.19. The x-axis is the value of the threshold and the y-axis is the segmentation accuracy. The wider the peak, the more thresholds which will give a certain accuracy. We can see that the $(\bar{x}_o - \bar{x}_b) = 20$ target is difficult to segment using a threshold based segmenter because there is only one threshold value that will give a segmentation accuracy greater than 70%. The $(\bar{x}_o - \bar{x}_b) = 30$ target is easier to segment because there are several threshold values which will give greater than 85% segmentation accuracy. The remaining two images are increasingly easy to segment because as the plots show, there are more values that will give a greater than 85% segmentation accuracy. The plots in the right column of Figure 2.1.20 show the number of thresholds vs. A_S. The plots show that there are:

1. very few values (only 1) for the $(\bar{x}_o - \bar{x}_b) = 20$ image,
2. $\approx$15 values for the $(\bar{x}_o - \bar{x}_b) = 30$ image,
3. $\approx$20 values for the $(\bar{x}_o - \bar{x}_b) = 40$ image, and
4. greater than 30 values for the $(\bar{x}_o - \bar{x}_b) = 50$ image,

which give a segmentation accuracy of greater than 75%. Therefore the images increasingly easier to segment (using a threshold based segmenter) as you move from the top image to the bottom image of Figure 2.1.20.

2.1.2.2.2. Conclusions

The ST metric correctly predicted which of the synthesized images are more difficult to segment. These were the same images that fooled the $TBIR^2$ measure. The major flaw with this method is that it tells us how many thresholds will give us a certain segmentation accuracy, but it won't tell us how hard it is to find those thresholds. For example the ST metric will be fooled if there is no overlap in the pixel values for the foreground and the background. In such a case it is very easy to pick a threshold. For example, suppose all the background pixels for a given image have the value 10, and all the target values are 200, then any threshold between 10 and 200 would give perfect segmentation. Consider a second image with the same background values and the target values equal to 20. It is as easy to segment as the previous image, however our measure would say that it is harder to segment because there are fewer thresholds that will give the proper segmentation.

Although the ST metric is based a single threshold value, it works well for the synthesized images. Future work will include:

1. testing the ST metric on real FLIR images to see how well it measures segmentability, and
2. looking into making it a more robust measure by basing it on a Bayes Classifier for minimal error. Such an approach views segmentation as a problem of classifying each pixel as either background or target. This makes it possible to find optimal thresholds.

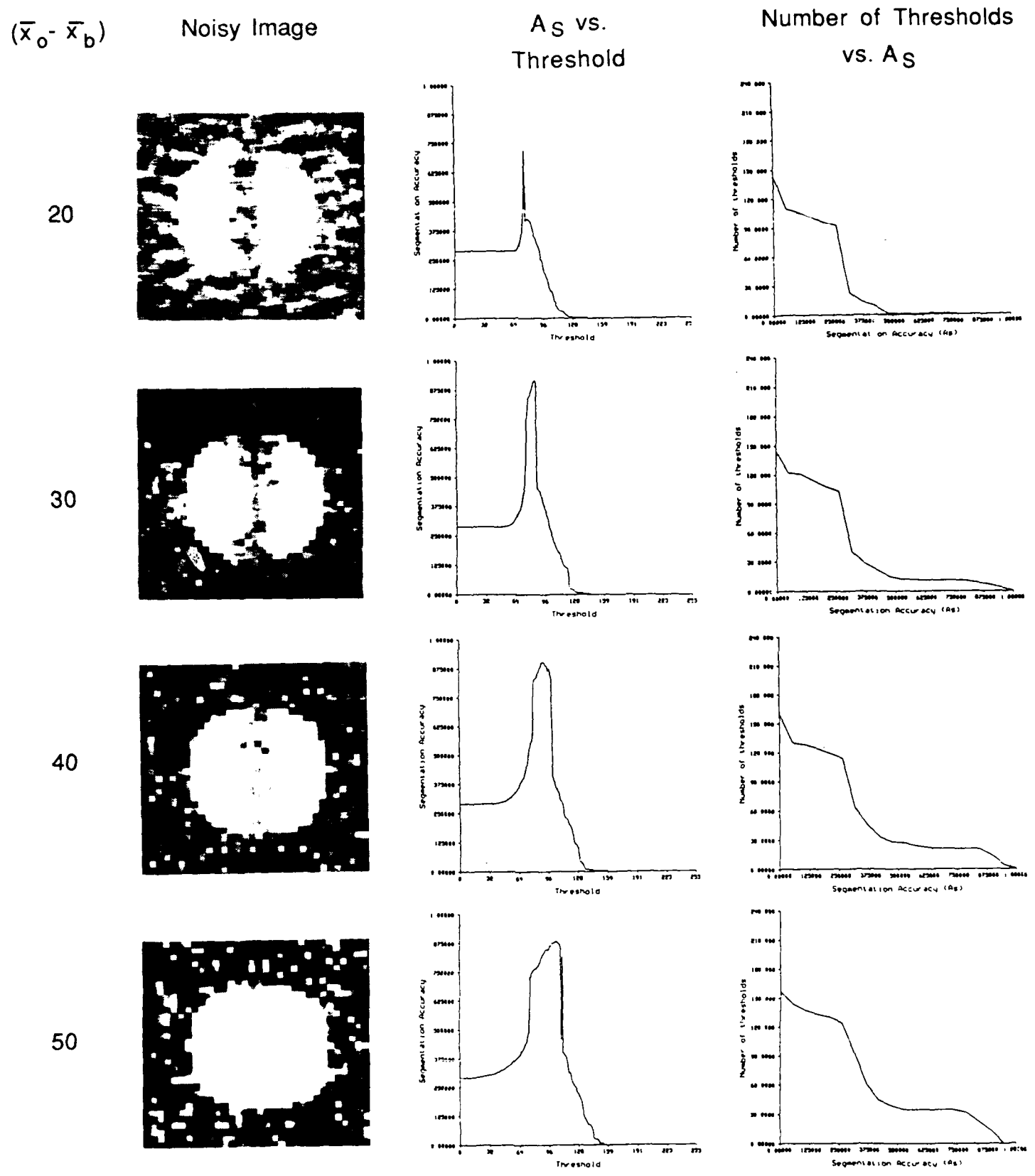


Figure 2.1.20 Plots showing the ST metric for the same images as in Figure 2.1.19.

2.2. TWO FLIR SEGMENTERS

Although most of the effort this time has been in processing LADAR data, work is still continuing in FLIR processing. We believe that one must be able to perform reasonable low level processing on a given sensor *before* trying to fuse it with other sensors. This section presents our experiences with using two different segmenters (edge guided threshold, and tree traversal) on FLIR images. These experience should prove to be useful when fusing FLIR with LADAR.

2.2.1. An EGT Based Segmenter for FLIR Data

A tunable edge-guided threshold-based (EGT) segmenter was developed for the automatic extraction of target silhouettes from FLIR images. The segmenter was designed to be part of an intelligent target recognizer. It is "tunable" in that one may specify the expected brightness range of the targets and the expected target areas. The expected brightness is specified as a percentage range. For example, from time of day and weather information, and possibly knowledge of how active the targets have been, we might expect the targets to consist of the hottest 10% to 20% of the pixels in the image. This percentage estimate also depends on the expected size of the targets relative to the image size. The expected target area (in pixels) is also specified as a range of values, and may be calculated from range information. Once potential target regions have been extracted, the segmentation results may be evaluated by an expert system and the segmenter called again with new parameters.

2.2.1.1. The Algorithm

The first step in the segmentation process is to perform edge detection on the original gray scale FLIR image. The edge detection process produces candidate object edge pixels by adaptively thresholding the Sobel edge magnitude image. The original image is convolved with the horizontal and vertical Sobel edge masks to produce the horizontal and vertical edge gradients (**H** and **V**), as illustrated in Figure 2.2.1. The absolute values of **H** and **V** are then used to calculate the edge magnitude **M**. The mean value of **M** over the edge magnitude image is computed, and an edge threshold of 2.25*Mean applied to **M** produces the final edge image.

Once edge pixels have been found, histograms of the gray scale values of the edge and nonedge (object) pixels in the original FLIR image are computed separately for the purpose of selecting a threshold to use to segment the image. This is preferable to computing the histogram of the entire original image because edge pixels typically have gray values between those of the regions that they separate, and so they tend to make the modes of the regions blend together by filling in the valley between them. This makes threshold selection more difficult. Computing separate histograms for object and edge pixels should result in more distinct (nonoverlapping) modes in the object histogram, and peaks in the edge histogram corresponding to the valleys between these modes (see Figure 2.2.2). The gray values at which these resulting peaks in the edge pixel histogram or deeper valleys in the object pixel histogram occur are good

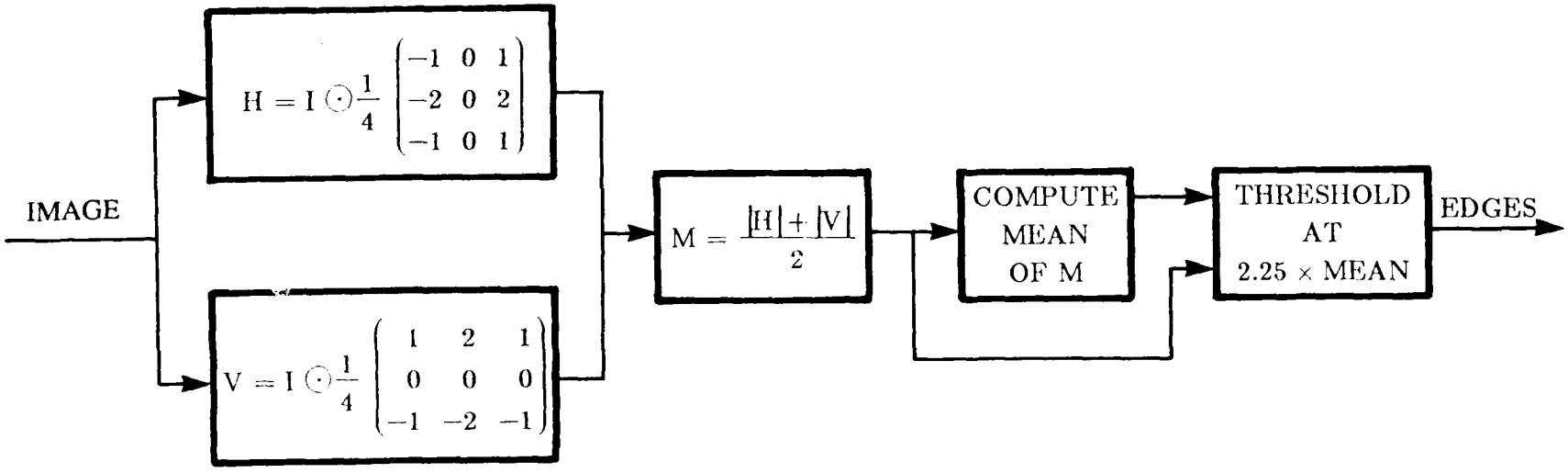


Figure 2.2.1 Edge detection process block diagram.

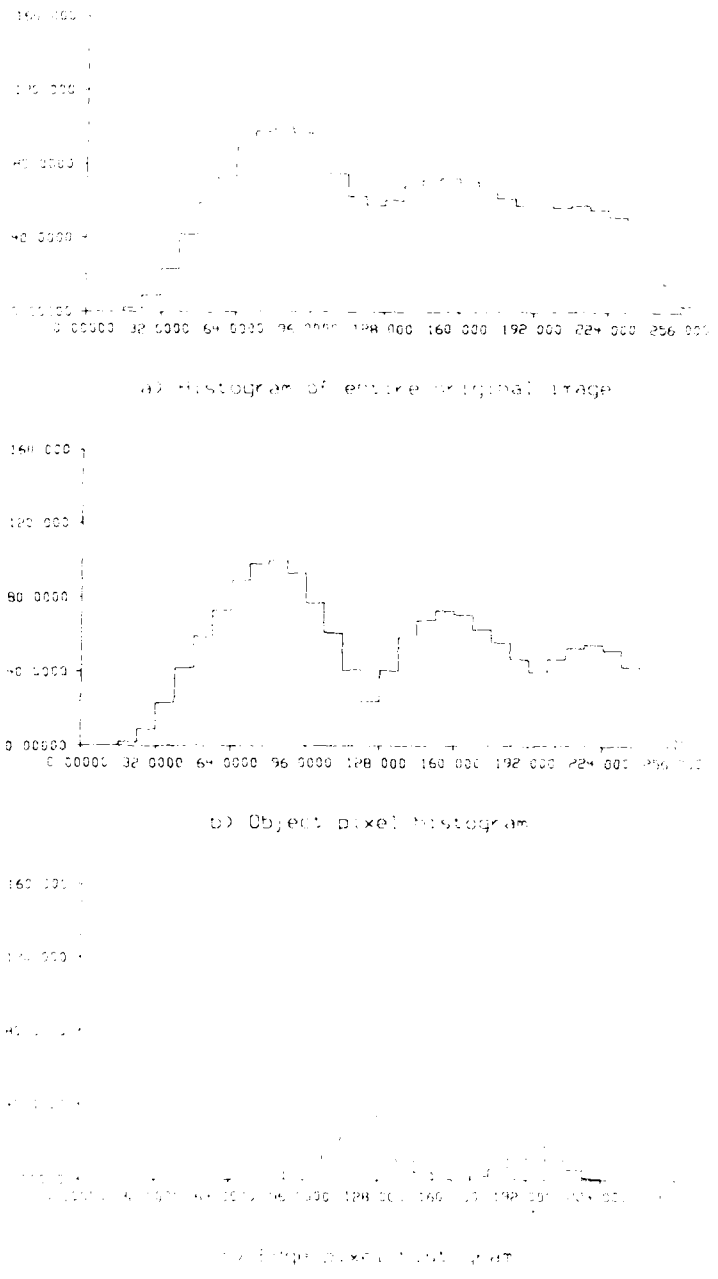


Figure 2.2.2 Illustration of effect of histogramming object and edge pixels separately.

candidates for thresholds for segmenting the image.

The image threshold is then selected as follows. Suppose the lower and upper brightness percentage limits are *LPCT* and *UPCT* (e.g. to specify that a target should consist of the brightest 10% to 20% of the image pixels, *LPCT* =0.10 and *UPCT* =0.20). Upper and lower pixel count limits are then calculated as *LOWER=LPCT*X*Y* and *UPPER=UPCT*X*Y*, where *X* and *Y* are the image dimensions. One then starts at the rightmost object histogram bin (corresponding to the brightest gray values) and works towards the left, keeping a running sum of the number of pixels in each bin. The bins for which the running sum first exceeds *LOWER* and *UPPER* are marked, thereby specifying the interval of interest in the histogram in which we expect to find a good threshold. We select as our threshold the gray value of the bin with the minimum value in this interval. This value should correspond to the deepest valley in the object histogram in the interval of interest.

The threshold is then applied to the original image, with pixels whose value is less than the threshold being set to zero and the others retaining their original value. A median filter may be applied to the result in order to reduce some of the border noise and holes usually accompanying a threshold segmentation. Whether or not this filter is applied depends on the expected target size. The filter would not be applied if the target was small enough to become disconnected by it.

Finally, connected component labeling is performed and component areas are calculated. The target area limits are *AMIN* and *AMAX*, and are specified at the same time as *LPCT* and *UPCT*. Note that these target area limits have nothing to do with *LOWER* and *UPPER*, which were used to find a proper image threshold. Only those components within the specified area limits are kept as possible target silhouettes. Since gray level information was kept when generating the silhouettes, the label and final silhouette images are the only ones necessary for calculating both binary and gray-level features.

2.2.1.2. Results

Here we present results obtained by running the EGT segmenter on both simulated and actual FLIR images. Figure 2.2.3 shows the histograms for the entire image, the edge pixels only, and the nonedge (object) pixels only for a tank from our Eglin Turntable data set (file *eglin15*). Note that histograming the object and edge pixels separately helped bring out a small valley in the object histogram near the bin corresponding roughly to pixels with gray value 96. The brightness range specified was 10-20%, which was found to correspond to the gray level range 84-95, and the area range was 500-2500 pixels. As indicated, a threshold of 94 was chosen automatically. Figure 2.2.4 shows the original FLIR image, and intermediate and final segmentation results.

A simulated FLIR image of a tank was obtained by digitizing a blurry image of a dark tank model on a light background, resampling it from 512 by 480 down to 128 by 120, and inverting it. The EGT segmenter was run on this image with brightness range 10-20% and area range

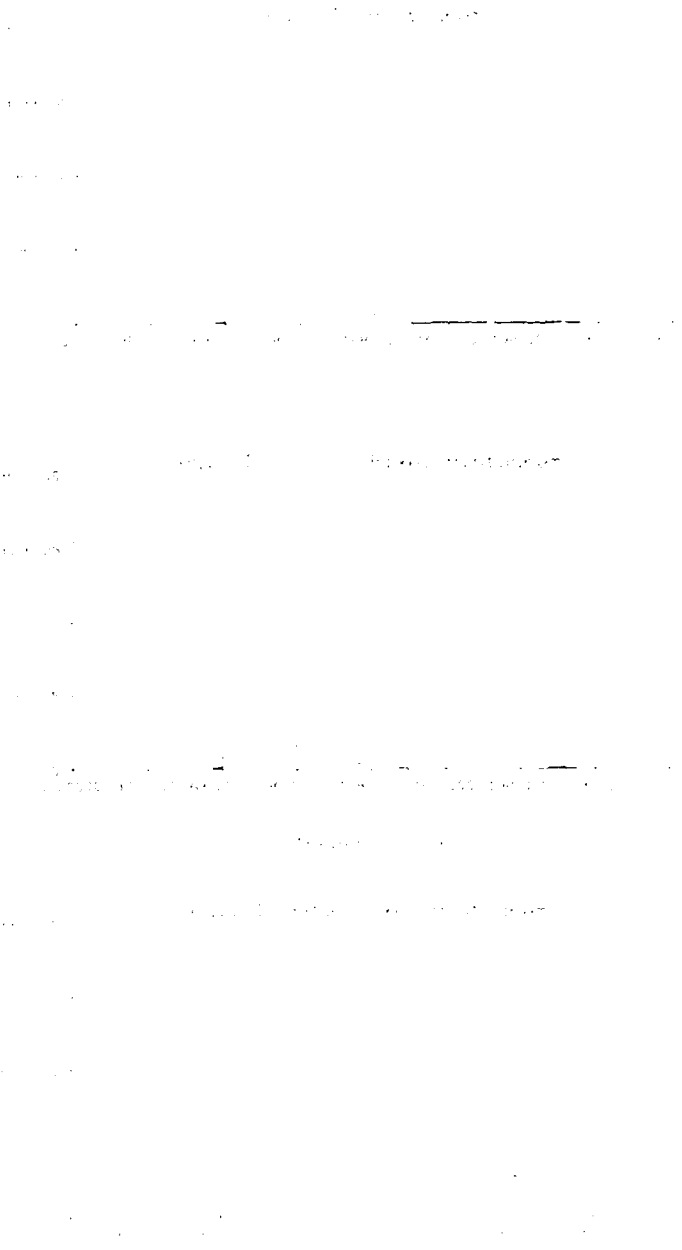
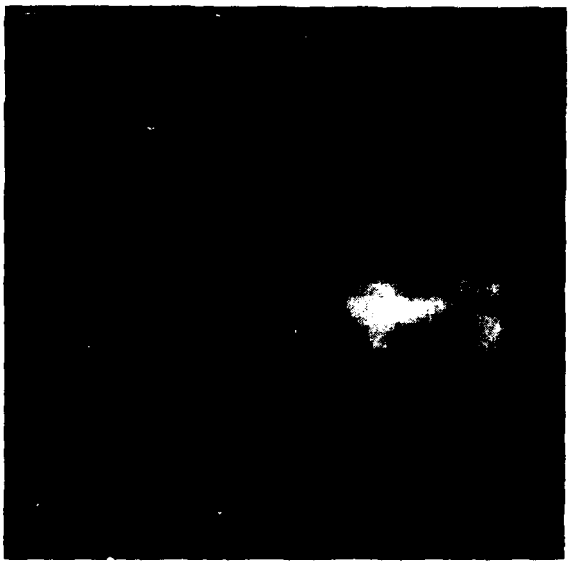


Figure 2.2.3 Histograms for FLIR image from Eglin Turntable data set.



original



sobel edges



thresholded



segmented

Figure 2.2.4 Original FLIR image from Eglin Turntable data set and processing results.

500-2500. The brightness range was found to correspond to gray level value range 148-199, and a threshold value of 182 was chosen automatically. Figure 2.2.5 demonstrates the deepening of the valley between the modes of the object histogram due to histograming the edge pixels separately, and Figure 2.2.6 contains the original, intermediate, and final result images.

2.2.1.3. Comparison with Hughes Segmenter

In previous reports we showed composite images of 50 APC's, 50 tanks, and 50 trucks used for an interclass separation experiment. These targets were hand extracted by first enclosing the targets within tight rectangular windows and then within each window using the complex likelihood segmenter described in the Bandwidth Reduction and Intelligent Target Tracking (BRITT) Phase One Final Report by Hughes Aircraft Company's Electro-Optical & Data Systems Group [Hughes84]. The 150 target images used were from the BRITT data set.

We also ran the EGT segmenter on this data set. The brightness range specified was 0-10% and the area range was 100-2500 pixels. The same limits were used for all 150 images (the segmenter was NOT tuned for each image). Even with this disadvantage (remember the Hughes segmenter was provided windows around the targets via human input which guarantee the target to be in the center of the window and at least a five pixel border between the target and the window), our simple segmenter produced results comparable to those achieved by Hughes. Figures 2.2.7, 2.2.10, and 2.2.13 are composites of the original FLIR images, Figures 2.2.8, 2.2.11, and 2.2.14 are Hughes segmentation results, and Figures 2.2.9, 2.2.12, and 2.2.15 are EGT results. Because the same limits were used for the entire data set (which contained targets at several ranges) sometimes the EGT silhouettes were slightly larger or smaller than they should be. Also, some of the target images with low target-to-background contrast were broken up into several components because a median filter was applied to clean up some of the threshold-produced noise and make the silhouette borders appear better. If a component of such a broken up target was below the lower area limit, it was classified as an invalid target region and thrown away. These were the two chief causes of the few poor EGT segmentation results.

2.2.1.4. Summary

We have presented an edge guided threshold based segmenter for FLIR images. Although it is much simpler than the segmenter used by Hughes, its performance is comparable to that of the Hughes' segmenter, at least for the images that both were tested on.

One of the notable features of our segmenter is that it is "tunable", that is, one can specify the expected brightness (or darkness) range of the target. This tunability will play an important role in our production system based approach to fusing LADAR and FLIR data[AndKak87]. As data is gathered from both sensors, hypotheses will be made as to what the target will look like. The FLIR segmenter can then be tuned to try to accurately segment the target and then the hypotheses can be verified.

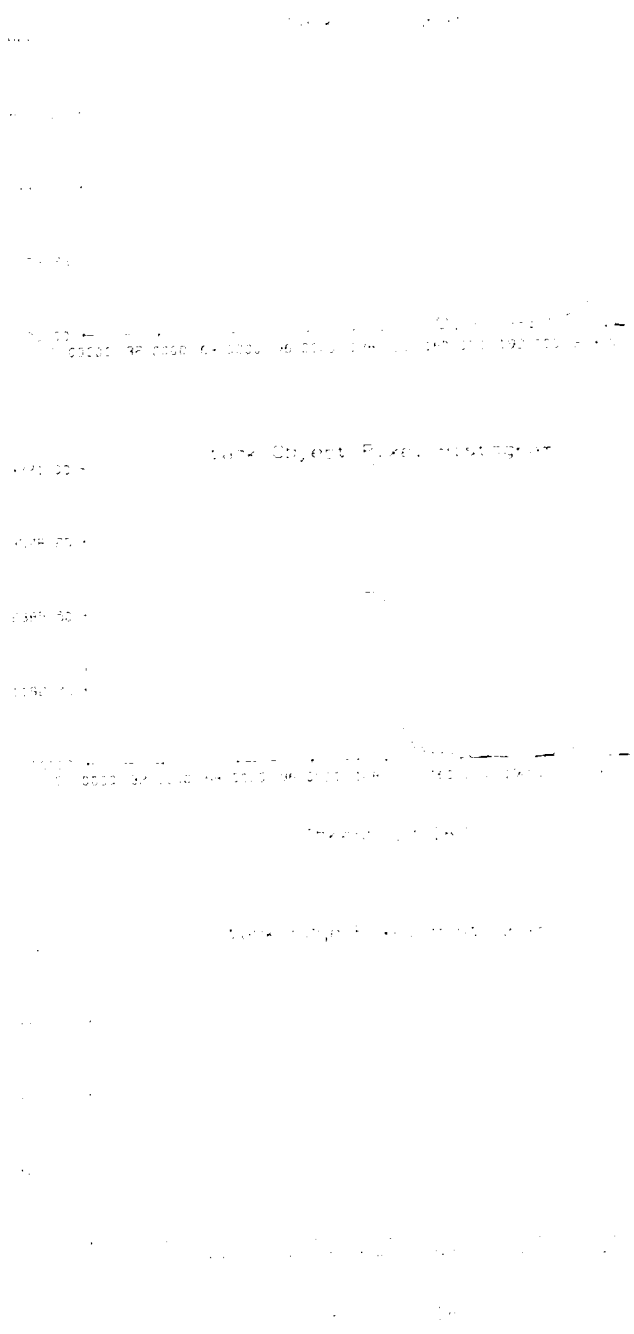
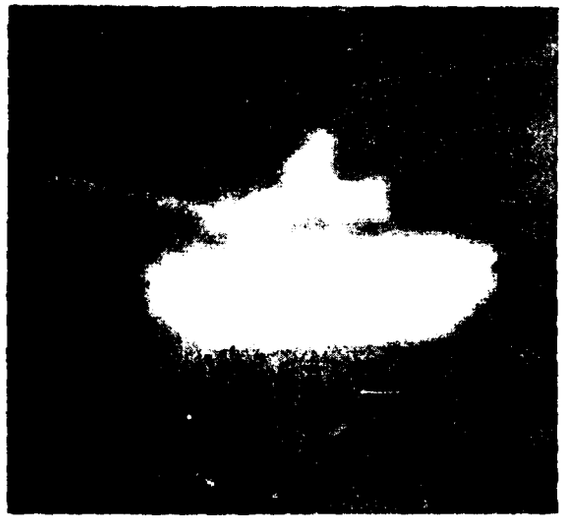


Figure 2.2.5 Histograms for simulated FLIR image of a tank.



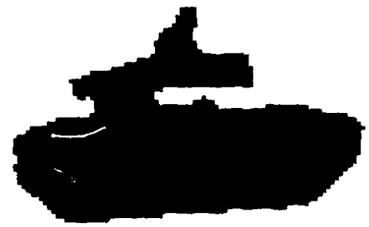
original



sobel edges



thresholded



segmented

Figure 2.2.6 Simulated FLIR image of a tank and processing results.

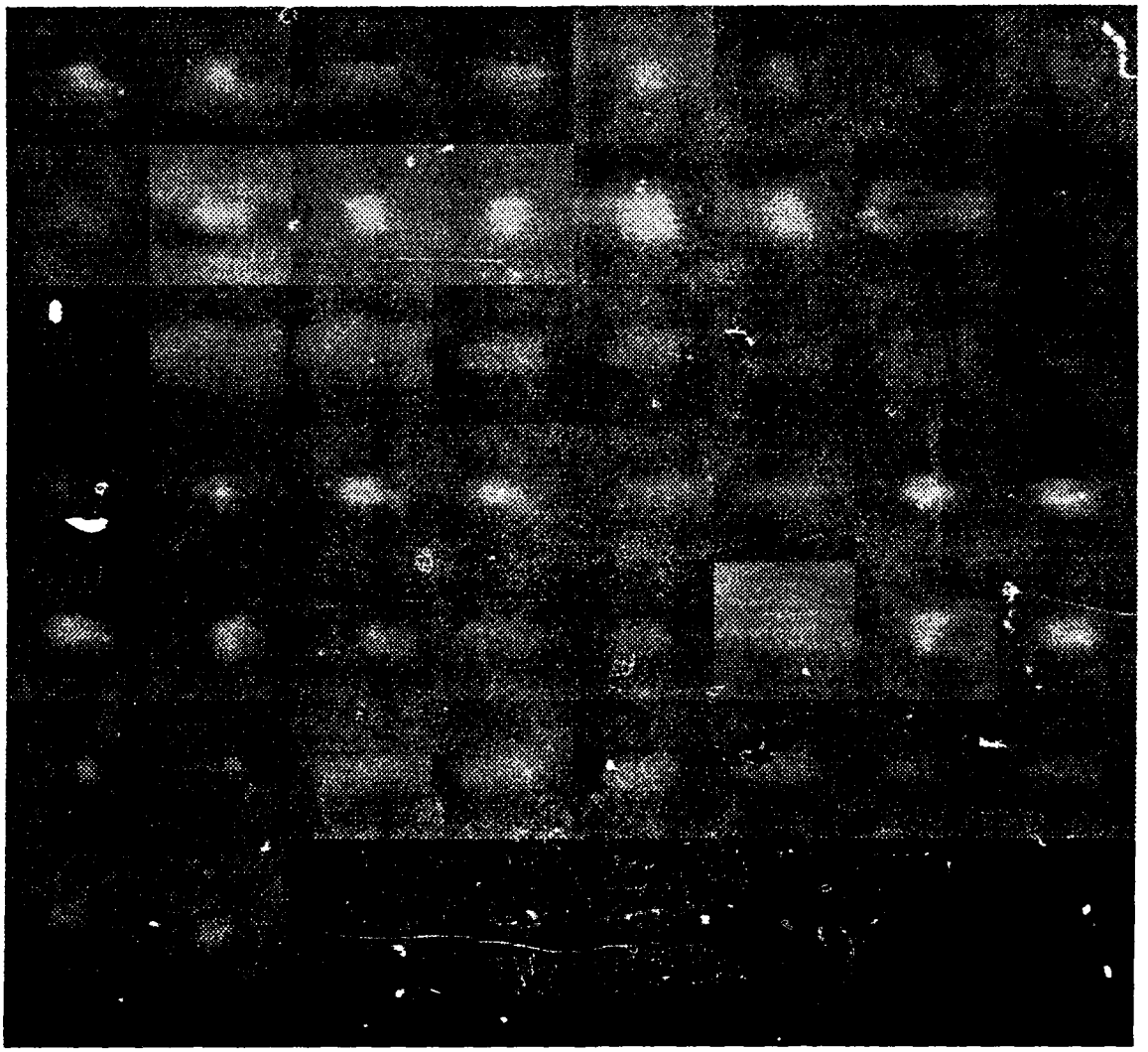


Figure 2.2.7 Composite of 50 original FLIR images of APC's from the BRETT data set.

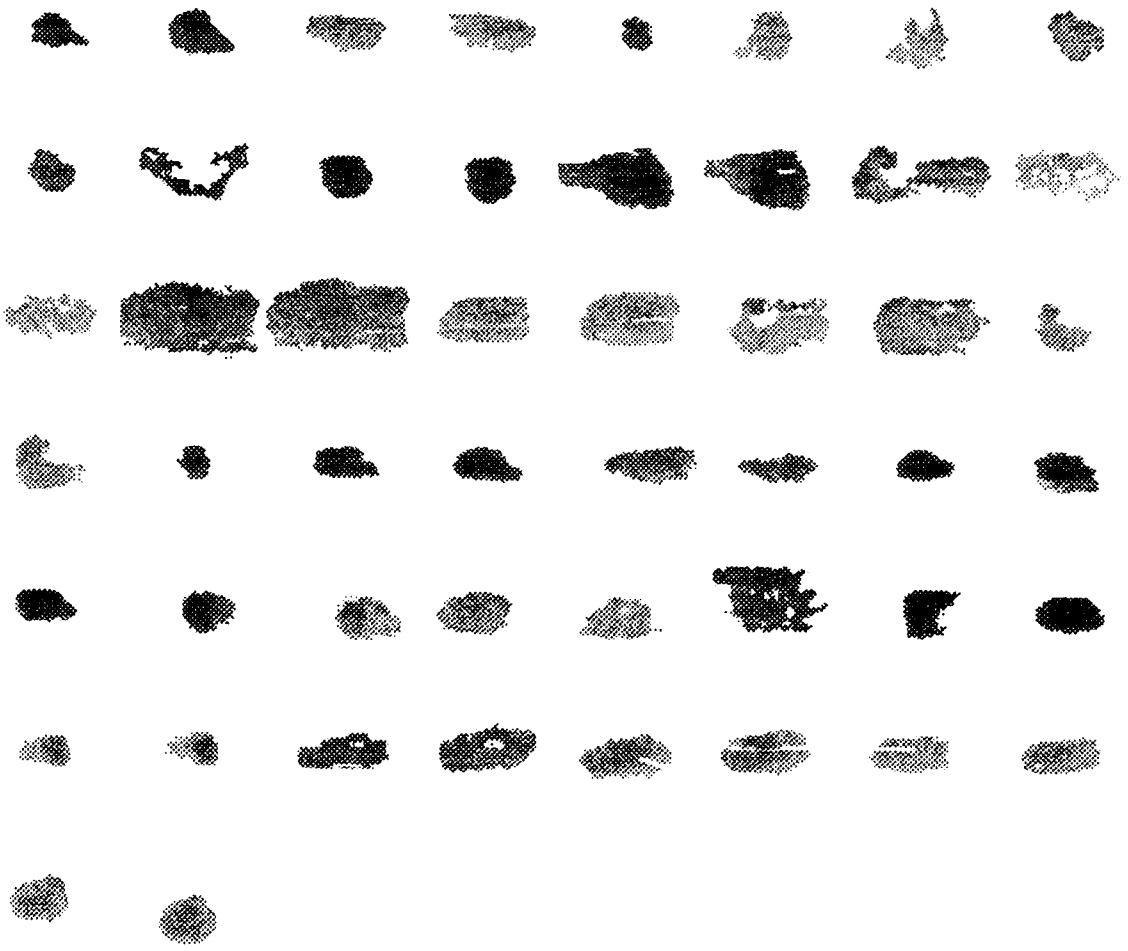


Figure 2.2.8 Composite of the APC segmentation results from the Hughes likelihood segmenter.

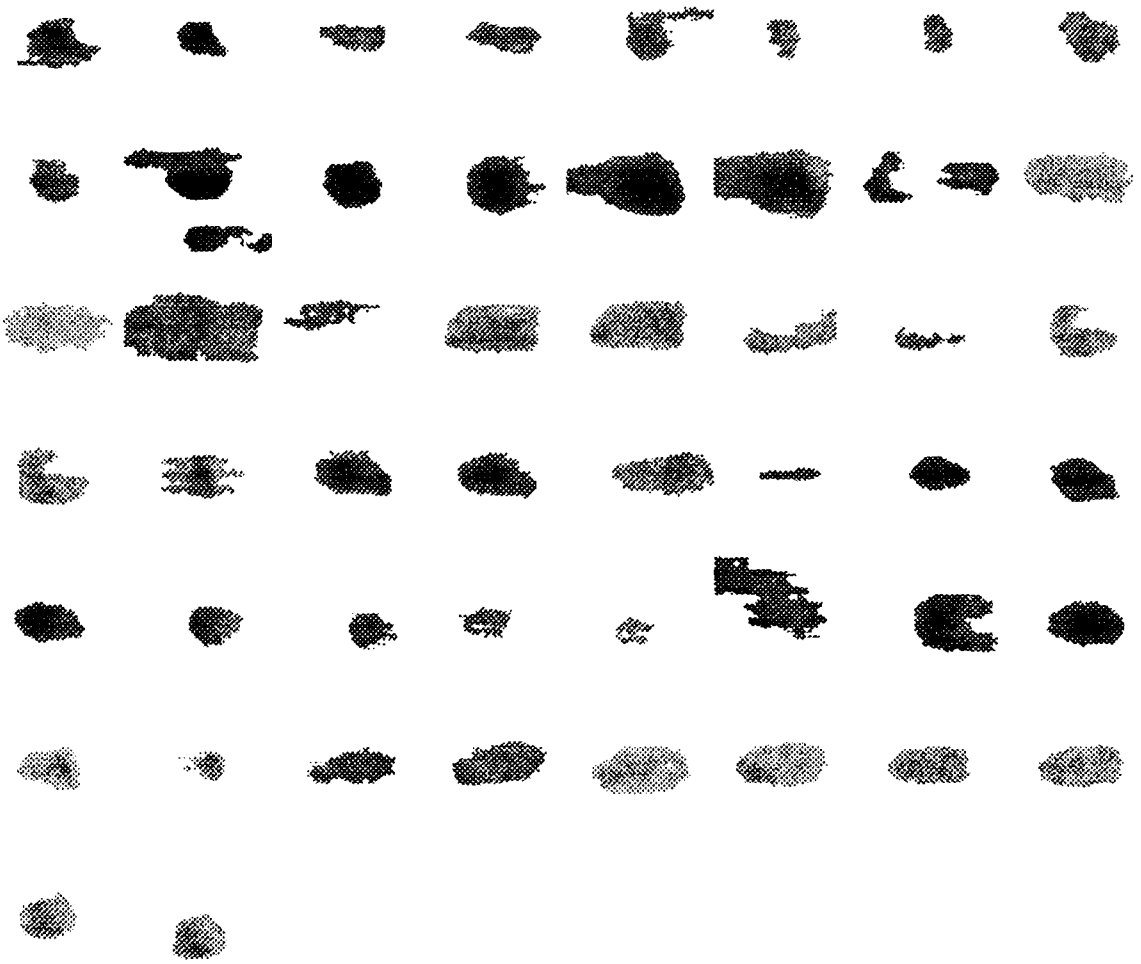


Figure 2.2.9 Composite of the APC segmentation results from the edge-guided threshold segmenter.

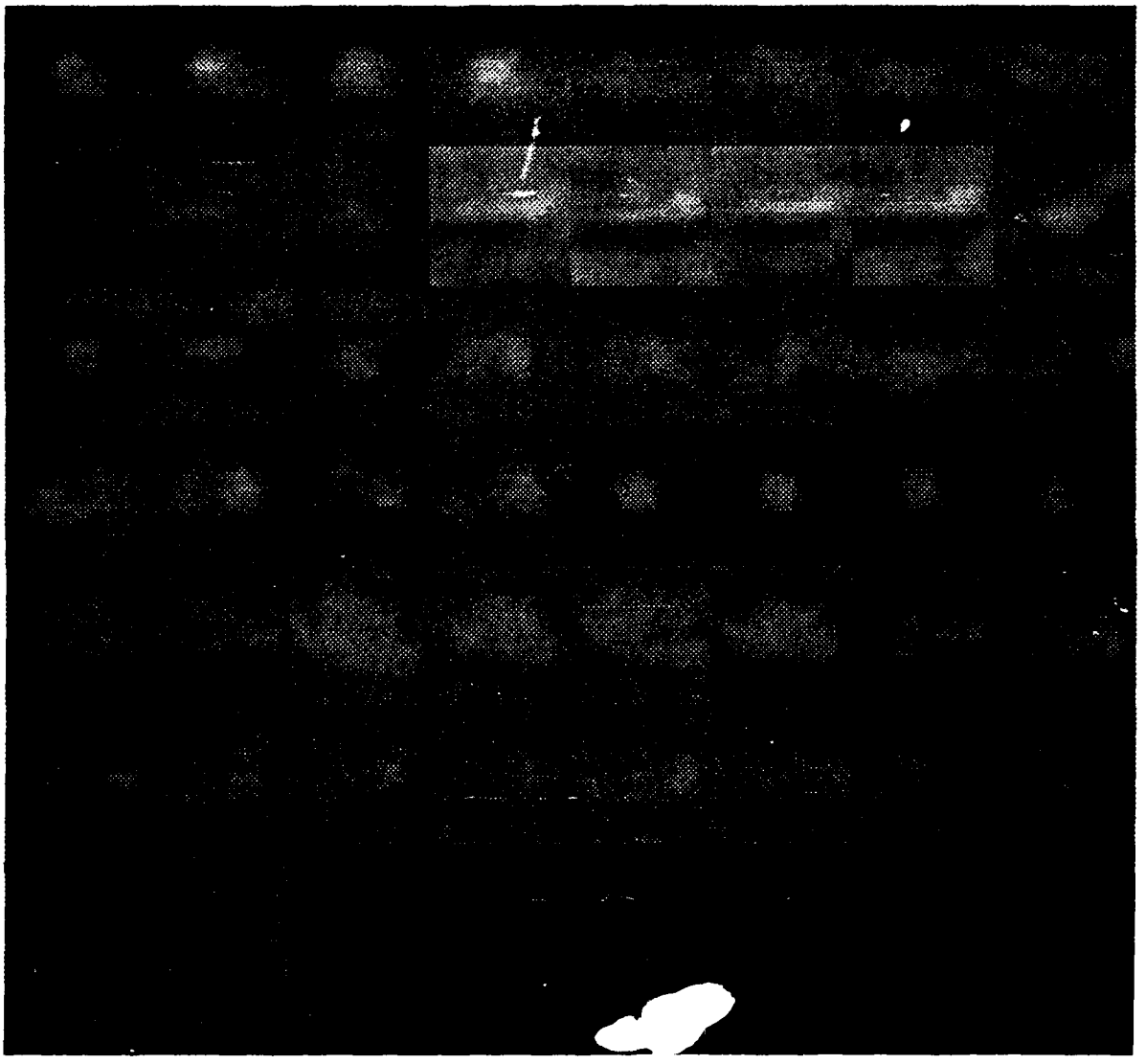


Figure 2.2.10 Composite of 50 original FLIR images of tanks from the BRITT data set.

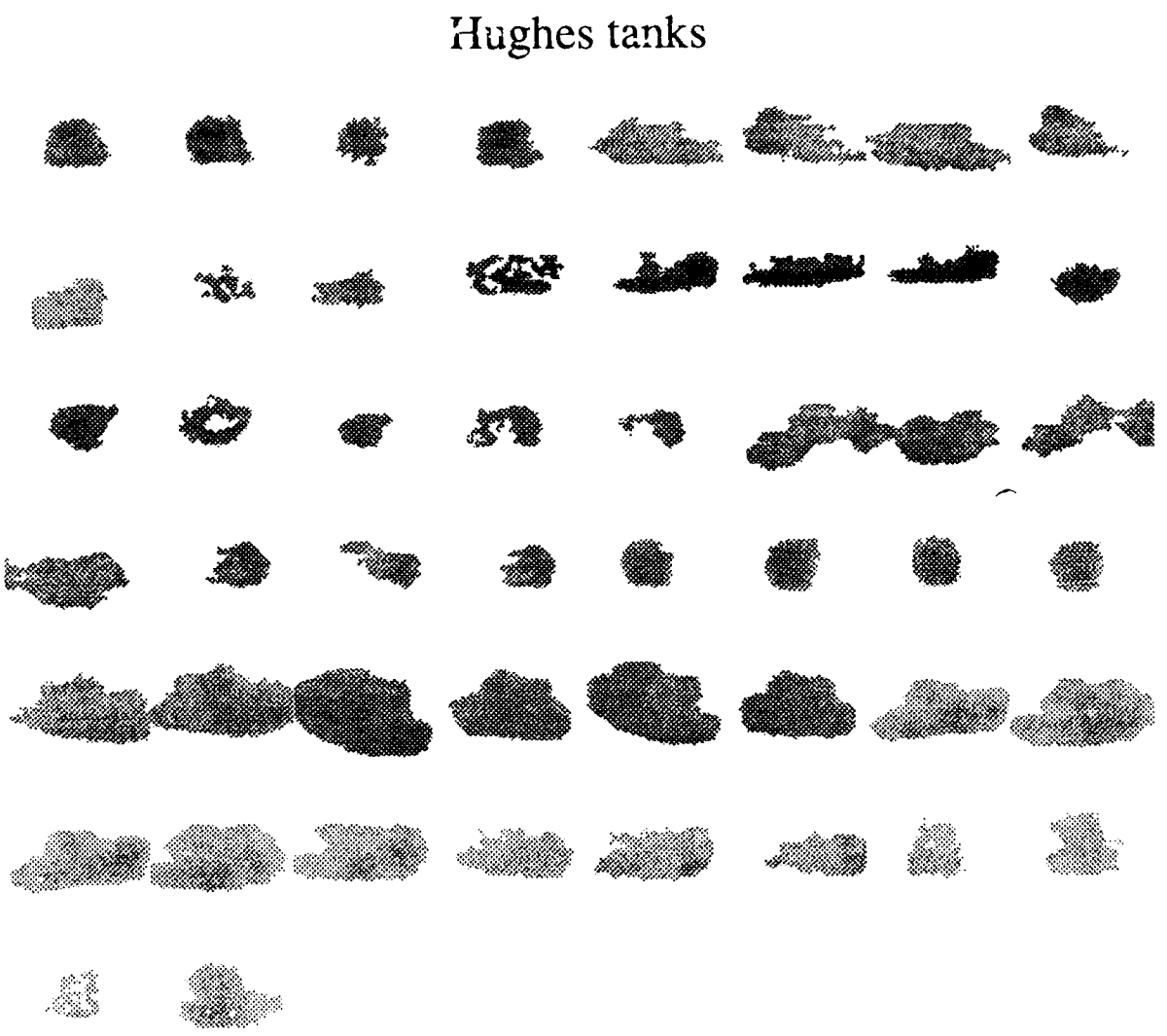


Figure 2.2.11 Composite of the tank segmentation results from the Hughes likelihood segmenter.

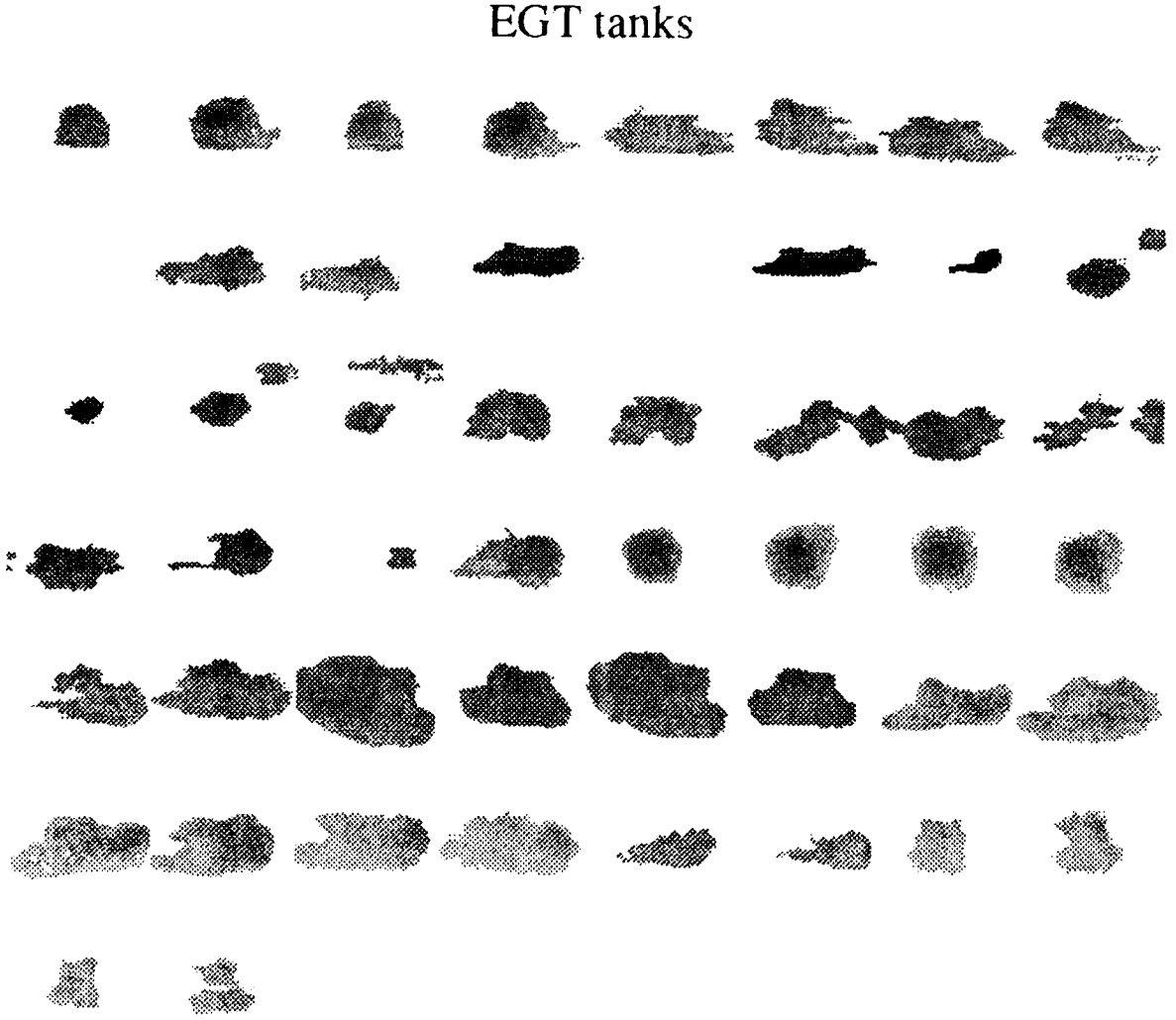


Figure 2.2.12 Composite of the tank segmentation results from the edge-guided threshold segmenter.

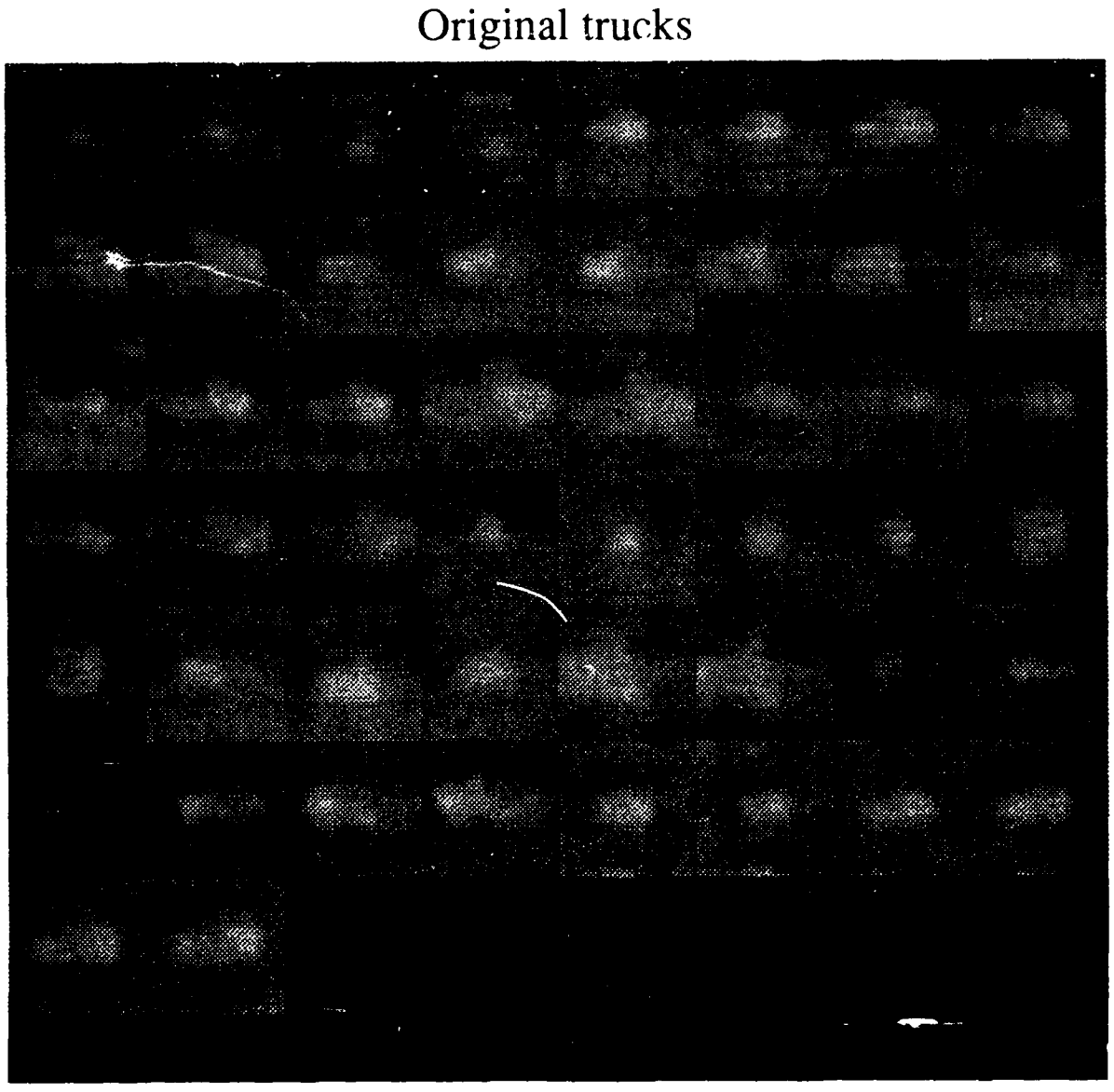


Figure 2.2.13 Composite of 50 original FLIR images of trucks from the BRITT data set.

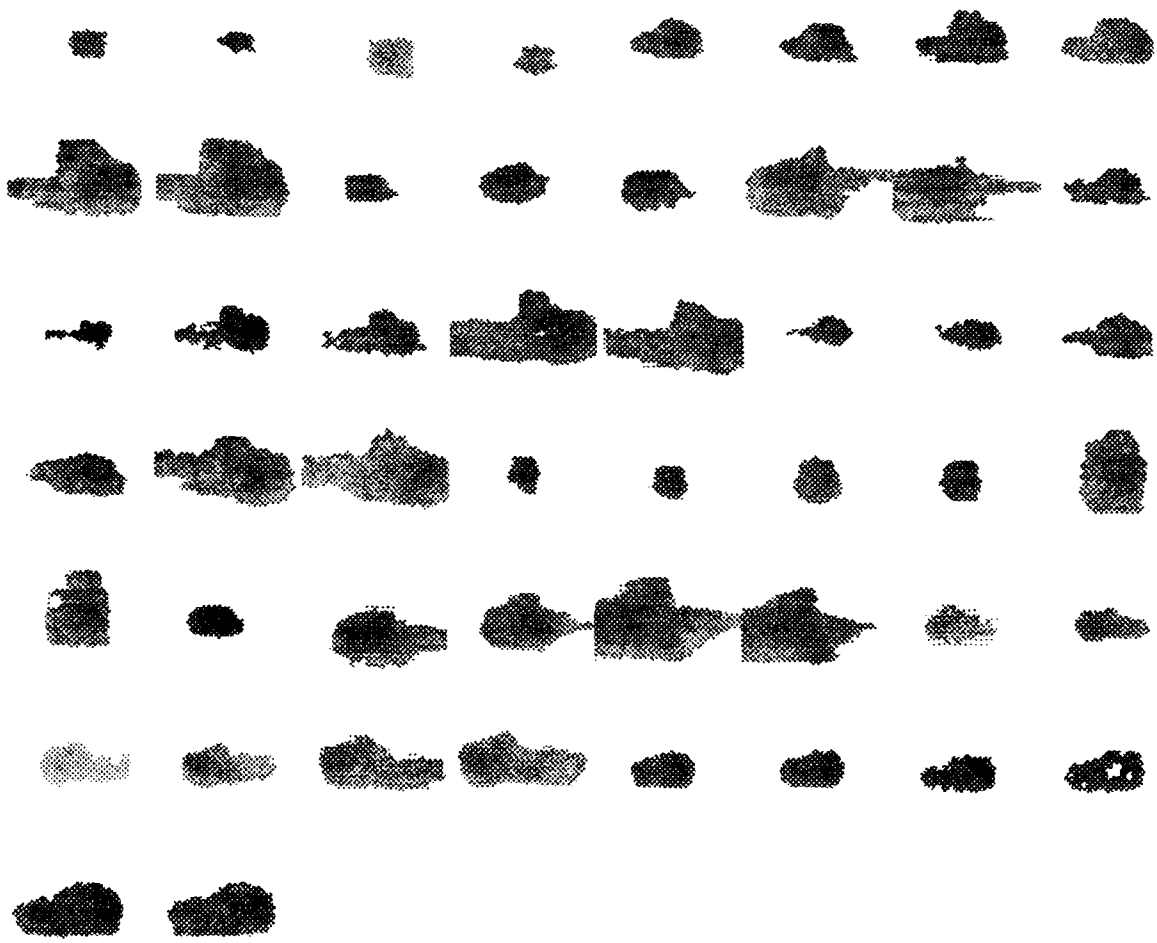


Figure 2.2.14 Composite of the truck segmentation results from the Hughes likelihood segmenter.

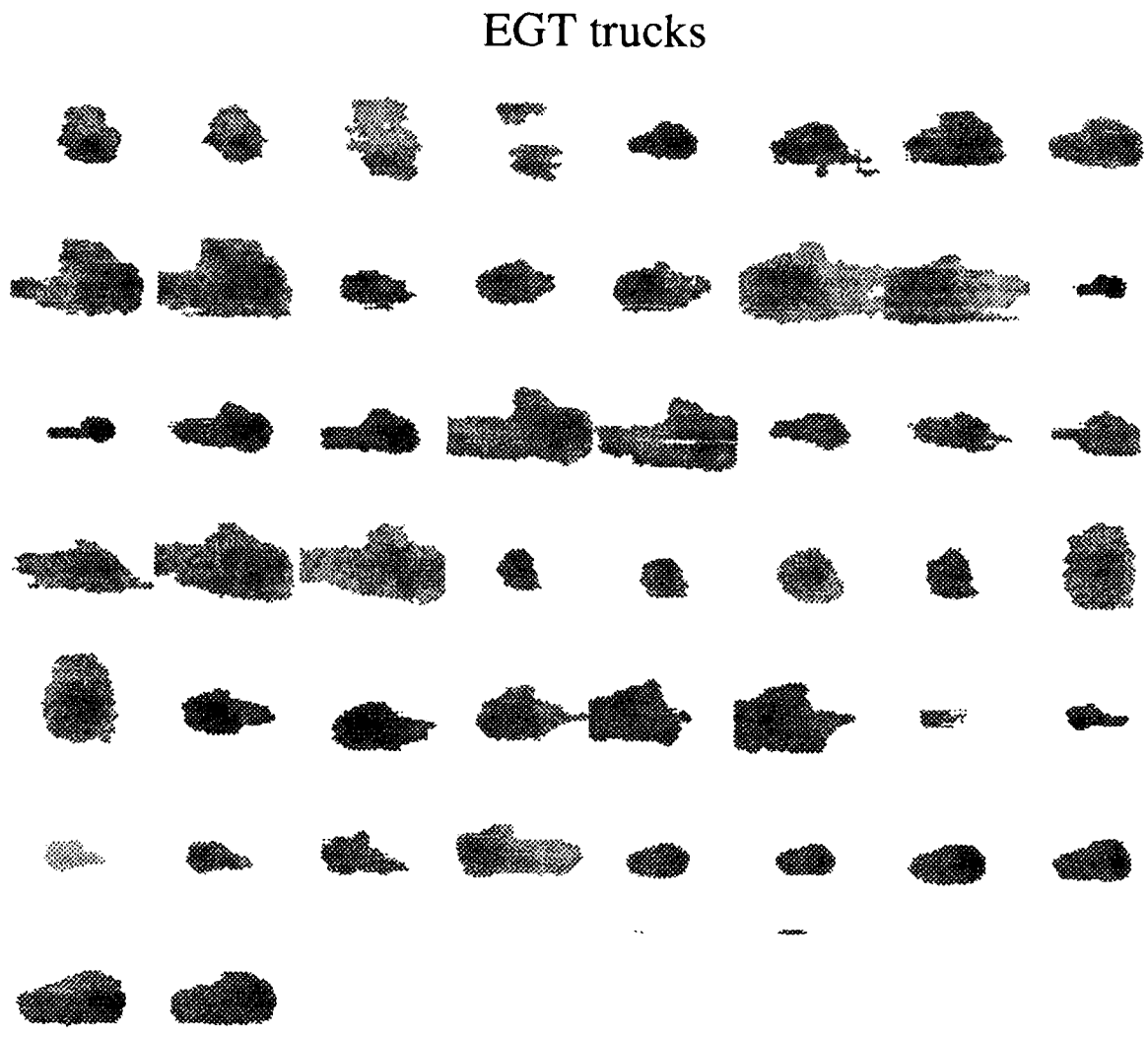


Figure 2.2.15 Composite of the truck segmentation results from the edge-guided threshold segmenter.

2.2.2. FLIR Segmentation by Tree Traversal

This section presents the tree traversal segmentation algorithm in [HoPa76], and evaluates its merits as a FLIR segmenter. Although only FLIR data is used here, the algorithm is general enough to apply to many different types of images including LADAR by merely changing the segmentation criteria.

The following is a discussion of the algorithm and some FLIR segmentation results. The implementation of the algorithm presented here uses criteria suitable for segmentation of reflectance or FLIR imagery. Although not pursued yet, it may be possible to "tune" portions of the implementation to take advantage of some special FLIR characteristics. This is a topic for future work.

2.2.2.1. The Algorithm

The algorithm consists of three main steps:

(1) Split and merge using a tree representation of the image.

(2) Merging based on adjacency in the image plane.

(3) Further merging of image plane regions based on some "nearest neighbor" criterion.

In steps one and two the difference between minimum and maximum gray level is the criterion used on the FLIR data. In step one, if the difference between the maximum and minimum gray levels of a region is too great, the region is split. For both steps one and two, regions may be merged provided the difference between the maximum and minimum gray levels of the resulting region is small enough.

The "nearest neighbor" criterion of step three is average region gray level. Two adjacent regions can only be merged if the difference between their respective average gray levels is small enough. The following sections discuss each step in more detail.

2.2.2.1.1. Split and Merge Using a Tree Representation

In this portion of the algorithm all operations on segments of the image are performed within the confines of a tree structure. Thus in order to begin, the tree structure must be initialized to represent the necessary information about the image. This means that an initial segmentation of the image which assigns each pixel of the image plane to a node of the tree (see Figure 2.2.16) must be chosen. This actually corresponds to selecting a starting level within the tree structure. Each node of this level contains: the (x,y) position of the block it represents, the number of pixels to a side of this block, and the largest and smallest gray level within the block.

Following this initial segmentation of the image plane, each node representing a square block of the image is examined to determine whether the block should be broken into four smaller blocks based on the min/max criterion described above. Using this same criterion, four blocks whose nodes share a common parent node are examined to determine if they should be

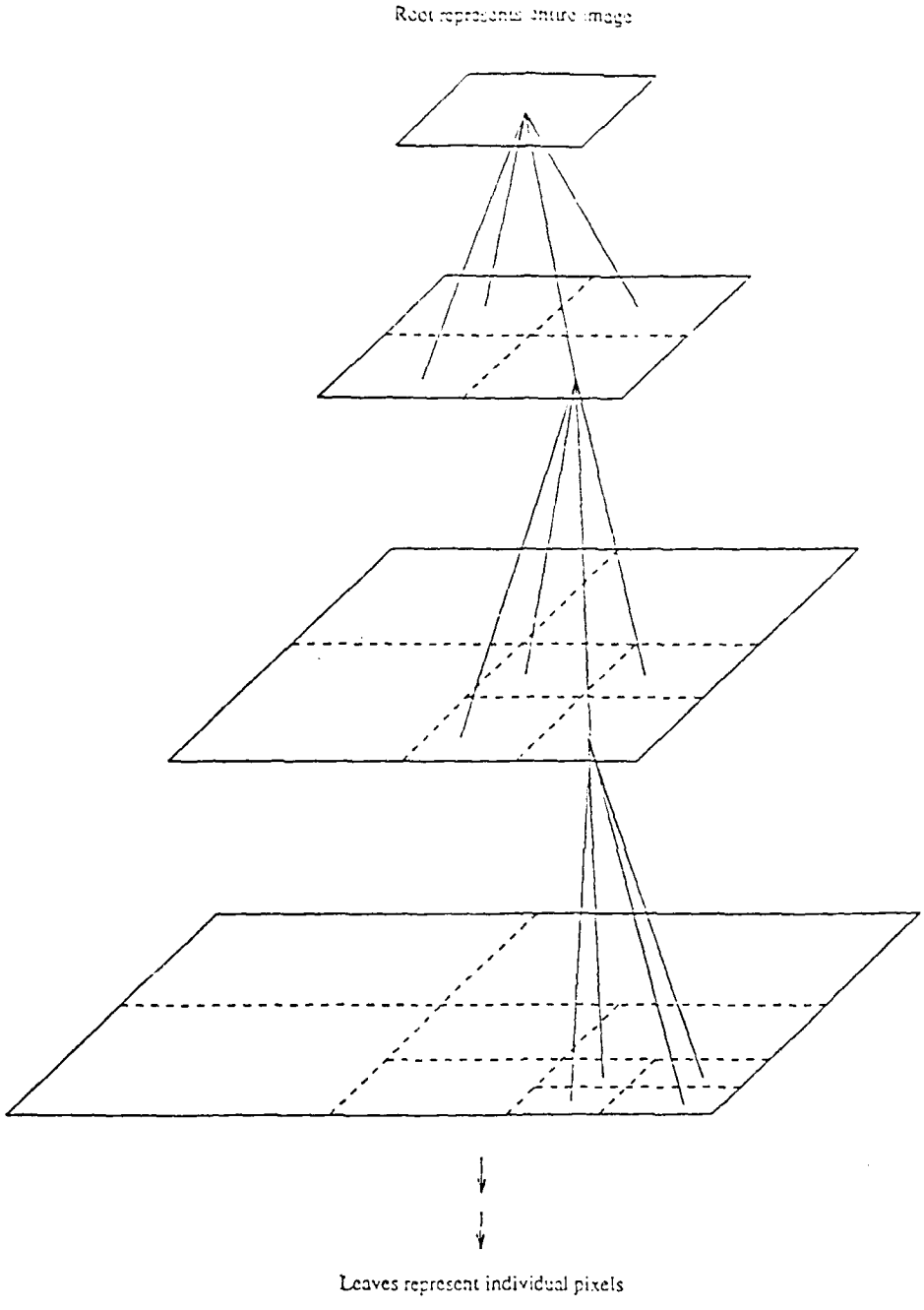


Figure 2.2.16 Visual representation of tree data structure.

merged into one larger block.

Note that a block which is created by a split operation cannot be the object of a merge operation within the same tree structure. Nor can a block that is the result of a merge operation be subject to a split. Therefore, only one pass is necessary, and the nodes forming the final cutset of the tree structure represent the result of the "split and merge" portion of the algorithm. See Figure 2.2.17 for an example of the results of this step.

An important effect of the tree structure representation is the limitation that it imposes on the merge operation. Note that only nodes sharing a common *immediate* parent are considered for merging. Because of this, the next step of the algorithm departs from the tree representation and again applies the min/max criterion to determine if blocks of the final cutset should be merged into larger, not necessarily square, regions.

2.2.2.1.2. Grouping of Final Cutset Segments

After completion of the previous step the image consists of square segments that range in size from single pixels, to the entire image. This is the final cutset mentioned above. Each of these segments is now examined to determine if it can be merged with the blocks adjacent to it in the image plane.

The present implementation of the algorithm does not use any particular order in examining these segments. This is a portion of the algorithm in which it may be possible to take advantage of special FLIR imagery characteristics as well as other *a priori* information in order to improve segmentation results. Some ideas are discussed in the future work section.

2.2.2.1.3. Further Region Grouping Based on "Nearest Neighbor" Criterion

Now the image consists of irregular regions in which the difference between maximum and minimum gray level is within tolerance. To begin the next step of the processing, the average gray level of each region is computed. In this step the criterion used is "closeness" in terms of average gray level. All adjacent regions are checked, and the "closest", if it is "close" enough, is merged with the region in question. After updating the average gray level of this new region, all of its neighbors are ranked according to closest average gray level, and the operation is repeated in this fashion until merging is no longer possible. The next region is then examined in the same way, and this is repeated until all regions of the image have been considered.

Here again the merging operation proceeds from region to region in an arbitrary order. As for the previous step, possible ordering criteria based on FLIR imagery characteristics as well as other *a priori* information, are discussed in the future work section.

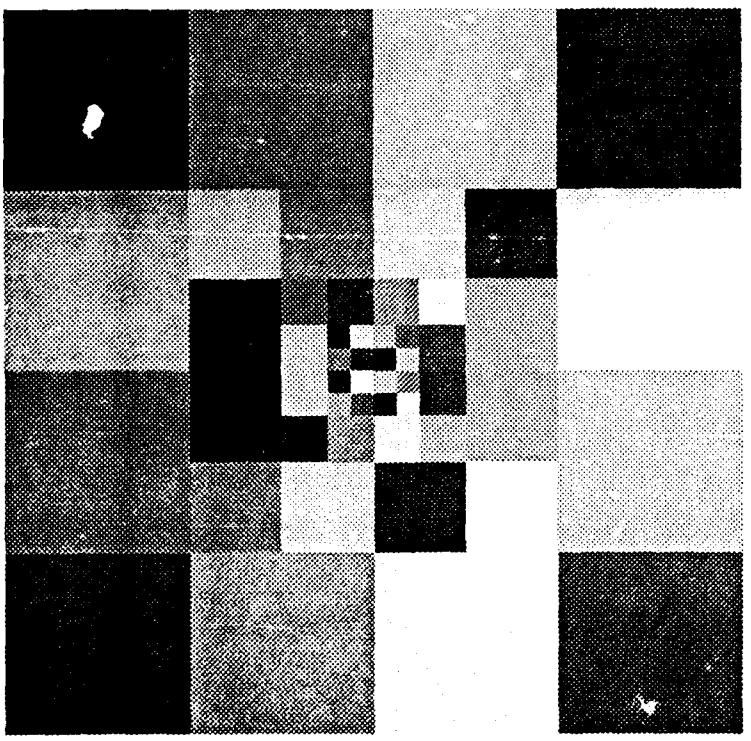


Figure 2.2.17 Test input image, and "segmentation" by split and merge step. (Arbitrary gray level assignment to distinguish regions.)

2.2.2.2. Some Segmentation Results

To test the flexibility of the algorithm in segmenting different types of FLIR imagery, several representative FLIR images taken from the BRITT data set are segmented below (see Figures 2.2.18 - 2.2.21). Table 2.2.1 summarizes the performance of the segmenter for each of these images.

As seen in these examples the segmentation results of the algorithm can be quite noisy. Especially for low contrast images. Most non-target regions may be eliminated, however, by invoking size constraints, perimeter-area ratio constraints, etc.

Although a sophisticated algorithm was not implemented here, a very simple cleanup using size and brightness constraints was used to enhance, in most cases, the segmentation results. The size constraint discards regions outside the 100 to 2500 pixel-area range.

NOTE: The brightness constraint applied here discards regions with an average gray level below that of the overall image. This could not be used in general since targets may occasionally be darker than background.

One of the most important features of this segmentation algorithm is its ability to segment regions within regions. This is shown in Figure 2.2.19 where the hot engine of the truck is segmented from the truck body. Such information could prove to be very valuable in later processing where "hot spot" locations within a target may be very useful for classification purposes.

NOTE: This information is not retained by the cleanup process in these examples simply because the routine is not sophisticated enough to recognize such situations.

It is important to note that the quality of the segmentation results presented here is due to some experience in selecting thresholds for the criteria. The results are *very* sensitive to the selection of these thresholds.

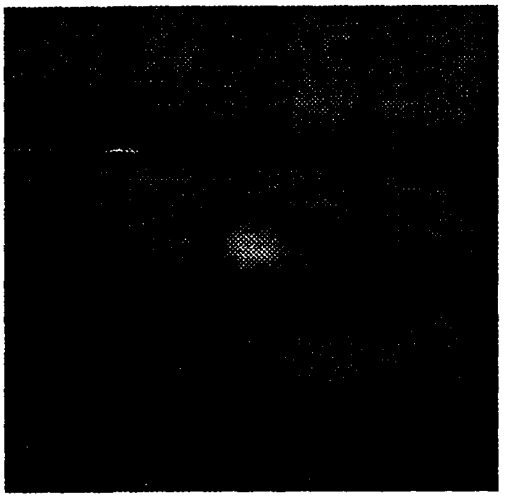
2.2.2.3. Comparison with EGT and Hughes Segmenters

To determine how well this segmenter really performs it was run on the same three sets of 50 images that the EGT and Hughes segmenters were run in Section 2.1.1.1. The results appear in Figures 2.2.22 - 2.2.30.

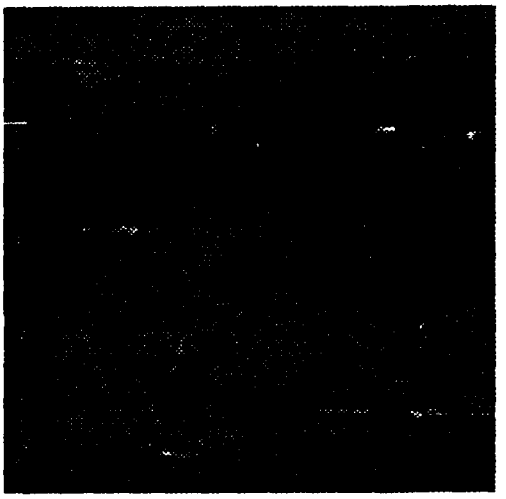
2.2.2.3.1. Threshold Selection

The original FLIR images were separated according to the characteristics mentioned in Table 2.2.1 (i.e. contrast, striatedness, etc.), and thresholds were chosen based on experience. This, admittedly, gives the tree traversal algorithm an advantage over the constant threshold used in the EGT results. About ten segmentations were improved by it.

As before, a cleanup routine was run on the segmentation results. Since the routine is not, nor is it meant to be, very sophisticated, it sometimes discards some of the small interior regions of targets. Recall that the ability to extract regions from within regions is one of the more



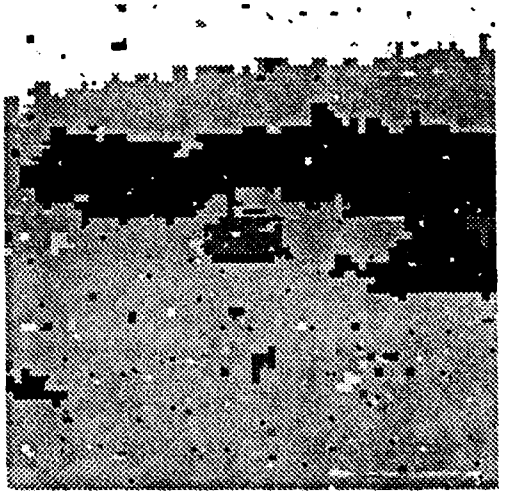
briu040



briu238



segmented



segmented



after cleanup



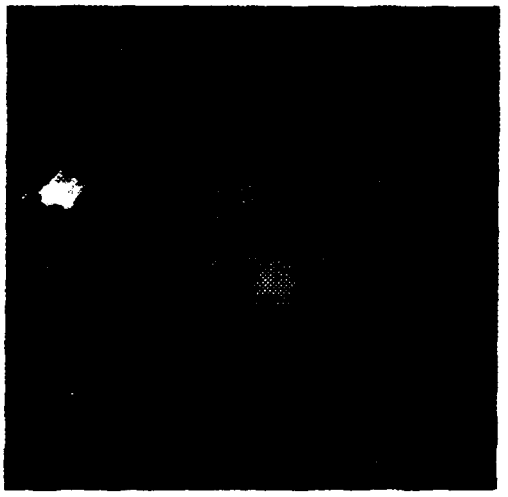


after cleanup

Figure 2.2.18 A "hot" and a "cool" target.



britt137



britt029



segmented



segmented

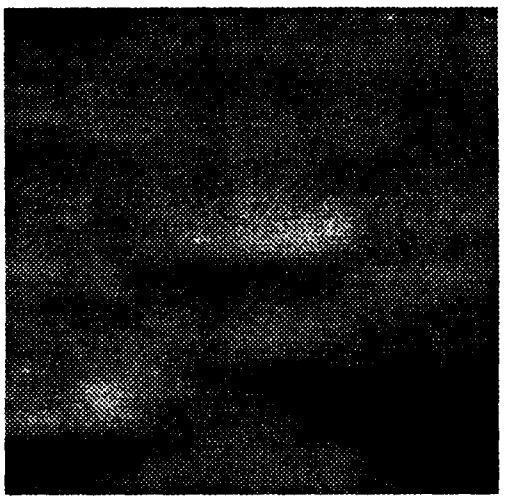


after cleanup

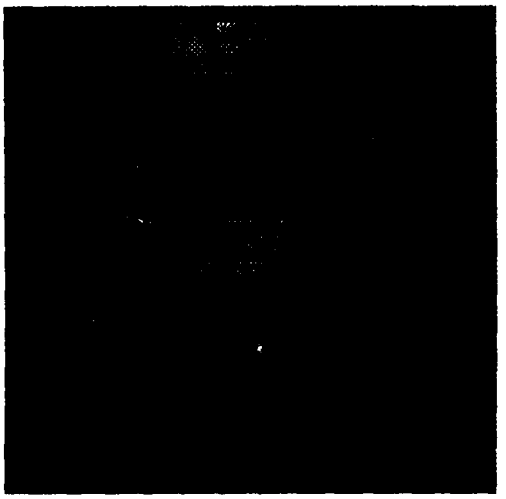


after cleanup

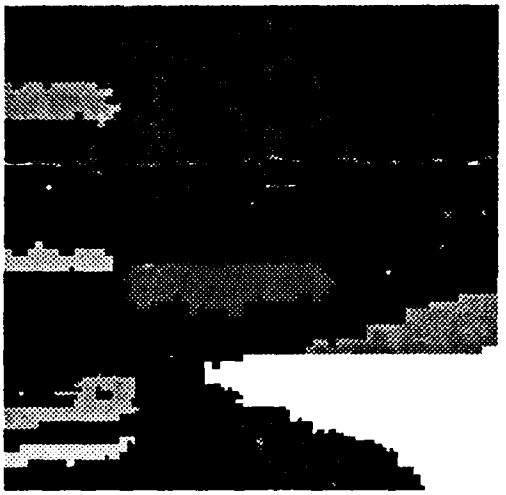
Figure 2.2.19 Multiple region targets.



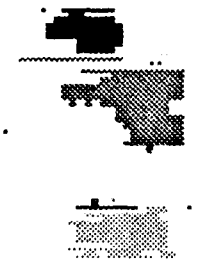
britt277



britt515



segmented



segmented

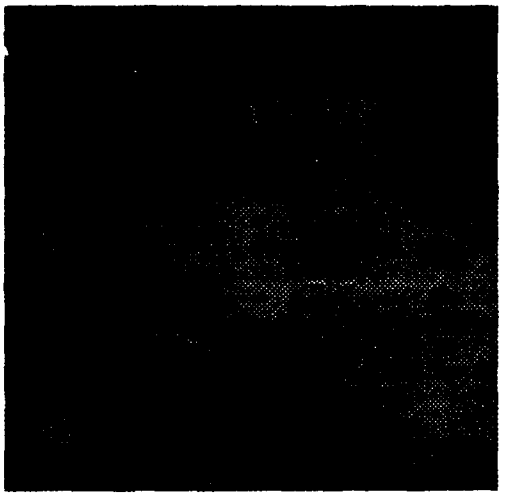


after cleanup

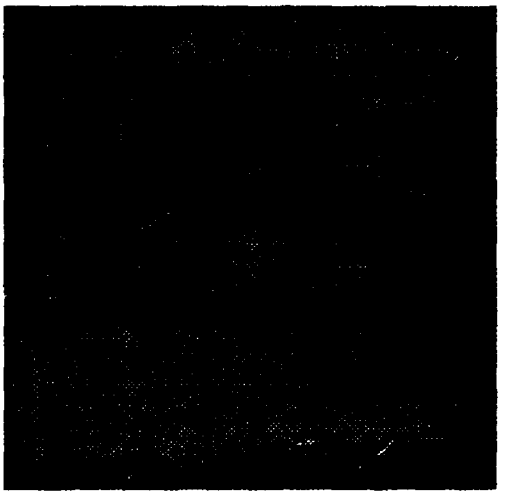


after cleanup

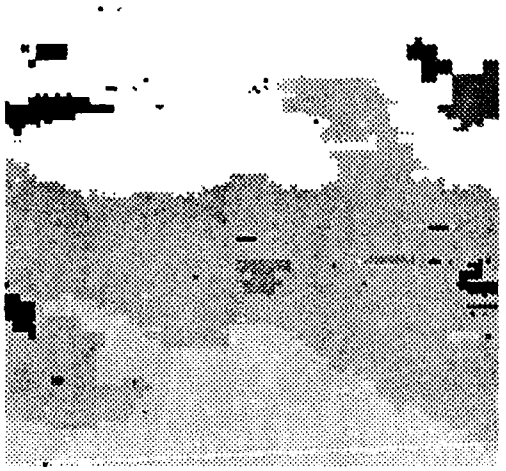
Figure 2.2.20 Multiple targets.



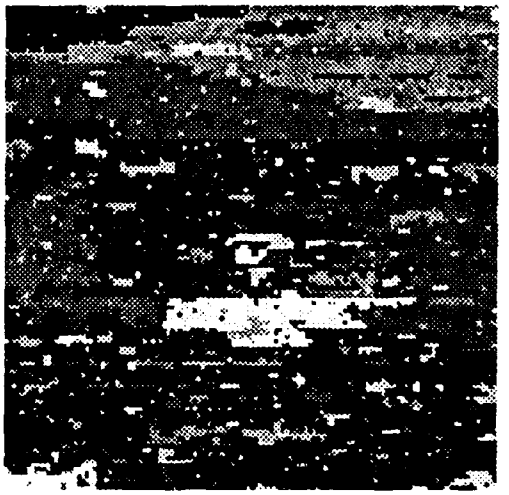
britt347



brit003

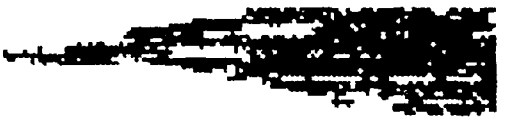


segmented



segmented







after cleanup

after cleanup

Figure 2.2.21 Very difficult segmentations.

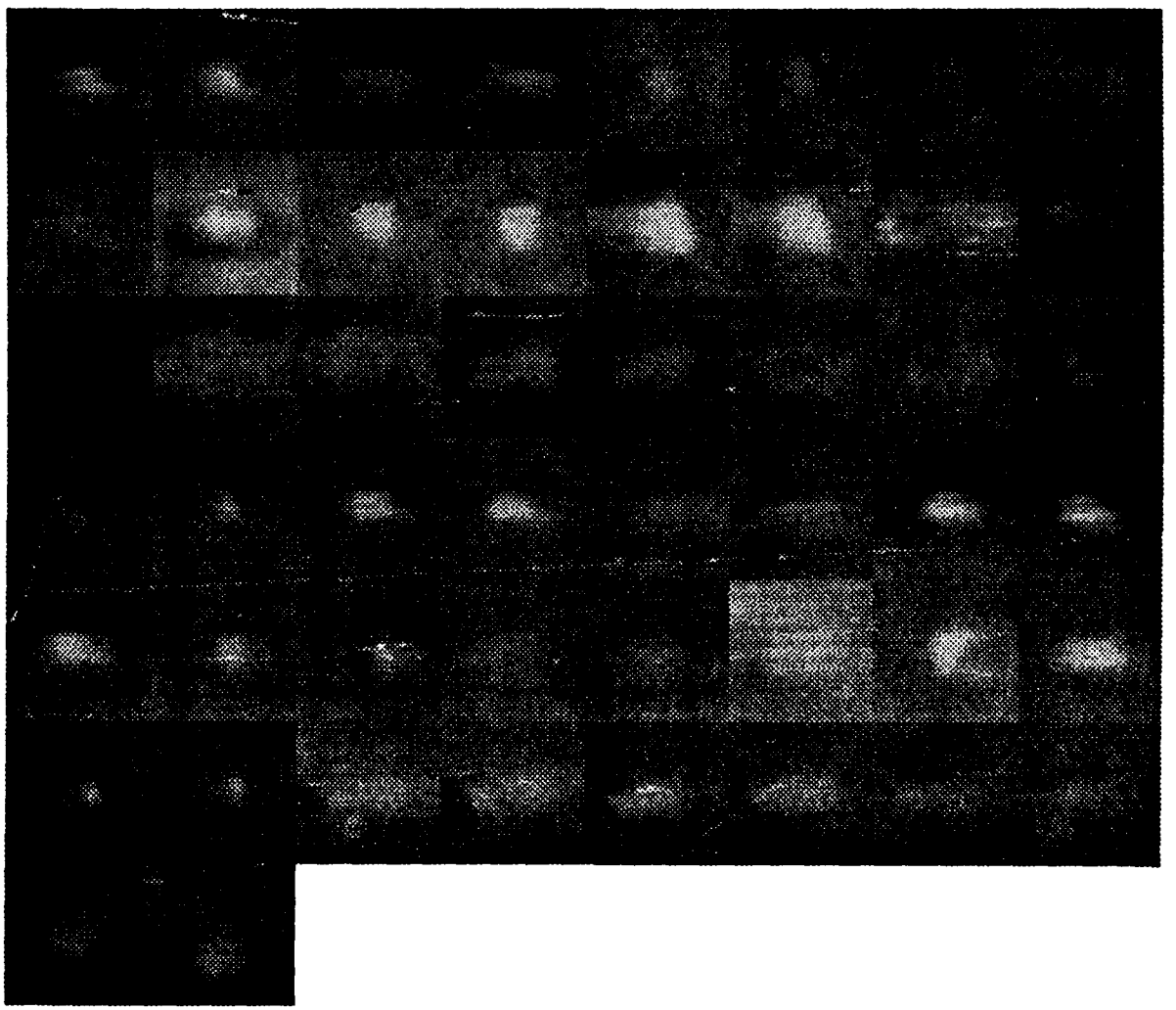


Figure 2.2.22 Composite of 50 original FLIR images of APC's from the BRITT data set.

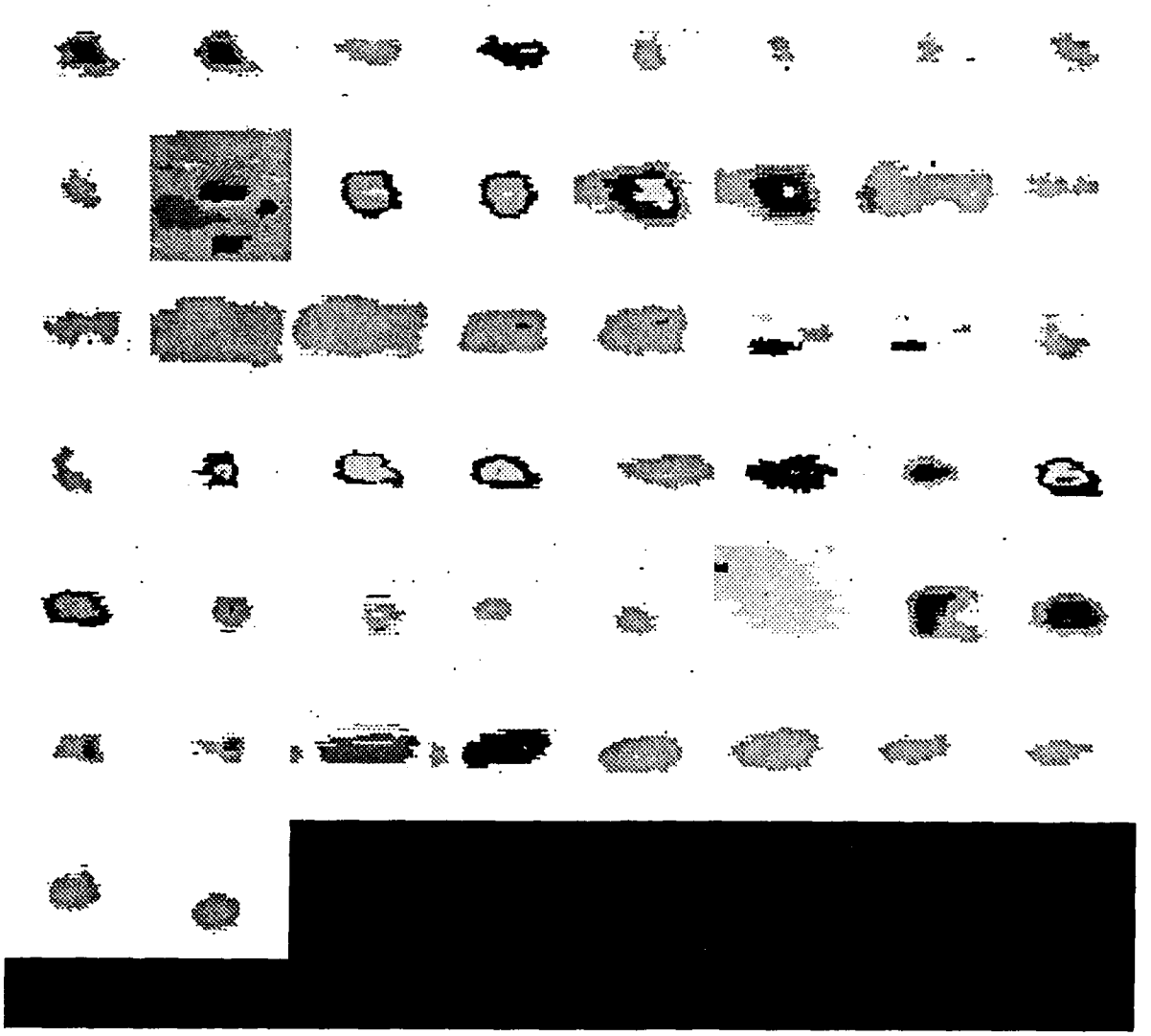


Figure 2.2.23 Composite of the APC segmentation results.

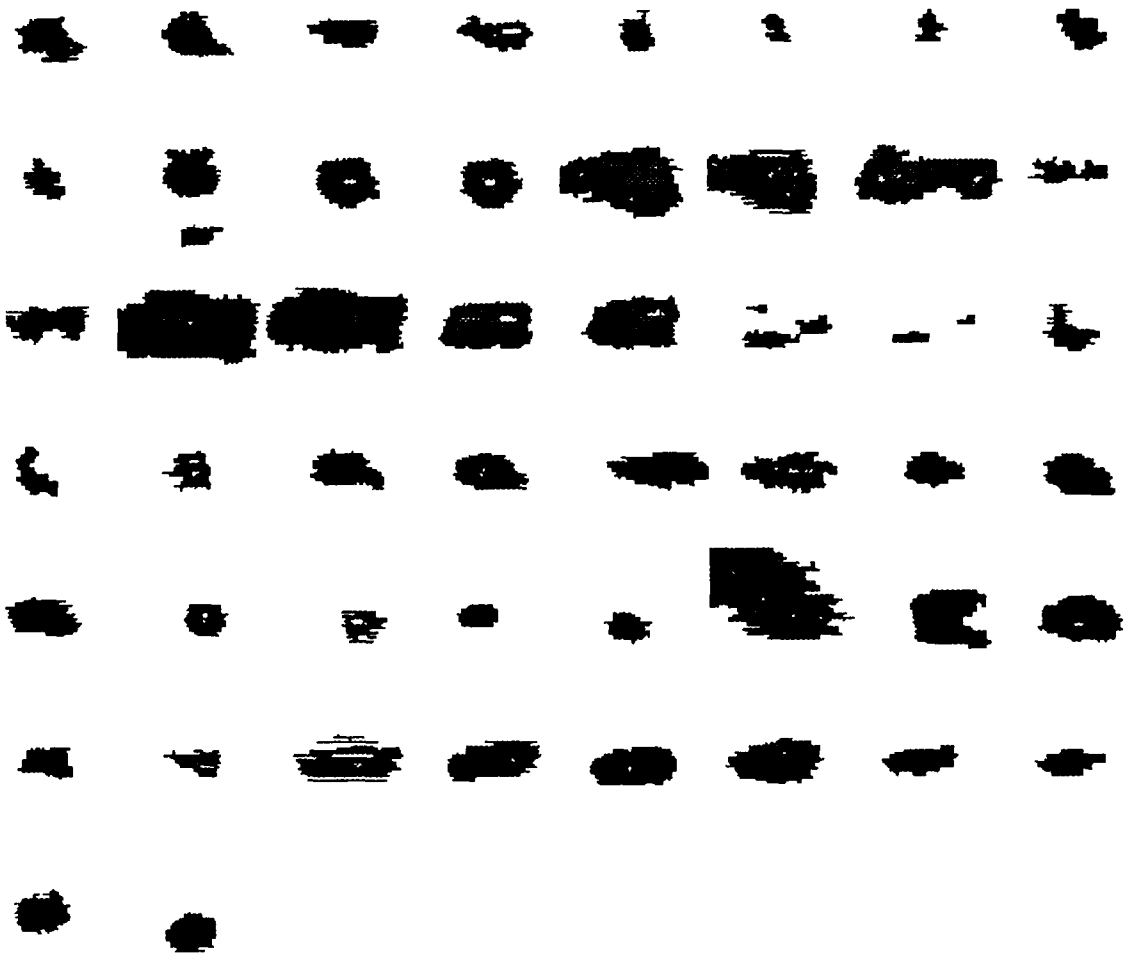


Figure 2.2.24 Composite of the APC results after cleanup.

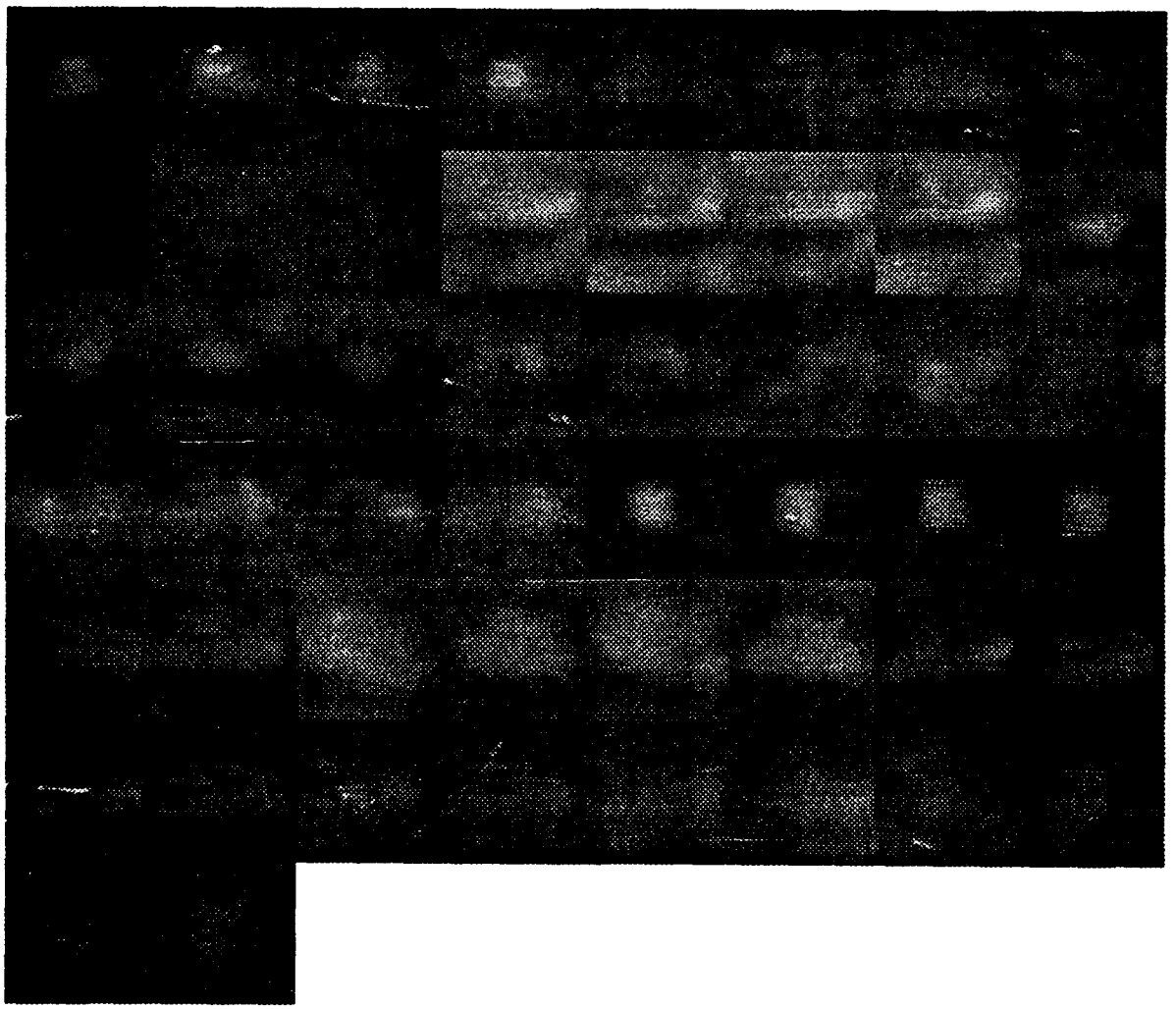


Figure 2.2.25 50 original FLIR images of tanks from the BRITT data set.

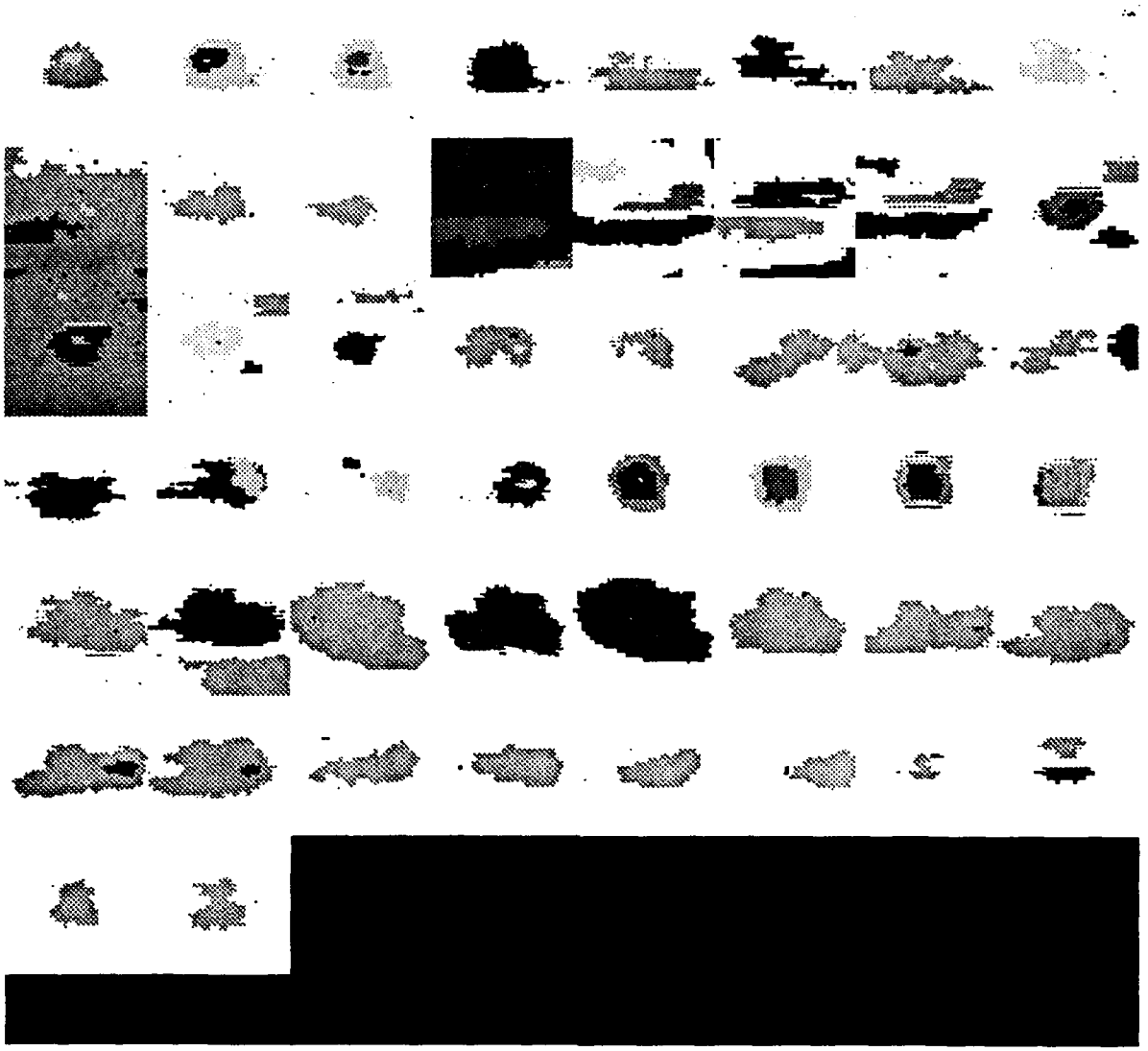


Figure 2.2.26 Composite of the tank segmentation results.

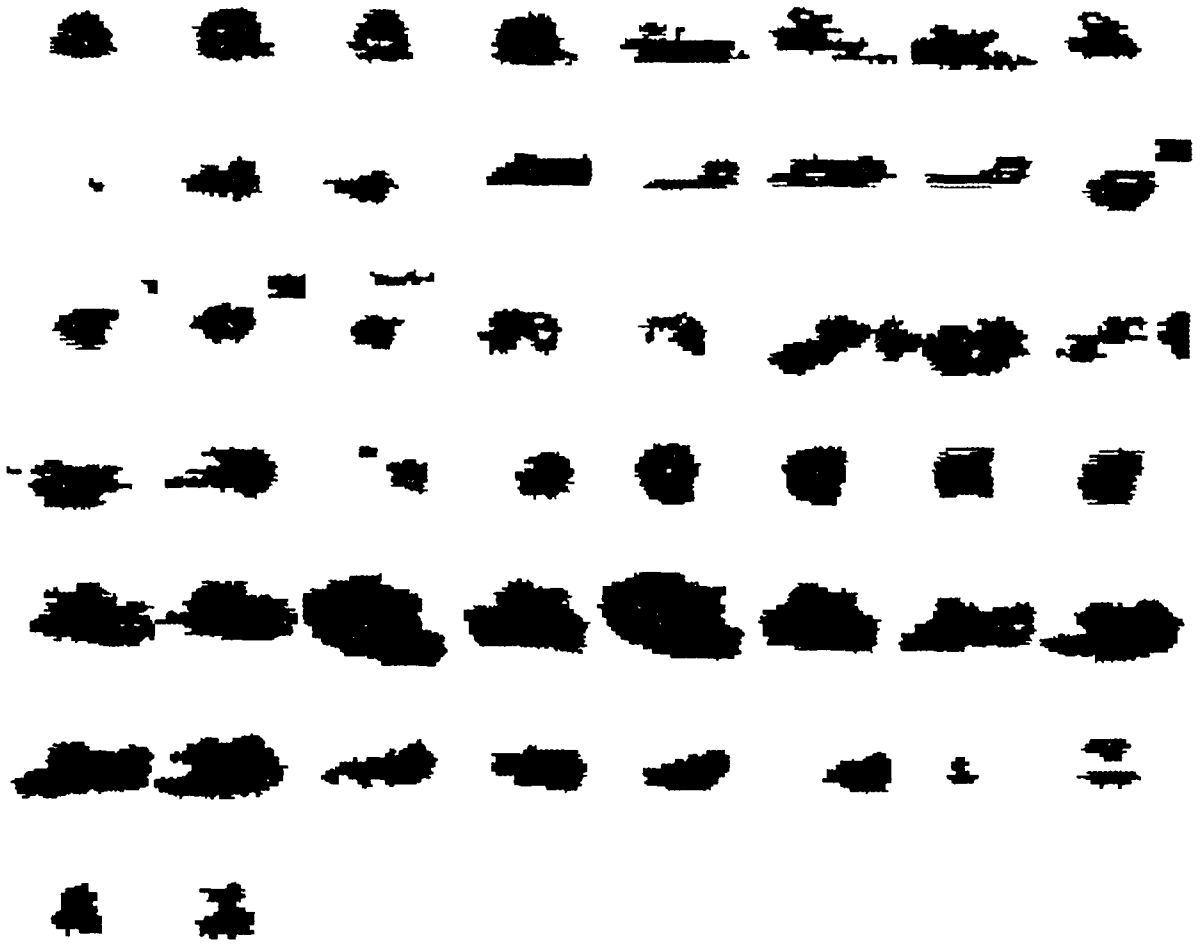


Figure 2.2.27 Composite of the tank results after cleanup.

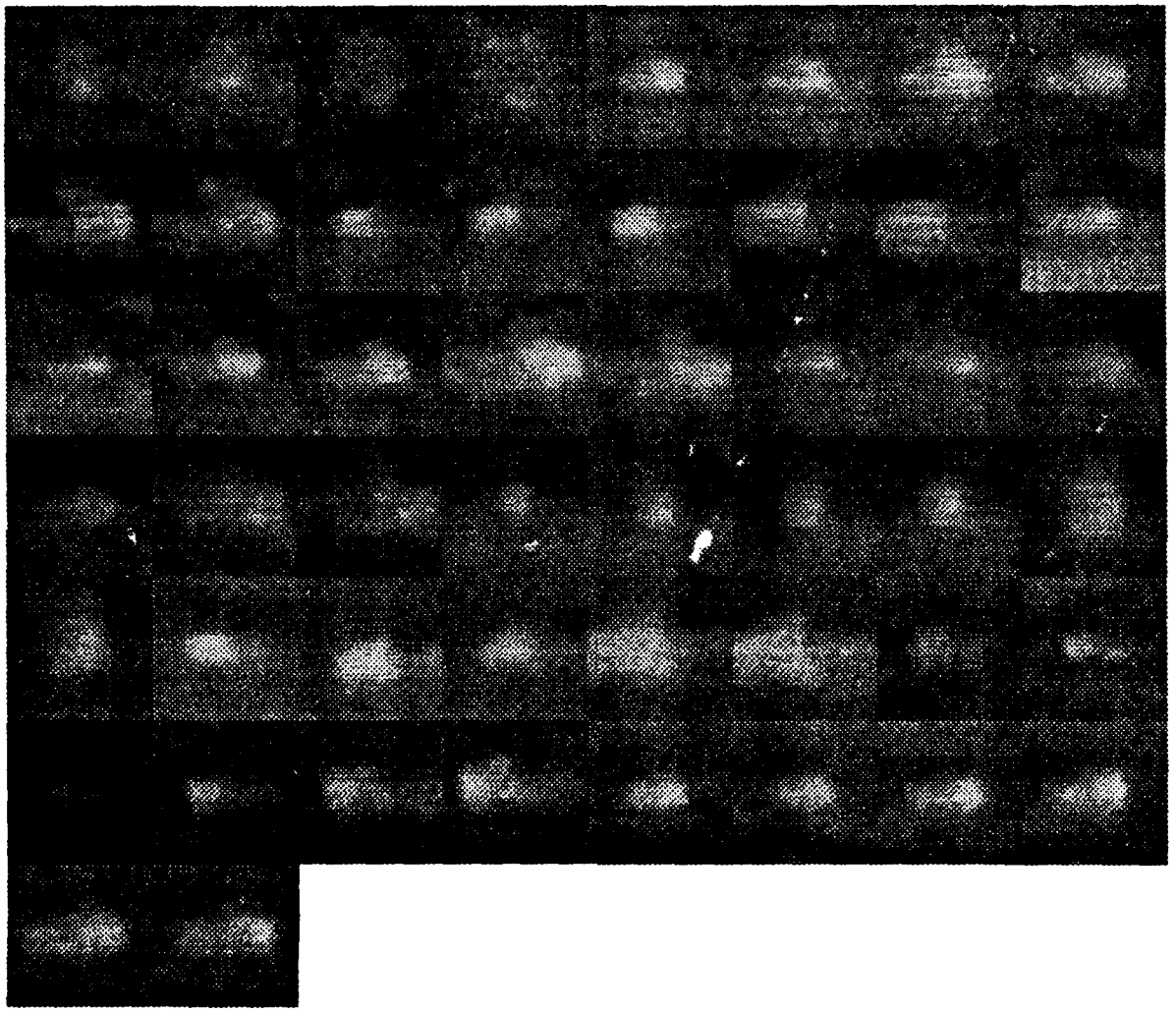


Figure 2.2.28 Composite of 50 original FLIR images of trucks from the BRITT data set.

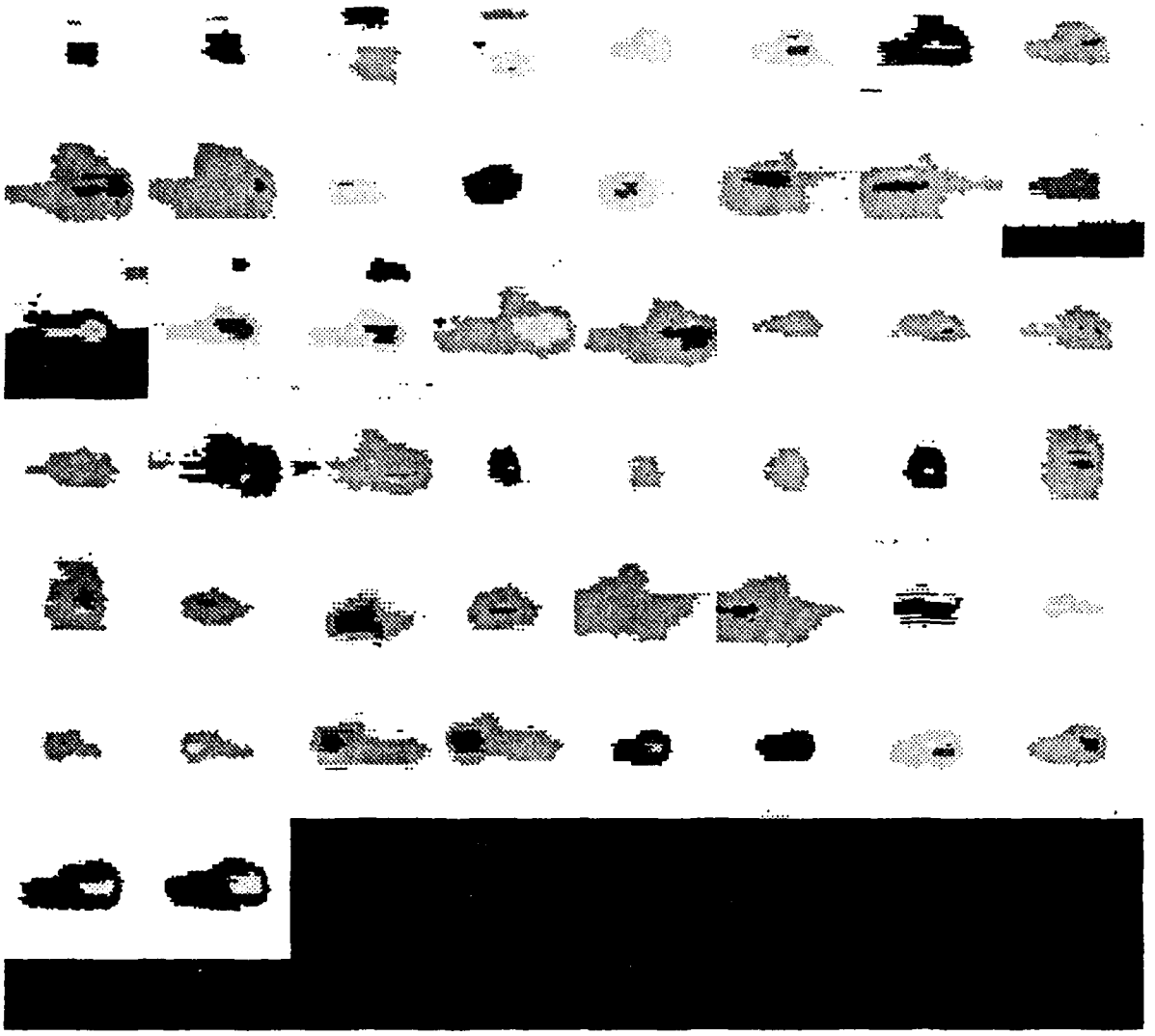


Figure 2.2.29 Composite of the truck segmentation results.

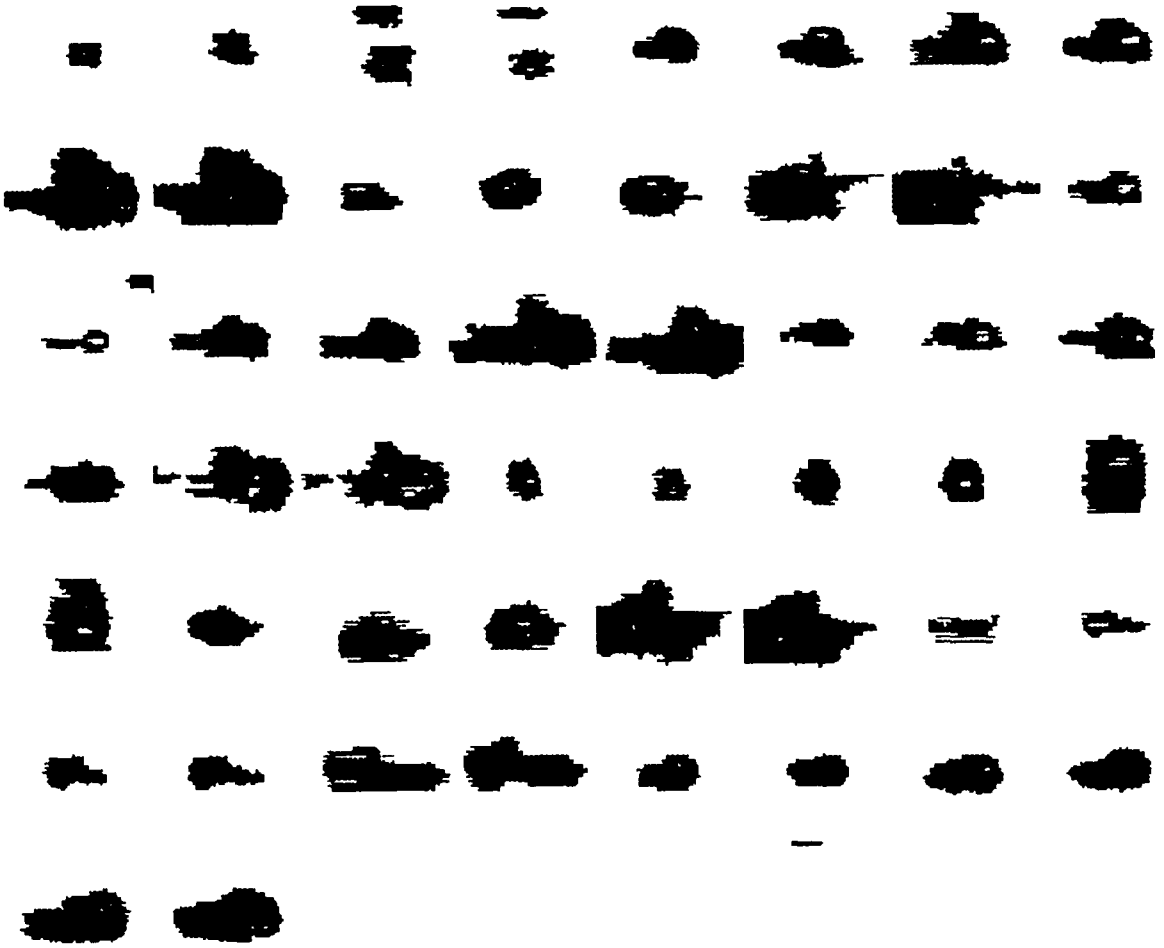


Figure 2.2.30 Composite of the truck results after cleanup.

Table 2.2.1 Comments on the tree traversal segmenter's performance on the BRITT data.

Images	Ground Truth	Thresholds	Comments
britt040	apc dead center	lmax - minl=10 aver. thresh. = 40	This is a high contrast image which means there is a large difference between target and background average gray level. A high average criterion threshold produces a sharp segmentation.
britt238	apc dead center	lmax - minl=10 aver. thresh. = 5	This is a low contrast image so the average criterion threshold is much lower causing the background to be broken into multiple regions. Most of these regions are removed by the cleanup routine, however.
britt137 britt029	truck dead center	lmax - minl=10 aver. thresh. = 20	Within these targets there are "warm" and "hot" regions. The engine, for example, is much warmer than the rest of the target. The average criterion is selected to separate the target from the background while also separating the engine from the target body. This ability to segment a target into regions is a very nice property that may be useful for classification purposes.
britt277	tank dead center tank lower left	lmax - minl=10 aver. thresh. = 20	Here there are many bright spots visible to the naked eye. The average criterion threshold causes the background to be broken into multiple regions. Again, most of these are removed by the cleanup routine, but there is apparently some background clutter which does not disappear. Also, note that the dark region in the center of the original image, and "pulled out" by the segmentation, is discarded by the cleanup routine. This could be the tread portion of the tank, and therefore, should not have been ignored.
britt515	apc dead center tank top center	lmax - minl=20 aver. thresh. = 20	The striated nature of this image calls for modification of the min/max criterion threshold. The criterion is increased from that used in the previous images in order to bridge the striated regions.
britt347	tank dead center	lmax - minl=10 aver. thresh. = 20	The target region was found in the segmentation. However, it was removed by the cleanup routine since it was smaller than 100 pixels.
britt003	apc dead center	lmax - minl=6 aver. thresh. = 5	Here the contrast is simply to low to pick out the target.

desirable aspects of this segmentation algorithm. For this reason the cleanup routine was given a size constraint of 20 to 2500 pixels for this experiment. Of course the smaller lower bound tends to increase the number of extraneous regions retained.

2.2.2.3.2. Comparison Summary

Following the cleanup routine the segmentation results tend to be comparable to those of the EGT and Hughes segmenters. This is impressive since the algorithm is very general and not specifically written for FLIR segmentation. In addition to this, the algorithm also segments hot and cold regions *within* a target. In other words, multiple levels of segmentation are possible. Not just target and non-target distinctions.

In contrast to the EGT segmenter, however, this algorithm is very sensitive to threshold selection. This could make the segmenter difficult to use in an ATR environment since thresholds must be chosen very accurately in order to produce useful results.

2.2.3. Future Work

As mentioned during the discussion of the tree traversal segmentation algorithm, portions of the implementation could be modified to take advantage of FLIR characteristics and any other *a priori* information. For example, if interest points within the image are identified prior to segmentation attempts, this information could be used to direct the region growing of steps two and three. Points of interest would be examined first during these region growing processes, and, especially for low contrast imagery, this would reduce the possibility of "interest regions" being absorbed into the background. With regard to the classification problem, the ability to segment hot and cold regions within a target will probably become important. Engine location, for example, will be very useful information when determining target class. Of course before the tree traversal segmentation algorithm can be used in an ATR environment, methods of automatic threshold selection must be developed.

We now have available three segmenters which work well with FLIR imagery. In the future we will be using these segmenters (one or more of them) to assist in fusing the FLIR data with the LADAR data.

2.3. THE USE OF HIGH LEVEL REASONING TO IMPROVE THE CLASSIFICATION OF FLIR DATA

Current production algorithms for processing FLIR images work in a bottom-up approach. That is, they start at the pixel level and segment the image into regions and then label (i.e. classify) each region based on its contents. The labeling of a given region is generally done in a context-independent manner without consulting the spatial inferences in neighboring regions. More consistent labeling and therefore better recognition accuracy may be obtainable by using global information about the scene. Such an approach could examine each region individually and give each a label and a confidence value for that label. Next all the regions could be

examined to see if the labeling is consistent. If the labeling is not consistent, hypotheses for relabeling could be generated based on partial evidence in each region and global information about the scene. The regions could then be reexamined to try to verify these hypotheses.

Hypothesize-and-verify approaches are difficult to incorporate in purely bottom-up algorithms, therefore we are investigating the use of hierarchical data structures which will lend themselves to the construction of scene hypothesis at different scales. We are currently investigating LoG channels (Laplacian-of-Gaussian), pyramids, and quadtrees which could comprise the lower levels of a hierarchy.

Section 2.3.1. presents some possible symbolic data structures and discusses the advantages of each. Section 2.3.2. presents the various pixel level data hierarchies. At some point we feel it will be necessary to switch from a pixel description of an image to a symbolic description where each location in the image will have information describing which higher level objects in the scene it is part of. Such information will greatly help the reasoning process. Section 2.3.3. presents a high-level reasoning system which converts edge descriptions to symbols and uses global information to construct hypothesis to aid in make the edge labeling consistent in a scene.

2.3.1. Data Structures for Symbolic Reasoning

The following is a brief description of some preliminary ideas for a data structure for symbolic reasoning. This *symbolic list* provides a link between the pixel level segmentation and the high level reasoning by taking pixel level features (such as lines, edges, and regions) and storing them in a data structure so that a high level reasoning process can easily access related information. Once an image is decomposed into edges and regions, graph theory provides a natural means for describing the image. The following section gives some common graph theory definitions which will then be used in describing the symbolic data structure.

2.3.1.1. Some Graph Theory Definitions

A segmented image may be thought of as a planar graph. There are *edges* (line segments), *vertices* (line segment intersection points), and *faces* (regions). The following paragraphs describe these objects.

An *edge* is a line segment terminated by two vertices or by a single vertex if it is a loop. A *vertex* is the endpoint of a single edge, an arbitrary single point in a closed edge (loop), or the intersection point of three or more edges. If the *degree* of a vertex is defined as the number of edges incident with it, each loop counting as two edges [BonMur76], then with the above definitions of edge and vertex the only way for a vertex to be of degree 2 is if it is part of a loop. A vertex of degree 1 is simply a way of specify the endpoint of a "dangling" edge. Aside from the above two cases (the termination point of an edge or its intersection with itself), the only other place vertices occur is at the intersection of 3 or more edges. This is important in that no matter how many crazy turns an edge makes, a vertex will only be defined if it intersects

something. This keeps 90-180 degree bends in an edge from being defined as vertices.

Let us consider regions next. First let us define a *walk*. A *walk* is a non-null sequence $W = v_0 e_1 v_1 e_2 v_2 \cdots e_k v_k$ whose terms are alternately vertices and edges, such that for $1 \leq i \leq k$ the ends of e_i are v_{i-1} and v_i. v_0 and v_k are the origin and terminus of W, respectively, and W is said to be of length k [BonMur76]. A *region* therefore may be specified by the closed walk (terminus = origin = v_0) defining its border. Let us define a *region* as such a walk W of length k surrounding area A(W) such that there are not two shorter walks X of length i and Y of length j, i < k and j < k, with $A(X) \bigcup A(Y) = A(W)$, if there are any closed walks contained in walk W, the regions defined by those walks are not included in the region defined by W. (i.e. regions are non-overlapping.)

2.3.1.2. The Data Structures

Figure 2.3.1 shows a target which has been segmented into edges, vertices, and regions, each of which has been labeled. The edges, vertices, and regions of any image which has been segmented as in Figure 2.3.1 can be represented symbolically by the following data structures.

1. A List of Edges

   The **edge** list stores a count of the number of edges in the image and has a pointer to a linked list containing all the edges. The edge label, length of edge, and the location of endpoints are stored for each edge as shown in Figure 2.3.1.

2. A List of Vertices

   The **vertices** list keeps a count of the number of vertices and points to a linked list of them. The degree of a given vertex, the vertex label, and the location of the vertex are stored for each vertex (see Figure 2.3.1).

3. A List of Regions

   The **regions** list contains a count of the regions in the image and a pointer to a linked list of regions, each containing the region label, a count of edges, and a pointer to a linked list of edges and vertices tracing the walk that defines the region. (See Figure 2.3.1.)

The three lists above provide a compact means of storing the individual edge, vertex, and region data. The following tables store information about the relationships between the various edges, vertices, and regions.

1. Adjacency Table

   The **adjacency** table (for an example see Table 2.3.1 and Figure 2.3.1) has one column and one row for each vertex in the image. Entry **adjacency**[*i*][*j*] gives the number of edges joining v_i and v_j. This table is useful for finding loops since a non zero entry at **adjacency**[*i*][*i*] shows that vertex v_i is part of a loop.

2. Region Boundary Table

   The **boundary** table (see Table 2.3.2 for an example) has one column for each edge and

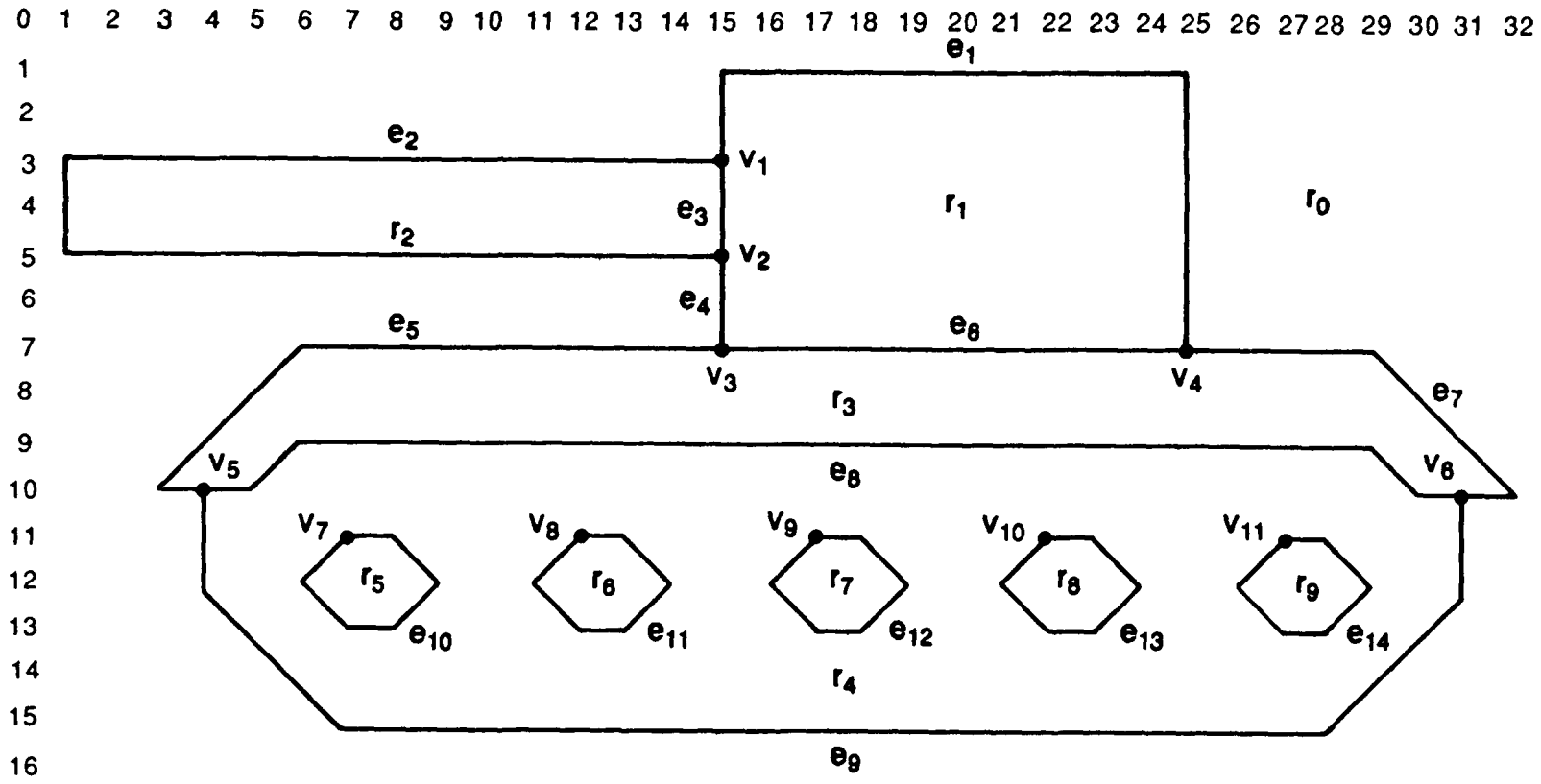


Figure 2.3.1 Sample target which has been segmented into edges, vertices, and regions, and the corresponding symbolic data structures.

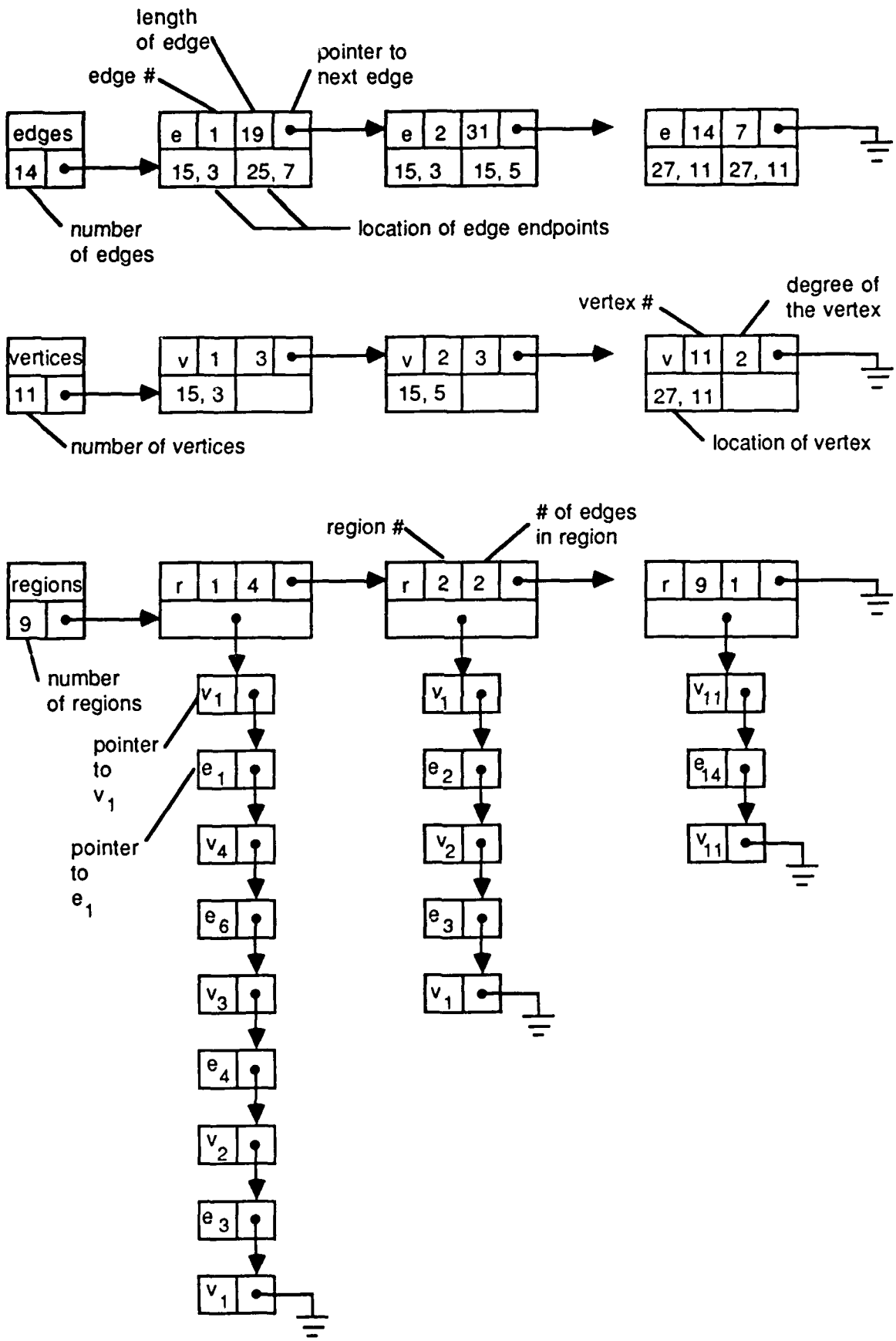


Figure 2.3.1 Continued.

Table 2.3.1 The **adjacency** table for Figure 2.3.1. It has one row and one column for each vertex.

	vertices										
Adjacency	1	2	3	4	5	6	7	8	9	10	11
1	0	2	0	1	0	0	0	0	0	0	0
2	2	0	1	0	0	0	0	0	0	0	0
3	0	1	0	1	1	0	0	0	0	0	0
4	1	0	1	0	0	1	0	0	0	0	0
5	0	0	1	0	0	2	0	0	0	0	0
6	0	0	0	1	2	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	1	0
11	0	0	0	0	0	0	0	0	0	0	1

one row for each region in the image. If **boundary**[i][j] = 1, edge e_j is part of the boundary of region r_i.

This table can be used to easily compute the number of edges around region r_i by summing the entries in row i. It can also be used to see if a region completely contains another region. To test to see if region r_i is completely surrounded by another region, find a list of edges bounding r_i by looking for the set of edges, E, which have a 1 in row r_i of the boundary table. If each edge in E bounds the same region, $r_j \neq r_i$, then r_j completely surrounds r_i.

3. The Incidence Table

The **incidence** table (Table 2.3.3 is an example) has one column for each edge and one row for each vertex. **incidence**[i][j] equals the number of times v_i and e_j are incident.

2.3.1.3. An Example of Reasoning with the Symbolic Data Structure

A number of useful reasoning processes can be easily supported by the above lists and tables. For example, suppose we are trying to recognize the target in Figure 2.3.1. Suppose a rule in our knowledge base was:

Due to the nature of treads on a tank, you will find several loops in a horizontal line near the bottom of the target.

Table 2.3.2 The **boundary** table for Figure 2.3.1. It has one column for each edge and one row for each region.

	edges													
boundary	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	1	0	1	1	0	1	0	1	0	0	0	0	0
1	1	0	1	1	0	1	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	1	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 2.3.3 The **incidence** table for Figure 2.3.1. It has one column for each edge and one row for each vertex.

	edges													
incidence	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	0	1	1	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	1	1	1	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	2	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	2	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	2	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	2	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	2

Our reasoner wants to find support for this rule, so it looks for *loops* by consulting the diagonal of the **adjacency** table. It finds that v_7 - v_{11} are all loops (they all have a non zero value on the diagonal). The **incidence** table shows that v_7 - v_{11} are incident on edges e_{10} - e_{14} respectively. Consulting the **boundary** table shows that edges e_{10} - e_{14} all bound region r_4 and individually

bound regions r_5 - r_9 respectively. This means r_5 - r_9 are all completely contained in r_4. Consulting the coordinates for e_{10} - e_{14}, (stored in the **edge** list), shows that e_{10} - e_{14} are horizontal. Therefore we have found support for the above rule by simply consulting the lists and table.

2.3.1.4. Conclusions

We have described the symbolic structures we are currently using and have shown that they are useful for supporting the queries of a reasoning process. These structures will change as different methods of reasoning are tried and weaknesses in the structures are found. It is our hope that combining the symbolic data structures with the hierarchical methods will result in a powerful, yet compact, method for storing the relevant information in an image so that it can be used by some sort of high level reasoning program.

2.3.2. Pixel Level Hierarchical Data Structures

The lower few levels of our hierarchy will probably **not** contain any symbolic information, instead it will contain the low level pixel information. The following two sections discuss three popular data structures for pixel level hierarchies. Section 2.3.2.3. approaches the problem of determining where in a hierarchy (which level) to start looking for targets.

2.3.2.1. LoG Channels

LoG channels are a popular research topic because they process images much like the human visual system does. A LoG channel is computed by taking a Laplacian of a Gaussian transform; the Laplacian is the orientation independent second order derivative operator, and the Gaussian provides a *tunable* smoothing, i.e. it allows one to select how much smoothing to use. Figure 2.3.2 shows a LoG model of the human visual system. Initially processing is done at the coarsest levels where everything becomes blob-like. At this level blobs in the left image are easily matched with blobs in the right image. As more information is needed, the higher resolution levels are consulted. Figure 2.3.3 shows a FLIR image and four of its LoG images. A hierarchical vision system would examine each of these representations and try to construct hypotheses about the contents of the scene.

It is interesting to note that while it may be difficult to hypothesize the existence of a road at the lowest channels (because the lowest channels are more dominated by small pixel-to-pixel variations and are less capable of grouping together pixels to construct large scale detail), the same task could prove relatively easy in a coarser channel.

2.3.2.2. Pyramids and Quadtrees

Pyramids and quadtrees may be the most promising hierarchical data structures for reasoning in computer vision because they are naturally a hierarchy with less information stored in the upper level and more details stored at the lower levels. Such an arrangement allows for quick

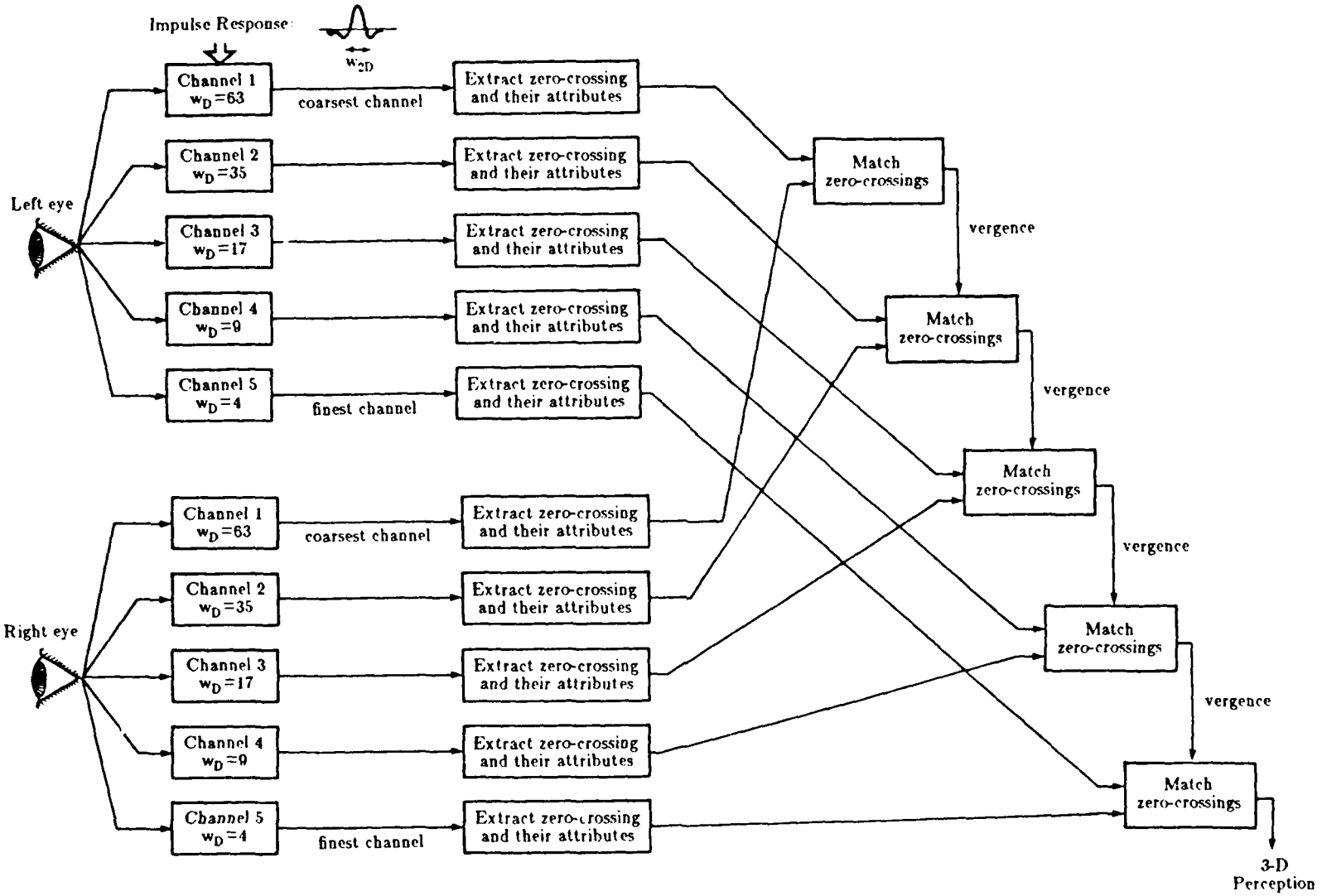


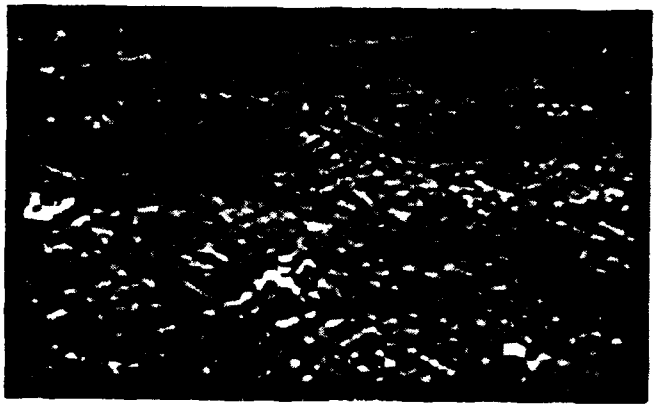
Figure 2.3.2 LoG channel model of Human visual system.



(a)



(b)



(c)



(d)



(e)

Figure 2.3.3 LoG channels of a FLIR image. (a) W_{2D}=32, (b) W_{2D}=16, (c) W_{2D}=8, (d) W_{2D}=4, (e) Original image.

location of relevant parts of the image while ignoring irrelevant details.

Figure 2.3.4 shows a 256 x 256 FLIR image of three targets which is mapped to each higher level by taking an average over the four pixels below it. A hierarchical vision system would scan the 4 x 4, 8 x 8, and 16 x 16 level and not find much of interest. This is because there is nothing large in the image. The bright spots and dark spots at the 32 x 32 level may contain something of interest. The pyramids below these spots could then be examined in hopes of finding a target. The pyramid has reduced the search space from blindly searching all 256 x 256 = 64k pixels of the original image to searching 4 x 4 + 8 x 8 + 16 x 16 + 32 x 32 = 1360 pixels. After searching these pixels the search was directed to area of the image that were most likely to contain information of interest.

This example has shown how a pyramid constructed by averaging pixels as a map between levels can reduce the number of pixels to be searched. (A similar example could be constructed for quadtrees.)

2.3.2.3. Classification on a Pyramid

A question to ask is, **How do you know which level to start searching on?** To answer this question we constructed pyramids out of each of the targets used in the BRITT data classification experiment. Each level of each of the pyramids were constructed by averaging the four pixels from the level below as discussed in the previous paragraphs. We then ran the classification experiments on each of the levels. We had hoped to see the classification errors rise slowly as we moved up the pyramid since there was less information at each level. Then we expected to see the error rate jump once we hit a level where too much information was averaged out. However, Table 2.3.4 shows that results of are not the results we expected.

Table 2.3.4 Results of classification experiments on pyramid built from BRITT data. The values in the table are the percentage of the targets misclassified.

	feature set 1		feature set 2		feature set 3	
level	upper	lower	upper	lower	upper	lower
0	62.3%	51.0%	46.7%	42.7%	63.6%	11.1%
1	61.8	53.2	48.0	44.4	57.8	53.8
2	60.2	52.7	47.6	31.1	59.6	32.4
3	61.3	57.9	49.3	41.8	61.3	61.3
4	59.1	46.2	56.0	54.7	64.9	64.9

The results show that the error rates do not really increase as we move up the pyramid to levels

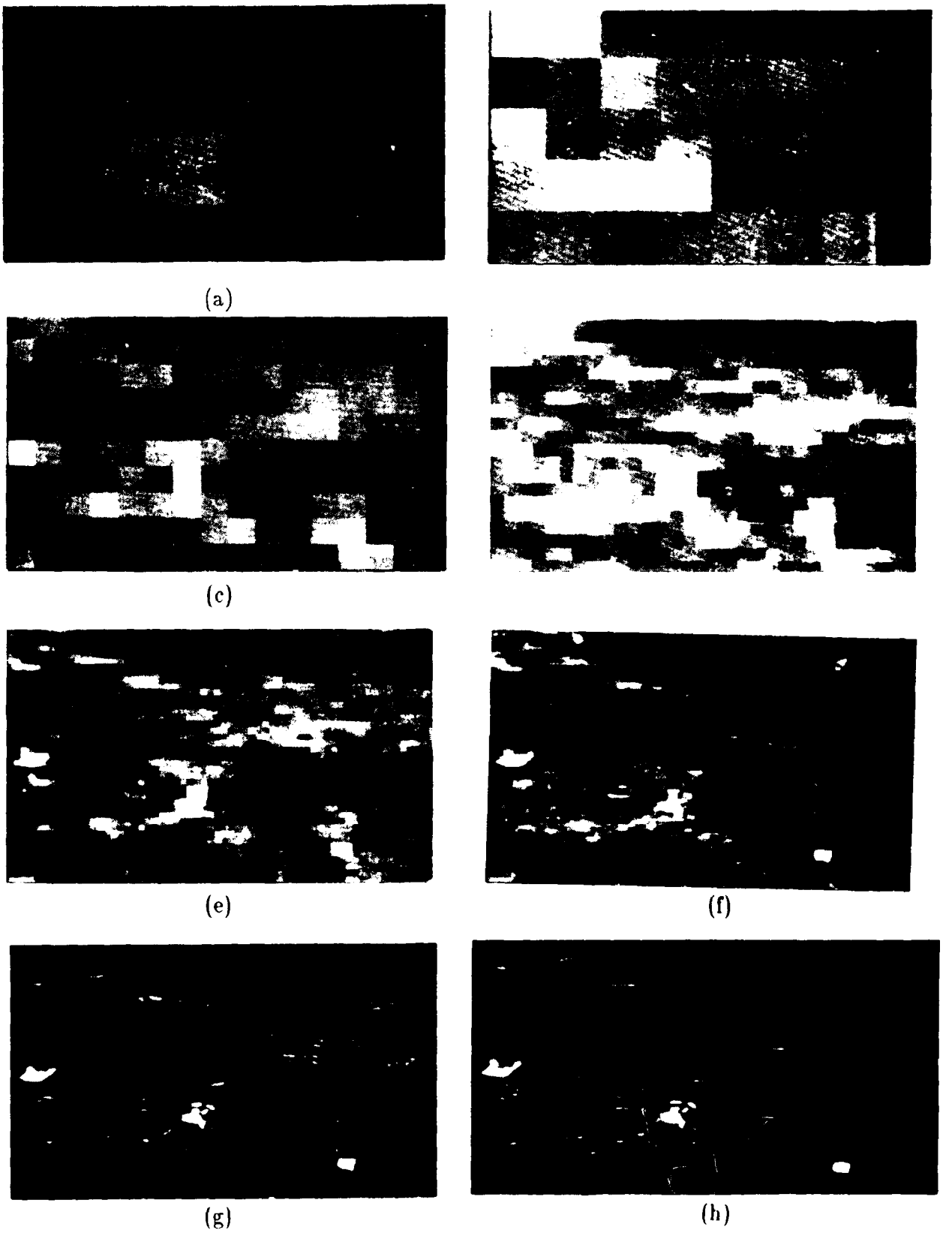


Figure 2.3.4 Different levels of a pyramid representation of a 320 x 496 FLIR image padded to 512 x 512. (a) 4 x 4, (b) 8 x 8, (c) 16 x 16, (d) 32 x 32, (e) 64 x 64, (f) 128 x 128, (g) 256 x 256, (h) original 512 x 512 image.

with less information. One could hastly conclude that the important information is preserved from level to level in the pyramid. However, take for example the 62.3% lower bound error rate for feature set 1 on level zero. This is a very poor error rate (as discussed in Section 2.1.1.4.4.) the error rate is so poor on level 0, it should be no surprise that it does not get much worse at higher levels simply because **it is hard to make any more errors**. With three classes to pick from, a random guess would result in an error rate of 66.6%. And level 0 already has an upper bound of 62.3%.

One can draw one more conclusion from these results: There is a mismatch between the digital sampling rates implied by the matrix sizes used for FLIR data and the intrinsic resolution of a FLIR sensor. As demonstrated by the feature set 2 above, a reduced representation of a FLIR image on a much smaller matrix does not cause any noticeable reduction in the classification information.

2.3.3. Using a Global Map to Improve Edge Labeling

FLIR images rarely have enough clarity and pixels on target to extract and accurately label detailed edges as shown in Figure 2.3.1. More often the line segments will consist of several edges separated by gaps, edges with many noise edges attached, or both. Many schemes have been devised to eliminate noise edges and close gaps between edges in the same line. A better approach to the problem is to include global information about the area being processed. For example assume we have a global map of the area being viewed by the sensor and a rough idea of where the sensor is in the map. The map would contain objects such as rivers, roads, lakes, mountains, etc. The location of the sensor could be found by some global navigation system. The map and approximate location are then used to build an approximate model of what is being viewed by the sensor and the model is used to guide the labeling of the edges. The regions bounded by the edges could be labeled as sky, mountains, plains, forest, etc. Such classification could then be used to direct the target detector where to look for targets.

In the Robot Vision Lab we are building a general purpose software tool called PSEIKI [AndKak87] (a **P**roduction **S**ystem **E**nvironment for **I**ntegrating **K**nowledge with **I**mages). Our plan is to apply PSEIKI to NVL projects where symbolic reasoning and integration with world knowledge can be applied. PSEIKI is a rule based system written in OPS.

The following section describes a system which takes some images at various locations on a sidewalk and uses a global map of the sidewalk to help label the edge segments which belong to each edge of the sidewalk. We could have used FLIR data however, we do not have any global maps which correspond with the FLIR images in our database.

2.3.3.1. System Goals

The first goal of PSEIKI is: given an approximate location on the sidewalk and a corresponding view, find the edges of the sidewalk. Figure 2.3.5 shows the kinds of views that were taken of the sidewalk. The images on the right, used for drawing inferences about the

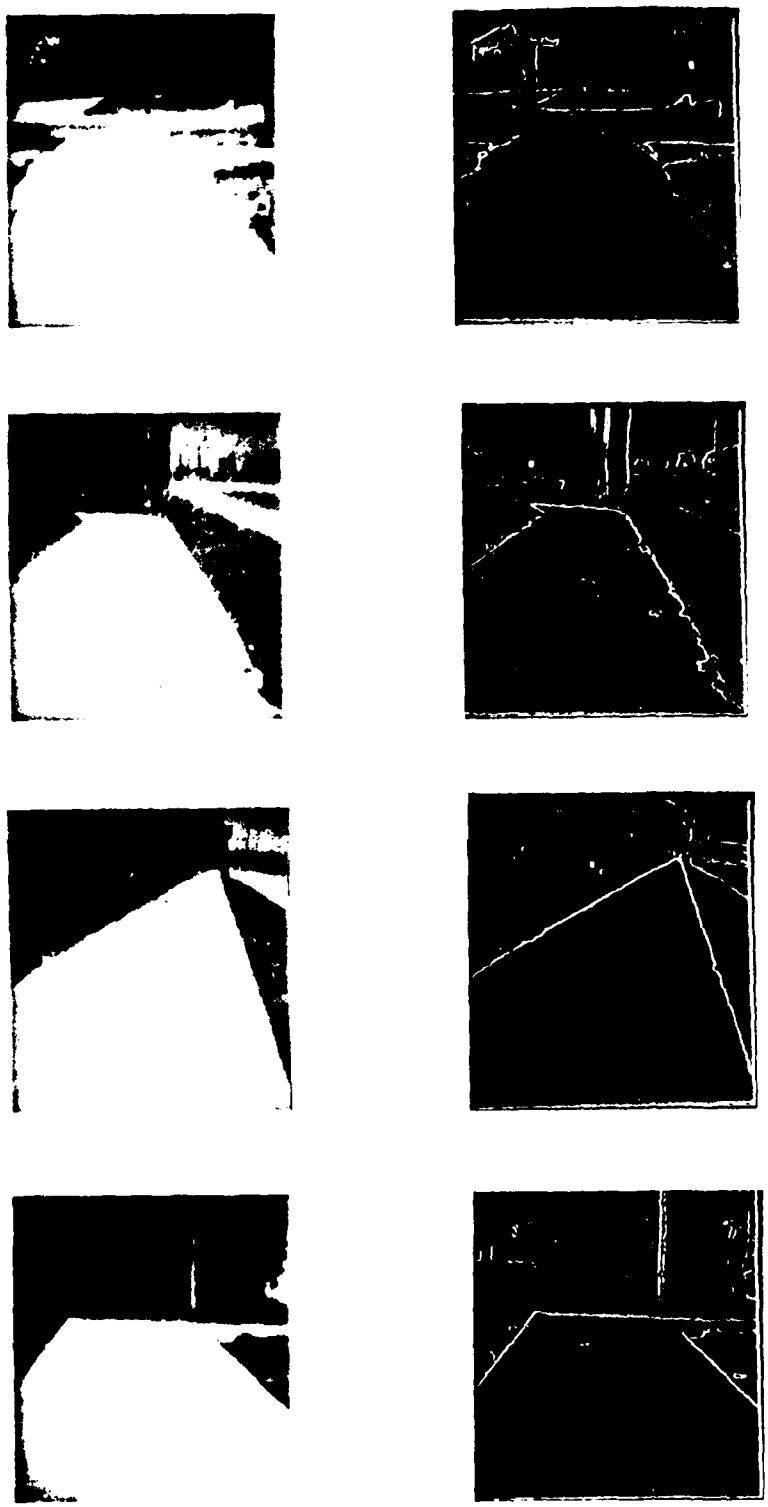


Figure 2.3.5 Typical images to be processed by PSEIKI.

location and the direction of the sidewalk, were obtained by applying edge detection and thinning operators to the images on the left. The architecture for the edge labeling process is shown in Figure 2.3.6. The process has two subsections: the preprocessor/pixel-to-symbol converter and a rule-based edge labeler.

2.3.3.2. Preprocessing and Conversion to Symbolic Form

The preprocessor accepts digitized images and outputs binary edges suitable for conversion into symbolic form. The edges are detected by applying a Sobel operator to the digitized gray scale image. These edges are then thinned via Eberlein's algorithm [Eber76] and thresholded. The resulting binary images are thinned again to produce edges that are at most one pixel wide. Small edges are also deleted by the preprocessor. At this point, the image is ready to be converted into symbolic form.

The conversion to symbolic form is accomplished via an algorithm based on the Nevatia-Babu line-finder [NeBa80]. In this process, the following steps are performed. First, some pixels are labeled as vertices. The pixels so labeled are edge endpoints and the points at which two or more edges intersect. The edges in the segmented image are then traced from the starting to ending vertices and are represented as broken line segments. After each edge is converted to symbolic form, it contains the following information: edge number, start vertex, end vertex, length and strength (average gradient magnitude). Likewise, each vertex contains the following information: row coordinate, column coordinate, vertex number and degree.

2.3.3.3. Rule Based Edge Finder

The rule-based labeling system is written in OPS5 [BFKM86] and is split into three subsystems. The first subsystem is statement driven and does not employ an inexact reasoning scheme. Its purpose is to overcome segmentation deficiencies and reduce the amount of data seen by the sections that do use inexact reasoning. There are two main ways that the current segmentation is deficient. First the segmentation procedure leaves small edges caused by noise. Although many of these edges are eliminated during the segmentation process, others still remain because they are connected to longer segments. These noise-edges often look like the example in Figure 2.3.7. The first section of the expert system eliminates these "dangling" edges (all segments which are shorter than a specified length and have a degree one vertex). Figure 2.3.8 is a pseudo-code example of one the rules used to delete a small, dangling edge.

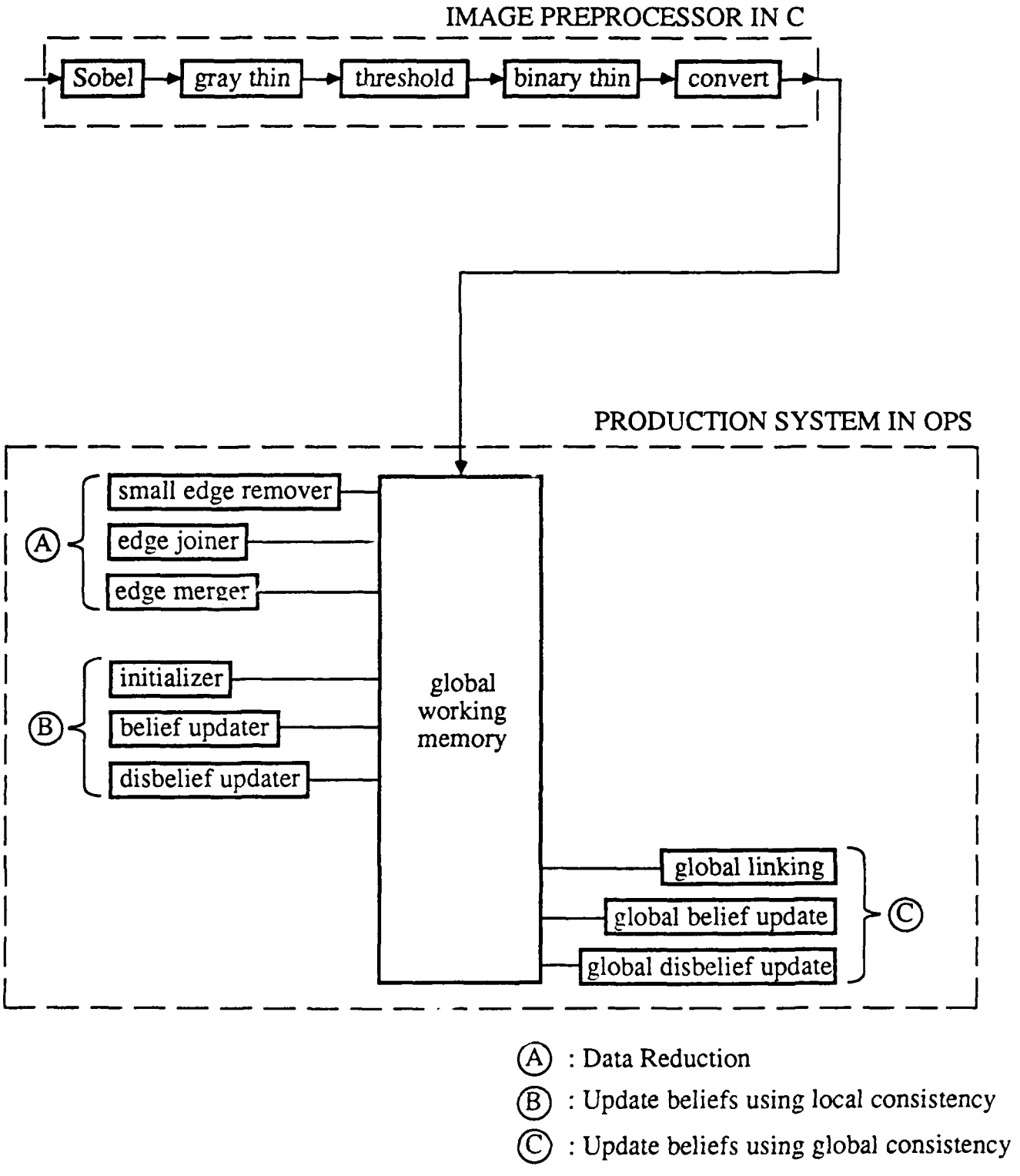


Figure 2.3.6 Architecture for edge labeling process.

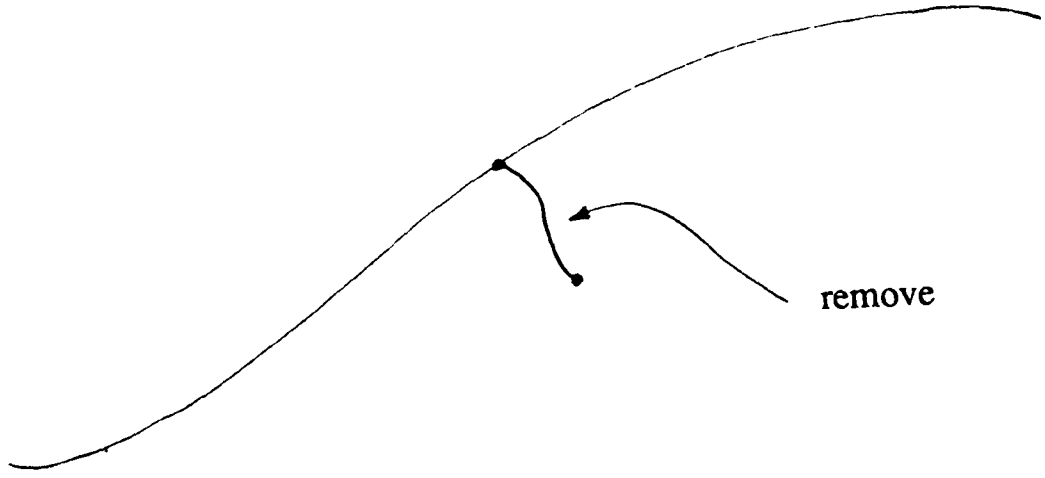


Figure 2.3.7 Example of a noise-edge.

```
RULE: remove-small-segments
   IF: current context is remove-small-segments
    AND: segment with name <small-segment> has
         length < 4 and
         start vertex with name <start> and
         end vertex with name <end>
    AND: vertex with name <start> has
         degree 1
   THEN: delete segment <small-segment>
    AND: make contexts to decrement the degrees
         of vertices <start> and <end>
```

Figure 2.3.8 Rule to delete small edges.

The segmentation used also produces artifacts that break lines into smaller line segments. The system tries to compensate for this fact by rejoining these broken line segments. Finally the first section combines, into a single line, segments that are joined at a degree-two vertex. This action is demonstrated in Figure 2.3.9 and the pseudo-code for a rule that performs the merging action is shown in Figure 2.3.10.

```
RULE: merge-joined-segments
   IF: current context is merge-joined-segments
    AND vertex with name <common-vertex> has degree 2
    AND segment with name <segment1> ends at vertex <common-vertex>
    AND segment with name <segment2> ends at vertex <common-vertex>
   THEN: remove <common-vertex>, <segment1>, <segment2>
    AND create a new segment to replace <segment1> and <segment2>
```

Figure 2.3.10 Rule to merge connected segments into a single segment.

The overall result of these subprocesses is a cleaner image containing a substantially reduced number of line segments. Experimental results demonstrate that the amount of pruning is between 66% and 75%; this is obviously a large reduction in the amount of data. Figure 2.3.11 shows an example of input to and output from the first section of the expert system.

The second subsystem performs segment labeling and confidence estimation. The system uses the expected position in the global map to estimate where it should see the sidewalk's edges in its field of view. (for example, if the sidewalk is expected to have a right hand turn, it should see the left and right edges for the sidewalk before the turn and the top and bottom edges for the sidewalk after the turn.) The expert system was designed to use the position of expected (model) edges to label those found in the segmented image. The edge labels assigned consist of the following information: the name of the corresponding model edge and a certainty factor between 1 and −1 describing the confidence attached to that labeling. All segments are initially

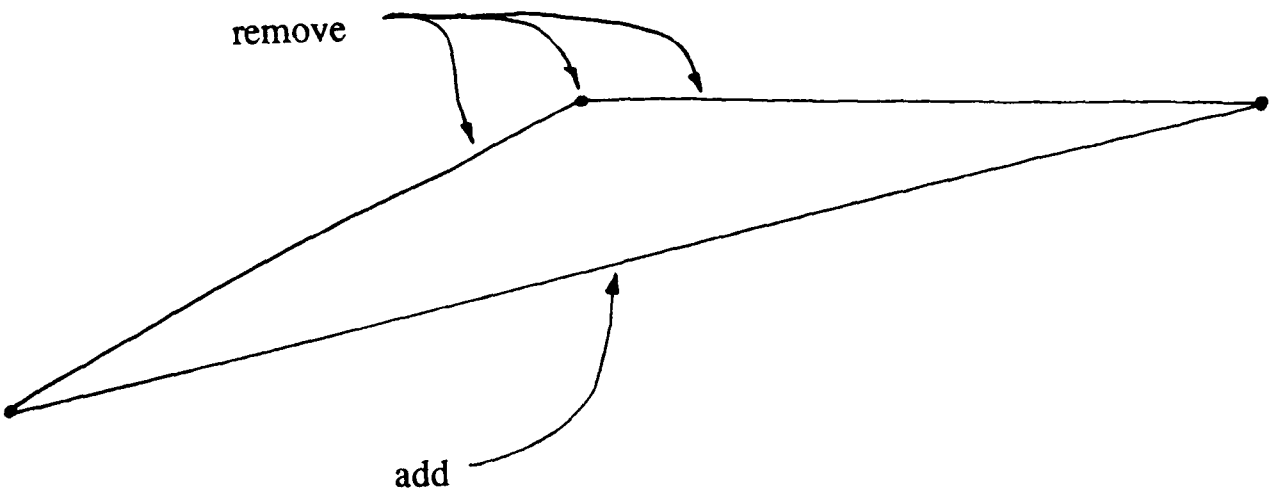


Figure 2.3.9 Example of edge merging.

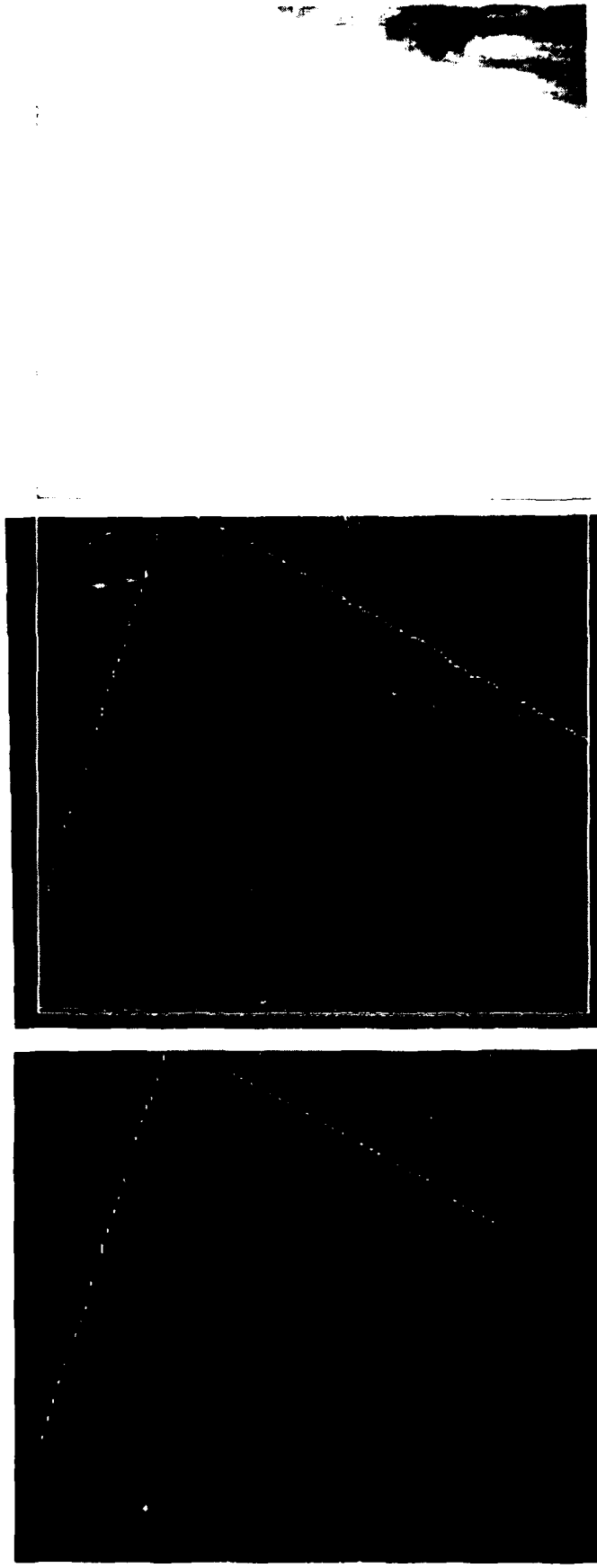


Figure 2.3.11 Example of the first section of the expert system. (a) Input image. (b) Output of preprocessor. (c) Output of first section of expert system.

labeled as the model edge to which they correspond most closely.

The initial labels and confidence estimates are based on the collinearity between the edge in question and the model edge. Collinearity is defined as follows

$$Col(<segment\,1>,<segment\,2>) = \frac{D_{max} - D_{seg}}{D_{max}} \cos\theta$$

where θ is the angle between the detected and the model segment; D_{seg} is the distance from the middle of the segment to the model edge and D_{max} is the maximum possible distance from the model edge to the detected edge (see Figure 2.3.12). If the edge is further than D_{max} away from the model edge, the collinearity is defined to be zero.

After the initial labels are assigned, they can not be changed. However, every segment's confidence value is updated based on the consistency between the labels of all other segments and the expected geometry. The belief that a segment's label is correct based on new evidence is computed separately from the confidence that it is incorrect (its disbelief). New and old evidence is combined using a variant of the Dempster-Shafer theory of evidence [Shaf76]. The following rules demonstrate how the (dis)belief that a segment's label is correct is updated using labels of other segments. If a segment is thought to lie on the same model edge, the confidence that the label is correct is increased if the segments are highly collinear. This is expressed as shown in Figure 2.3.13

```
RULE: update-belief-compatible
   IF current context is update-certainty of <segment1>
    AND: label of <segment1> is <label1>
    AND: there is a segment named <segment2>
         with label <label2> and
         certainty factor <cf> > 0.2
    AND: the angle between the models
         with labels <label1> and <label2> = 0
   THEN: new certainty that label for <segment1> is correct is equal to
         Col(<segment1>, <segment2>) * <cf>
```

Figure 2.3.13 Rule used to update belief of segments believed collinear.

If the two segments are believed to correspond to different model edges, a segment's confidence is updated based on how closely the angle between the two segments matches the corresponding model angle. The new belief is defined as the cosine of the difference between the expected and measured angles multiplied by the confidence value of the updating edge. Figure 2.3.14 shows an example of a rule that is used to update a segment's belief if the two segments correspond to different model edges.

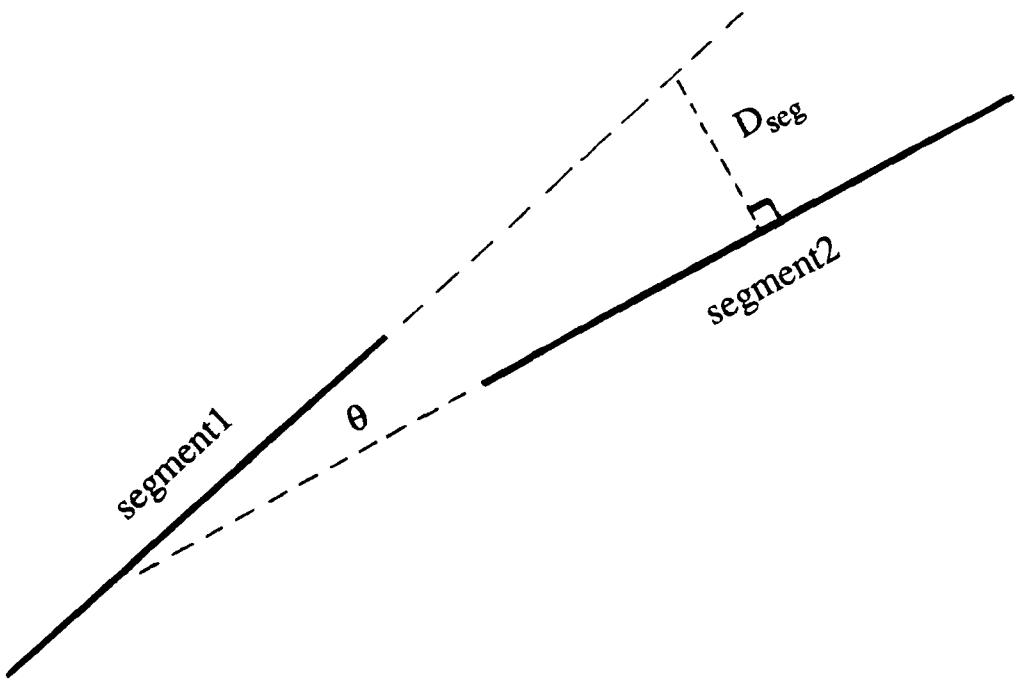


Figure 2.3.12 Geometry used in definition of colinearity.

```
RULE: update-belief-incompatible
   IF current context is update-certainty of <segment1>
    AND: label of <segment1> is <label1> and
         length <length>
    AND: there is a segment named <segment2>
         with label <label2> that
         makes an angle of <true-angle> with <segment1>
         and has a certainty factor <cf> > 0.2
    AND: the angle between the models
         with labels <label1> and <label2> is <model-angle>
   THEN: new certainty that label for <segment1> is correct is equal to
          cos(<true-angle> - <model-angle>) * <cf> * <scale-factor>
```

Figure 2.3.14 Rule used to update belief of segments believed noncollinear.

Therefore, if the angle expected between the edges is exactly the angle found, the new belief that the labeling is correct is equal to the belief that the updating edge's label is correct. Likewise, the new disbelief is defined as the sine of the difference of the angles scaled by the confidence of the updating edge's label. This is demonstrated by the rule in Figure 2.3.15.

```
RULE: update-disbelief-incompatible
   IF current context is update-certainty of <segment1>
    AND: label of <segment1> is <label1> and
         length <length>
    AND: there is a segment named <segment2>
         with label <label2> that
         makes an angle of <true-angle> with <segment1>
         and has a certainty factor <cf> > 0.2
    AND: the angle between the models
         with labels <label1> and <label2> is <model-angle>
   THEN: new certainty that label for <segment1> is incorrect is equal to
          sin(<true-angle> - <model-angle>) * <cf> * <scale-factor>
```

Figure 2.3.15 Rule used to update disbelief.

For example, if the expected angle between segments was 60° but the measured difference was 30° the new disbelief would be one half the belief that the updating segment's label was correct.

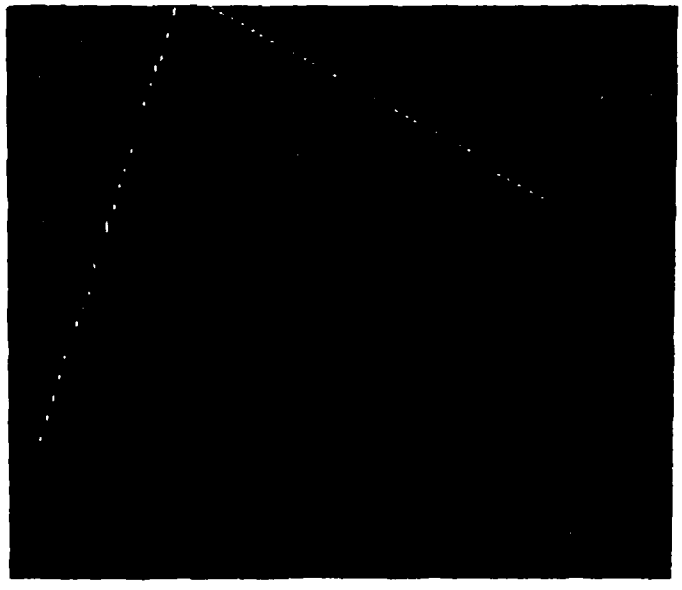
The third subsystem used in PSEIKI is not yet complete; it will duplicate the processes used in the first two but will work on groups of edges instead of singletons. Work also must be done to incorporate camera calibration information into the reasoning system; this will remove distortions due to perspective imaging.

2.3.3.4. Experimental Data

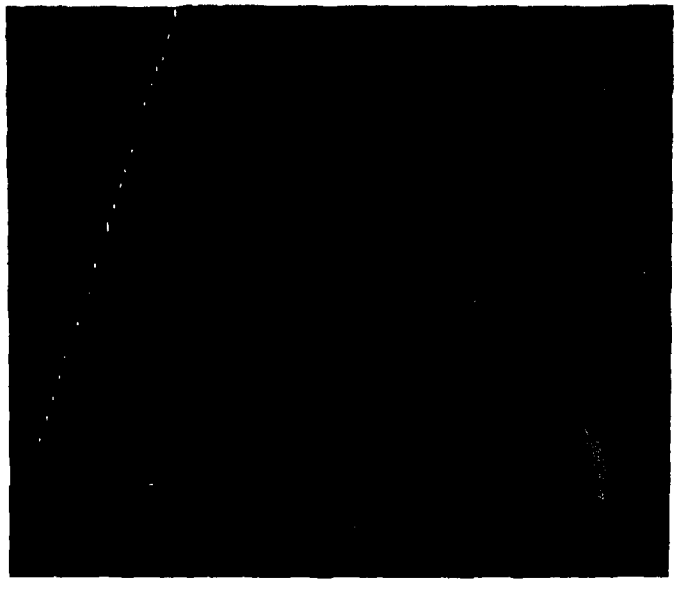
The expert system was run on data from a number of digitized images, one of which is presented here. The image shown was taken slightly to the left of the center of of a straight section of sidewalk. This tests the expert system in the case of a positional error. Figure 2.3.11.a shows the unprocessed image. The image in Figure 2.3.11.b shows the output of the image segmenter, as can be seen it contains a large number of broken, noisy line segments. The output of the first section of the expert system is shown in Figure 2.3.11.c. The number of segments was reduced from 99 to 25 segments in this image. Figure 2.3.16 shows the segments labeled as left and right edges of the sidewalk for all labels with positive certainty. As can be seen by the results, the system performed admirably on this image. It found both edges of the sidewalk and only included one segment that was not a sidewalk edge.

2.3.3.5. Conclusion

The present expert system demonstrates the validity of using a production system to implement a geometric reasoning system. Such a system could a applied to FLIR images to identify major components of the image such as the sky, mountains, forests, etc. Each of the three components of the ATR process (detection, segmentation, and classification) could produce more accurate results once these regions are found and correctly labeled. For example, the detector would know not to look in the sky for tanks. The segmenter could use special segmentation algorithms tuned to the type of region it is segmenting. Finally the classifier could use the type of region information to tune its classification process.



(a)



(b)



(c)

Figure 2.3.16 Example of edges labeled by the second section of expert system. (a) Input to second section of expert system. (b) Segments labeled as the *left* edge. (c) Segments labeled as the *right* edge. vertices

3. LASER RADAR RANGE DATA PROCESSING

We believe that in the multi-sensor ATR systems of the future, LADAR will play an important role for verification of targets detected by FLIR; LADAR will also play an important role in estimating the aspect of a target -- an important piece of information about the hostile intent of the target. In this section, we will discuss the extensive amount of work we have done on segmentation of targets from LADAR range images; we will also report on the exploratory work carried out by us on different reasoning strategies that could be employed for identifying targets and estimating their aspects. It appears to us that for targets positioned within a kilometer from a sensor with current resolution capability, it should be possible to carry out identification and aspect estimation by reasoning in terms of target surfaces and their interrelationships. However, for targets farther than a kilometer, sensors of today do not permit a reliable extraction of surfaces and recognition and aspect estimation must depend upon just the silhouette.

3.1. DATA DESCRIPTIONS

LADAR data from two field tests were used in addition to synthetic data. The following sections describe the data that was collected.

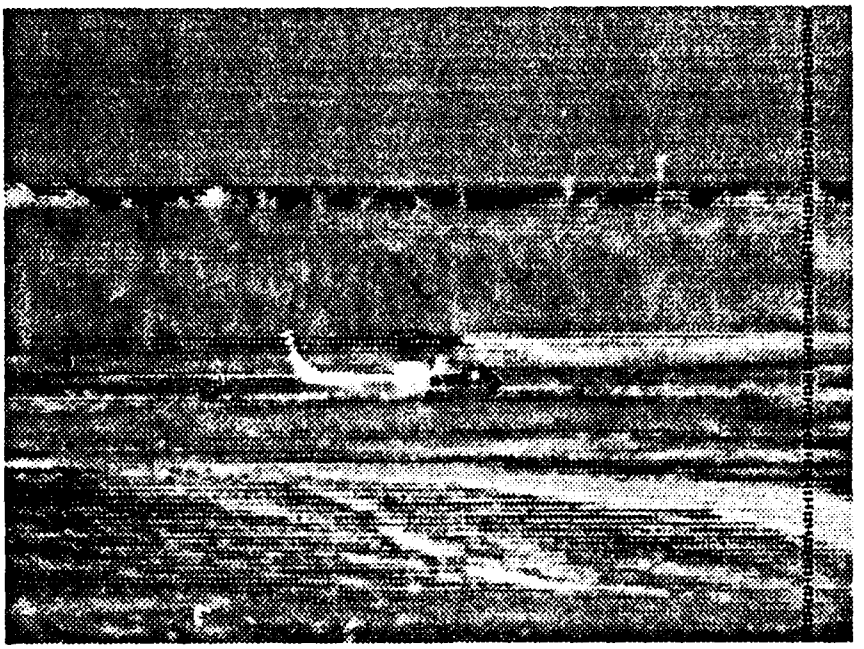
3.1.1. Description of the 1986 A.P. Hill Laser Radar Data

The laser radar range images described as the 1986 AP Hill Laser Radar Data were taken by the LADAR sensor described in [Rayt] at Fort A. P. Hill, Virginia on the 24th through the 28th of March 1986. Six images were selected from all the data taken at A. P. Hill field test. These images, shown in Figures 3.1.1-3.1.6, were selected to give a representative sampling of targets, ranges, clutter, fields of view, and occlusion. Note that these figures show both a FLIR image (top) and a LADAR image (bottom), **only the range images were used for these experiments.** Each LADAR image is 160 by 96 pixels, with 8 bits per pixel. The LADAR sensor can collect range images at different fields of view (FOV); Table 3.1.1 shows the instantaneous FOV's and the FOV of each image for the three FOV's used in Figures 3.1.1-3.1.6.

Table 3.1.1 Field of views for the LADAR images in Figures 3.1.1-3.1.6.

FOV #	Instantaneous FOV Azimuth x Elevation (mrad)	FOV (mrad)
1	0.1 x 0.1	16 x 9.6
2	0.1 x 0.2	16 x 19.2
5	0.2 x 0.2	32 x 19.2

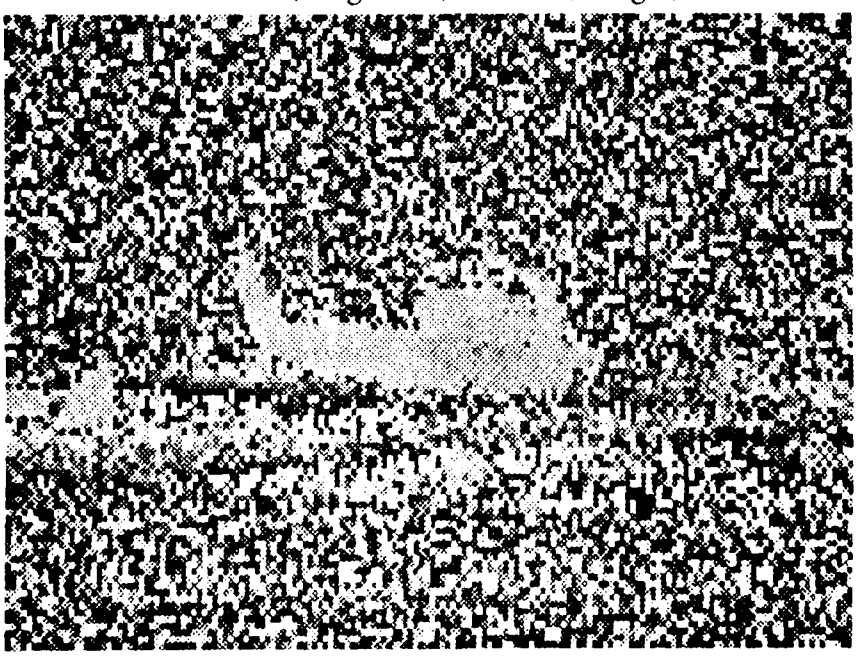
File = rvl1:~nvl/range-data/324/flir/images/im03



Size = 320 x 192

TARGET= U1H HELICOPTER
ASPECT= SIDE VIEW
RANGE= 1KM
FRAME 09951 RUN NO. 3
DATE 03/24/86 TIME 1335.46.069
TEST SITE FT A.P. HILL, VA
THE TEST SITE IS AT THE DROP ZONE AT A.P. HILL

File: rvl1:~nvl/range-data/324/laser/images/im03



Size = 160 x 96

TARGET - UH-1
ASPECT - BSR
FIELD-OF-VIEW - FOV5

Figure 3.1.1 FLIR and Laser Radar Range image for target ap1.32403.

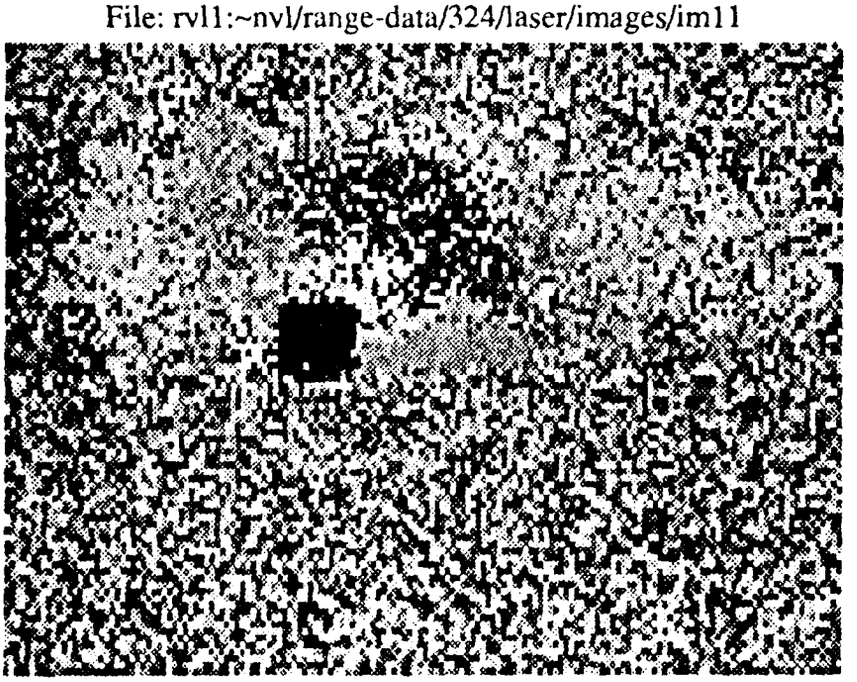
File = rvl1:~nvl/range-data/324/flir/images/im11



Size = 320 x 192

TARGET= M60A2
ASPECT= RIGHT VIEW
RANGE= 2050M
FRAME 86247 RUN NO. 8
DATE 03/24/86 TIME 1616.01.692
TEST SITE FT A.P. HILL, VA
THE TEST SITE IS AT THE DROP ZONE AT A.P. HILL

File: rvl1:~nvl/range-data/324/laser/images/im11



Size = 160 x 96

TARGET - M60A2
ASPECT - BSR
FIELD-OF-VIEW - FOV2

Figure 3.1.2 FLIR and Laser Radar Range image for target ap1.32411.

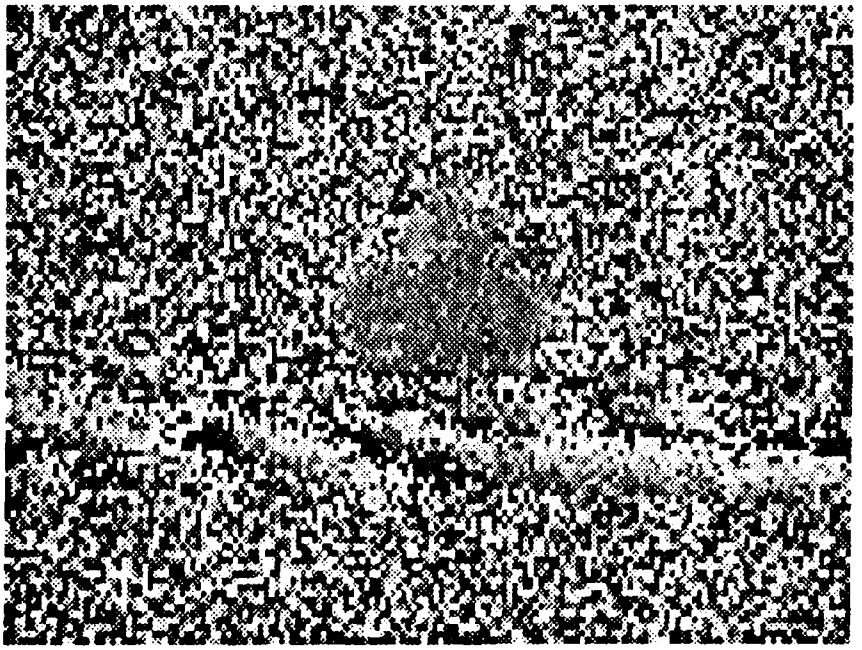
File = rvl1:~nvl/range-data/325/flir/images/im04



Size = 320 x 192

TARGET= M60A2
ASPECT= RIGHT VIEW
RANGE= 1935M
FRAME 27691 RUN NO. 2
DATE 03/25/86 TIME 0909.34.192
TEST SITE FT A.P. HILL, VA
THE TEST SITE IS AT THE DROP ZONE AT A.P. HILL

File: rvl1:~nvl/range-data/325/laser/images/im04



Size = 160 x 96

TARGET - M60A2
ASPECT - BSR
FIELD-OF-VIEW - FOV1

Figure 3.1.3 FLIR and Laser Radar Range image for target ap1.32504.

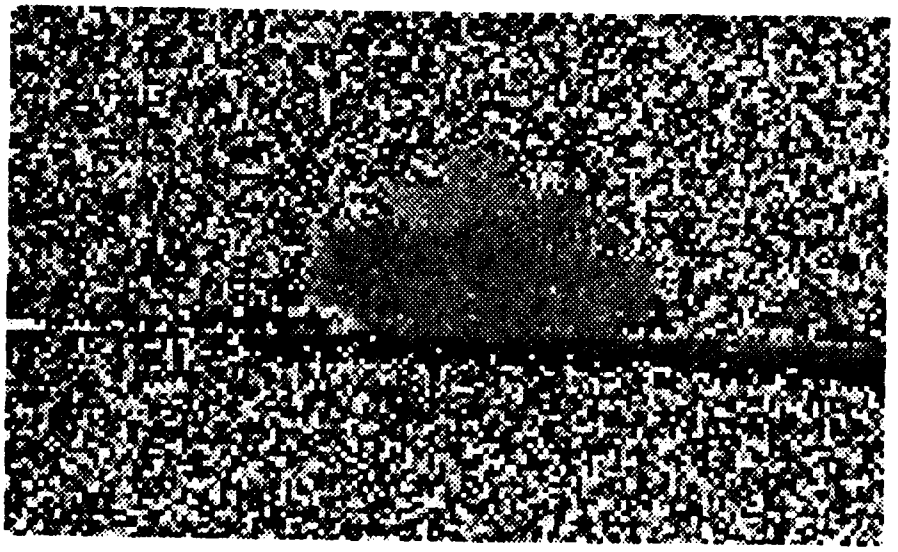
File = rvl1:~nvl/range-data/326/flir/images/im33



Size = 320 x 192

TARGET= M60A2
ASPECT= RIGHT VIEW
RANGE= 1190M
FRAME 30200 RUN NO. 18
DATE 03/26/86 TIME 0400.01.100
TEST SITE FT A.P. HILL, VA
THE TEST SITE IS AT THE DROP ZONE AT A.P. HILL

File: rvl1:~nvl/range-data/326/laser/images/im33

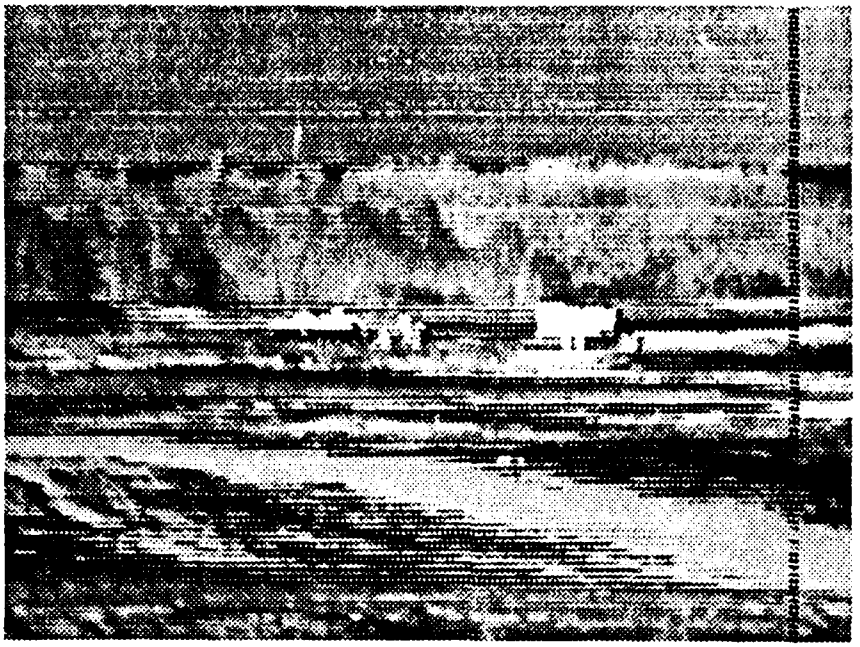


Size = 160 x 96

TARGET - M60A2
ASPECT - BSR
FIELD OF VIEW - FOV1

Figure 3.1.4 FLIR and Laser Radar Range image for target ap1.32633.

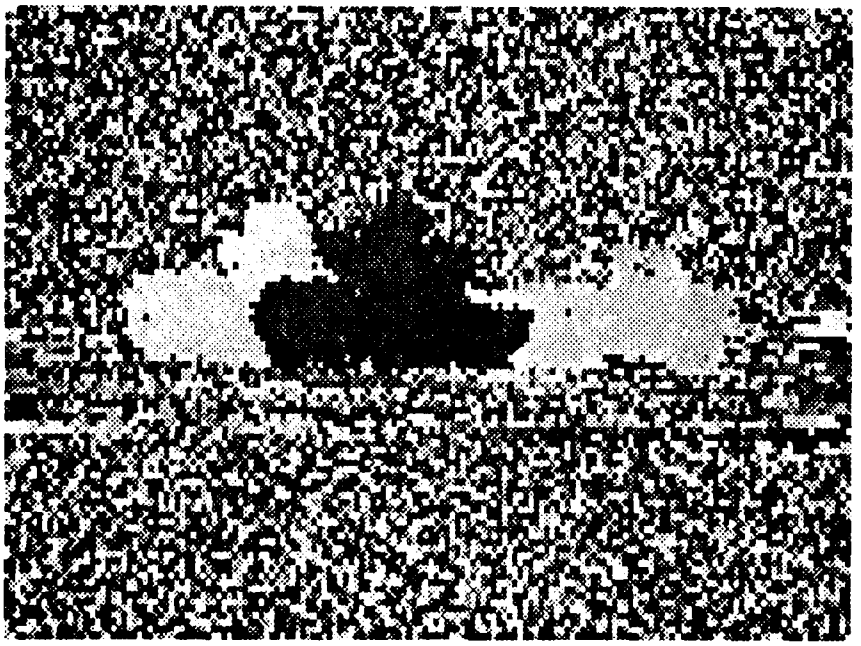
File = rvl1:~nvl/range-data/328/flir/images/im37



Size = 320 x 192

TARGET= CENTER OF CLUSTER
ASPECT= RIGHT VIEWS
RANGE= 1540M
FRAME 30322 RUN NO. 7
DATE 03/28/86 TIME 0030.00.102
TEST SITE FT A.P. HILL, VA
THE TEST SITE IS AT THE DROP ZONE AT A.P. HILL

File: rvl1:~nvl/range-data/328/laser/images/im37

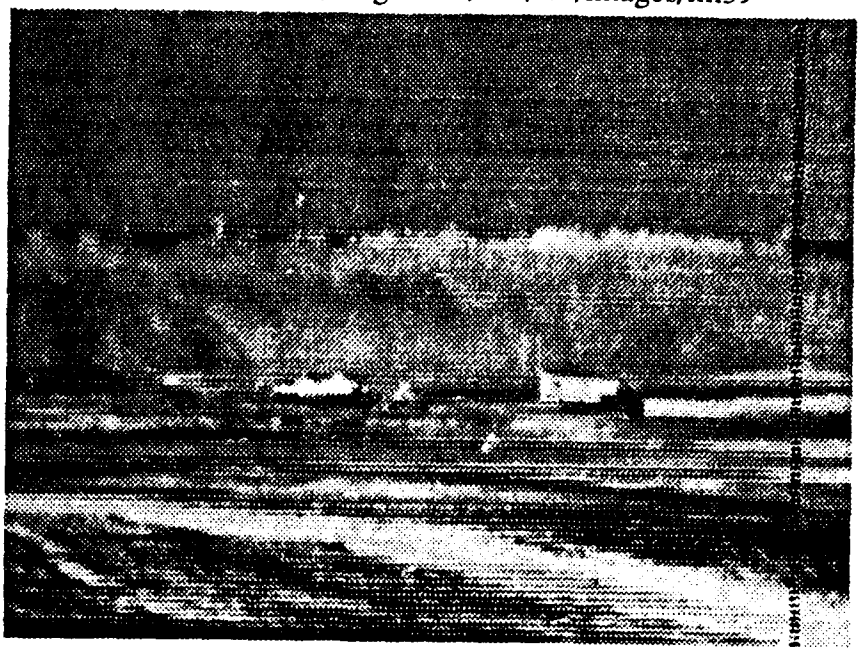


Size = 160 x 96

TARGET - 2-M60A2/2.5TT
ASPECT - BSR
FIELD-OF-VIEW - FOV1

Figure 3.1.5 FLIR and Laser Radar Range image for target ap1.32837.

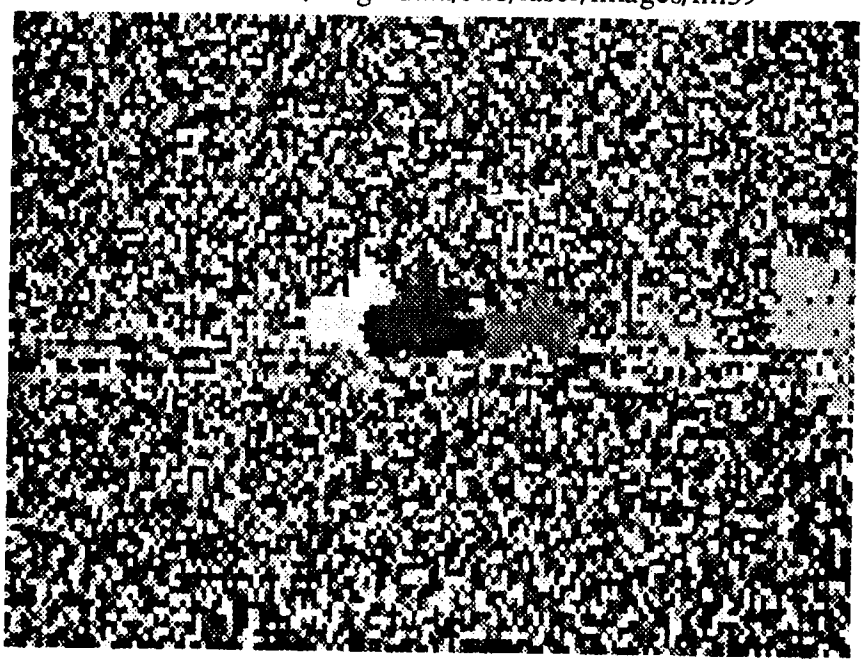
File = rvl1:~nvl/range-data/328/flir/images/im39



Size = 320 x 192

TARGET= A CLUSTER OF OBJECTS
ASPECT= RIGHT VIEWS
RANGE= 1540M
FRAME ?>353 RUN NO. 10
DATE 03/28/86 TIME 216:.40.270
TEST SITE FT A.P. HILL, VA
THE TEST SITE IS AT THE DROP ZONE AT A.P. HILL

File: rvl1:~nvl/range-data/328/laser/images/im39



Size = 160 x 96

TARGET - 2-M60A2/2.5TT
ASPECT - BSR

Figure 3.1.6 FLIR and Laser Radar Range image for target ap1.32839.

3.1.2. Description of the 1987 A. P. Hill Laser Radar Data

The laser radar range images described as the 1987 AP Hill Laser Radar Data were collected at Fort A. P. Hill, Virginia in June of 1987. For the experiments four images (shown in Figures 3.1.7-3.1.10) were selected to give a variety of ranges, targets and weather conditions. The following sections give information about which images were used and how they were modeled using the ground truth information. Detailed information about the field test can be found in [NeSm87].

3.1.2.1. The Types of Data Collected

During the June 1987 field test five types of data were collected:

FLIR,
LADAR *AM*,
LADAR *FM*,
LADAR *return intensity*, and
millimeter wave (MMW) radar.

The FLIR, MMW and LADAR images were collected from three different sensors each in a different van and are therefore not pixel registered. The three LADAR signals were collected by the same sensor and are all pixel registered. Although this report centers on the processing of the LADAR data, future work will include fusing other sensors such as FLIR and MMW radar with LADAR images. The FLIR images in Figures 3.1.7.1-3.1.10 are shown only to give additional information about the targets in the field test and were not used in any of the processing in this section. The following paragraphs discuss the three types of LADAR data.

The LADAR *AM* data is like the data collected during the 1986 A. P. Hill field test discussed in Section 3.1.1. This data gives very fine *relative* range resolution (as good as 7.2 cm between adjacent range values), but is based on the phase angle of the return signal so that there are range ambiguities caused by phase wrap around. The AM data was collect using 8 bits per pixel. The ambiguity interval is 1875 cm.

The LADAR *FM* image gives *absolute* range at a much coarser resolution than the AM data. It was collected using 12 bits per pixel. **However neither the AM nor the FM images were distributed.** Instead, the LADAR *AM* and the LADAR *FM* images were combined to form a *LADAR resolved range* image which, in theory, should give absolute range with very fine resolution.* The LADAR resolved range image (referred to as LADAR in the rest of this section) was distributed using 32 bits per pixel where each pixel value represents the absolute distance in centimeters to the object.

---

* In practice, the noise component of the FM signal has been measured to be as much as 9 meters on a target at 1400 meters and therefore offsets many of the advantages of the AM component's fine resolution

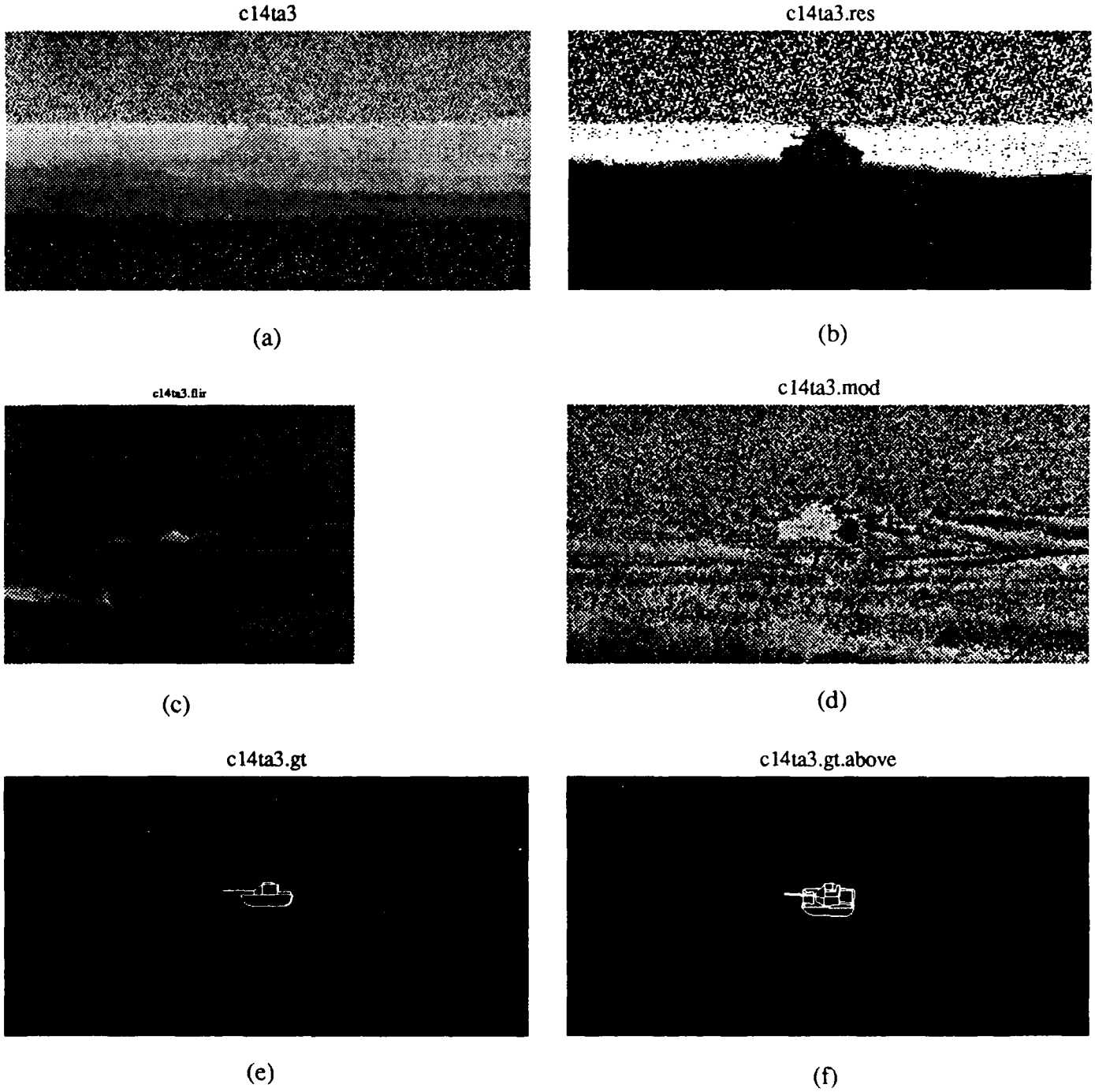


Figure 3.1.7 Image *c14ta3* a) Original 32 bit per pixel image linerly rescaled to 8 bits per pixel, b) Range slice with black being everything in front of 1400m and white being everything behind 1471.4 meters, c) FLIR image of same target, d) original image mod 1875. e) Model of ground truth as seen by the sensor. e) Model of ground truth rotated by 45° to see the orientation of the target.

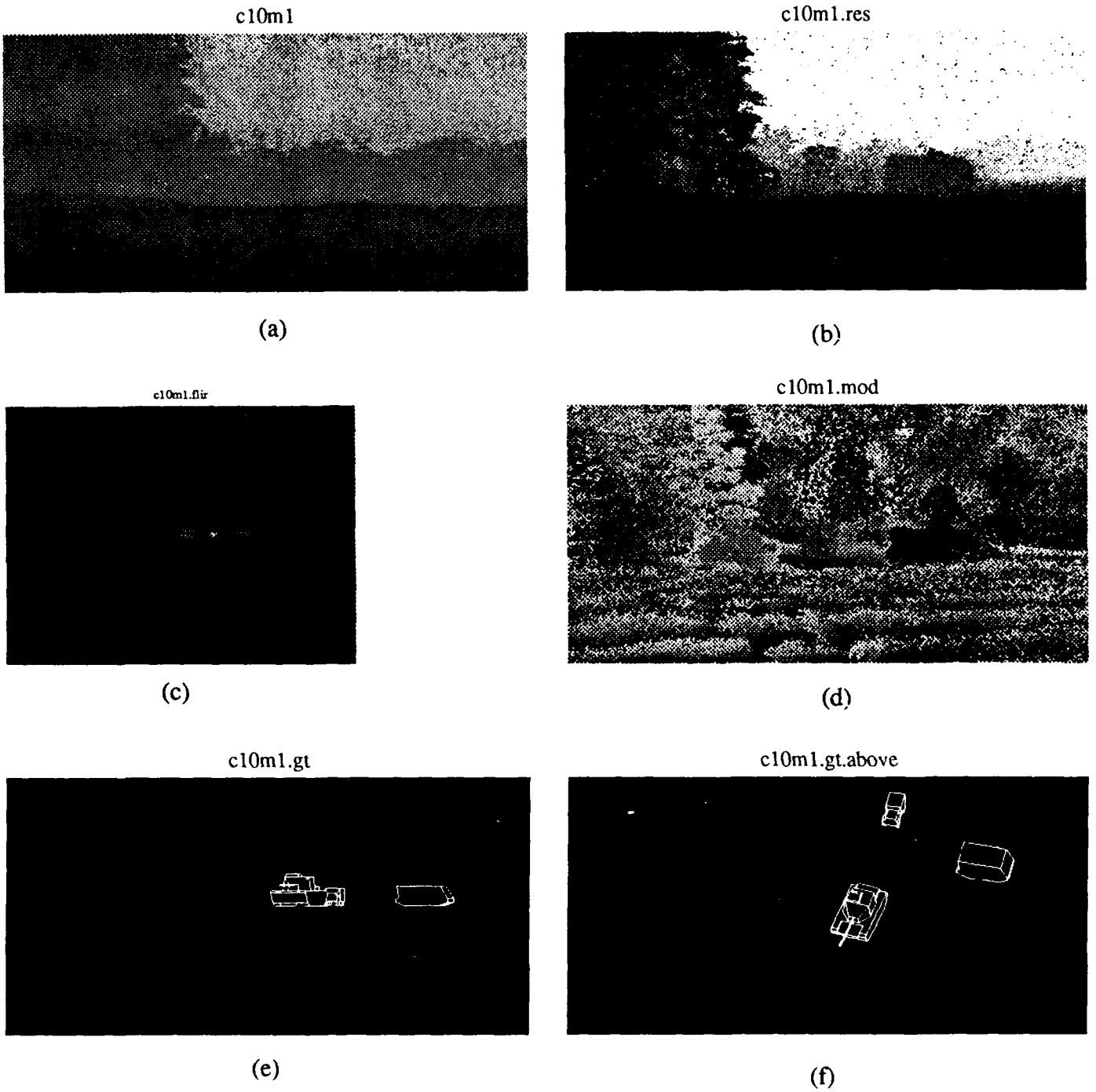


Figure 3.1.8 Image *c10m1* a) Original 32 bit per pixel image linerly rescaled to 8 bits per pixel, b) Range slice with black being everything in front of 1015m and white being everything behind 1086.4 meters, c) FLIR image of same targets, d) original image mod 1875. e) Model of ground truth as seen by the sensor. f) Model of ground truth rotated by 45° to see the relative spacing and orientation of the targets.

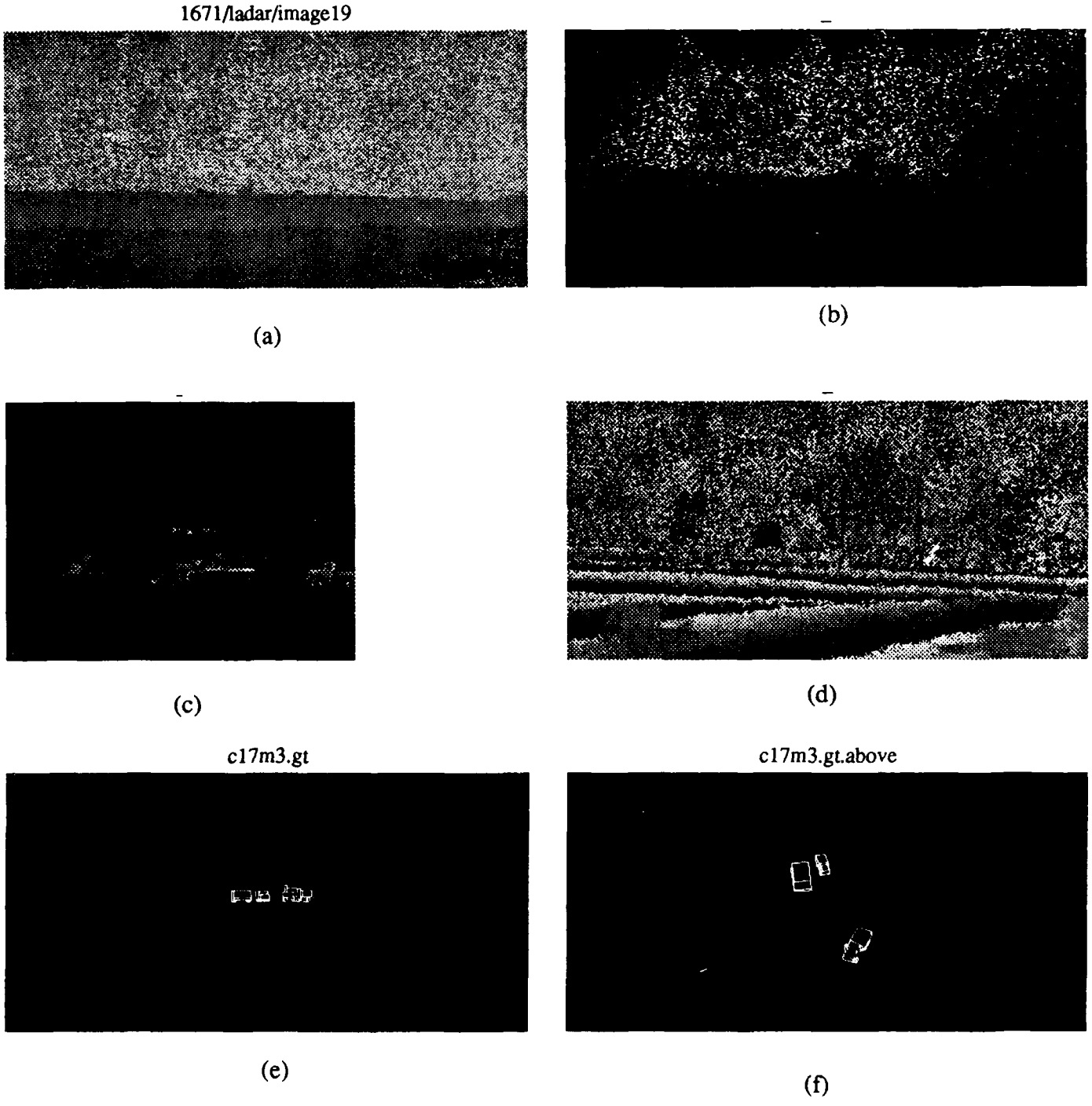


Figure 3.1.9 Image *c17m3* a) Original 32 bit per pixel image linerly rescaled to 8 bits per pixel, b) Range slice with black being everything in front of 1650m and white being everything behind 1721.4 meters, c) FLIR image of same targets, d) original image mod 1875. e) Model of ground truth as seen by the sensor. f) Model of ground truth rotated by 45° to see the relative spacing and orientation of the targets.

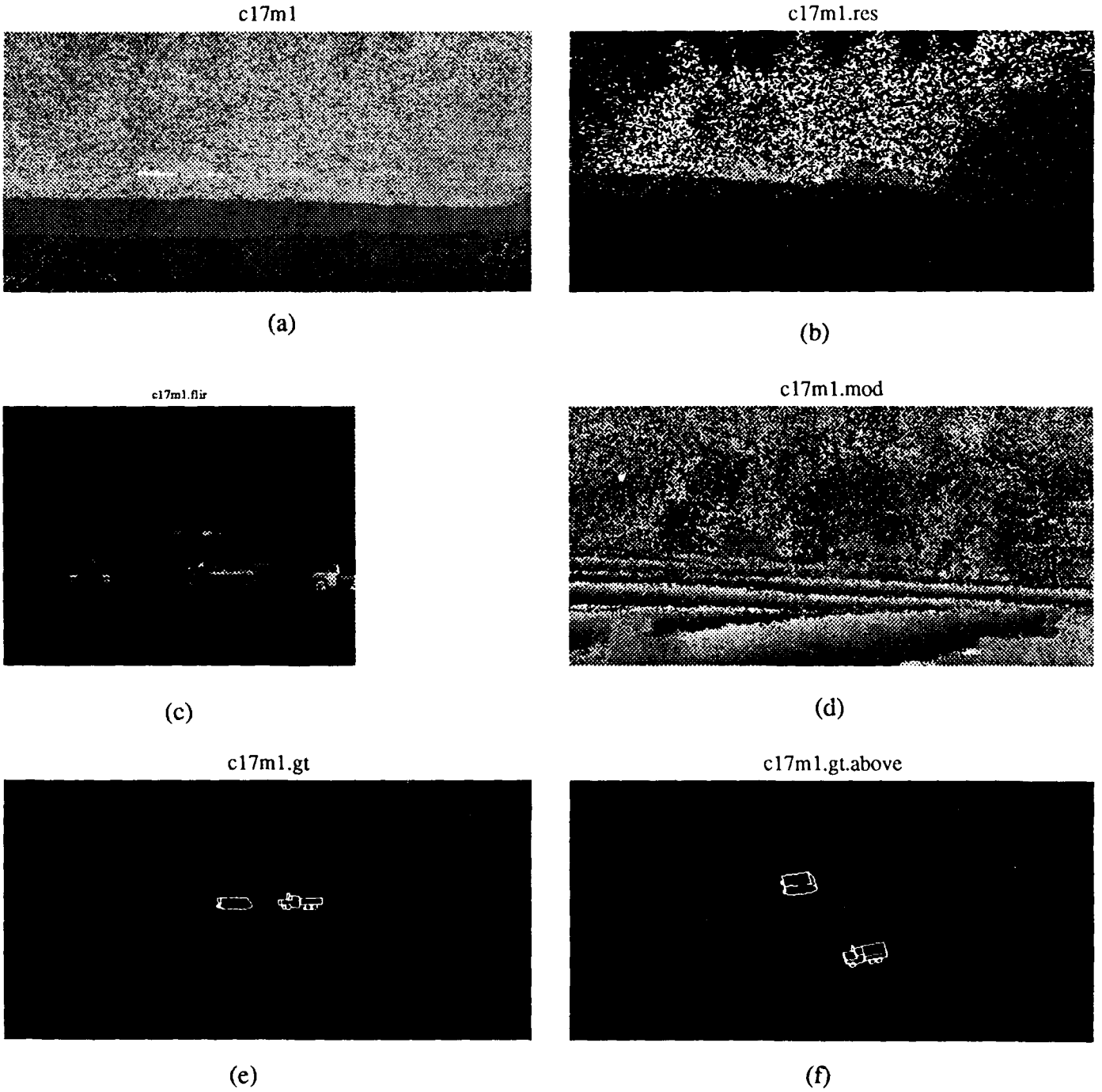


Figure 3.1.10 Image *c17m1* a) Original 32 bit per pixel image linerly rescaled to 8 bits per pixel, b) Range slice with black being everything in front of 1625m and white being everything behind 1696.4 meters, c) FLIR image of same targets, d) original image mod 1875. e) Model of ground truth as seen by the sensor. f) Model of ground truth rotated by 45° to see the relative spacing and orientation of the targets.

In the third type of LADAR data, the LADAR *return intensity*, each pixel is linearly proportional to the strength of the return LADAR signal. The larger the pixel value, the stronger the return signal. Such information might be useful as a confidence value associated with each returned range pixel value. If the return signal is strong, a high confidence value can be associated with the corresponding resolved range value. Unfortunately the intensity data that was collected is not usable since there were some problems during the field test which resulted in not all of the data being linear [Phil87]. This data was therefore not used.

The LADAR sensor is capable of collecting images using different resolutions and different frame sizes. All the experiments presented in this section used data with a resolution of 0.05 mrad in both directions and a frame size of 511 by 256 (horizontal by vertical). (Called type *7C* in [NeSm87].)

3.1.2.2. The Test Images

The images in Figures 3.1.7-3.1.10 all have the same arrangement with (a) being the 32 bit LADAR image and (c) being the 8 bit FLIR image of the same scene. The following paragraphs present the general approach used to produce the other images. Table 3.1.2 is a summary of the LADAR and FLIR image header information. Table 3.1.3 gives specific comments on each of the test images.

The upper left image (a) in Figures 3.1.7-7.3.1.10 is the 32 bit per pixel LADAR range image. Because of the large dynamic range, it was linearly rescaled to display as 8 bits per pixel. Rescaling tends to cause the targets to blend into the background and are often not visible in the image. The upper right image (b) is a LADAR range slice of the same image. The range slice was found by apply the following formula to each of the pixels:

$pixel_{slice}$ = (*pixel* - offset) / scale
if $pixel_{slice} < 0$
    then $pixel_{slice} = 0$
    else if $pixel_{slice} > 255$
        then $pixel_{slice} = 255$

The resulting image has all pixels in front of *offset* set to black and all those behind the slice (those greater than *offset* + *scale* * 255) set to white. Figures 3.1.6-3.1.10 list the front and the back of the range slice in meters. A *scale* of at least 7 should be used since the range data is given in centimeters, but the sensor can only resolve to 7.2 centimeters. Since the lower 2 bits of the AM relative range sensor might be noisy, at scale of 28 was used when displaying images. **Note: all processing was done on the 32 bit data. The rescaling was only used for display purposes**

The left image on the second row (c) is a FLIR image taken of the same configuration of targets using a narrow field of view. The right image on the second row (d) is computed from the LADAR image on a pixel by pixel basis using the following formula:

Table 3.1.2 Summary of LADAR and FLIR header files for Figures 3.1.7-3.1.10.

	c14ta3	**c10m1**	**c17m3**	**c17m1**
FILE NAME	LD60977C14TA3	LD61577C10M1	LD61677C17M3	LD61777C17M1
TARGET TYPE	M60A1 TANK	M551 TANK, CJ5 JEEP, M113 APC	M113 APC, 2.5 TON TRUCK CJ5 JEEP	M113 APC, 2.5 TON TRUCK
SITE	1400 M	1020 M	1700 M	1700 M
RANGE	1427 M	n/a	1693	1693
DATE	9-JUN-1987	15-JUN-1987	16-JUN-1987	17-JUN-1987
WEATHER	HAZY/ OVERCAST	HOT/HAZY/ HUMID	HAZY	CLEAR
ladar				
TIME	15:20:44.53	14:43:57.42	14:56:10.89	09:59:03.50
FOV	0.8 X 1.6 DEG 7C	0.8 X 1.6 DEG 7C	0.8 X 1.6 DEG 7C	0.8 X 1.6 DEG 7C
DATA SHIFT	-18	-18	-18	-18
MIRROR REFLECTION	NO	NO	NO	NO
MODE	IMAGE	IMAGE	IMAGE	IMAGE
RESOLVED RANGE	CENTIMETERS	CENTIMETERS	CENTIMETERS	CENTIMETERS
flir				
TIME	1520.39.424	1454.43.928	1456.00.272	0958.53.059
FOV	N	N	N	N
BRIGHTNESS	240	200	190	190
CONTRAST	120	40	70	80

Table 3.1.3 Comments about the test images selected from the 1987 A.P. Hill field test.

Figure	Comment
3.1.7 a,b	This is a single M60A1 at 1400 meters. As with most LADAR, the sky is easily found because it appears a random return values on the upper part of the image.
3.1.7 d	The target is on a wrap around line since the dark right side must be farther than the white left side. This is one of the problems with the relative range AM data. However this does show that large geometric features (such as the various planer surfaces of a target) will appear in LADAR data even at 1400 meters.
3.1.7 e,f	This field test involved taking images with the turret at different rotations. The ground truth data is unclear as to the location of the reference to which the rotation angles were measured. Since our model currently does not rotate the turret apart from the rest of the tank, the ground truth image might be inaccurate.
3.1.8 a,b	There are three targets (M60A1, Jeep, M113) at about 1000 meters.
3.1.8 d	The mod 1875 image makes the targets more visible than the range slice image.
3.1.8 e,f	Although the locations of the targets match the LADAR Log ground truth, the relative positions of the three targets appear wrong in the ground truth image. The FLIR image (c) clearly shows that all three targets are unoccluded. It is possible the angle to the sensor was miss-measured.
3.1.9 a-d	There are three targets (M113, Jeep, 2.5 ton truck) taken at 1700 meters on a humid hazy day. The targets are almost undetectable accept of in the FLIR image (c) in which they are clearly visible.
3.1.10 a-d	There are two targets (M113, 2.5 ton truck) at 1700 meters. The weather was clear and the targets are more visible than the targets in Figure 3.1.9.

$pixel_{AM} = pixel \bmod 1875$

Since the resolved range image is in centimeters and the AM ambiguity interval is 1875 cm, the AM portion of the signal can be found by taking the *mod 1875* of each pixel. That is why these images look similar to the images in Section 3.1.1. Close inspection of the *mod 1875* images reveals that there is less noise on target than shown in the range slice image. We believe this is caused by the noise present in both the *AM* and *FM* components of the return signal. The *FM* signal is used to decide which absolute range bin to which the relative *AM* signal corresponds. The resolved range image, which is a combination of both the *AM* and *FM* signals, therefore has the noise of both signals. When the *mod 1875* image is taken, the effects of the *FM* signal (and its noise) are removed which results in a cleaner image. The *FM* noise has been measured to be as much as 9 meters at a range of 1400 meters. [Phil87].

The images on the bottom row (e and f) were constructed using the ground truth data from the Laser Radar Log [NeSm87]. This data gives the type and position of each target relative to a reference marker as show in Figure 3.1.11. The data for image *c10m1* is:

SV = 132 degrees (position of sensor)
Right target
    APC
        RR - 52 feet (52 feet to right rear)
        CT = 5 degrees
        ST = 55 degrees
Center target
    Jeep
        LF - 58 feet (58 feet to left front)
        CT = 335 degrees*
        ST = 55 degrees
Left target
    Tank
        RF - 1 foot (1 foot to right front)
        CT = 355 degrees*
        ST = 150 degrees

* The LADAR LOG reports the Jeep CT to be 355 ° and the Tank CT to be 335 °, however these figures were corrected as shown during a phone conversation with Bob Dockery from the Center for Night Vision.

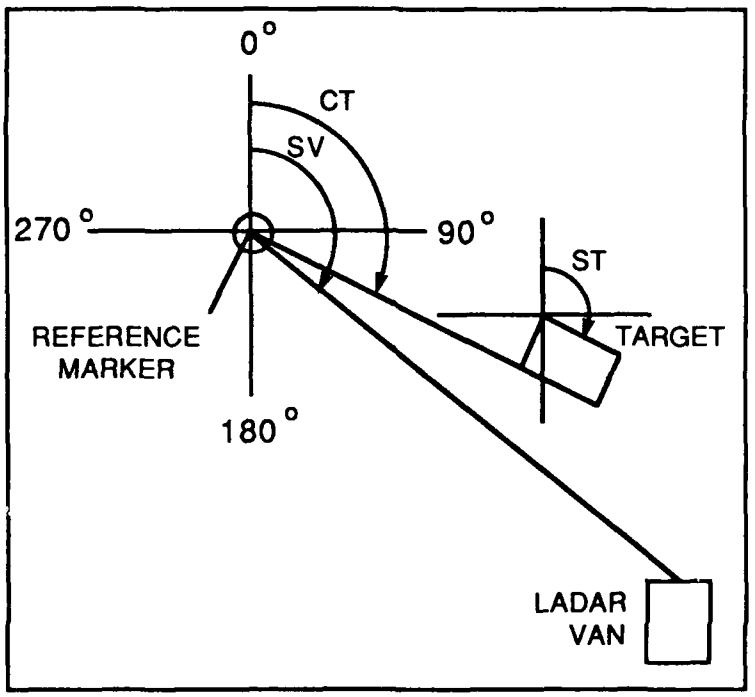


Figure 3.1.11 Relative placement of targets as given in the LASER RADAR LOG [NeSm87].

The following function, written in LISP, converts this data into PADL statements which positions the targets.

```
(defun group (van stream &rest targets)
                                ; van - angle from reference to LADAR van
                                ; stream - place to write output data
                                ; targets - list of targets in scene
  (let  ((count 0))             ; number of targets
    (mapc #'(lambda (target)    ; convert each target in the list of target
              (format stream    ; write the following to stream.
                      ;; The following is code written for PADL.
                      "target~a = ~a_~a at (degz=~a, movx=~a, degz=~a)~%"
                      count
                      (type target)
                      (corner target)
                      ;; The rotation about the center must be
                      ;; corrected because the ct below will turn the
                      ;; target too.
                      (- (ct target) (st target)) ; Rotation
                                                  ; about center
                                                  ; of target
                      (* (offset target) 0.3048) ; Convert feet to meters
                      (- (ct target)))
              (setq count (1+ count)))
          targets)
    (format stream "all = (target0")
    (dotimes (i (1- count))
      (format stream " un target~a" (1+ i)))
    (format stream ") at degz=~a~%" van)))

(defmacro type (x)          ; Type of target
  '(car ,x))
(defmacro corner (x)        ; Corner to measure to.  (one of lf, rf, lr, rr)
  '(nth 1 ,x))
(defmacro offset (x)        ; Distance from corner to reference marker.
  '(nth 2 ,x))
(defmacro ct (x)            ; Angle from reference to corner of target
  '(nth 3 ,x))
(defmacro st (x)            ; Rotation about the target's axis.
  '(nth 4 ,x))
```

This input:

```
;;; This is a padl description of the targets for TAPE DF1572
;;; Time: 1430

(group 132 t
     '(m113  rr 52   5  55)
     '(m151  lf 58 335 145)
     '(m60a1 rf  1 355 150))
```

produces the following PADL code for image *c10m1*.

```
target0 = m113_rr at (degz=-50, movx=15.8496, degz=-5)
target1 = m151_lf at (degz=190, movx=17.6784, degz=-335)
target2 = m60a1_rf at (degz=205, movx=0.3048, degz=-355)
all = (target0 un target1 un target2) at degz=132
```

The first line of input to the *group* routine states that the van with the LADAR sensor is a 132 degrees from the reference. The second line of input to the *group* routine places the *r*ight *r*ear corner of the *M113* APC at 52 feet and 5 degrees from the reference. The APC is rotated about its axis 55 degrees. This translates into PADL code by arbitrarily identifying the APC as *target0*. *target0* places *m113_rr* (the right rear of a M113) by rotating it −50 degrees about its axis. (It is a negative amount since PADL defines rotations opposite from that used in the LADAR Log. It is only 50 degrees not 55 since the second rotation will take it the needed extra 5 degrees.) Next the APC is moved 15.8496 meters (52 feet) from the reference and then rotated 5 degrees about the reference. The same is done for the M151 and the M60A1.

This data was used by the Electronic Terrain Board Model discussed in Section 4 to produce the ground truth images. The left image is of the scene as viewed by the sensor, and the right image is the same view rotated 45° to give a better view of the relative spacing and orientation of the targets. Figure 3.1.12 is the targets in image *c10m1* as viewed from above. The marking show that the targets are placed as described in the LADAR LOG, however when viewed from the sensor position, the targets do not appear to have the same relative positions. We are unable to explain the differences at this time. One possibility is that our selection of wire frame models is limited so the models used in the ground truth images most likely do not match exactly the actual targets used in the field test. If, for example, our M60A1 model was wider than the actual target it would occlude the Jeep as shown.

3.1.2.3. Preprocessing the Data

Close inspection of the targets in the range slices of the resolved range data shows that the targets, though distinctive in three of the four test images,* all contain considerable impulse

* Apparently image *c17m3* was taken on a very hazy day and the targets are not as visible in it.

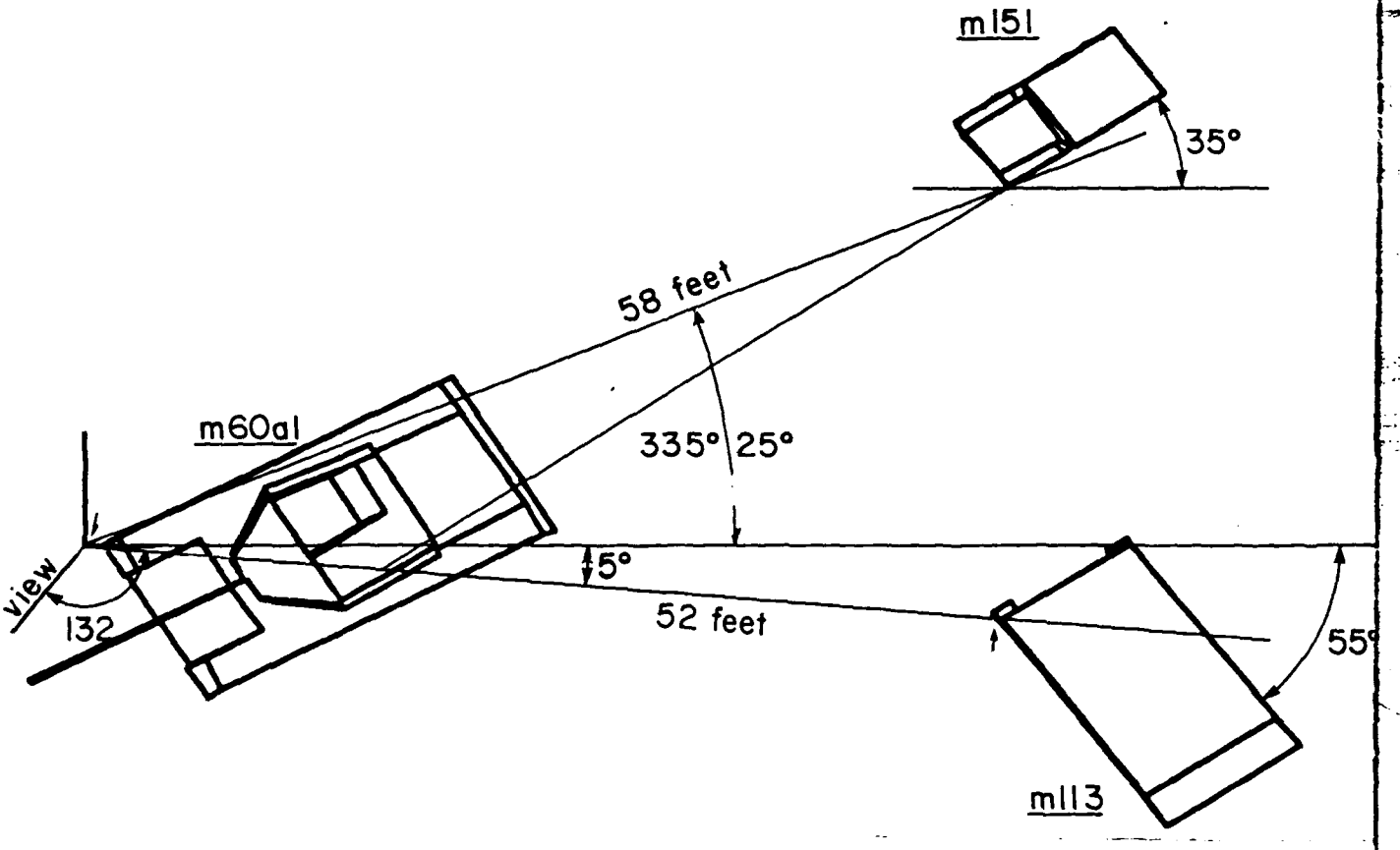


Figure 3.1.12 Targets in image *c10m1* as viewed from above.

noise. Images (c) and (d) in Figures 3.1.13-3.1.16 show that using a 3 by 3 and a 5 by 5 median filter on the images removes much of the noise. To better see the affects of the filtering operation, the tank and the APC from image *c10m1* and the APC and the 2.5 ton truck from image *c17m1* enlarged in Figures 3.1.17-3.1.20 Some observations are in order. Note the jagged left edge of the tank in Figure 3.1.17(b) and the jagged right edge of the APC in Figure 3.1.18(b). The jaggedness is caused by the sensor scanning alternate lines in different directions. (i.e. If the even lines are scanned from left to right, then the odd lines will be scanned from right to left.) Unfortunately near the edges of the image the alternate lines do not line up exactly as shown in these images. This lack of alignment is obscured in the resolved range image, but made clear in the *mod 1875* image. Apparently this problem didn't occur in the *c17m1* images.

Median filtering removes much of the impulse noise as shown in Figures 3.1.21-3.1.24 unfortunately it may also be removing much of the fine structure information on the target. The great dynamic range of 32 bit data presents some problems not present in 8 bit data. In this data a noise spike can cause a pixel value to appear a kilometer or two away from the target. Such noise must be removed, but at the same time the fine "10's of centimeters" resolution must be preserved since it might contain structural information about the target.

The *mod 1875* images provide the first clue as to how the process the 32 bit data. The targets are often more visible in these images that in the original resolved range images. The impulse noise on target appears to be much lower which means the FM data must have contained noise which caused the pixel value to be put in the wrong range bin. Our *median based range bin corrector* (MBRBC) attempts to fix this problem by looking the the median value of all the pixels in a window around a given pixel. If the given pixel is more than one ambiguity interval (1875 cm) away from the median, an integer number of ambiguity values will be added (or subtracted) from the pixel value until it is within an ambiguity interval of the median. If the image has no range bins errors, the pixel value will be within an ambiguity interval to the median and with therefore not be changed. If the pixel is misplaced by a range bin error it will be placed back into the correct bin (assuming the median of the neighbors is within the correct bin). Images (d) and (f) in Figures 3.1.13-3.1.16 show the test images after using the MBRBC filter. These images show that there is no great improvement in the overall appearance, however, Figures 3.1.25-3.1.28 show the close ups of the four targets used before and their MBRBC images. Here it is clear that the standard median filter tends to remove what might be structural details and smooth out the image. The MBRBC images have the same noise spikes removed, but still have the information that might allow fine structural details to be extracted.

3.1.2.4. Comments on the Quality of the Data

The LADAR sensor requires careful tuning before it will produce good images. The quality of the data taken during the 1987 field test appears to vary greatly from one set of images to another. An example is images *c17m1* and *c17m3* which were taken at the same distance and field of view, but the targets in *c17m3* are much more visible that those in *c17m1* . This could

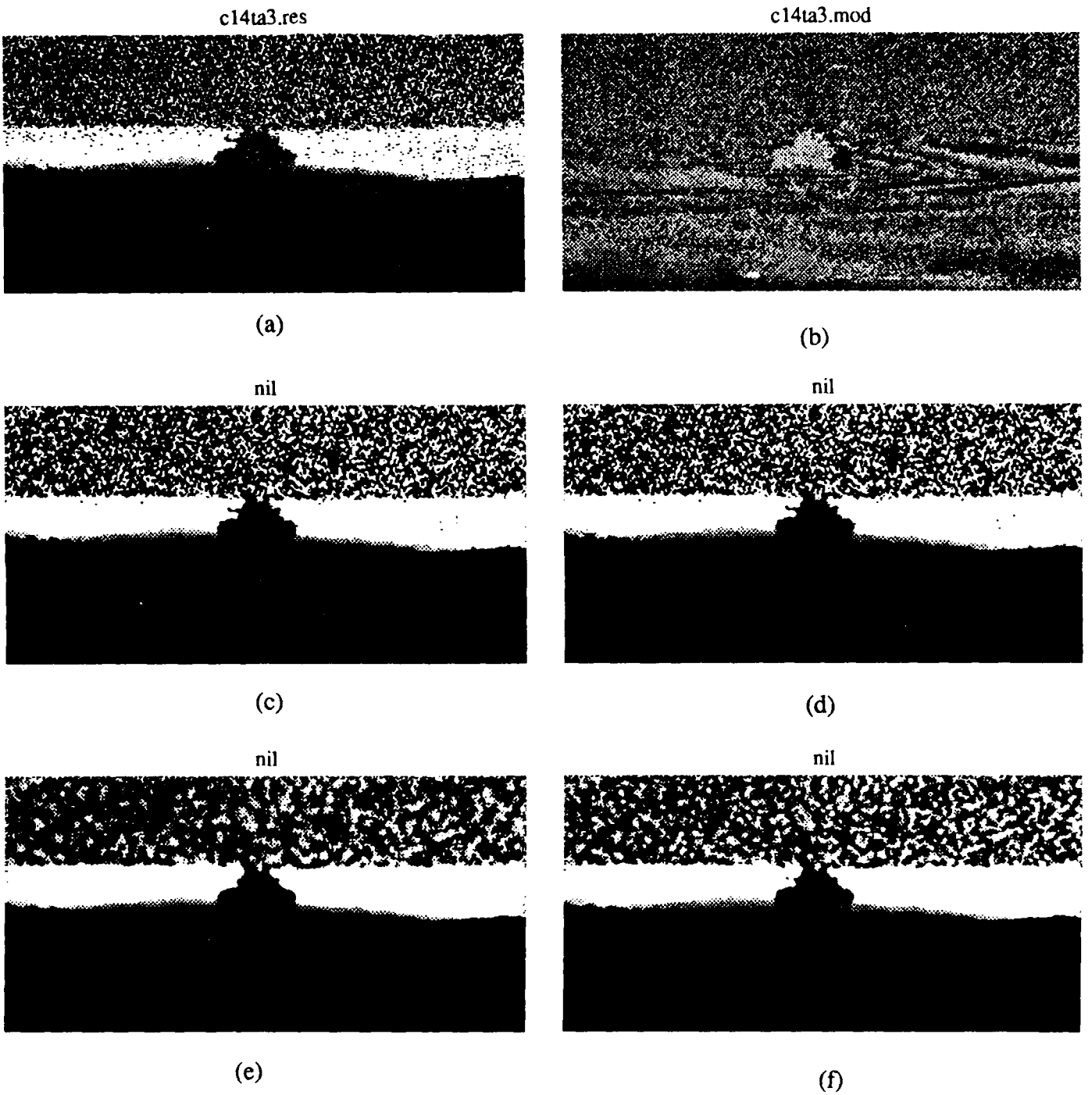


Figure 3.1.13 Image *c14ta3* a) Range slice as in Figure 3.1.7, b) mod 1875, c) Median using 3 by 3 window, d) Smart median using a 3 by 3 window, e) Median using a 5 by 5 window, f) Smart median using a 5 by 5 window.

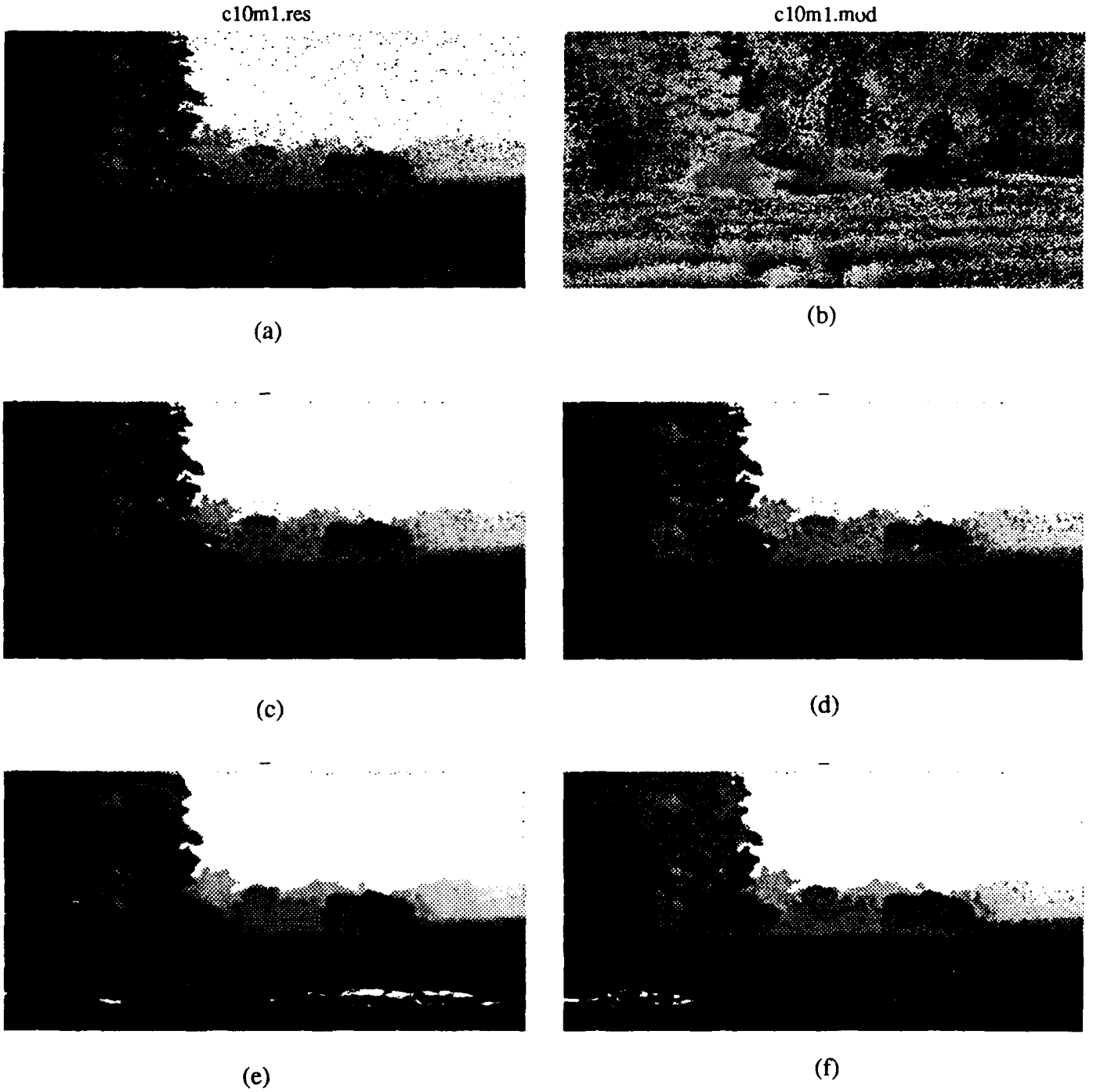


Figure 3.1.14 Image *c10m1* a) Range slice as in Figure 3.1.8, b) mod 1875, c) Median using 3 by 3 window, d) Smart median using a 3 by 3 window, e) Median using a 5 by 5 window, f) Smart median using a 5 by 5 window.

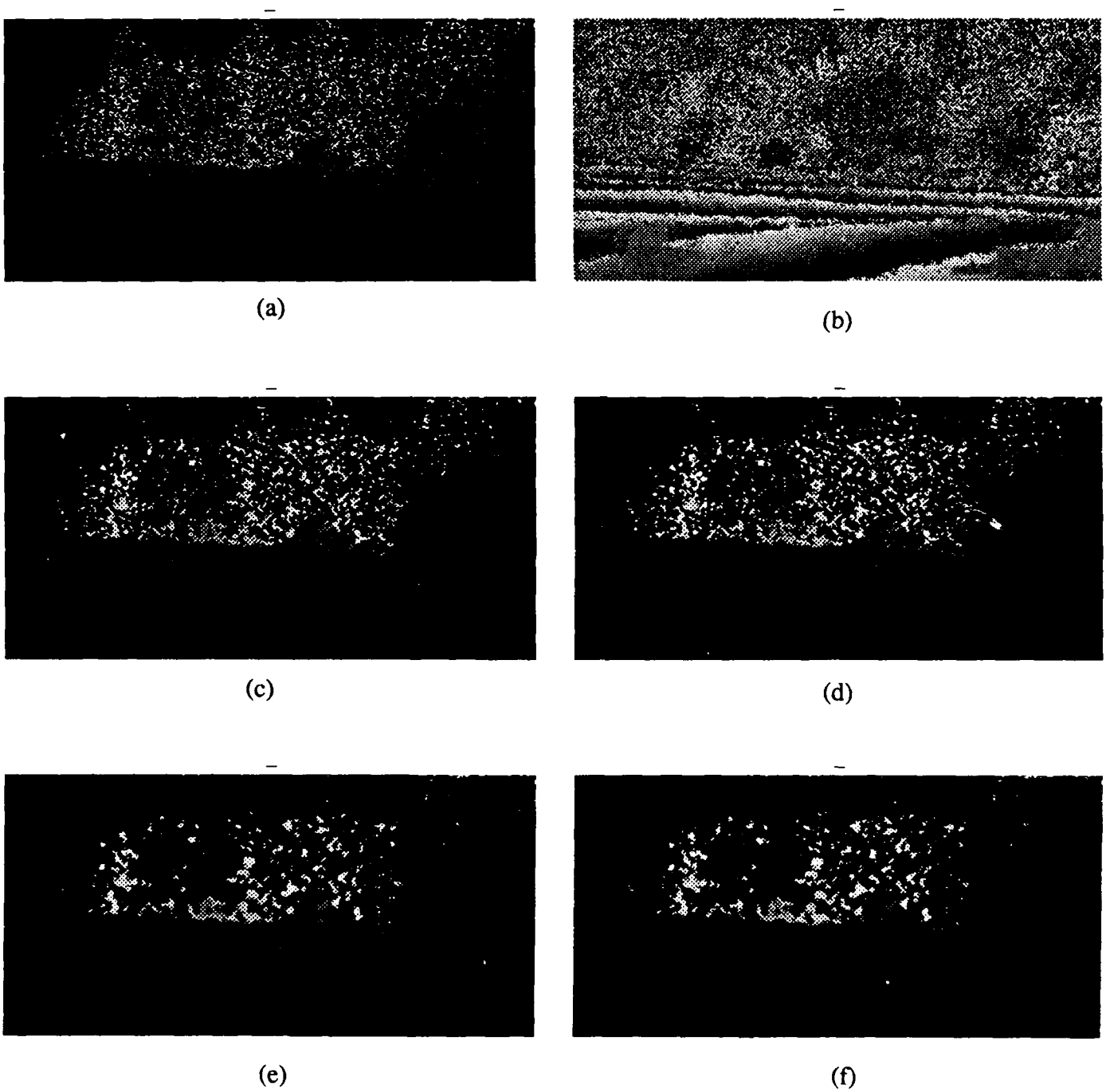


Figure 3.1.15 Image *c17m3* a) Range slice as in Figure 3.1.9, b) mod 1875, c) Median using 3 by 3 window, d) Smart median using a 3 by 3 window, e) Median using a 5 by 5 window, f) Smart median using a 5 by 5 window.

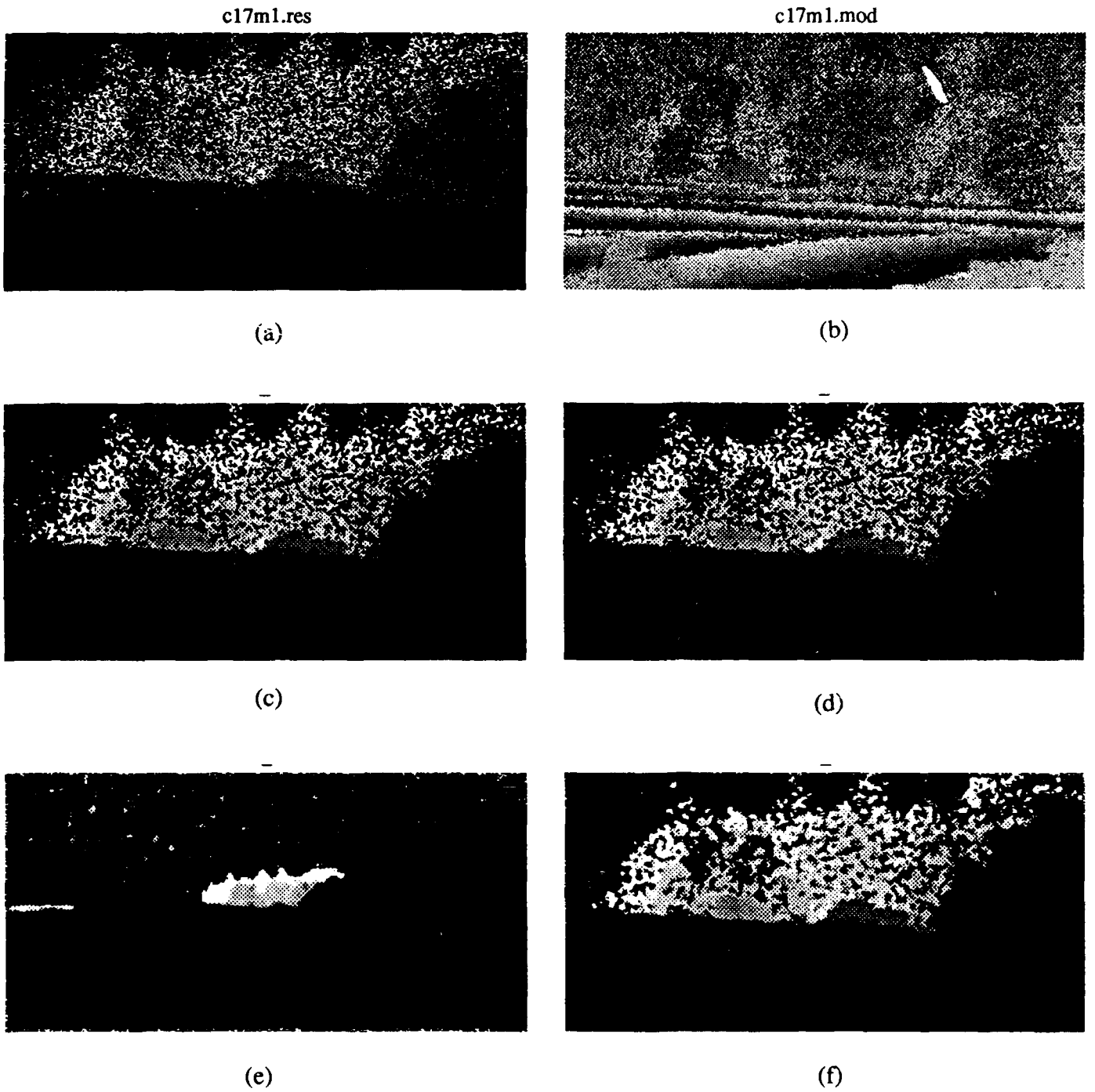
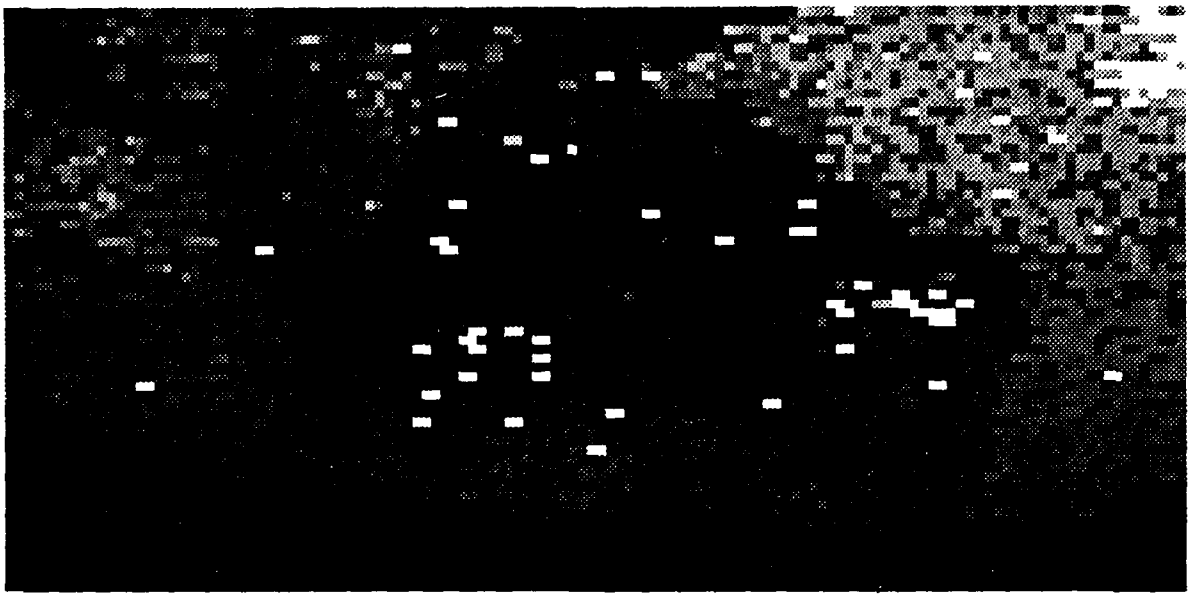
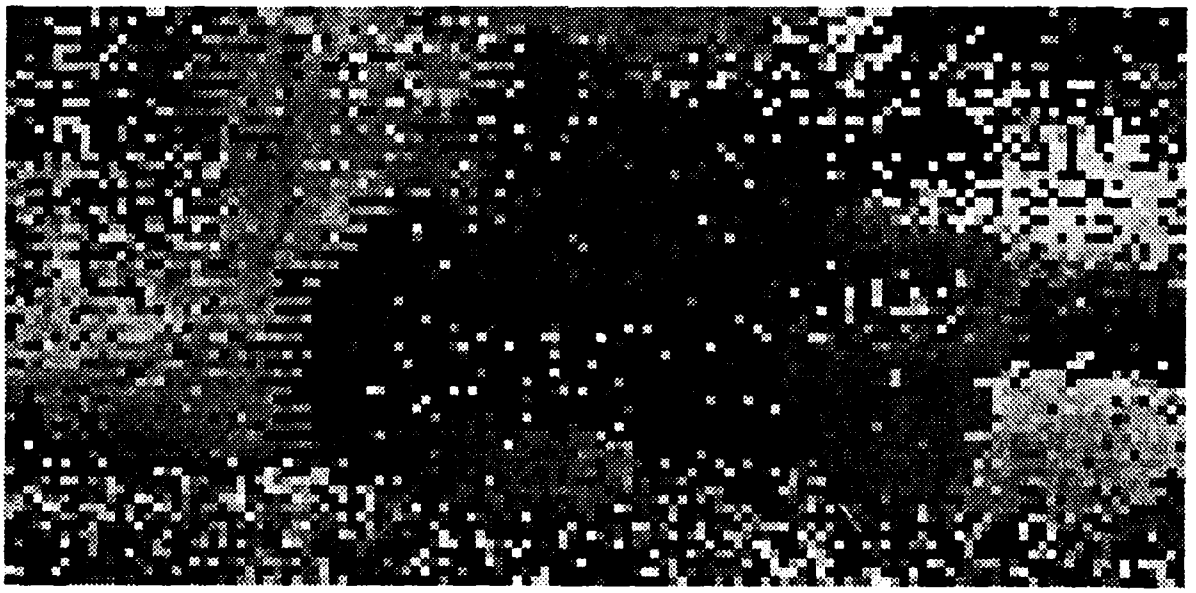


Figure 3.1.16 Image *c17m1* a) Range slice as in Figure 3.1.10, b) mod 1875, c) Median using 3 by 3 window, d) Smart median using a 3 by 3 window, e) Median using a 5 by 5 window, f) Smart median using a 5 by 5 window.



(a)

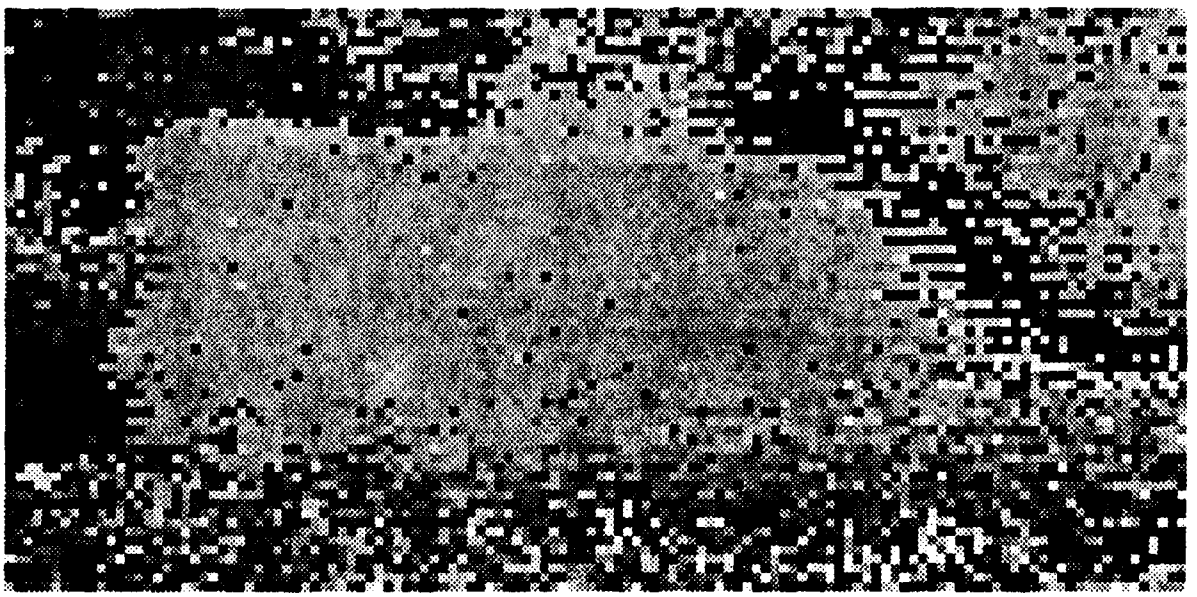


(b)

Figure 3.1.17 Tank from image *c10m1*. The upper left corner of this image is pixel 100, 110 from image *c10m1* a) Rescaled image, b) mod 1875 image.

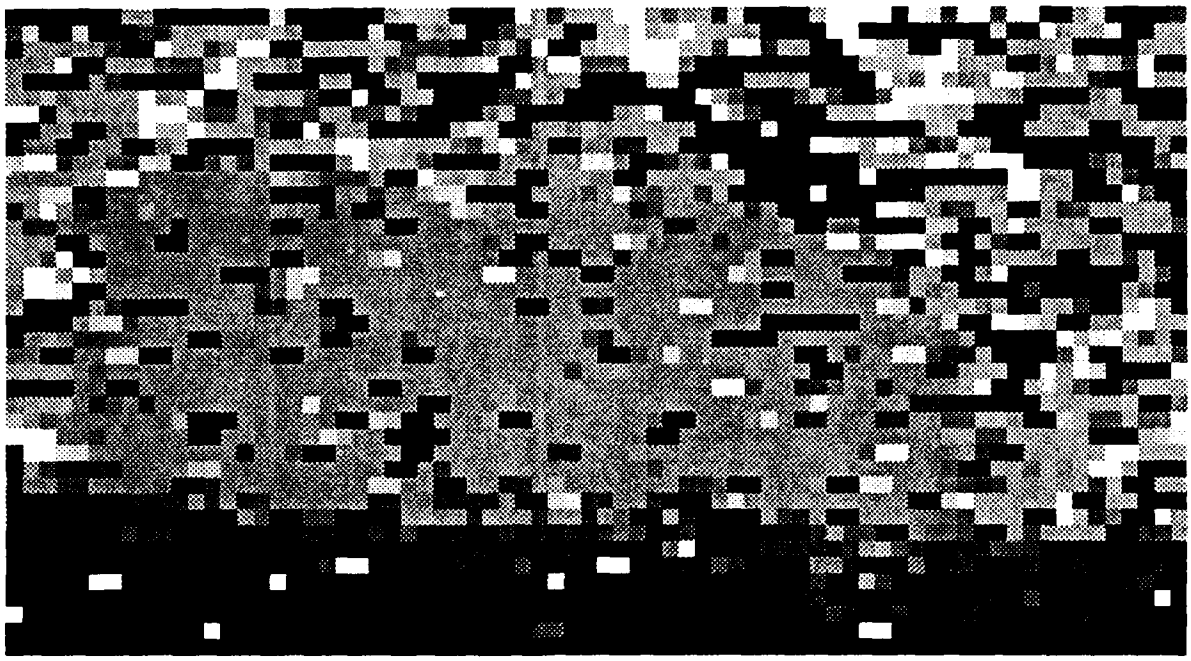


(a)

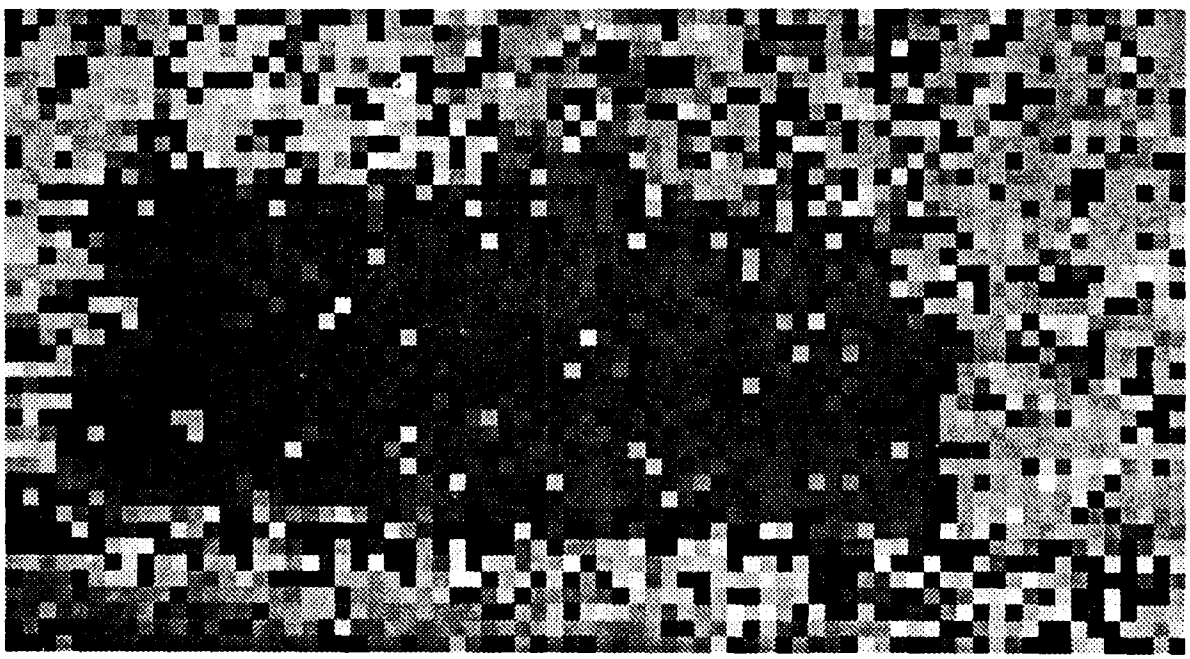


(b)

Figure 3.1.18 APC from image *c10m1*. The upper left corner of this image is pixel 300, 110 from image *c10m1*. a) Rescaled image, b) mod 1875 image.

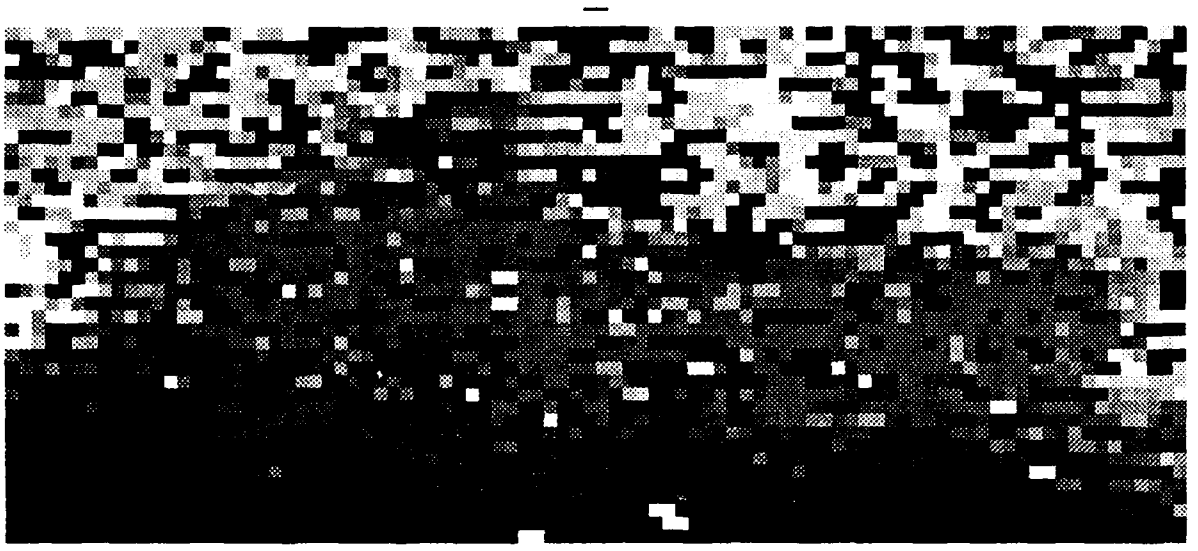


(a)

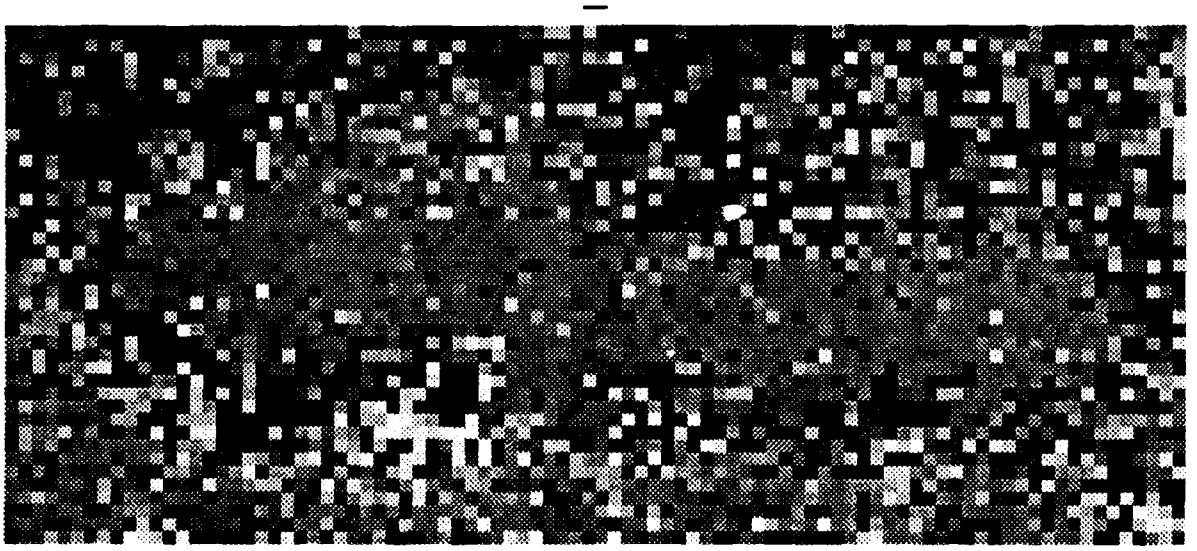


(b)

Figure 3.1.19 APC from image *c17m1*. The upper left corner of this image is pixel 144,115 from image *c17m1*. a) Rescaled image, b) mod 1875 image.



(a)



(b)

Figure 3.1.20 2.5 ton truck from image *c17m1*. The upper left corner of this image is pixel 251, 120 from image *c17m1*. a) Rescaled image, b) mod 1875 image.

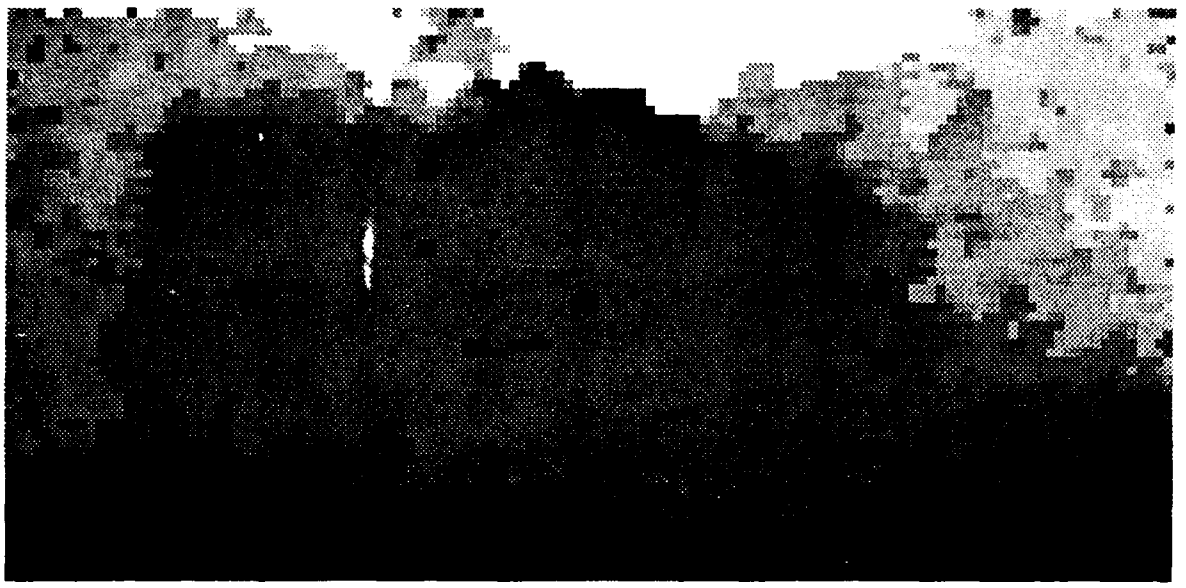


(a)

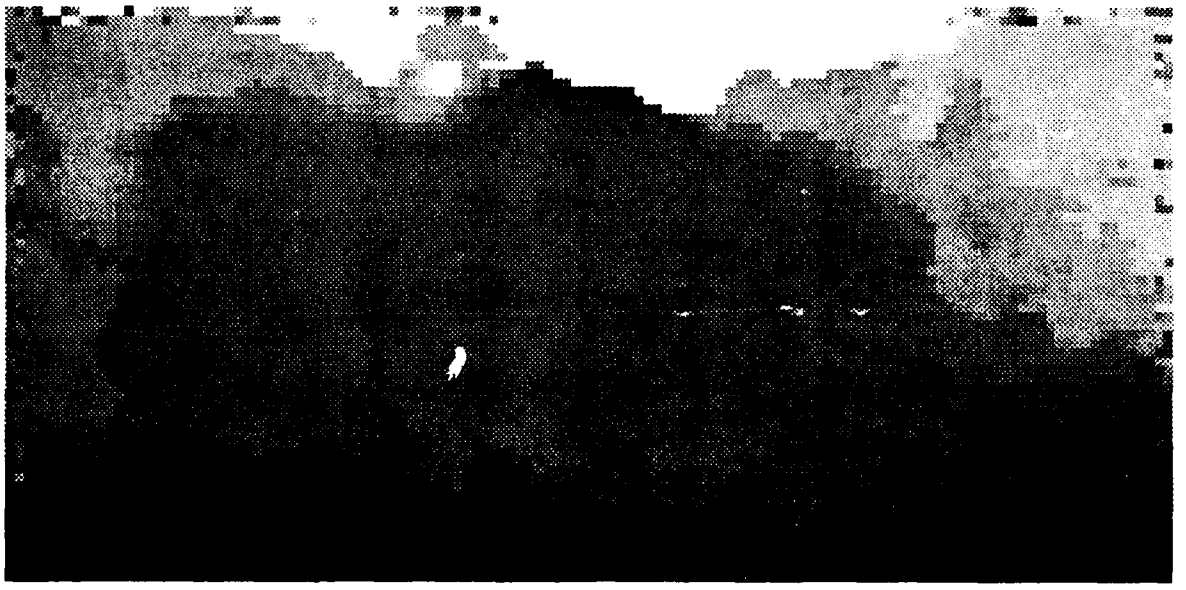


(b)

Figure 3.1.21 Median filtered tank from image *c10m1*. The upper left corner of this image is pixel 100, 110 from image *c10m1* a) 3 by 3 window, b) 5 by 5 window.

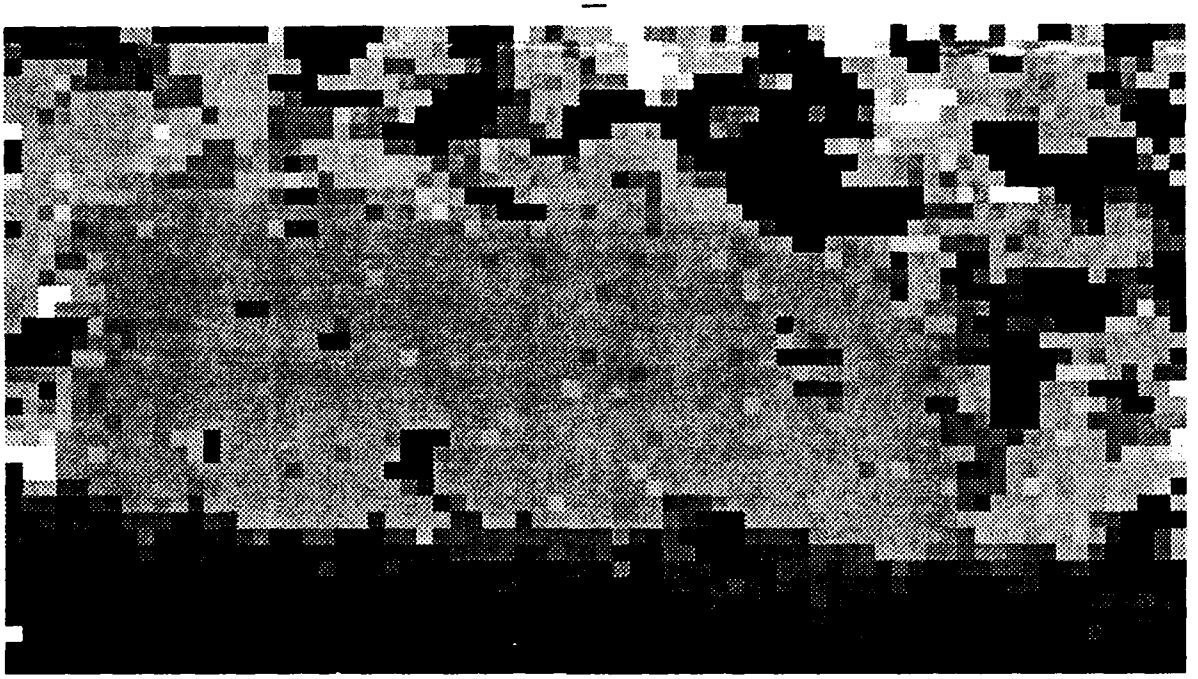


(a)

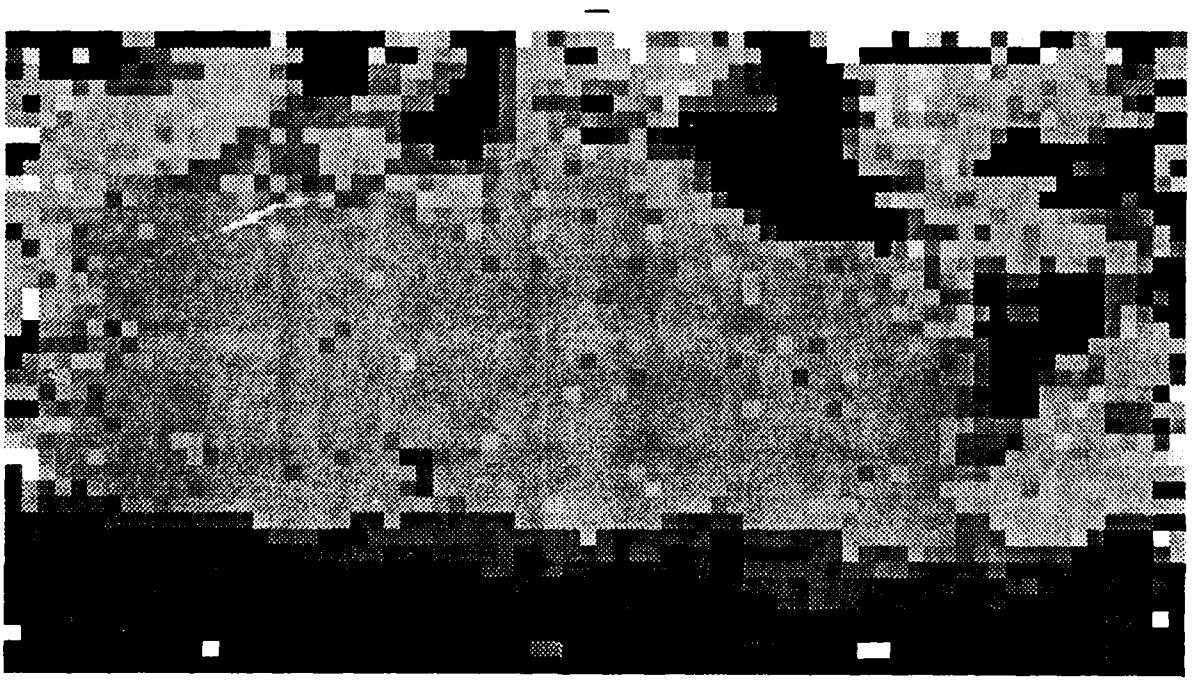


(b)

Figure 3.1.22 Median filtered APC from image *c10m1*. The upper left corner of this image is pixel 300, 110 from image *c10m1*. a) 3 by 3 window, b) 5 by 5 window.

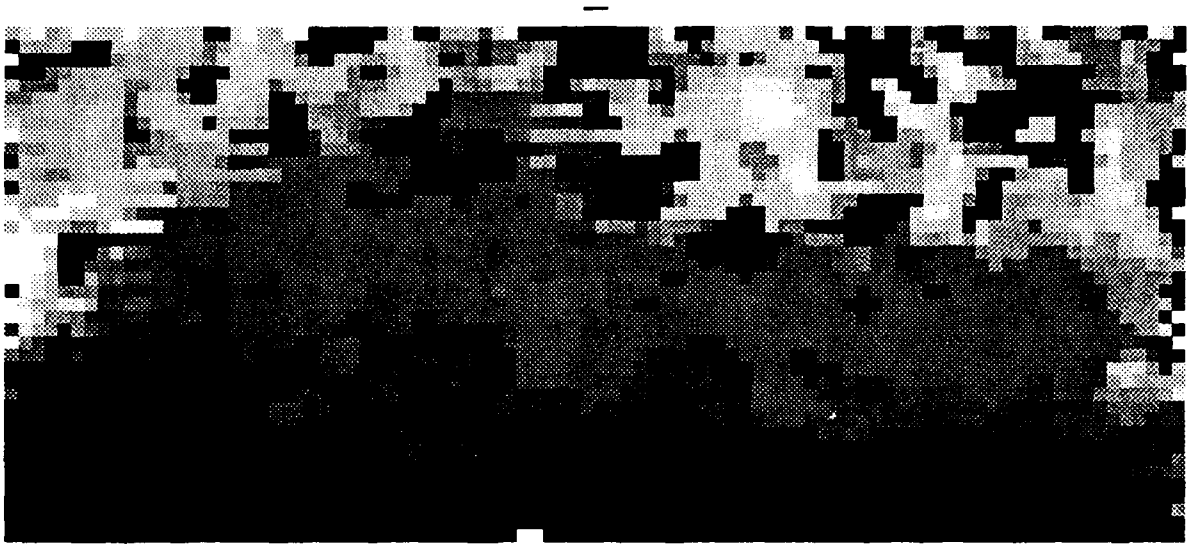


(a)

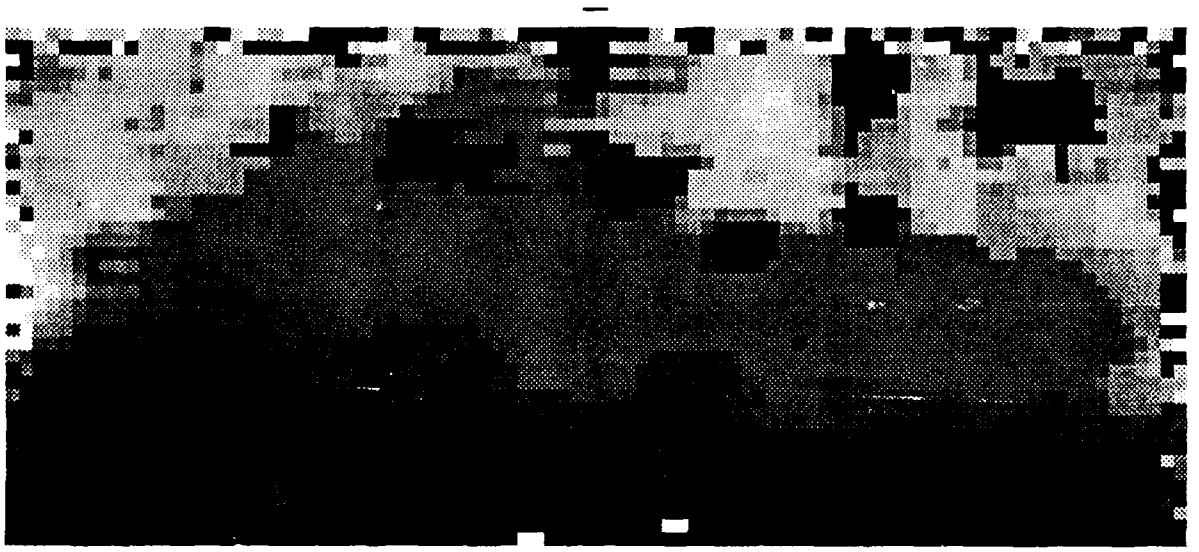


(b)

Figure 3.1.23 Median filtered APC from image *c17m1*. The upper left corner of this image is pixel 144,115 from image *c17m1*. a) 3 by 3 window, b) 5 by 5 window.



(a)



(b)

Figure 3.1.24 Median filtered 2.5 ton truck from image *c17ml*. The upper left corner of this image is pixel 251, 120 from image *c17ml*. a) 3 by 3 window, b) 5 by 5 window.

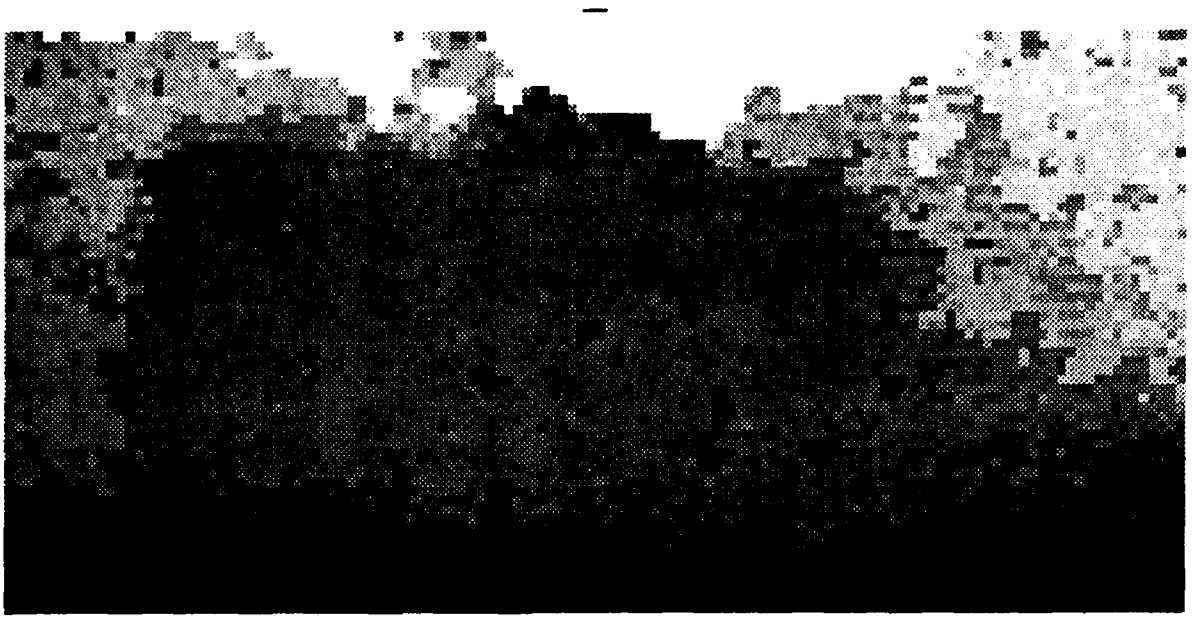


(a)



(b)

Figure 3.1.25 MBRBC filtered tank from image *c10m1* . The upper left corner of this image is pixel 100, 110 from image *c10m1* a) 3 by 3 window, b) 5 by 5 window.

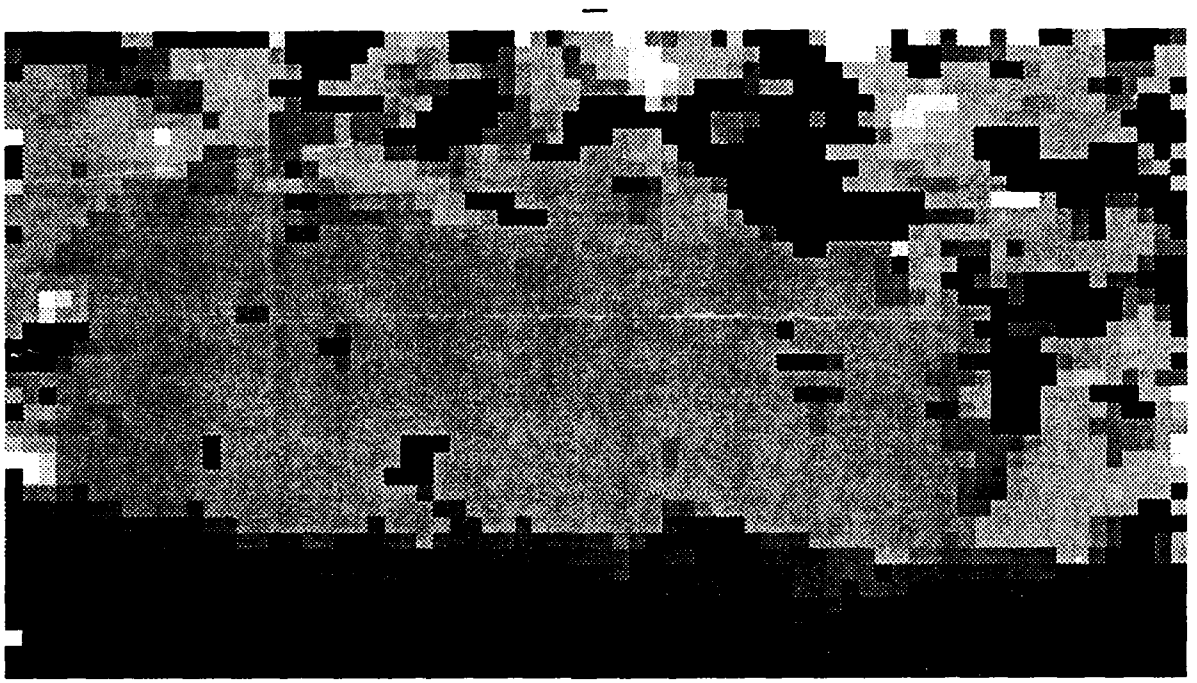


(a)

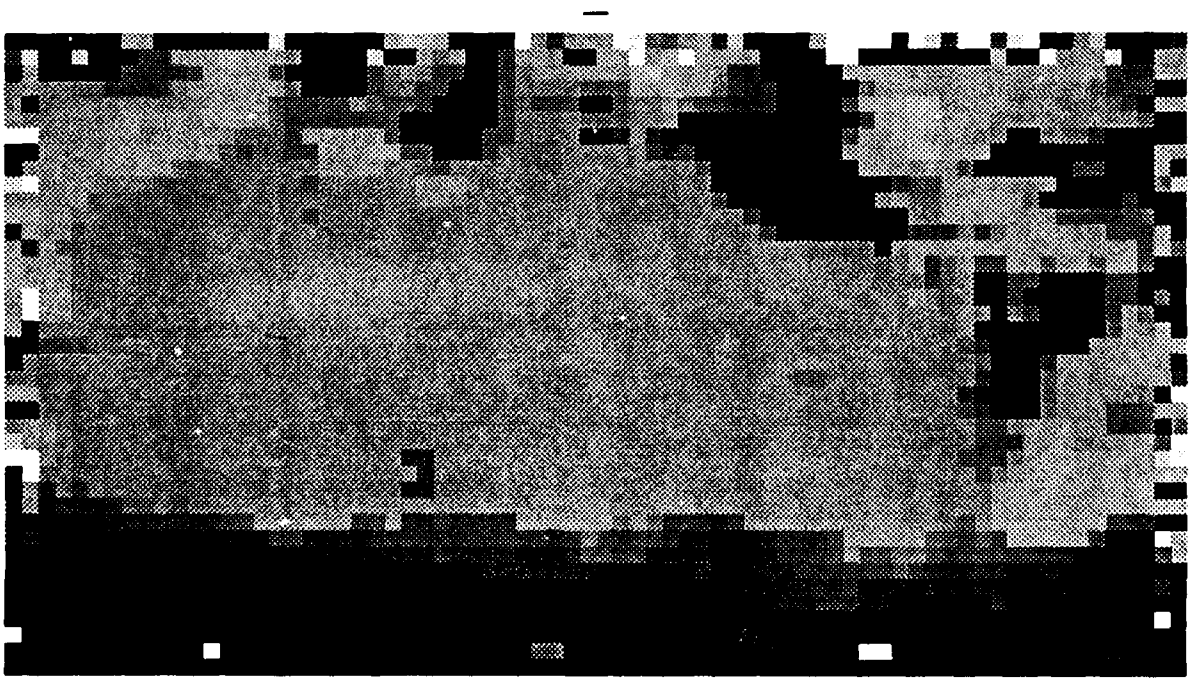


(b)

Figure 3.1.26 MBRBC filtered APC from image *c10m1*. The upper left corner of this image is pixel 300, 110 from image *c10m1*. a) 3 by 3 window, b) 5 by 5 window.

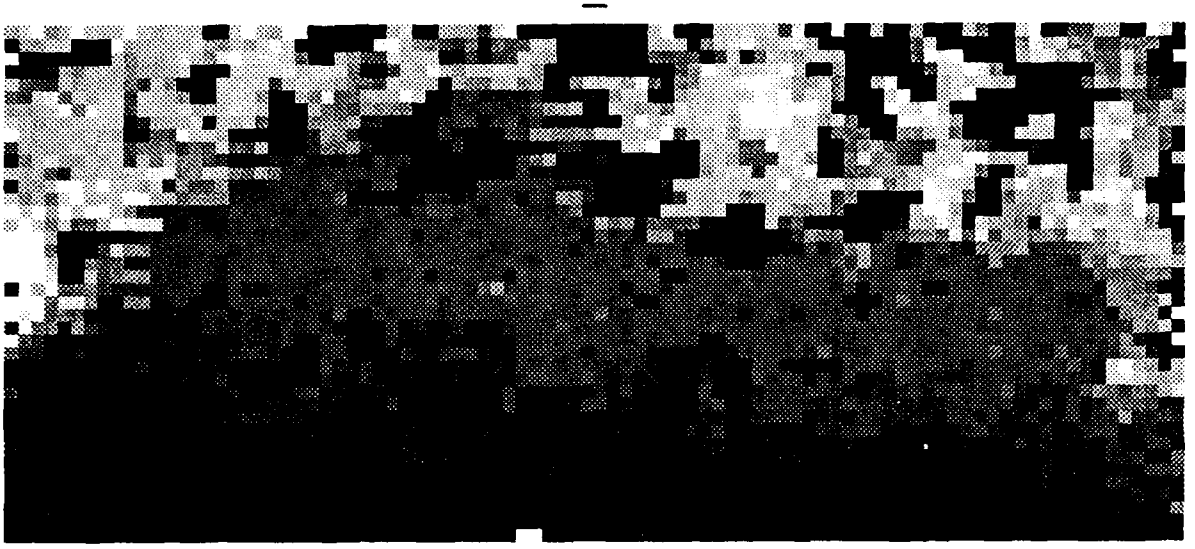


(a)

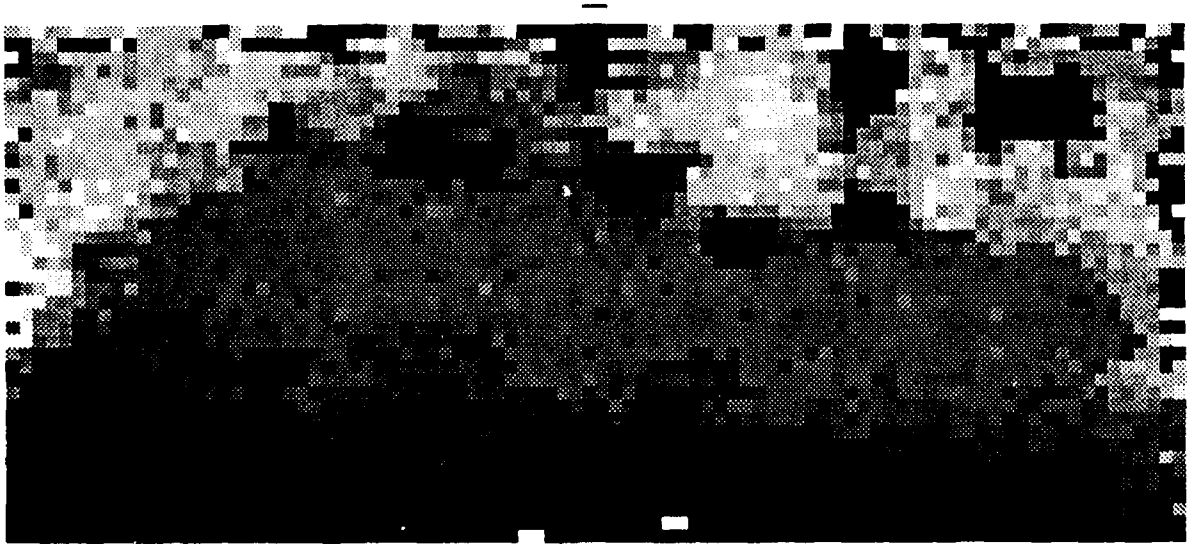


(b)

Figure 3.1.27 MBRBC filtered APC from image *c17m1*. The upper left corner of this image is pixel 144,115 from image *c17m1*. a) 3 by 3 window, b) 5 by 5 window.



(a)



(b)

Figure 3.1.28 MBRBC filtered 2.5 ton truck from image *c17m1*. The upper left corner of this image is pixel 251, 120 from image *c17m1*. a) 3 by 3 window, b) 5 by 5 window.

be caused by the weather being clear the day *c17m1* was taken and hazy the day *c17m3* was taken, or it could be caused by improper tuning. Such variation in results, whether caused by tuning or the weather, make it difficult to estimate the maximum distance such a sensor can be used.

Another problem with the data is the misalignment between the even and odd scan lines. A simple solution is to only use every other scan line, but such an approach then throws out half of the data.

3.1.2.5. Future Work

Since this is the first report in which the 1987 A. P. Hill data was examined, there are many areas that need more work. The following paragraphs details on the following ideas:

1. Create ground truth images using the Electronic Terrain Board Model.
2. Devise a method to measure the effectiveness of the processing filters.

The Electronic Terrain Board Model need to be improved so that it can produce images, based in the ground truth information given for each image, which are accurate enough to be considered *ground truth images*. This model needs to include both the targets and the terrain We expect these changes to be brought about by switching to the BRL-CAD modeler which will allow the very detailed BRL models to be used.

We need to develop a method of measuring the effectiveness of the median and MBRBC filters. Visually speaking, the median filter appears to remove some fine structure information, however we have no way of measuring how much information is lost. One good method would be to

1) use the ground truth data to know location and orientation of the planer regions,
2) filter the LADAR data with the preprocessor to be tested,
3) extract the geometric features as presented in Section 3.3.1,
4) compare the extracted features to those in the ground truth data.

Such a procedure should give a very accurate measure to a filter's effectiveness provided the models used in the ground truth data are close enough to the targets which actually appeared in the image.

3.2. EVALUATION OF LADAR IMAGES

3.2.1. Preliminary Results of Measuring Classifiability vs. Range in LADAR

One of the unanswered questions about range data is: "At what range can targets be accurately classified?" Of course the answer depends on may factors including the sensor, the weather, and the target. In the work presented here an attempt has been made to find an upper

bound on the classification accuracy by measuring the classifiably of noiseless synthetic data. The only noise present in such data is the spatial quantization noise caused by having fewer pixels on target as the target is farther away. The following section presents the results of the study.

3.2.1.1. The Synthetic Images

The synthetic images were generated as shown in Section 4.1. The four targets used in these experiments are the M60A1, M113, BMP, and BRDM2. Each target was viewed from zero elevation (slant angle of zero). The targets were rotated from zero to 360 degrees and an image was taken every size degrees for a total of sixty images per class. Section 4.1.3 discussed how to add noise to an image, however for these experiments, no noise was added.

3.2.1.2. The Segmentation and Features

Since synthetic data is used, the data is perfectly segmented, therefore no segmenter is needed. The same two feature sets were used as in Section 3.3.4.2.

3.2.1.3. Classification Results

Tables 3.2.1 and 3.2.2 show the results of the experiment for each feature set.

Table 3.2.1 Classification accuracy of noiseless synthetic LADAR data. 60 each of M60A1, M113, BMP, MRDM2s using shape features.

Shape features (Table 3.3.10)		
Distance	Lower Bound	Upper Bound
500m	0.0	0.42
1km	0.0	0.42
2km	0.0	0.0
3km	0.83	1.67
4km	4.17	5.83
5km	0.00	0.00

As expected (due to lack of sensor noise) the classification accuracies were very high. In general the accuracies decreased as the distance to the target increased. Unexpected results appear at 5km using the shape feature set and at 4km using the Beta feature set. We can offer no explanation for this happening at this time.

3.2.1.4. Conclusion and Future Work

Since noiseless data is being used conclusion about *real* data cannot be made. (Other than the obvious "the classification accuracy will decrease with distance.")

Table 3.2.2 Classification accuracy of noiseless synthetic LADAR data. 60 each of M60A1, M113, BMP, MRDM2s using Beta features.

Beta features (Table 3.3.11)		
Distance	Lower Bound	Upper Bound
500m	0.0	0.0
1km	0.83	0.83
2km	2.08	2.50
3km	1.67	4.17
4km	1.25	3.33
5km	3.33	5.00

Future work in the area will start with using synthetic data with noise added. Section 4.1.3 gives details on how such data is being created.

3.2.2. Further Experimentation to Determine Classifiability vs. Range

In continuing the study of classifiability vs. range, the classification experiment in Section 3.2.1 was expanded by generating synthetic targets and artificially degrading them with noise as in Section 4.1.3. The same orientation angle is still used, but the samples now include targets from four target classes at ranges of 0.5, 1.0, 2.0, 3.0, 4.0, and 5.0 km.

In addition, each synthetic LADAR image from a particular target class and range was corrupted with nine different noise characteristics, bringing the total sample size to 216. The noise characteristics were created by combining various levels of "drop out" and Gaussian noise. Realistic levels of these noise types were determined using the table of results presented in Section 4.1.3. From this data, overall minimums, maximums, and averages were determined for each noise type. For convenience we define three noise levels (*low*, *medium*, and *high*). These levels are shown in Table 3.2.3.

Table 3.2.3 Noise levels used in classifiability experiment.

Noise Parameters		
Level	Drop Out Probability	Gaussian Standard Deviation
low	0.005	7.00
medium	0.043	10.45
high	0.102	17.20

3.2.2.1. Segmentation

The planar patch fitting algorithm of Section 3.3.3.1 was again used to segment the samples. Unfortunately, the automatic threshold selection of this algorithm worked very poorly on these images, and the segmentation became an iterative process of choosing segmentation thresholds and observing results. The best segmentation of this step was then further processed with the aid of connected component labeling to remove holes in the silhouette as well as false target regions outside the silhouette. Figures 3.2.1 and 3.2.2 show some examples of synthetic imagery and the corresponding final segmentations.

3.2.2.2. Classification Results

Although separate experiments using both the shape and moment feature sets were run as before, problems which will require further investigation were encountered with the moment feature set experiment. The covariance matrices computed for these experiments were nearly singular, thus making the classification results useless.

Three experiments using the shape feature set produced valid results, however, and these appear in Tables 3.2.4, 3.2.5, and 3.2.6. Table 3.2.4 directly addresses the question of classifiability vs. range, while Table 3.2.5 examines the effects of noise on classifiability. Table 3.2.6 conveys the results of an overall classification experiment in which samples where not separated according to range or noise.

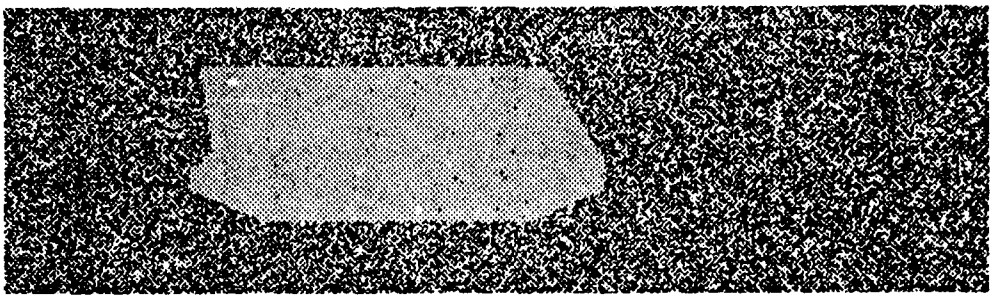
Table 3.2.4 Classifiability vs. Range study.

Classifiability vs. Range Shape feature set	
Range (km)	Error (Lower bound)
0.5	0.0 % (0/36)
1.0	2.8 % (1/36)
2.0	0.0 % (0/36)
3.0	19.4 % (7/36)
4.0	2.8 % (1/36)
5.0	0.0 % (0/36)

3.2.2.3. Conclusion

Originally the experiment presented here was to be performed on a much larger scale with 60 orientations for each target class. At this point, however the processing steps involved are not sufficiently refined and automated to make such an experiment possible for this report†.

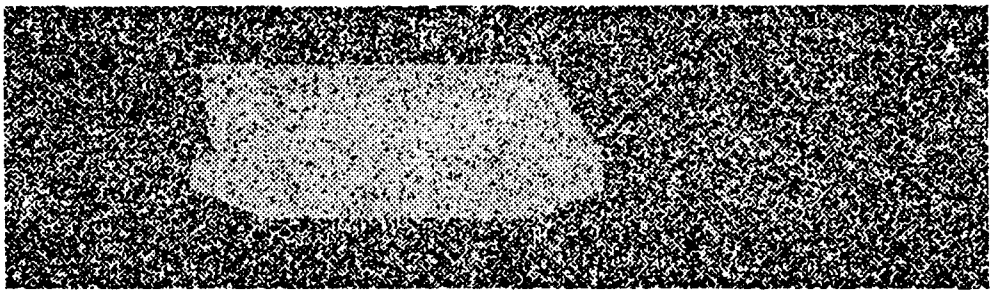
† Currently such an experiment would require an estimated *month* of continuous computer



Synthetic image, Noise: L/L



Segmented image, Noise: L/L

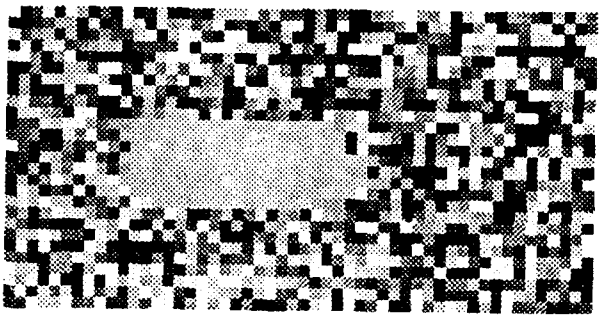


Synthetic image, Noise: H/H



Segmented image, Noise: H/H

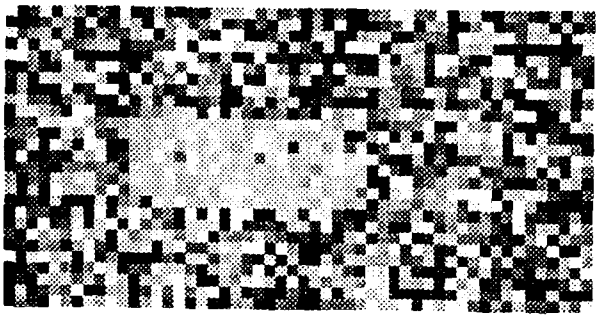
Figure 3.2.1 Segmentation examples. Synthetic image of M113 at 0.5 km.



Synthetic image, Noise: L/L



Segmented image, Noise: L/L



Synthetic image, Noise: H/H



Segmented image, Noise: H/H

Figure 3.2.2 Segmentation examples. Synthetic image of M113 at 5.0 km.

Table 3.2.5 Classifiability vs. Noise study. Noise characteristics are specified as: Drop Out Probability/Gaussian Standard Deviation. Letters indicate the noise level, i.e. L_ow, M_edium, or H_igh.

Classifiability vs. Noise Shape feature set	
Noise	Error (Lower bound)
L/L	4.2 % (1/24)
L/M	4.2 % (1/24)
L/H	4.2 % (1/24)
M/L	4.2 % (1/24)
M/M	0.0 % (0/24)
M/H	8.3 % (2/24)
H/L	4.2 % (1/24)
H/M	4.2 % (1/24)
H/H	4.2 % (1/24)

Table 3.2.6 Overall classification experiment. Classification experiment using all samples, i.e. no separation by range or noise.

Overall Classification Experiment Shape feature set	
Lower bound	Upper bound
4.2 % (9/216)	13.4 % (29/216)

Because of this, the sample set here is still too small to be of great help. In each range set, for example, there were only 36 samples. Each noise set only had 24 samples. This was the reason for not estimating the upper bound error in these two experiments. This estimate requires one sample to be left out, and with so few samples to begin with, the results would be questionable.

It is strange that no trend is evident in either of these experiments. This could be due to the small sample sizes. Another interesting point is the error jump in the range experiment for the 3 km case. The reason for this is still unknown, but the small sample size could have exaggerated the problem.

---

time alone, not including the user interaction needed to be sure all thresholds were set properly.

3.2.2.4. Future Work

The next step in this study of classifiability vs. range is the addition of orientation. Ideally the experiment will be an expansion of the work presented in Section 3.2.1 where synthetic images of the same four target classes were used. In this study, 60 different orientations of each sample were included. To run the above experiment with this many orientations would require processing 12,960 images. Obviously this experiment may need to be trimmed down to a more reasonable level, but in any case, some aspects of the classification process (such as segmentation) must be improved before such a large scale study can be attempted.

3.2.3. Optimal Sampling of the Feature Space

With the availability of synthetic imagery, endless data can be generated for a given experiment. Unfortunately experiments run on endless data complete after endless time. The synthetic test data must be carefully chosen to represent the types of images which will be viewed in real situations. This section presents two approaches to intelligently reducing the amount of data used in a classification experiment. The first approach, presented in Sections 3.2.3.1 - 3.2.3.6, uses a new method based on the Reduced Parzen Classifier. The second approach is classical clustering which is surveyed in Section 3.2.3.7.

3.2.3.1. Motivation

This section pursues the design of a statistical classifier for use in the *classifiability of LADAR silhouette data vs. range* experiment of [KaYo88]. The thrust of the experiment is to measure the decreasing classifiability of noisy synthetic LADAR silhouette data with increasing distance to the target. The initial approach was to train the classifier using 60 images of each target. An image was generated for every six degrees of target rotation. It was noted that this *uniform* sampling of rotation space was not the optimal approach to solving the problem. In this section we show a method of selecting a subset of available data to use as design samples for the classifier.

Originally it was hoped that this work would lead to a drastic reduction in the amount of processing necessary to train the statistical classifier by identifying a set of "key orientations" for the design samples that could be used for all ranges and noise levels. The results presented here indicate that this reduction in processing will not be possible, but it does appear that it may be possible to use fewer than 60 design samples in training the classifier. This would reduce the computation necessary for classification. In any case, the results of this work are interesting and have proven useful in understanding the estimation of class densities for purposes of classification.

3.2.3.2. The Feature Space

This section is merely an effort to better understand the problem at hand by drawing attention away from the target orientation question and focusing it on the feature space. It is the result of much thought toward better understanding the problem of density estimation and, although not actually implemented, should serve as an instructive prelude to the algorithm presented in the next section.

Since our statistical classification experiments are based on target class density estimation, it seems reasonable to pose the problem as follows:

> *Given that r* samples are sufficient to represent a class density p(x), and a "good" density estimate $\hat{p}_N(x)$ is available, what r sample subset of the N available samples gives the best density estimate $\hat{p}_r(x)$?*

Breaking the problem down a bit further: If j samples (j<r) have been chosen to estimate the class density $p(x)$, which of the N-j remaining samples should be chosen next? Is there a portion of the class density that is under represented? Of the N-j remaining samples, suppose we choose the least likely sample given the current class density estimate, $\hat{p}_j(x)$. In other words, choose the sample, S, from the N-j remaining samples such that $\hat{p}_j(S)$ is a minimum. This should force the r representative samples to be spread well over the N available samples, and could be accomplished through a simple application of Parzen density estimation techniques.

Although this procedure should space the r representative samples well across the class density's extent, nothing has been done to indicate sample populations in the regions surrounding the r representative samples. An obvious approach to this problem is to assign a weight to each representative sample based on how many of the N-r remaining samples are closer to it than any other representative sample. The commonly used distance metric, $(X-Y)^T\hat{\Sigma}^{-1}(X-Y)$, could be used here. The sample covariance, $\hat{\Sigma}$, is calculated using the N available samples.

3.2.3.3. The Reduced Parzen Classifier [FuHa]

Another approach to the problem of selecting r samples is to determine which of several r element subsets produce the best density estimate, $\hat{p}_r(x)$. The Reduced Parzen Classifier (RPC) [FuHa] algorithm uses this approach, and is based on computing a figure of merit by which density estimates are compared. This figure of merit is actually an entropy expression.

$$\int \ln\left[\frac{\hat{p}_r(x)}{\hat{p}_N(x)}\right]\hat{p}_N(x)dx \leq 0$$

The inequality, $ln(a) \leq a-1$, may be used to show the bound on this expression. Strict equality

* How to select the size of the reduced sample set, r, is briefly addressed in Section 3.2.3.6.

holds iff $\hat{p}_r(x)=\hat{p}_N(x)$. This expectation is replaced by a sample mean calculation in the actual algorithm.

$$J=\frac{1}{N}\sum_{i=1}^{N}(ln[\hat{p}_r(X_i)]-ln[\hat{p}_N(X_i)])$$

As in the last section, density estimates are computed using Parzen estimation techniques. If the computational burden were not an issue this criterion could be maximized over all possible r element subsets of the N available samples to select the optimum reduced sample set. To make the computation expense reasonable, however, a simplification is made in the algorithm:

(1) Arbitrarily select an initial assignment of r samples from the N sample data set. Call the r sample set STORE and the remaining N-r samples TEST.

(2) For each element, X_t, in TEST, compute the change in J that results if the sample is transferred to STORE. $\Delta J_1(X_t)=J_{r+1}(X_t)-J_r$.

(3) Pick the element, X_t, corresponding to the largest ΔJ_1 (and call it X_t^*).

(4) For each element, X_s, in STORE, compute the change in J that results if the sample is transferred to TEST. $\Delta J_2(X_s)=J_r(X_s)-J_{r+1}$.

(5) Find the element, X_s, corresponding to the largest ΔJ_2 (and call it X_s^*).

(6) The change of J due to these two operations is $\Delta J=\Delta J_1+\Delta J_2$. If X_s^* exists such that $\Delta J>0$, transfer X_s^* to TEST, transfer X_t^* to STORE, and go to step 2.

(7) Otherwise, find the element, X_t, corresponding to the next largest ΔJ_1 (and call it X_t^*).

(8) If X_t^* exists, go to step 4.

(9) Otherwise, stop.

As pointed out in [FuHa], this algorithm does not necessarily select the optimal reduced sample set, but experimentation has shown the algorithm to work well.

An important point to note is that, just as in the work of the last section, this algorithm depends upon a "good" density estimate, $\hat{p}_N(x)$. It is first used in the selection of the reduced sample set, and then in designing the Bayes classifier where a good sample covariance estimate is needed.

This is the primary reason that the initial hopes of this study are not realized. The results of the next section show that class densities are so drastically changed when features are extracted from noisy images that it makes no sense to estimate them with the same orientation samples chosen for a noiseless case. In other words, there does not appear to be a set of "key orientations" for designing a general classifier.

3.2.3.4. Experimental Results

The Reduced Parzen Classifier of Section 3.2.3.3 was implemented and the experiments of this section are meant to increase our understanding of the density estimation problem so that we may intelligently approach the classification vs. range experiment.

The primary questions addressed are:

(1) Given 60 images generated at six degree orientation intervals for a single target model at 0.5 km., which ten orientations best estimate the target's class density?

(2) Are the same ten orientations chosen for a different target?

(3) How does range affect which samples are chosen for the reduced sample set?

(4) How does noise affect the selection?

(5) How does the feature set used affect the selection?

(6) How does classification accuracy vary with the size of the reduced sample set?

The following experiments were performed to answer these questions:

(1) Silhouettes extracted from 60 noiseless, 0.5 km., BRDM2 synthetic images. *Area* and $H^2/Area$ features computed for these silhouettes. RPC algorithm used to select ten optimal samples for density estimation. Repeated for M60A1 target class.

(2) Experiment (1) repeated for 5.0 km. data.

(3) Experiment (1) repeated with high noise (PDO=0.102, GSD=17.2)* added to synthetic imagery.

(4) Experiment (3) repeated with *Area* and *Rectangularity* features.

(5) Experiments (3) and (4) repeated using reduced sample set sizes of 1,3,5,7,30 and 50 to train the classifier. Classification accuracies computed for each case.

All of these experiments were performed on two target classes in a two dimensional feature space, but may be generalized to M classes in an N dimensional feature space.

NOTE:

The feature samples of the following scatter plots have been transformed through a simultaneous diagonalization process which diagonalizes both covariance matrices leaving one equal to the identity matrix. Thus, the samples no longer possess the physical meaning indicated by the axes labels, but were derived from these features.

---

* As defined in [KaYo87]. PDO=Probability of Drop Out, GSD=Gaussian Standard Deviation.

3.2.3.4.1. Reduced Samples Sets for Two Noiseless Target Classes

Figures 3.2.3(b) and 3.2.4 show the ten orientations chosen to estimate the BRDM2 class density for a 0.5 km. sample set. The chosen samples fall very near one another in the feature space. Some, in fact, lie directly on top of one another since their orientations differ by exactly 180 degrees. These results seem to contradict the reasoning of Section 3.2.3.2 where it was suggested that the representative samples be spread well across the class density. It may be, however, that the weighting procedure of the same section would produce results similar to those seen here by "zeroing-out" fringe samples.

Figures 3.2.3(b) and 3.2.5 show the ten orientations chosen to estimate the M60A1 class density for a 0.5 km. sample set. Since the silhouette characteristics for the two classes are very different (at least we hope so for classification purposes), it is entirely reasonable that different reduced sample sets were chosen to represent the two classes.

3.2.3.4.2. Effects of Range

Does it make sense that the orientations chosen for the reduced sample set will be different at 5.0 km. than they were at 0.5 km? This probably depends on the target since certain target characteristics may be distinguishable at close range, but disappear at long range with fewer pixels on target. For example, the main gun of a tank may be invisible at 5.0 km., and this could significantly affect which silhouette angles best represent this target class.

Figure 3.2.6 shows the sets of ten samples chosen by the RPC algorithm for the two targets at 5.0 km. Although the orientations of the 0.5 km. reduced sample sets are different from those of the 5.0 km., Figure 3.2.7 shows that the feature samples fall in similar regions of the 5.0 km. scatter plots and produce equal resubstitution errors of 0%.*

3.2.3.4.3. Effects of Noise

Figure 3.2.8 shows that classification becomes much less accurate when noise is added to the target images making silhouette segmentation, and therefore feature extraction, much more difficult. It is also evident that the feature samples are not just perturbed by the noise, but the shape of the class density is drastically affected. Will the same orientations chosen to estimate noiseless class densities be most useful in representing noisy densities? Figure 3.2.9 clearly indicates that this is not the case.

The orientations of the reduced sample sets chosen using the noisy class densities are entirely different from those that were chosen from noiseless images. The resubstitution error is well over three times worse when the orientations chosen from the noiseless densities are used to design the reduced Parzen classifier for noisy samples.

---

* Resubstitution error is the error computed when the classifier is tested using samples of the design set [Fu72].

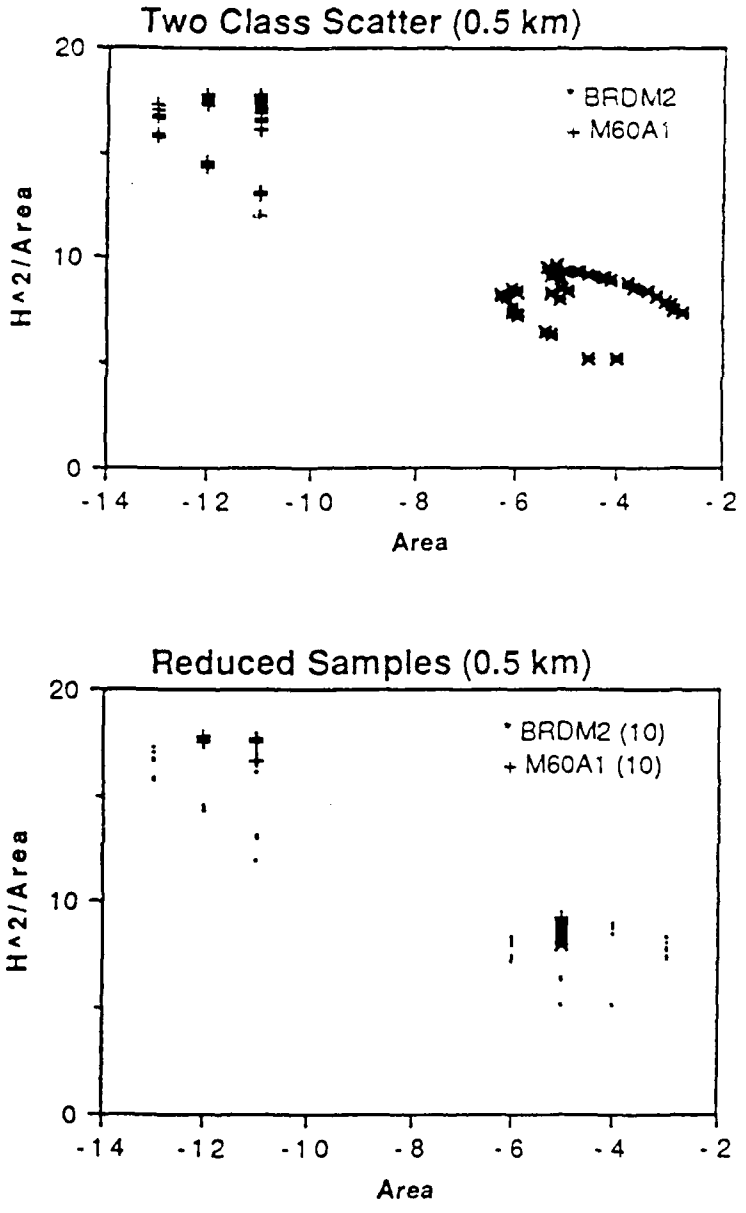
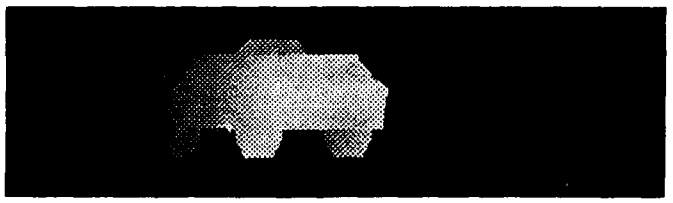
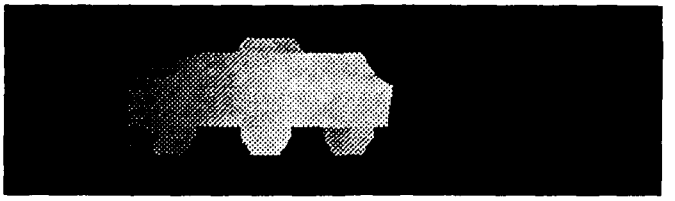


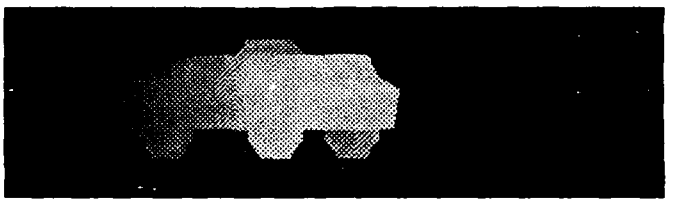
Figure 3.2.3 (a) Sixty samples of each target class. Resubstitution error=0%
(b) Ten samples of each reduced sample set. Dots denote samples NOT chosen for the reduced sample set. Resubstitution error=0%.
Corresponding target orientations:
BRDM2: 30,36,42,138,144,150,210,216,324,330 degrees
M60A1: 42,48,84,132,138,228,234,264,312,318 degrees



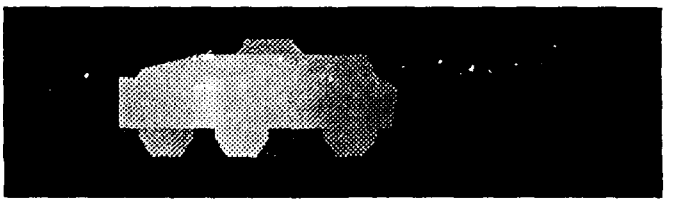
30 degrees



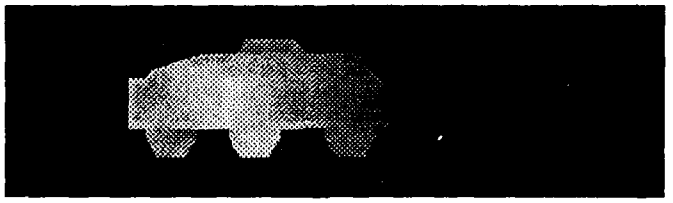
36 degrees



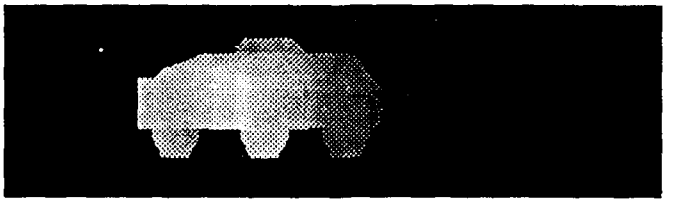
42 degrees



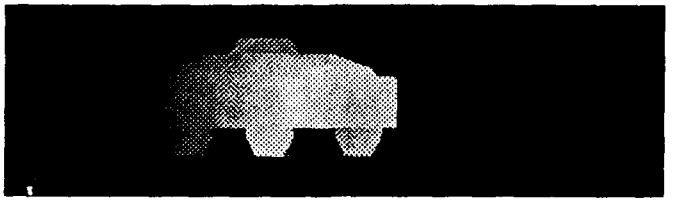
138 degrees



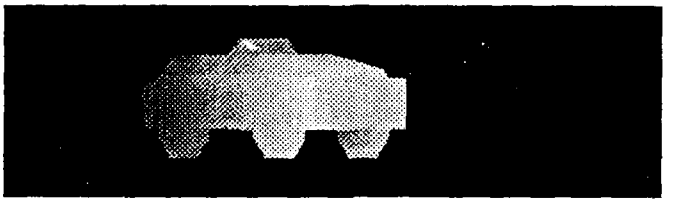
144 degrees



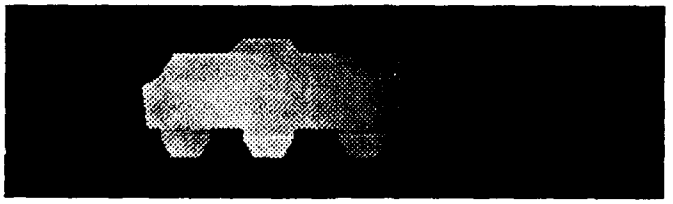
150 degrees



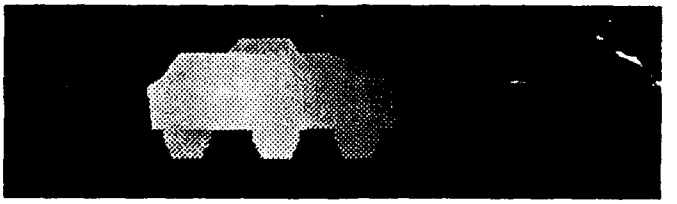
210 degrees



216 degrees

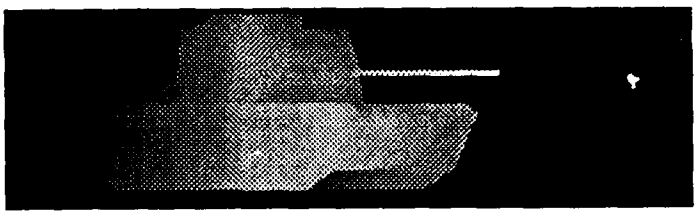


324 degrees

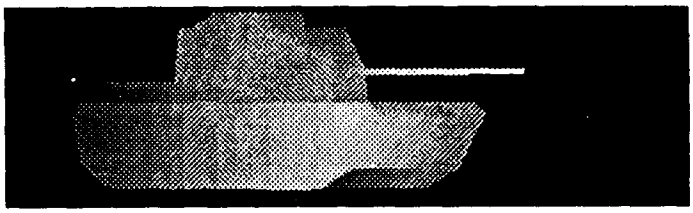


330 degrees

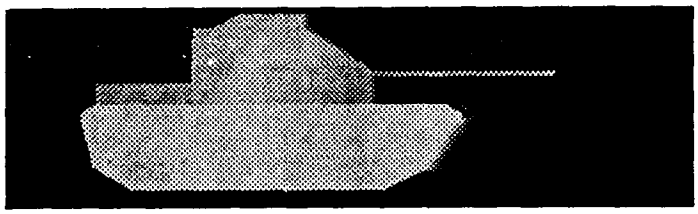
Figure 3.2.4 BRDM2 silhouettes chosen for 0.5 km. reduced sample set.



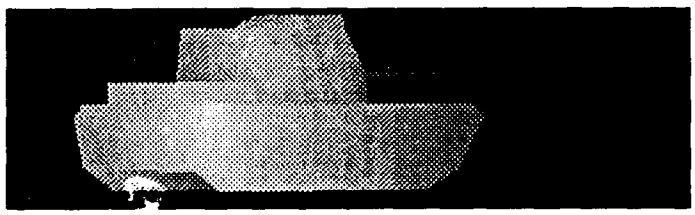
42 degrees



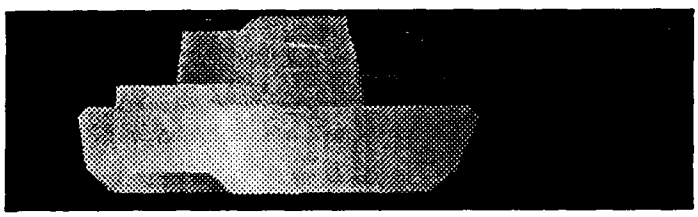
48 degrees



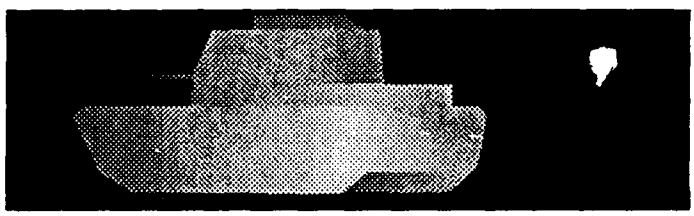
84 degrees



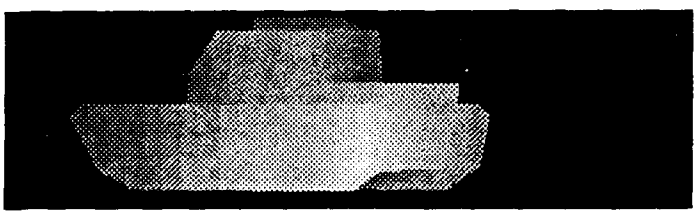
132 degrees



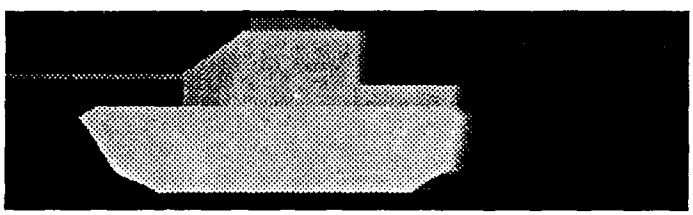
138 degrees



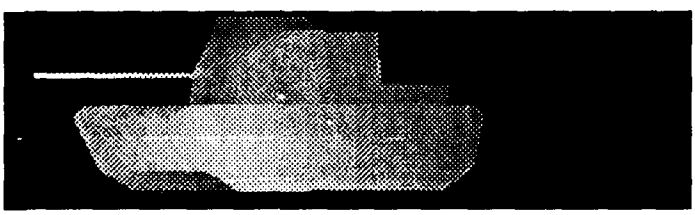
228 degrees



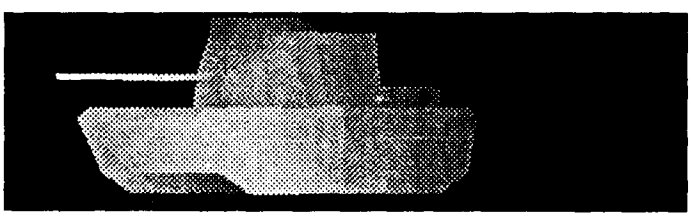
234 degrees



264 degrees



312 degrees



318 degrees

Figure 3.2.5 M60A1 silhouettes chosen for 0.5 km. reduced sample set.

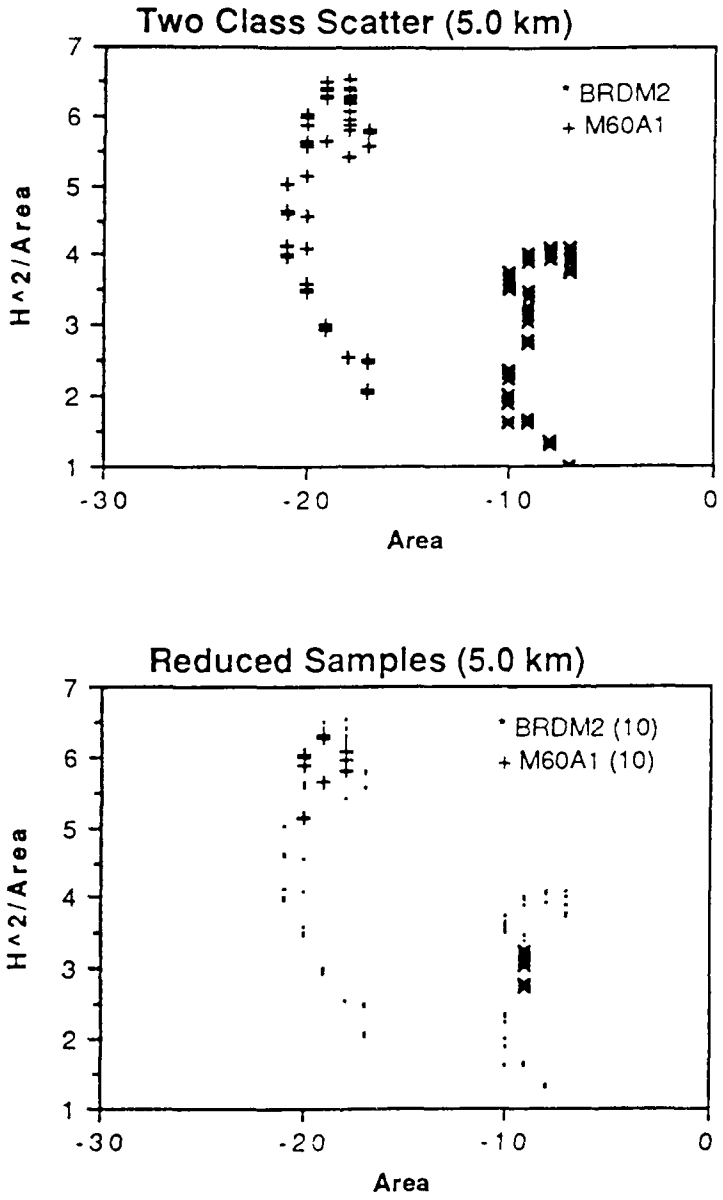


Figure 3.2.6 (a) Sixty samples of each target class. Resubstitution error=0%.
(b) Ten samples of each reduced sample set. Dots denote samples NOT chosen for the reduced sample set. Resubstitution error=0%.
Corresponding target orientations:
BRDM2: 30,36,42,132,138,144,204,210,318,324 degrees
M60A1: 78,96,104,132,228,234,306,312,318,324 degrees

Location of Reduced Sample Sets in 5.0 km. Class Densities

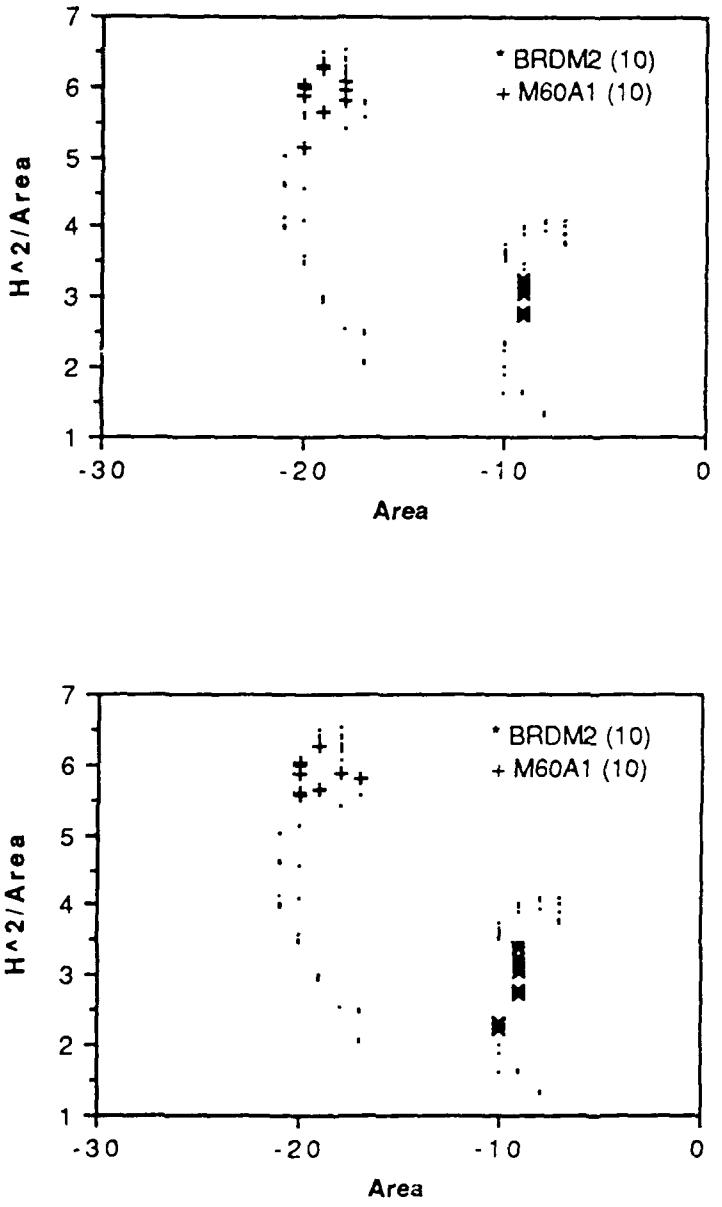
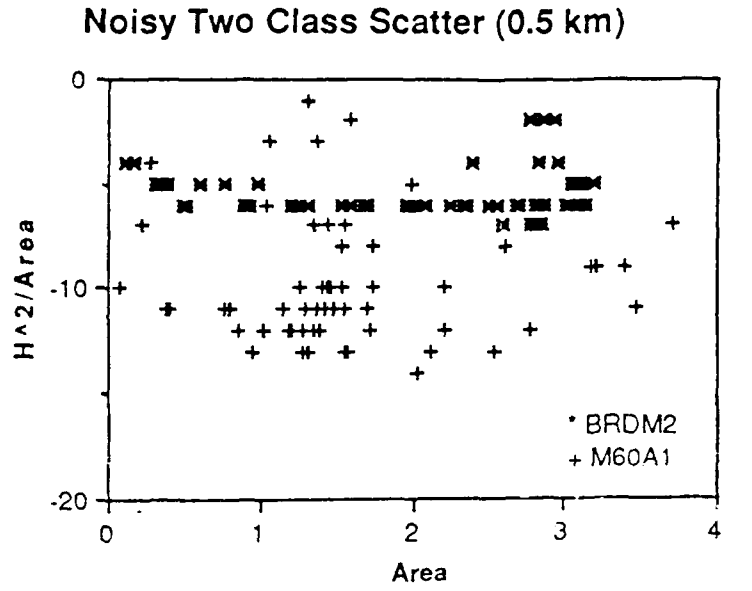


Figure 3.2.7 (a) Reduced sample sets chosen using 5.0 km. data. Resubstitution error=0%
(b) Reduced sample sets chosen using 0.5 km. data. Resubstitution error=0%
(Dots denote samples NOT chosen for the reduced sample sets.)



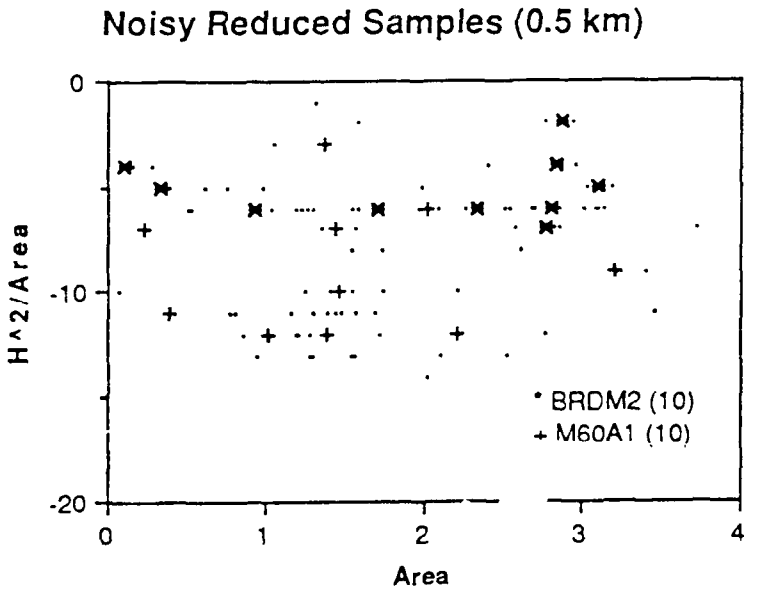


Figure 3.2.8 (a) Sixty samples of each target class. Resubstitution error=3.3%
(b) Ten samples of each reduced sample set. Dots denote samples NOT chosen for the reduced sample set. Resubstitution error=3.3%
Corresponding target orientations:
BRDM2: 0,84,264,282,300,306,318,330,342,354 degrees
M60A1: 84,126,150,174,192,210,264,288,318,354 degrees

Location of Reduced Sample Sets in 0.5 km. Noisy Class Densities

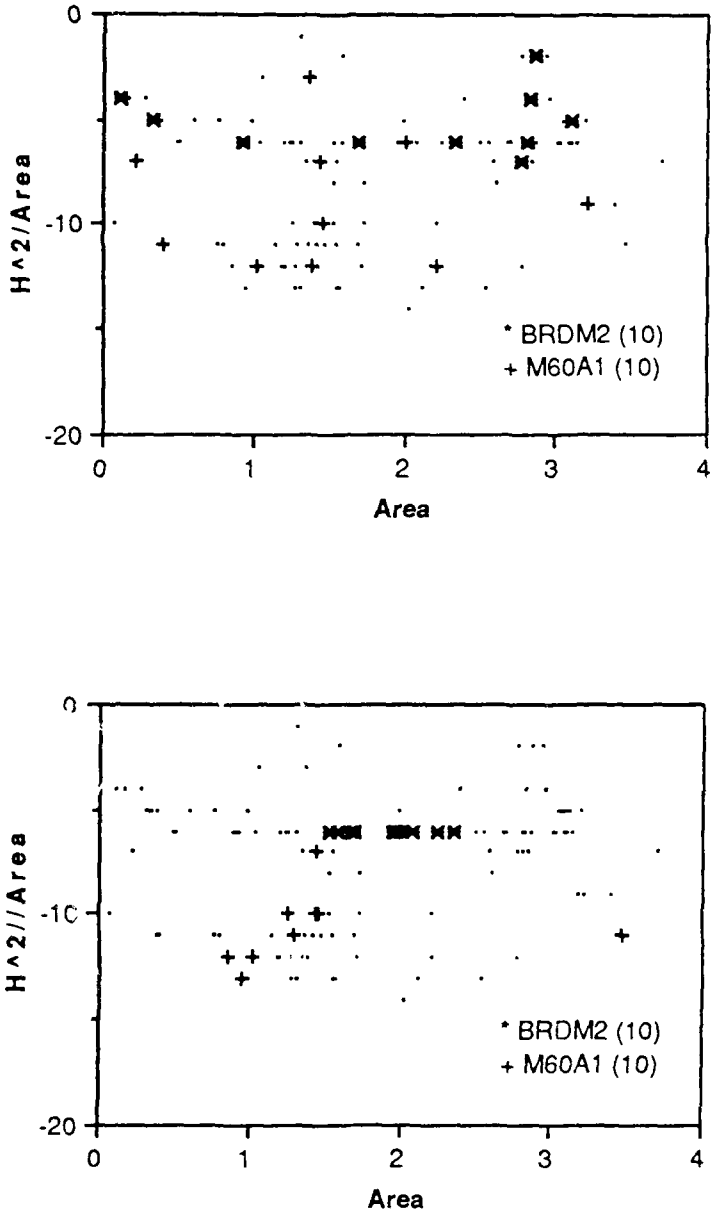


Figure 3.2.9 (a) Reduced sample set chosen using noisy 0.5 km. data. Resubstitution error=3.3%
(b) Reduced sample set chosen using noiseless 0.5 km. data. Resubstitution error=11.7%
(Dots denote samples NOT chosen for the reduced sample sets.)

3.2.3.4.4. The Feature Space and Selection of the Reduced Sample Set

Figure 3.2.10 examines the class densities for noisy samples using a different feature space. As expected, the orientations chosen for the reduced sample set in this feature space are different from those chosen in the previous feature space. Also, the overlap is different for the two feature spaces as evidenced by the resubstitution errors.

3.2.3.4.5. Classification Accuracy and the Size of the Reduced Sample Set

Although no theoretical analysis was performed to determine an optimal size for the reduced sample set, some interesting experimental results are discussed here. Since the samples of Figures 3.2.8(a) and 3.2.10(a) exhibited class overlap, they provided an excellent opportunity to determine how classification accuracy varies with the size of the reduced sample set.

Ten was arbitrarily chosen as the size of the reduced sample sets used in all of the previous plots, and, as shown by the graphs of Figure 3.2.11, is an interesting sample size. In the $Area-H^2/Area$ feature space, the classification accuracy is the same as that attained using all 60 samples. For the *Area–Rectangularity* feature space the ten sample set produces one of the best classification accuracies. Better than that attained using all 60 samples!

It is important to note that these experiments do not establish ten as an important number. The results presented here are for two dimensional feature spaces with only two target classes present. The classification accuracy vs. sample size could be very different in higher dimensional feature spaces with multiple target classes.

Nonetheless, it is interesting to see that reduced sample sets can improve classification accuracy. This may occur because of outlier or boundary samples in large sample sets which, in addition to being misclassified, may, when included in the design set, cause the misclassification of other samples.

3.2.3.5. SUMMARY

Several interesting results have been found using the reduced Parzen classifier of Section 3.2.3.3. It was interesting to see, for instance, that the feature samples chosen for the reduced sample sets were not spread across the entire class density, but were actually grouped together. This may not seem intuitive from a density estimation point of view, but the classification accuracies exhibited by the reduced sample sets certainly validate it from the classifier design perspective. It was also instructive to see that a carefully selected smaller set of design samples can produce classification results comparable with those of larger design sets.

Finally, the results of this study clearly show that it will not be possible to select a set of key target orientations which will provide optimal target discrimination under all noise conditions.

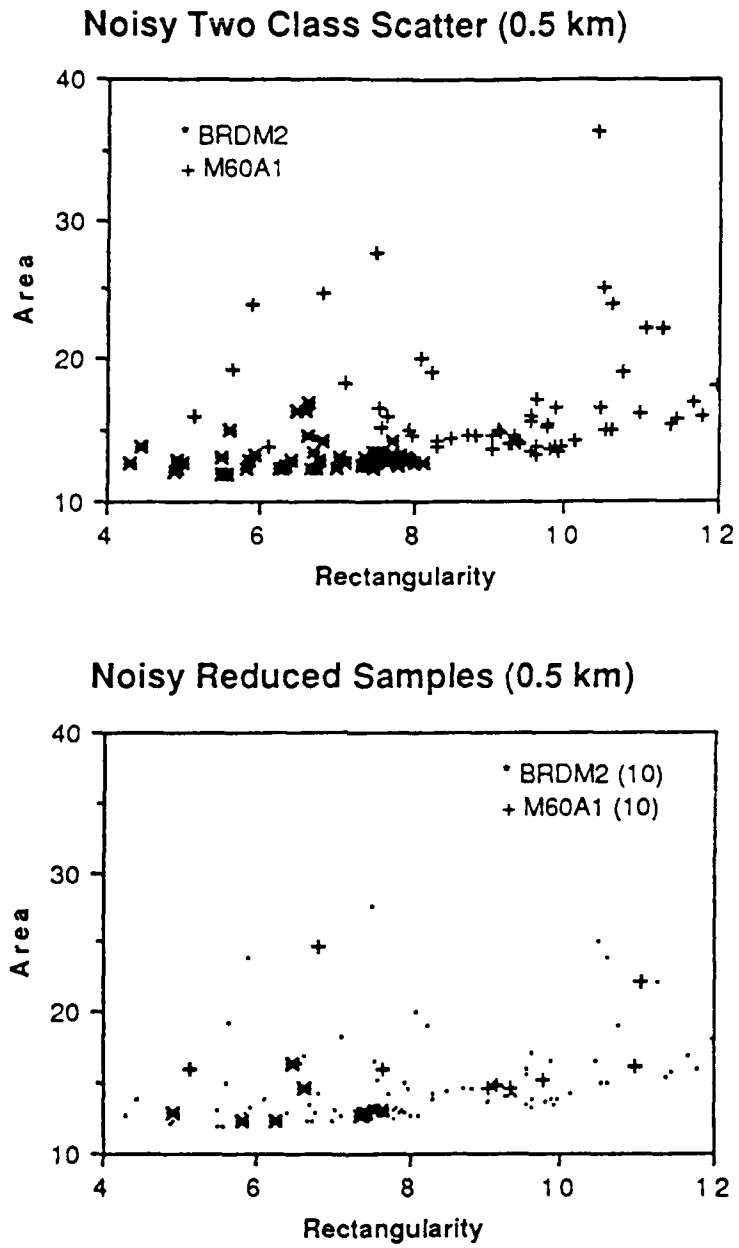
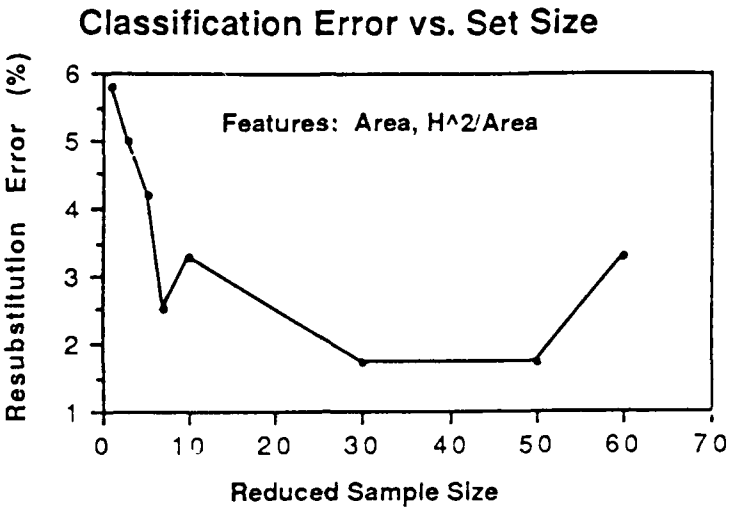


Figure 3.2.10 (a) Sixty samples of each target class. Resubstitution error=2.5%
(b) Ten samples of each reduced sample set. Dots denote samples NOT chosen for reduced sample set. Resubstitution error=1.7%
Corresponding target orientations:
BRDM2: 18,42,66,90,96,138,156,174,306,312 degrees
M60A1: 150,168,180,186,210,228,252,276,282,342 degrees



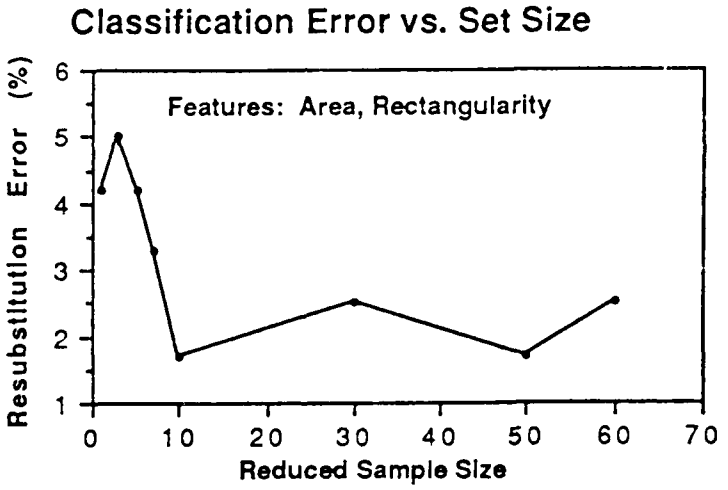


Figure 3.2.11 Some classification errors obtained using various reduced sample sizes.

3.2.3.6. Future Work

The results of this study should be verified in higher dimensional feature spaces with multiple target classes. Also, a more rigorous study to determine the optimal size of the reduced sample set should be performed. A first step might be to generalize the classification error vs. sample size experiment of this study, to the multiple target, N dimensional feature space situation.

3.2.3.7. A Survey of Classical Clustering Techniques

What follows is a look at some well-known clustering techniques. One primary use of clustering (unsupervised learning) has been pattern recognition when no *a priori* labels are available.

Our hope is that one, or a combination, of these classical techniques may be useful in partitioning a target class into regions of similar samples that may be represented by a set of distinguished viewpoints.

This goal, however, may meet the same difficulties seen in the study of [KaYo88]. Clustering is a completely different approach, however, and it may be reasonable to expect groups of similar samples to stay grouped under various levels of imagery noise and range characteristics even though the class density as a whole is completely distorted.

3.2.3.7.1. Outline of methods

Below is an outline of some of the oldest and most well-known clustering techniques. These techniques fall into one of two categories:

1) Hierarchical: Elements split/merged at one level remain split/merged at all lower/higher levels.

2) Non-hierarchical

I. **Hierarchical Clustering Techniques**

A) Agglomerative Methods (bottom-up)

1) Linkage

a) Single-linkage (Nearest-neighbor)

- Joins clusters by their two nearest elements
- Terminates when the shortest distance between the nearest elements of any two clusters exceeds some threshold
- Corresponds to the *minimum spanning tree* of graph theory if the threshold is not met and grouping continues through all elements
- Has a "chaining" characteristic which is good at finding long, winding clusters.

b) Complete-linkage (Furthest-neighbor)

- Joins clusters if their two most distant elements are closer than some threshold
- Terminates when no clusters may be merged under this threshold
- Each cluster corresponds to a *complete subgraph* in graph theory
- "Chaining" is discouraged.

c) Compromise

- Some type of averaging criterion is used rather than the extremum measurements
- Tend to handle outliers better.

2) Centroid Methods

- Clusters with the closest means (centroids) are merged.

3) Error Sum-of-Squares

- At each level of merging the clustering must be the one which increases this criterion by the *least*. (Ex. [Wa63])

B) Divisive (top-down)

- Little mention in the literature. Works on splitting rather than merging.

II. **Non-hierarchical**

A) Fixed number of clusters

1) Nearest Centroid Sorting

- Start with seed elements and merge in elements according to which cluster centroid is nearest. (Ex. [Fo65], [Ma67])

B) Variable number of clusters

- These algorithms usually ha[illegible]ne device for either:

1) reducing the number of [illegible]rs if two clusters are very near one another, or

2) increasing the number of clusters if a data unit is too far from all existing clusters.

(Ex. MacQueen's k-mean method with coarsening and refining parameters [Ma67], ISODATA (very elaborate and expensive) [BaHa65])

3.2.3.7.2. Some Criteria for Driving a Clustering Algorithm

The algorithms above use various criteria for determining which partition is optimum. The nearest centroid sorting methods, for instance, try to minimize the error sum-of-squares by constructing cloud-like clusters closely packed around a centroid. Some other criteria are:

- Minimize:

$$tr\ S_W\ ,\ |S_W|\ ,\ tr\ S_T^{-1}S_W\ ,\ \frac{|S_W|}{|S_T|}\ ,\ error\ sum\text{-}of\text{-}squares$$

- Maximize:

$$tr\ S_W^{-1}S_B$$

where

$$S_W=\sum_{i=1}^{c}\left\{\sum_{x\varepsilon\Omega_i}(X-M_i)(X-M_i)^T\right\};\ \Omega_i\ denotes\ cluster\ i$$

$$S_B=\sum_{i=1}^{c}n_i(M_i-M)(M_i-M)^T\ ;\ M=\frac{1}{n}\sum_{i=1}^{c}n_iM_i$$

$$S_T=S_W+S_B$$

The eigenvalues of S_W measure the spread of the samples (within their respective clusters) along the feature space coordinate axes. Minimization of $tr\ S_W$ is, in fact, the same as minimizing the error sum-of-squares. The eigenvalues of S_B, on the other hand, indicate the spread of the clusters within the entire class density which should be maximized. The $tr\ S_W^{-1}S_B$ criterion is a measure of the ratio of these with the between-class spread in the numerator.

Note that since the total-scatter matrix, S_T, is invariant to the cluster partitioning chosen, a linear transformation can be used to make the scatter white ($S_T=I$) thereby reducing these criteria to functions of S_W.

Other types of criteria have also been used. Some work has been done, for instance, by viewing classes as mixtures of multivariate *normal* distributions. Examples can be found in [ScSy71] and [Wo70].

3.2.3.7.3. Choosing a Method

Most clustering techniques are good at identifying a specific type of cluster. For example, the single-linkage method has a unique ability to find long, winding clusters through a characteristic called "chaining." The nearest centroid sorting methods, on the other hand, tend to group samples into "cloud-like" clusters.

The best strategy in approaching a clustering problem may be to use some combination of techniques. Three such strategies are listed by [And73]:

1) Obtaining a rough idea of the clusters with some inexpensive algorithm may be very useful in determining which of the more elaborate methods to use next. A nearest centroid sorting method may be most useful if the number of clusters is known, but otherwise some hierarchical technique may be best.

2) Once clusters are found their removal can facilitate later processing. After a particular technique has found certain types of clusters, other methods may be used to further group the data.

3) A hierarchical method may be used as a way of "seeding" nearest centroid sorting algorithms.

3.2.3.7.4. Summary

There are a multitude of algorithms available for performing cluster analysis and in many cases some combination of them is necessary to solve the problem. The works of Scott and Symons, and Wolfe which treated class densities as mixtures of clusters having *normal* distributions may be of particular interest in the problem of determining distinguished viewpoints for classifier design.

Before placing much confidence in the potential of the above techniques, however, some study should be performed to determine the effects of noise and range on cluster structure. It may be that the set of distinguished viewpoints will vary with these phenomena as seen in [KaYo88].

3.2.4. Target Detection Using Ladar Data

Recently, the Night Vision Lab has centered much interest on the detection of tactical targets using LADAR data. However, because of the active nature of a LADAR sensor and the amount of time that it needs to gather data, it is advantageous to limit the number of scans that the sensor makes across a scene during the detection process. In fact, it would be ideal if a tactical target could be detected using data from a single LADAR scan line. Although reliable detection using a single LADAR scan line would be ideal, it is generally believed that more than a single scan line needs to fall on a target for the detector to work reliably. Thus the following question arises: How many LADAR scan lines need to fall on a tactical target for it to be reliably detected?

We have undertaken an exhaustive study to investigate the performance of a target detector when data from a limited number of LADAR scan lines is available. The study is aimed at determining the performance of the detector as the number of lines that fall on a target is varied. We are also investigating detector performance when various preprocessing techniques are applied to the raw LADAR data.

The following section reports on the results of detecting a target using a single scan line from a LADAR sensor. Section 3.2.4.3 - 3.2.4.5 expands these experiments to see if the false alarm rate can be reduced by using multiple scan lines are on target. The description of the code used to generated these results in presented in Appendix B.

3.2.4.1. Target Detection Using a Single Line of LADAR Data

Night Vision Lab has centered much interest on the detection of tactical targets using a single stripe of LADAR data. We want to determine if a single scan line of LADAR data contains enough information to allow a system to determine the presence and approximately location of a target. In this section we present two approaches to this problem, the first is to simply measure the length between range discontinuities, and if the length falls between two thresholds, it is labeled as a target. The second method draws on detection theory in that it measures the characteristics of the background and the targets in a known image, and then classifies an unknown scan line by comparing it to the known characteristics. The following sections present more details about each of the approaches and shows that the second approach subsumes the first approach.

3.2.4.1.1. A Simple Target Detector

This section presents a possible solution to the detection problem that is deterministic and is easily expressed algorithmically. This method is similar to the approach currently used by commercial groups today. It works as follows: First, the scan line is broken up into piecewise linear segments (i.e. if the range values of a pixel is within a certain distance from those of its neighbors, it would be said to lie in the same piece). Next the detector makes a decision via the following rule

> IF segment length is > MIN_LENGTH
> AND segment length is < MAX_LENGTH
> AND there is a jump discontinuity > MIN_DISTANCE
> THEN the segment represents a target.

Because of the effect of the background noise and the statistical variation of the target range values, one would most likely find that such a detector is not robust.

3.2.4.2. A Detection Theory Based Approach to Target Detection

We have chosen to apply concepts from detection theory to the problem of robustly detecting tactical targets with a limited amount of LADAR data. We are able to apply these concepts by temporarily assuming that only a single scan line of LADAR data is used to detect the targets. If this assumption is true, then the detector is, in effect, working on one dimensional signals. Therefore, the problem to be solved is the classic problem of detecting **signals with unknown parameters in noise** [Trees68] where the signal parameters depend on the type of target, orientation of the target, and position of the scan line on the target. Detection theory describes the optimal method of solving this type of problem by using estimates of the probability density functions of the target and background. Once we have described this target detection method using a single LADAR scan line, we will show how to generalize it so that multiple LADAR scan lines may be used to make the

detection process more robust.

3.2.4.3. Target Detection Using a Single Scan Line of LADAR Data

When designing the detector, it is helpful to assume (temporarily) that all targets will be a constant distance from the LADAR sensor and the sensor will have a constant field of view. Based on these assumptions, it may also be assumed that at most N range pixels from a single-line fall on the target. Since jump discontinuity information provides strong evidence that a pixel belongs to a target, it is advantageous to guarantee that at least one jump discontinuity will be included in the data for any target pixel. This is accomplished by using the range values of a pixel and its M=N+2 neighbors on the same scan line. The data values for a pixel and its $M-1$ closest neighbors can be interpreted as a data vector representing a single point in M dimensional space. Therefore, the collection of all possible target (or background) data vectors comprises a probability density function in M dimensional space. Such density functions can be estimated and the estimates used by a robust target detector. One type of target detector that uses estimated density functions works as follows: Denote the density function corresponding to the target as

$$f_{target}(x_1, x_2, \ldots, x_M)$$

and the background density function as

$$f_{background}(x_1, x_2, \ldots, x_M).$$

If the following M points, $P_0 = (d_1, d_2, \ldots, d_M)$, correspond to the data vector from some pixel, then

$$f_{target}(d_1, d_2, \ldots, d_M)$$

and

$$f_{background}(d_1, d_2, \ldots, d_M)$$

can be computed at P_0. The pixel can then be said to belong to an object if

$$f_{target}(d_1, d_2, \ldots, d_M) > C \times f_{background}(d_1, d_2, \ldots, d_M) \tag{3.1}$$

where C is equivalent to a threshold and depends on a-priori class probabilities and the costs of false alarms and detection misses. In order to make the problem tractable, the functional form of the density functions are usually assumed to be Gaussian; thus, the density functions can be expressed as:

$$p_i(X) = \frac{1}{\sqrt{2\pi^n}\sqrt{|\Sigma_i|}} \exp\left[-\tfrac{1}{2}(X-M_i)^T \Sigma_i^{-1}(X-M_i)\right]$$

where Σ_i and M_i are the covariance matrix and mean of density i respectively and the superscript T stands for the transpose of a matrix. Substituting this functional form for the

distribution into equation (3.1) and taking the natural log of both sides allows us to classify a pixel as belonging to a target if:

$$\frac{1}{\sqrt{2\pi^n}\sqrt{|\Sigma_t|}} \exp\left[-\tfrac{1}{2}(X-M_t)^T\Sigma_t^{-1}(X-M_t)\right] > \frac{C}{\sqrt{2\pi^n}\sqrt{|\Sigma_b|}} \exp\left[-\tfrac{1}{2}(X-M_b)^T\Sigma_b^{-1}(X-M_b)\right]$$

The subscripts t and b denote the target and background densities, respectively. Taking the negative natural log allows the pixel to be classified as belonging to a target if

$$\tfrac{1}{2}\ln|\Sigma_t| + \tfrac{1}{2}(X-M_t)^T\Sigma_t^{-1}(X-M_t) < -ln(C) + \tfrac{1}{2}\ln|\Sigma_b| + \tfrac{1}{2}(X-M_b)^T\Sigma_b^{-1}(X-M_b)$$

This can also be expressed as

$$(X-M_b)^T\Sigma_b^{-1}(X-M_b) - (X-M_t)^T\Sigma_t^{-1}(X-M_t) + \ln\left\{\frac{|\Sigma_b|}{|\Sigma_t|}\right\} < 2ln(C)$$

The left-hand side of this equation is commonly called the **log-likelihood ratio**. Note that if the value for a pixel is replaced by its log-likelihood ratio,

$$(X-M_b)^T\Sigma_b^{-1}(X-M_b) - (X-M_t)^T\Sigma_t^{-1}(X-M_t) + \ln\left\{\frac{|\Sigma_b|}{|\Sigma_t|}\right\}$$

then the optimal value of C can be determined by performing a threshold over the entire image. The more positive values of the log-likelihood ratio will represent pixels that are strongly believed to be part of the background and more negative values of the log-likelihood ratio will represent pixels that are strongly believed to be part of the target. We will call the image formed by replacing every pixel by its log-likelihood value the **log-likelihood image**.

Note that by assuming that the distributions are Gaussian, the decision function is completely specified by the mean vectors and covariance matrices of the distributions (M_t, Σ_t, M_b, Σ_b). Thus, these are the only parameters that the detector needs in order to operate. This type of statistical detection scheme is fairly robust because of its ability to handle random variations of the signal coming off of the target. It also minimizes false alarms by compensating for the statistical nature of the background. Note that this detection scheme is easily augmented to handle randomly sized targets caused by objects at varying ranges and the imaging device using different fields of view. Randomly sized targets can be accommodated by resampling the scan line around the pixel being classified to produce the correct sampling density.

It should be noted that this approach subsumes simpler approaches like the first detector presented. To show this, we will look at a simple example. For the purpose of the example, assume that the background is purely random and that the target is a box of length L and is being scanned on one of its sides as is show in Figure 3.2.12. In the deterministic approach, based on recording lengths of constant range values, there are two properties that a linear segment must possess in order to be classified as a target. First of all,

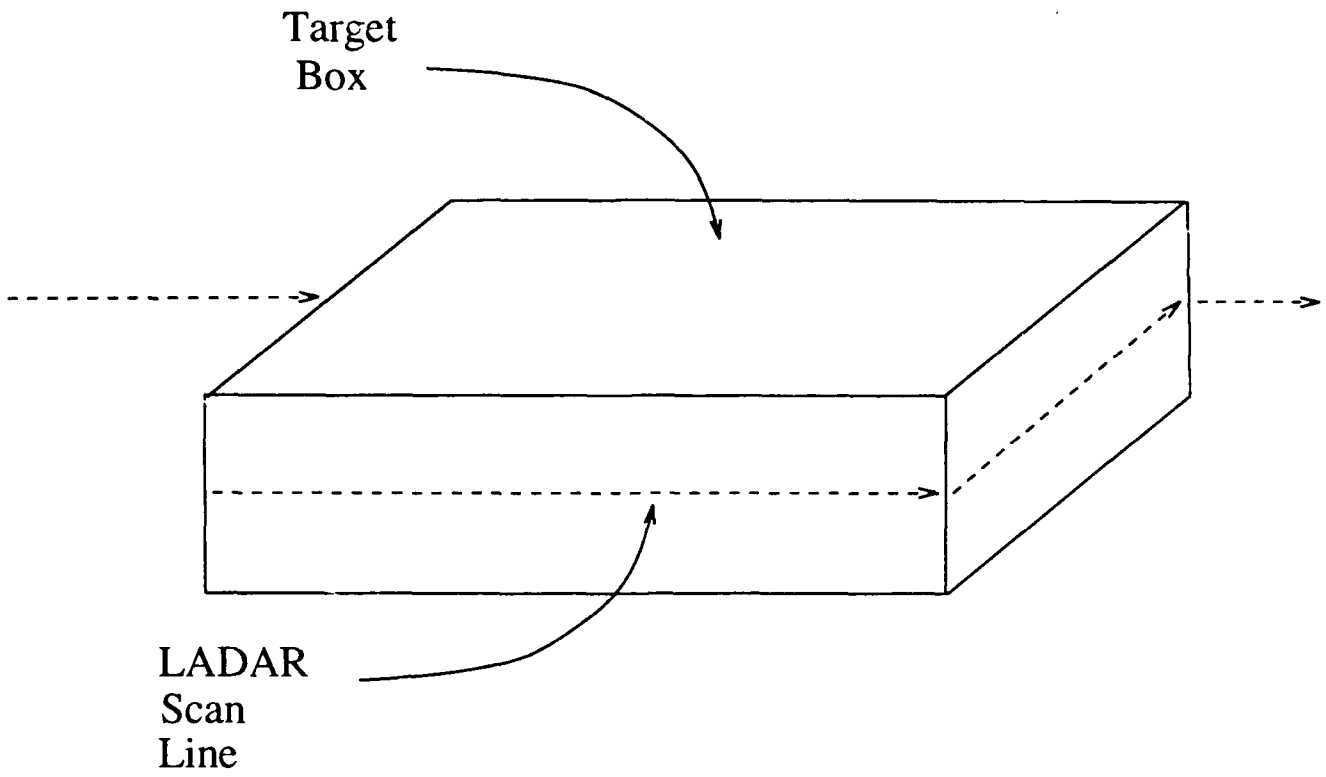


Figure 3.2.12 A single LADAR scan line scanning a box of length L.

the length of the segment must be between two bounds, MIN_LENGTH < ζ < MAX_LENGTH where ζ is the length of the segment actually detected. In the statistical detector, this same requirement is expressed by forcing target pixels to have ζ highly correlated pixels. The deterministic approach also requires that a jump discontinuity greater than the MIN_DISTANCE be present for a piece to be classified as a object. This requirement also appears in the statistical detector; it is expressed as highly uncorrelated pixels with mean values that differ greatly from the pixel being classified. In the deterministic approach the decision boundary for a pixel corresponds to the rectangular area shown in Figure 3.2.13c. In the statistical approach, the decision boundary can have arbitrary shape depending solely on the shape of the density functions. Some commonly used decision boundary shapes appear in Figure 3.2.13.

3.2.4.3.1. Results of Detection Experiments

Five experiments were run to test the detection procedure discussed above. Each experiment was run on six images representative of the images that we possess. These sample images are shown at the bottom of Figures 3.1.1-3.1.6. Two types of experiments were run; the first three experiments used the data vector generation procedure discussed in Section 3.2.4.3 The dimensionalities of the data vectors in these experiments were M = 25, 11 and 5. In each experiment the data was down-sampled to the correct dimensionality if necessary. After the images were run through the detector, some simple post-processing was done to improve the output. This post-processing consisted of labeling as false detections all detections that were less then 3 pixels long. Figure 3.2.14 (b-d) shows the output of the detector (after post processing) for the first three experiments. As is readily seen, the performance of the detector is satisfactory when large (M = 25) data vectors were used. The degradation of the detector's results when using small dimensional data vectors is probably due to the large loss of information that occurs when the data is drastically down-sampled. Table 3.2.7 lists the probability of detection and the false alarm rate for the first three experiments. Note that the parameters listed in Table 3.2.8 **do not** reflect the decrease in false alarm rate or probability of detection produced by the post-processing. Also note that changing the decision threshold would improve detection performance for small dimensional data vectors. It should also be noted that the detector completely missed some of the targets on scan lines in which a relatively thin portion of the target was scanned (e.g. on tank turrets).

A second type of detection experiment was also performed; this one was highly tuned to detect targets in data typical of the first LADAR data set. In these experiments, the detector worked on the same principle as many of the LADAR segmenters currently under investigation. Like the segmenters, the detector based its decision on the high variance of the background pixels. In this scheme, if the variance of the range values of the local neighborhood of a pixel was low, the pixel was labeled as belonging to a target; otherwise, it was labeled as background. For these experiments, the data vectors were composed of a

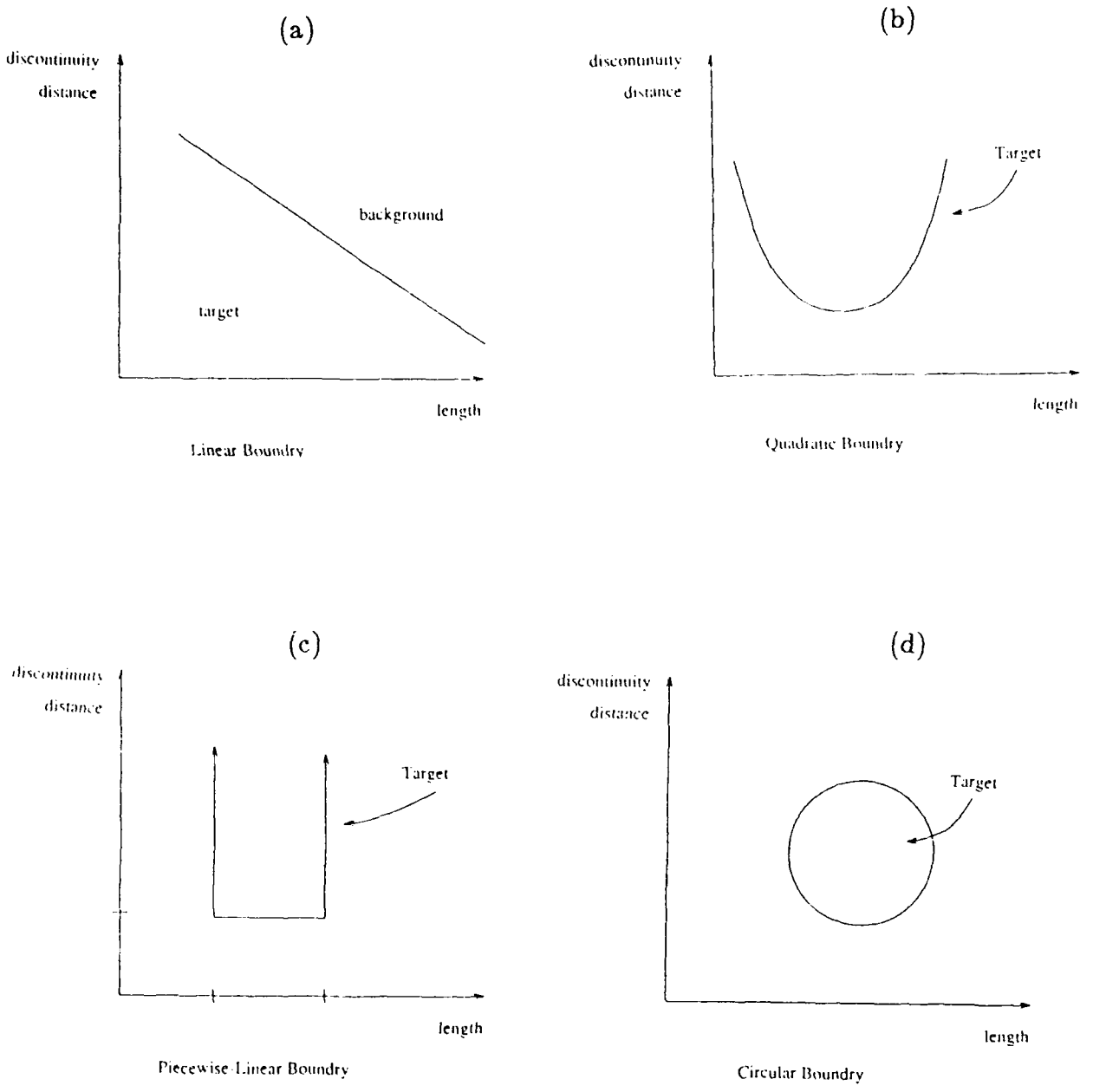


Figure 3.2.13 Commonly used decision boundaries.

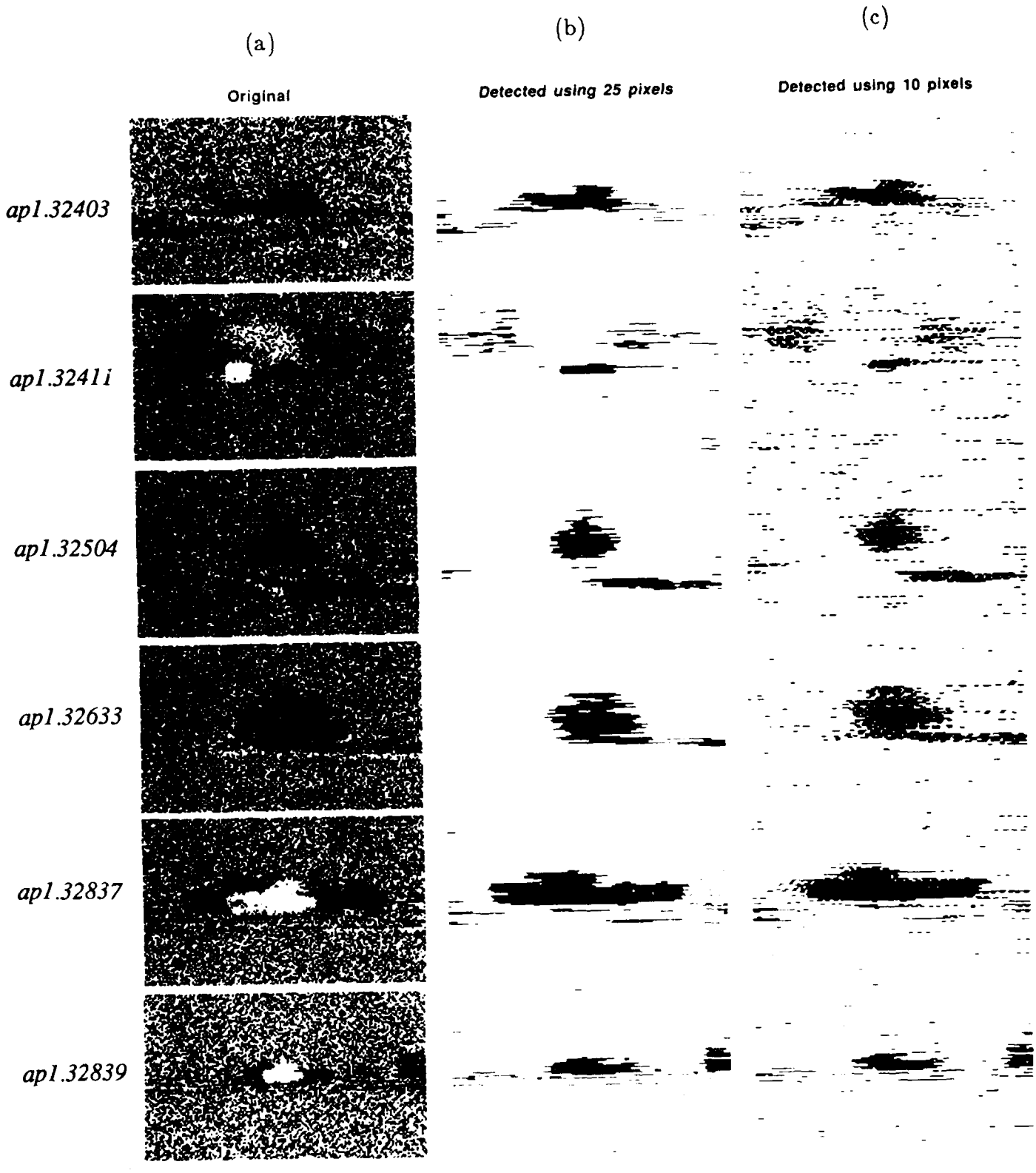


Figure 3.2.14 Output of a target detector which uses only a single scan line of LADAR data.

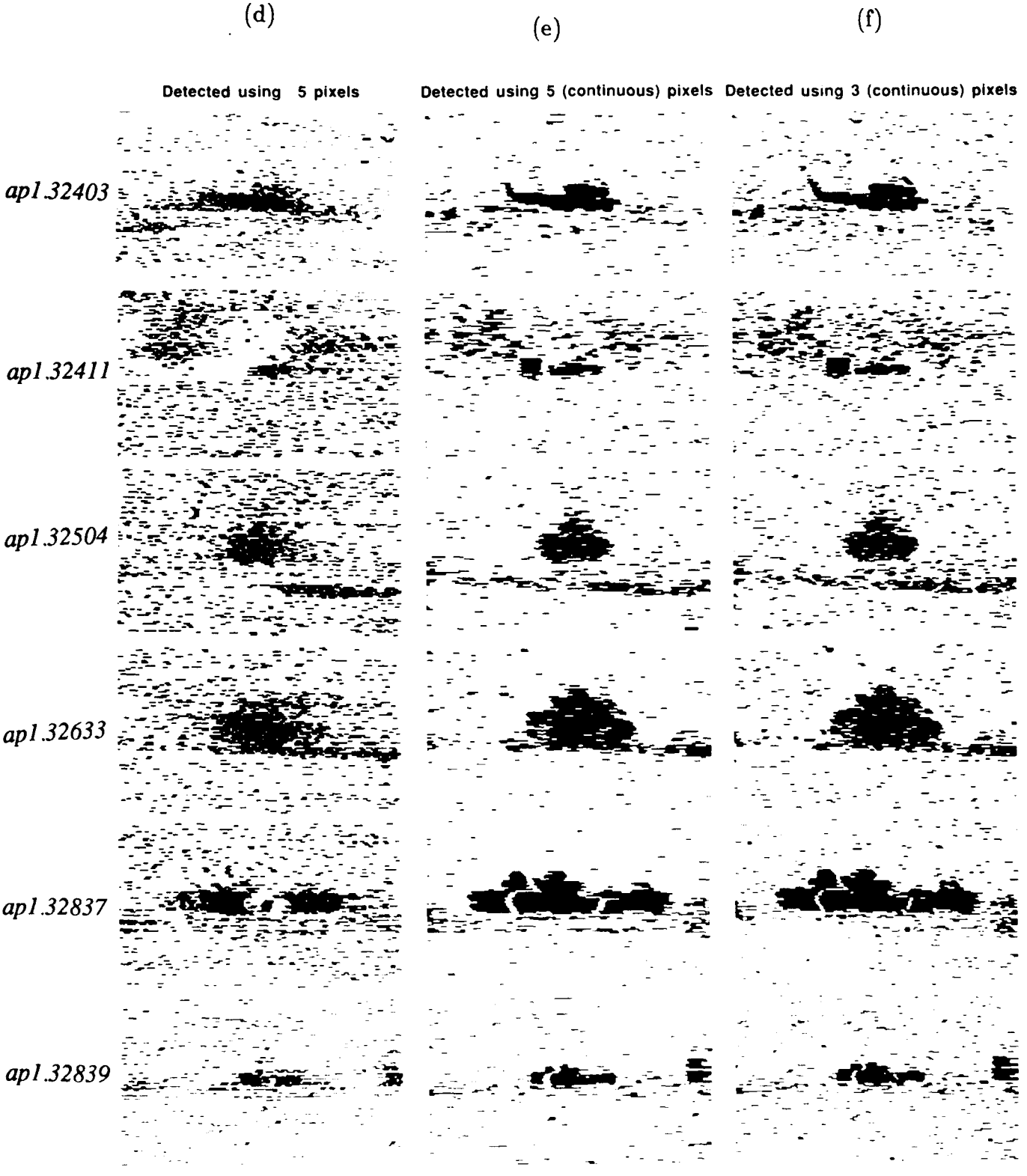


Figure 3.2.14 (continued)

Table 3.2.7 Probabilities of detection and false alarm rates for the first three experiments.

Experiment	Dimension	Probability of Detection	False Alarm Rate
1	25	85.8%	7.3%
2	11	82.3%	17.8%
3	5	73.8%	16.9%

pixel's range value and the range values of the pixel's $M-1$ *immediate* neighbors. Two experiments of this type were run. The data vector in the first experiment was 3 dimensional and consisted of the range value of a pixel and the range values of its left neighbor and its right neighbor. A 5 dimensional data vector was used in the second experiment. The vector was composed of the range value of a pixel and the range values of the two closest neighbors to its left and to its right. Figure 3.2.14 (e,f) shows the output of the detector after post-processing. Table 3.2.8 indicates the probability of detection and false alarm rate of the last two experiments. In these two experiments the targets were detected fairly reliably because of the high variance of the background pixels. It is expected that performance of this detection scheme will drop radically when absolute-range LADAR data becomes available.

Table 3.2.8 Probabilities of detection and false alarm rates for the last two experiments.

Experiment	Dimension	Probability of Detection	False Alarm Rate
4	5	90.9%	9.6%
5	3	84.6%	9.0%

3.2.4.3.2. Conclusions

These two types of experiments were run to demonstrate the flexibility of the detection programs and to show that detector performance depends on data vector composition. The same programs were used in both experiments; however, the data vector generator was prevented from down-sampling the scan lines in the second scheme. This approach to detection shows a relatively low false alarm rate, especially with methods 4 and 5 which use postprocessing with a small dimensional space to achieve an error rate under 10%.

3.2.4.4. Target Detection Using a Multiple Lines of LADAR Data – Approach 1

We have extended our work on LADAR detection to investigate the effects of using more than one scan line to detect a target. It is hoped that using the information from multiple scan lines will improve detector performance. For this report, two techniques for multi-line detection were investigated. Both techniques are simple extensions of the

statistical based approach discussed previously.

In the single line case, a range pixel was classified as either belonging to a target or to the background based on its range value and the range values of $M-1$ neighbors on the same scan line. An obvious extension to enable this procedure to handle data from multiple scan lines is to take M data points from all the scan lines being used to make the decision. Therefore, if L lines are being used in the decision process, the dimensionality of the data vector would now be:

$$Dim = M \times L$$

where Dim is the dimensionality of the multi-line vector, M is the dimensionality of the single-line vector and L is the number of scan lines.

The second extension to multi-line detection is slightly more complicated than the first. Conceptually, this extension can be viewed as L independent single line detectors whose outputs are combined to produce a single decision. The subdetectors for each line work exactly in the same manner as the single line detector previously described. After each subdetector decides if it has found a target, it sends its decision to the main detector along with the confidence associated with that decision. The main detector then combines the results from the subdetectors and arrives at a single decision on whether a target is present or not. Because the subdetectors in this extension are independent, the size of the data vectors are limited so that only M pixels at any time were needed by the detector; this is a large improvement over the first extension which needed $M \times L$ pixels for detection. In the previous detection experiments, the output of the detector indicated only if the pixel was to be classified as target or background and gave no indication of the certainty attached to that decision. This detection scheme weights the evidence provided by the subdetectors according to the confidence values attached to their decisions. It does this as follows: Define S_{target} as the sum of all the confidence values of the scan-lines that assert that a target has been detected and $S_{background}$ as the sum of all the confidence values of the scan-lines that deny the presence of a target. Finally define S_{total} as

$$S_{total} = S_{target} - S_{background}$$

Now, if S_{total} is positive then the assertion is made that there is a target somewhere in the lines that were scanned; otherwise, the presence of a target is denied.

Once the presence of a target is hypothesized, the detector tries to establish which scan-lines fell on the target. It accomplishes this by classifying as a target one pixel per scan line that contributed to S_{target}.

3.2.4.4.1. Experimental Procedure

A number of experiments were run to establish if either extension out performed the other. For each experiment, we have plotted the probability of detection and false alarm

rate when there were between one and five scan lines on target. In addition to the two extended detectors, the original classifier was also run to provide a baseline for comparison. Five pixels per scan-line were used in the experiments on the extended detectors. The size of the data vector for the baseline experiment depended on the number of scan lines used in the extended experiments. For example, if L scan lines were used in the multi-line experiments, $5 \times L$ were used in the baseline experiment. Thus the size of the final data vectors for the baseline and first extension were equal for all experiments. Because the dimensionalities of the two data vectors are equal, it should be possible to compare the results of the two techniques to see the affect of using multiple lines to detect the target. Otherwise, it would not be possible to determine if any increase in performance was due to additional information contained in the other scan lines or if it was due to the increase in dimensionality of the data vectors.

Four sets of experiments were run using the three detection techniques. Two of the four experiments were run using the simple post-processor described in Section 3.2.4.3.1. This post processor simply removes detections that are shorter than a specified threshold. The other two experiments did not use the post-processor. Theses two types of experiments were further subdivided based on how the probability of detection and false alarm rate were defined.

The first method of defining the above statistics was the same as the method presented for the single line case; that is, on a pixel by pixel basis. In this case, the probability of detection can be defined as

$$P_{detect} = \frac{T_{target}}{T_{true}}$$

where T_{target} is the number of target pixels classified correctly and T_{true} is the total number of target pixels present in the ground-truth image. Likewise, the false alarm rate can be defined as

$$P_{false} = \frac{B_{target}}{B_{true}}$$

where B_{target} is the number of background pixels classified as target pixels and B_{true} is the total number of background pixels present in the ground-truth image. We shall call this method of defining the statistics the *single-pixel method.*

The second method of defining the statistics is based on calling any group of connected target pixels a single detection. (a group of pixels on the same scan line are connected if all of them are labeled as target pixels and each of them is adjacent to at least one other pixel in the group.) If any pixel of a detection lies on the target then the target is labeled as found and the detection is said to be correct. In this method, the probability of detection can be defined as

$$P_{detect} = \frac{T_{target}}{T_{true}}$$

where T_{target} is the number of target detections touching the target and T_{true} is the total number of target stripes present in the ground-truth image. Likewise, the false alarm rate can be defined as

$$P_{false} = \frac{D_{background}}{D_{total}}$$

where $D_{background}$ is the number of detections that do not touch a target stripe, and D_{total} is the total number of detections found in the image. We shall call this method of defining the statistics the *multi-pixel method*. Note that there is a fundamental difference in how the false alarm rate is defined in the two methods. In the single pixel method, the false alarm rate is defined to be the percentage of misclassified background pixels. In the multi-pixel method, the false alarm rate is defined to be the percentage of misclassified target detections.

3.2.4.4.2. Experimental Results

Figures 3.2.15 and 3.2.16 show graphs from the experiment with no post processing using the single-pixel method of generating the statistics. These graphs are shown so an idea of how the different techniques compare can be gathered without any masking effects of the postprocessor being present. Note that P_{detect} for all three techniques is constant at about 70% independent of the number of scan lines used. However, the false alarm rate varies significantly depending on the technique and the number of scan lines used. In this case, the baseline technique gave the lowest false alarm rate followed by the first and second extensions in that order. Please note that the statistics yielded by all three detection techniques will be equivalent when only one scan line is used because both extensions reduce to the single-scan line detector in this degenerate case. Figures 3.2.17 and 3.2.18 are graphs of the statistics for the same experiment using the multi-pixel detection criterion. Figure 3.2.17 demonstrates that increasing the number of scan lines used for detection did not change the probability of detection for the two extended techniques but did decrease the probability of detection for the baseline experiment. This decrease is thought to be due to the decrease in the total number of detections that occurred in the base line case as the number of scan lines were increased. Note that Figure 3.2.18 shows that about 98% of all detections are false alarms. It also demonstrates that changing the number of scan lines used had little affect on the false alarm rate in all three techniques.

Figures 3.2.19 - 3.2.22 show how detection is affected when a simple post-processor is used to remove some false alarms. Figures 3.2.19 and 3.2.20 show the graphs of the single-pixel statistics when the postprocessor was used. These graphs show that the post-processor reduced both P_{detect} and P_{false}. They also show that almost no gains in

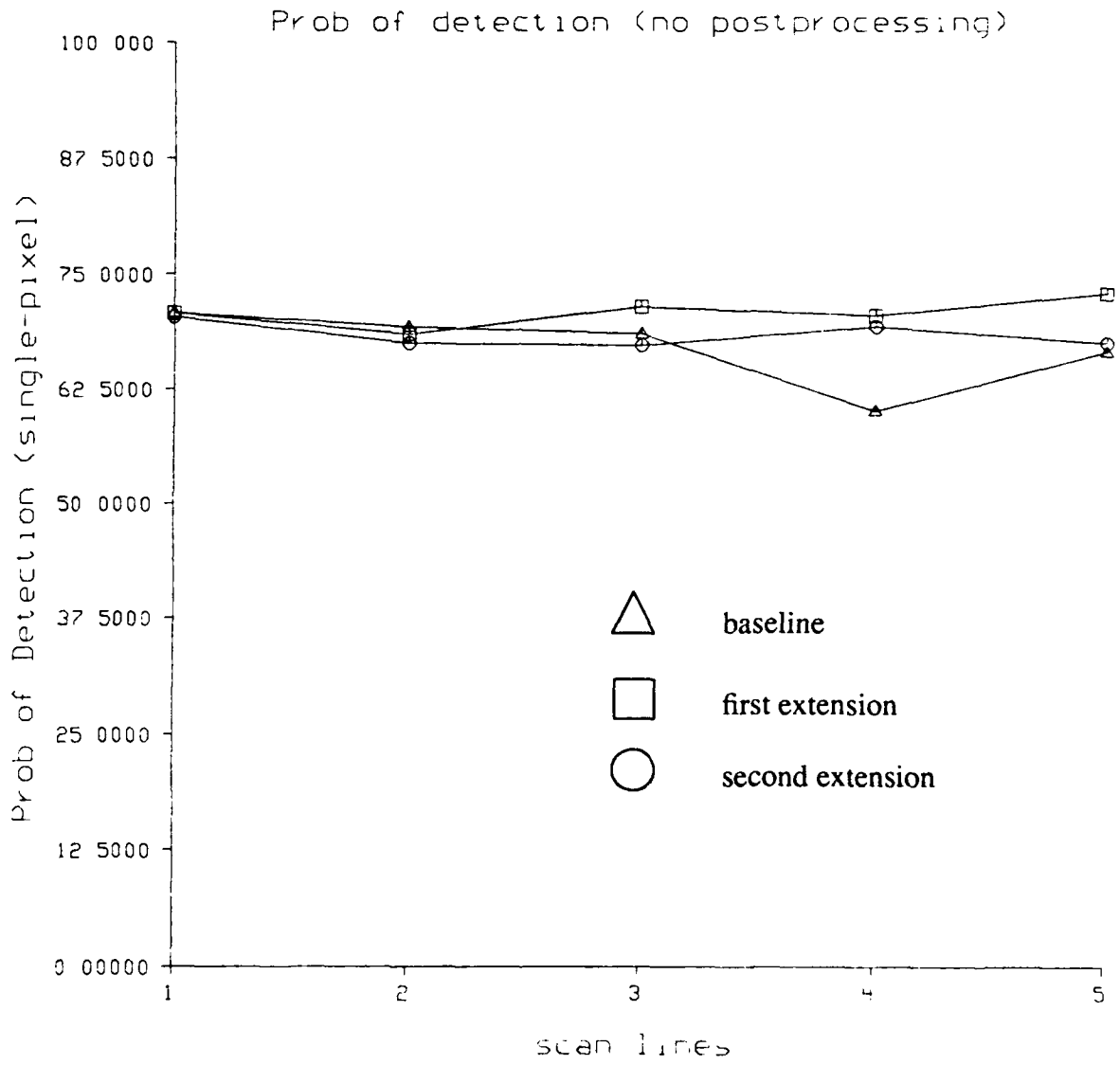


Figure 3.2.15 Probability of detection vs. number of scan lines using single-pixel method and no postprocessing.

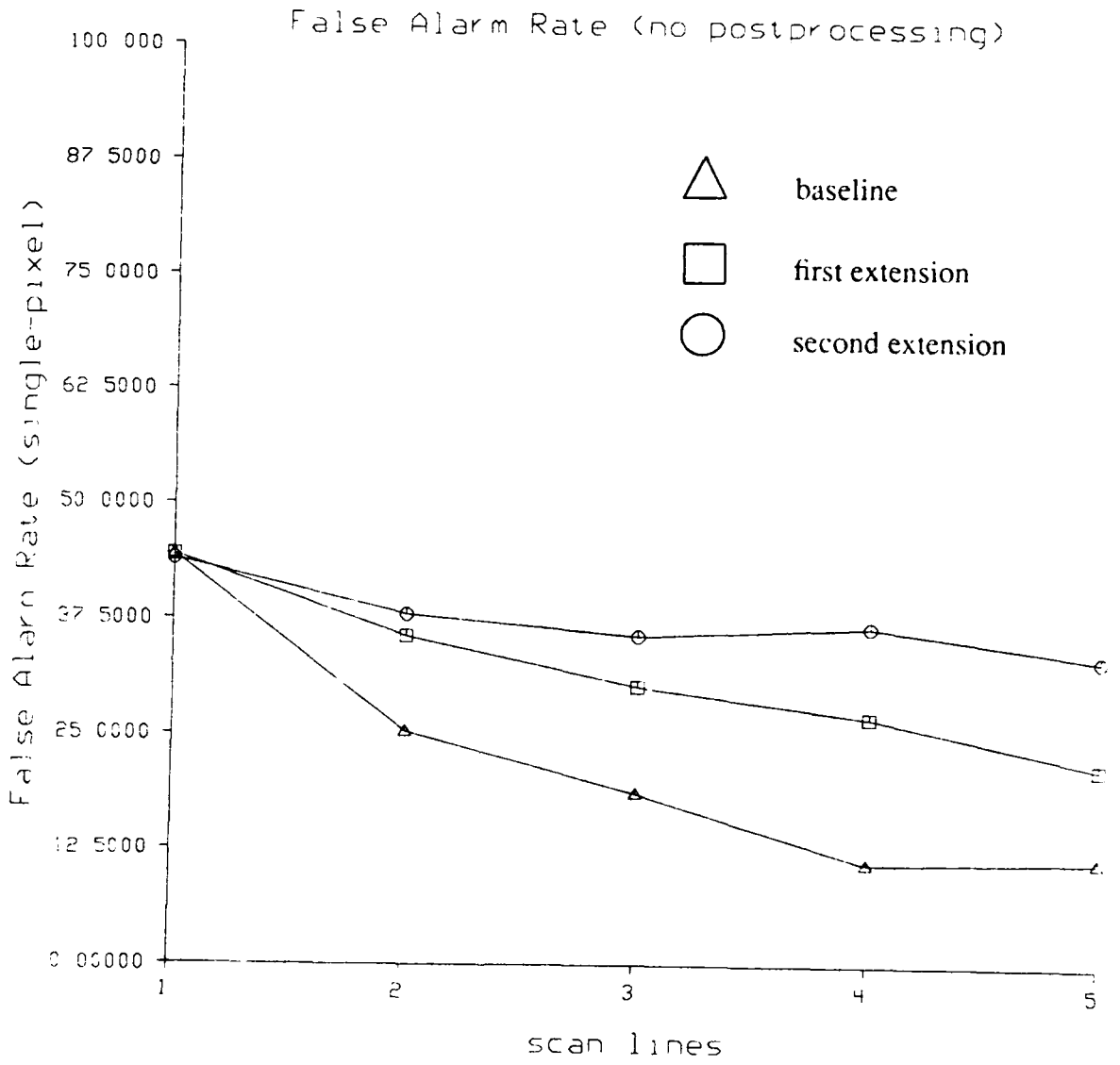


Figure 3.2.16 False alarm rate vs. number of scan lines using single-pixel method and no post-processing.

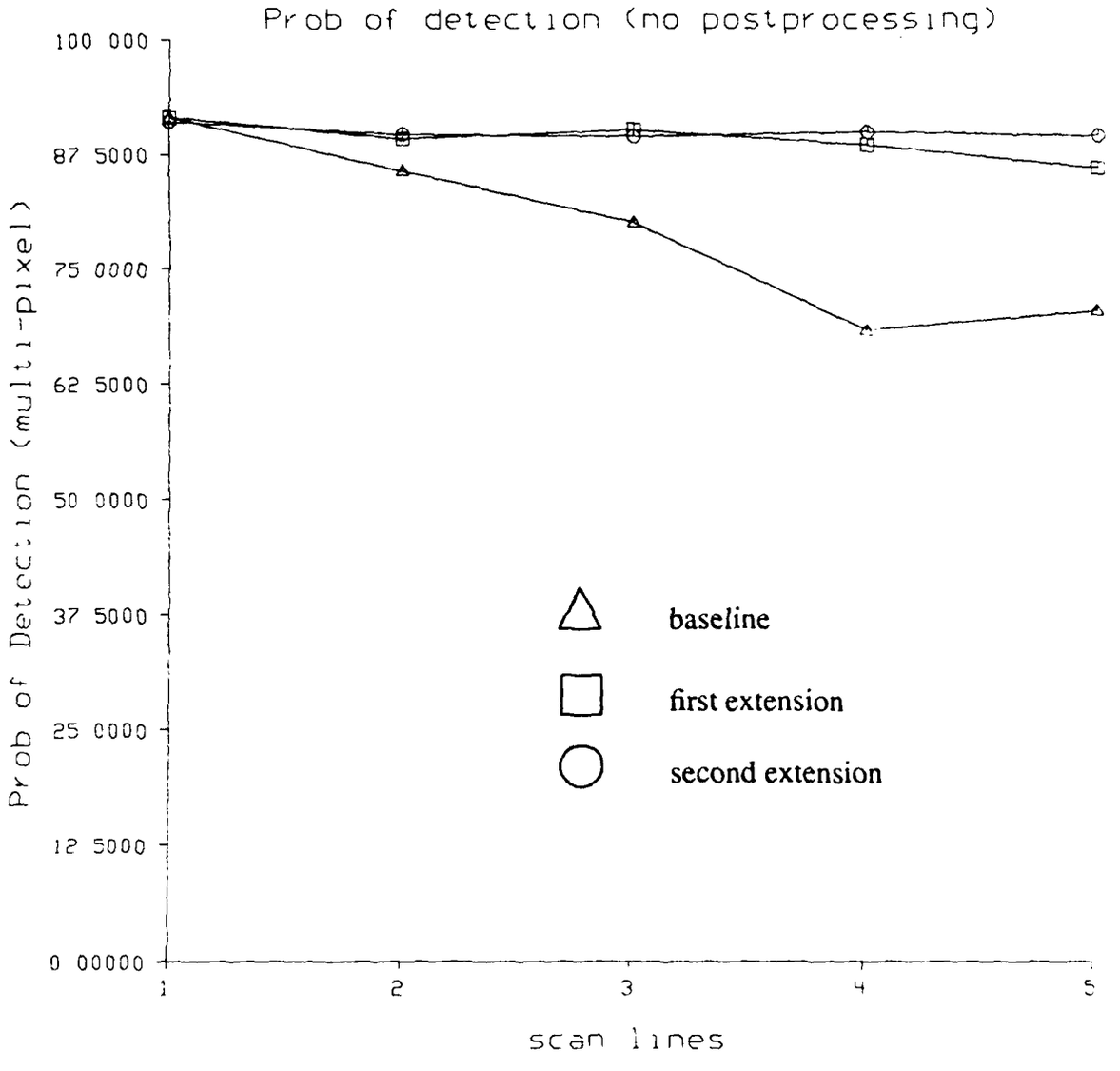


Figure 3.2.17 Probability of detection vs. number of scan lines using multi-pixel method and no postprocessing.

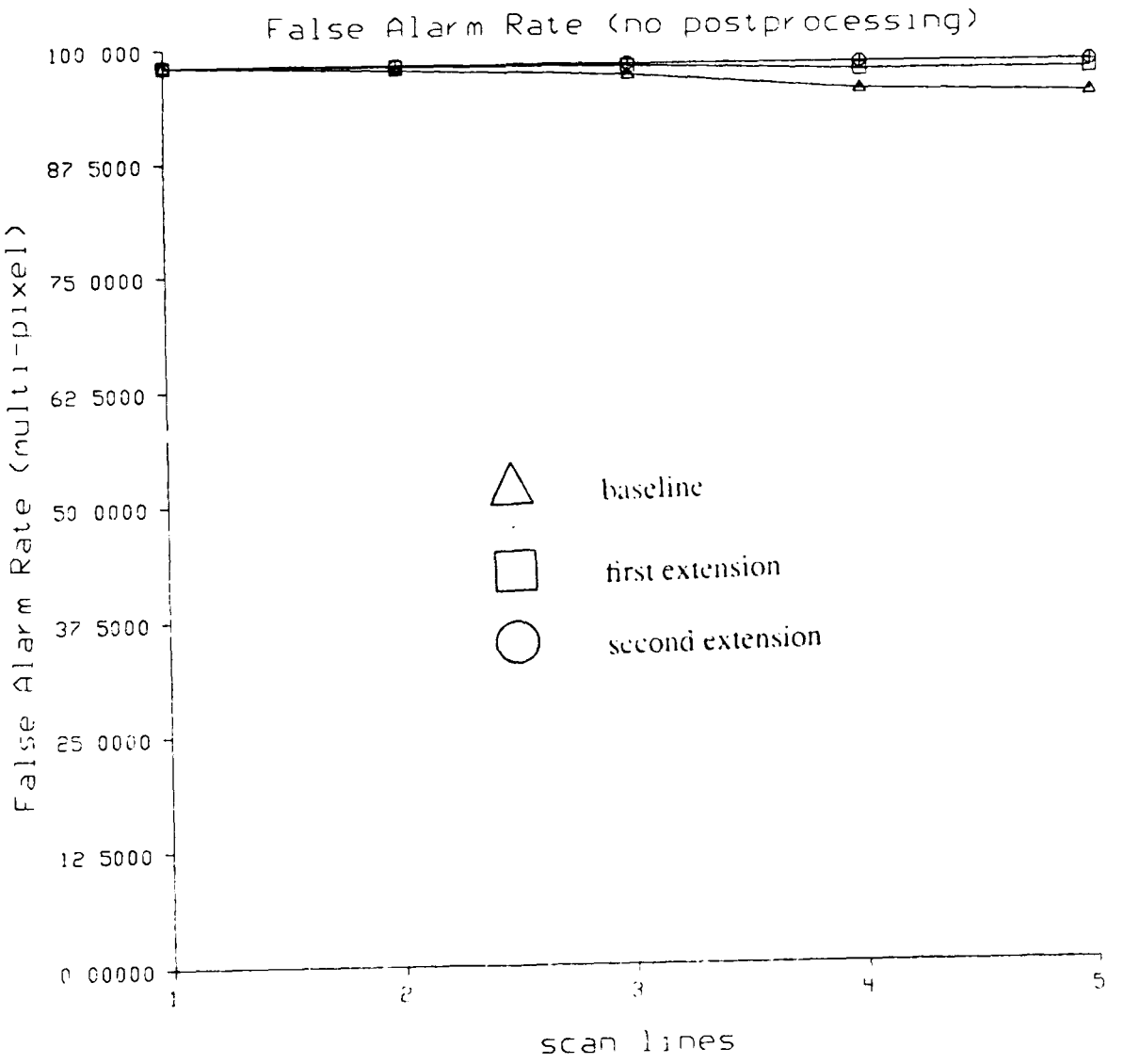


Figure 3.2.18 False alarm rate vs. number of scan lines using multi-pixel method and no post-processing.

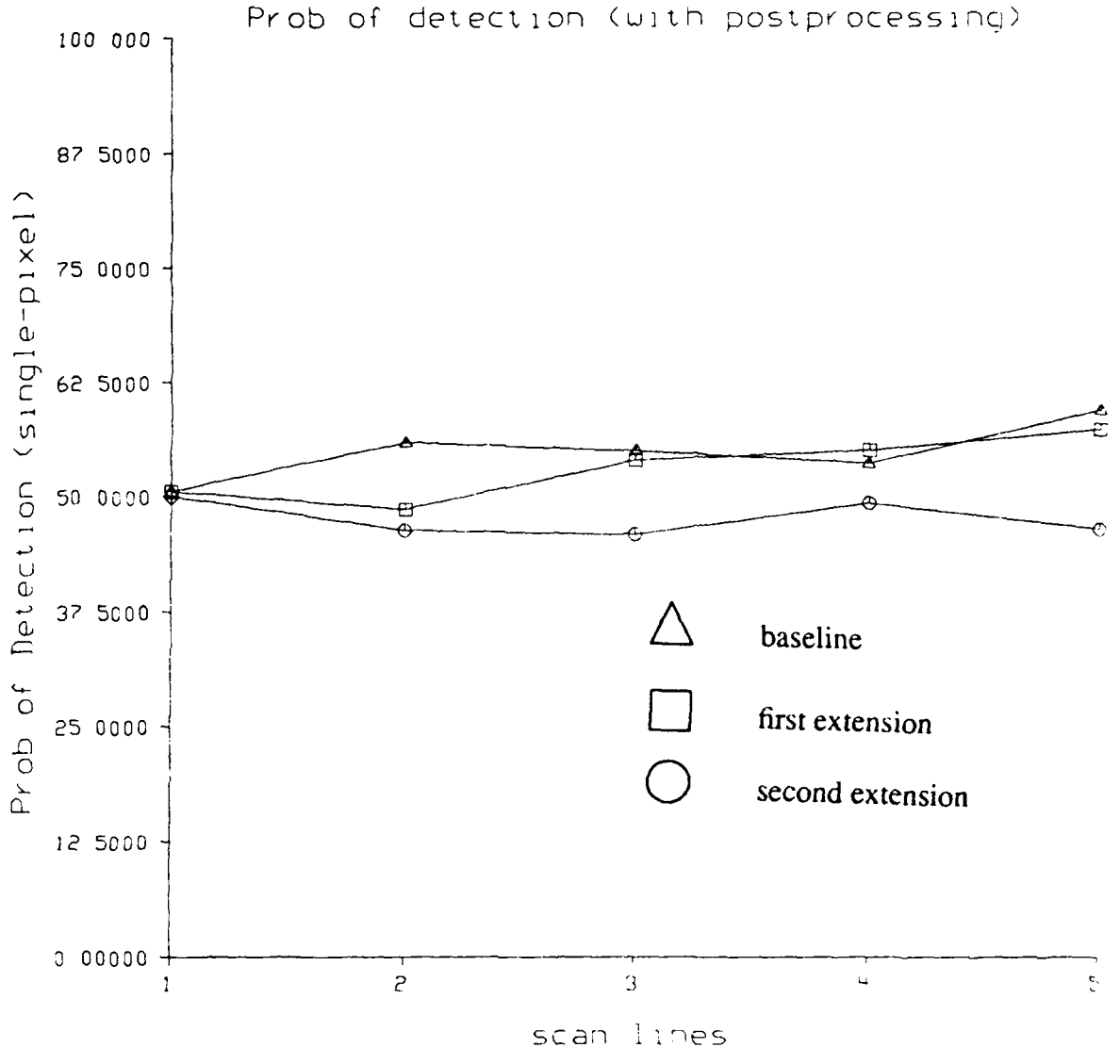


Figure 3.2.19 Probability of detection vs. number of scan lines using single-pixel method and with postprocessing.

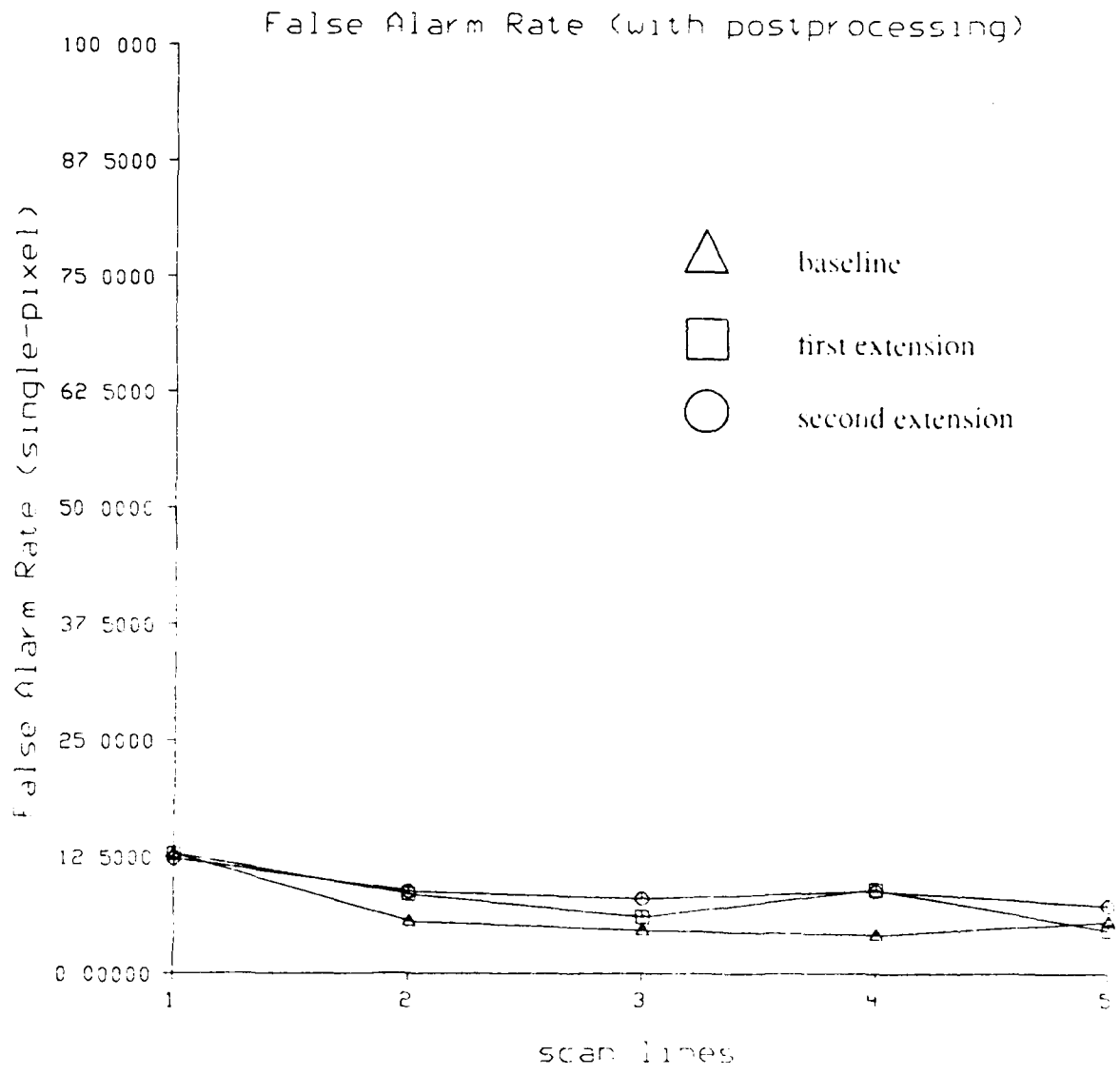


Figure 3.2.20 False alarm rate vs. number of scan lines using single-pixel method and with postprocessing.

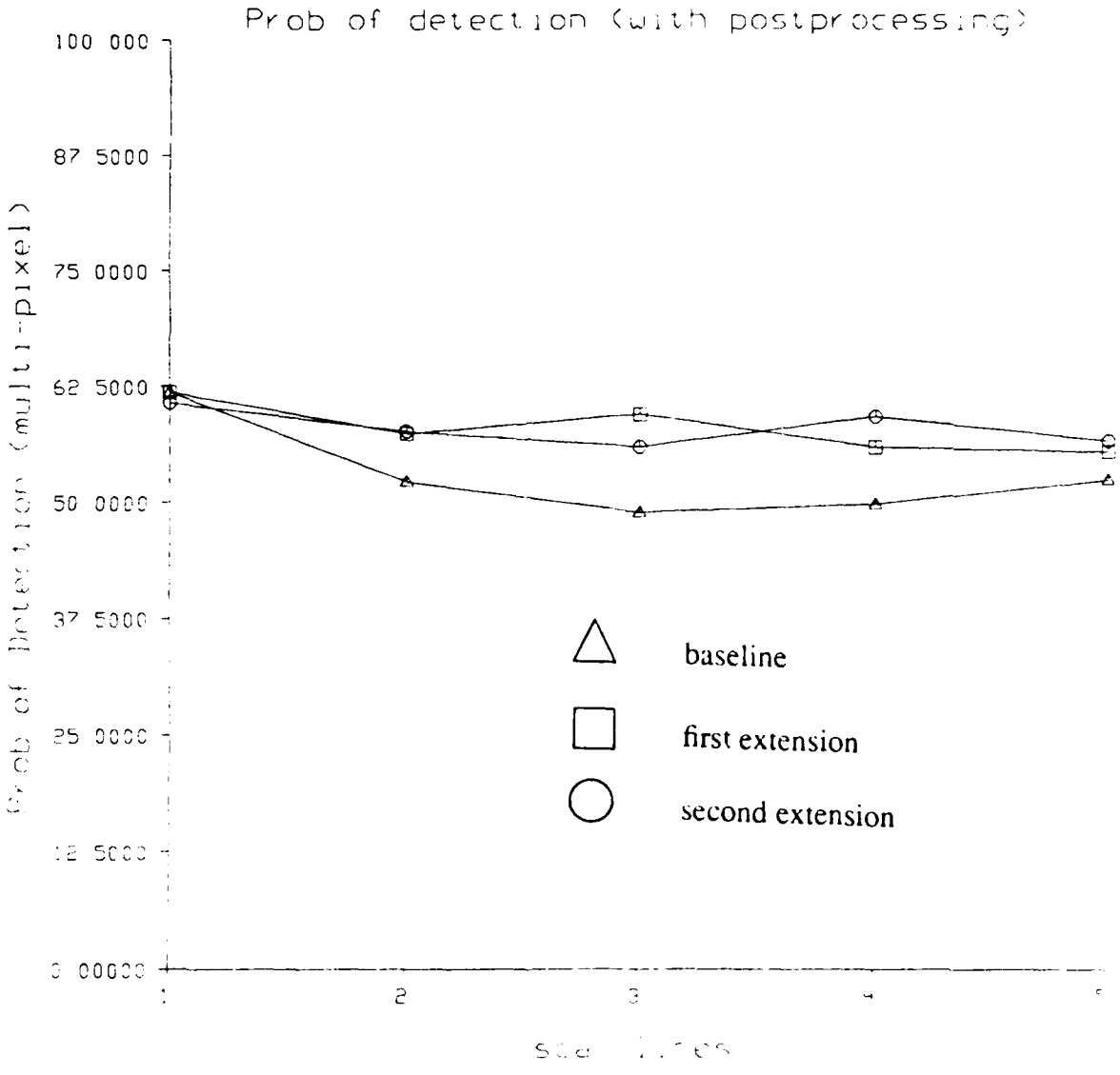


Figure 3.2.21 Probability of detection vs. number of scan lines using multi-pixel method and with postprocessing.

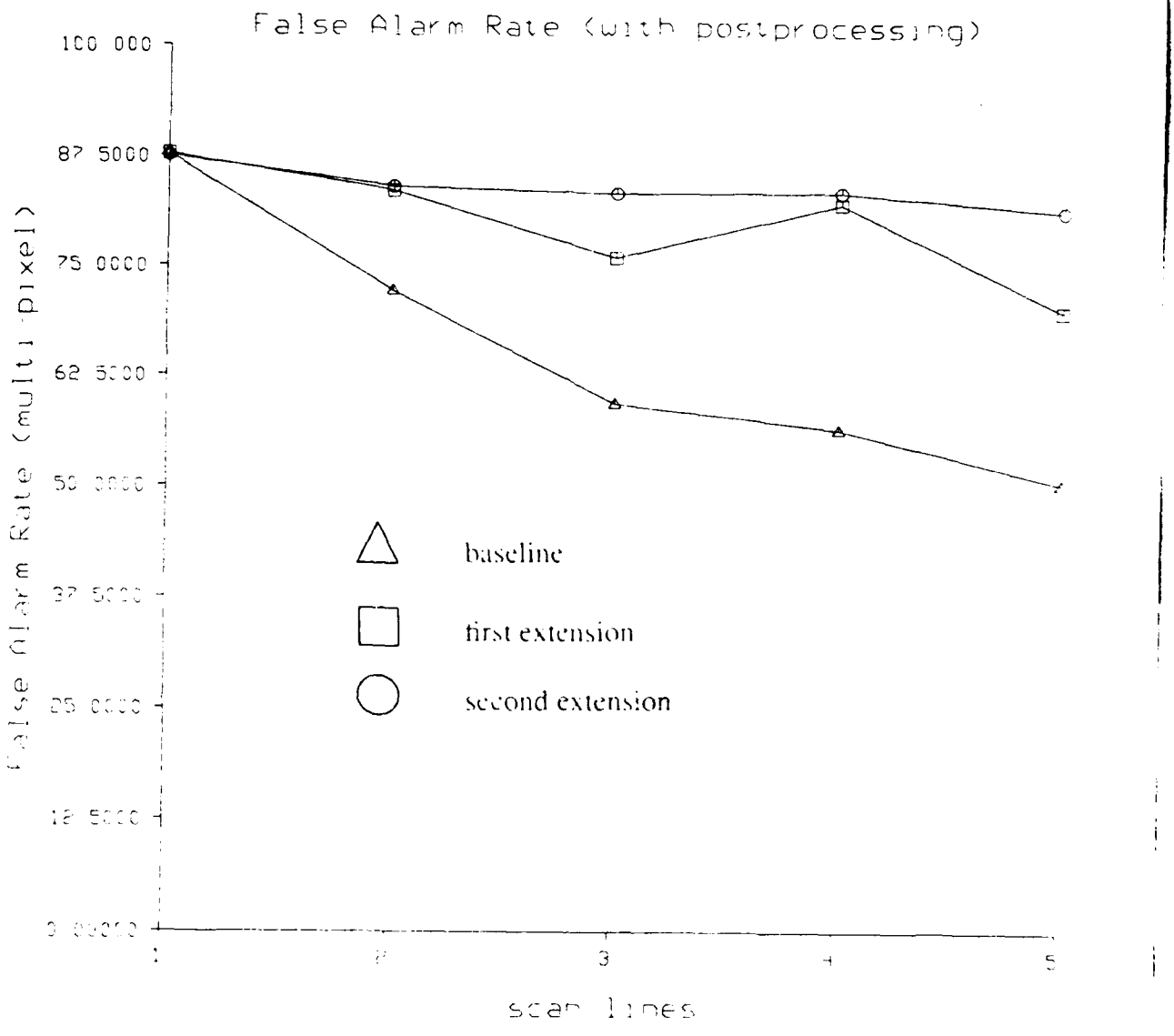


Figure 3.2.22 False alarm rate vs. number of scan lines using multi-pixel method and with rcp postprocessing.

performance are realized by increasing the number of scan lines when the post-processor was used. Finally, Figures 3.2.21 and 3.2.22 show the same experiment (with postprocessor) using the multi-pixel statistic generation technique. Both P_{detect} and P_{false} were reduced in this experiment also. However, The number of scan lines did affect the false alarm rate to different degrees for the various techniques. In this experiment, the baseline experiment benefited the most by increasing the number of scan lines used. The second extension also improved slightly with an increasing number of scan lines. The number of scan lines used had no affect on the performance of the first extended detector.

3.2.4.4.3. Discussion and Future Work

The detection results from both techniques were disappointing. Neither technique showed any significant improvement when the number of scan lines used was increased. Furthermore, if either technique did show some improvement with an increasing number of scan lines, the baseline experiment usually showed greater improvement. It is hoped that some other techniques can be used to extend the single line detector to show improved performance with multiple scan lines.

3.2.4.5. Target Detection Using Multiple Lines of LADAR Data – Approach 2

The single-line detector described here can be extended to detect targets using multiple lines of LADAR data if the value for each pixel is replaced by its log-likelihood ratio. The extension is achieved by combining the log-likelihood ratios for multiple lines produced by the single-line detector into a single value for each pixel. One method for combining the log-likelihood values from a single-line detectors, based on a **weighted-voting** scheme, was first described in [KaYo87]; an extension of that method, based on **meta-classifiers**, is used to combine the single-line values in this report. This combination technique is statistically based and mirrors the classification process performed by the single-line detector.

To combine the log-likelihood values from L scan lines, the log-likelihood values for the pixels undergoing combination are first grouped into a single L-dimensional data-vector. The data-vectors are built by sampling the values in the log-likelihood image perpendicular to the scanning direction (the values are sampled perpendicularly so that they all come from different scan lines). As in the single-line case, the data-vectors are assumed to be samples from one of two probability density functions, one for the target and one for the background. We can represent the log-likelihood density function for the target as

$$f'_{target}(l_1, l_2, \ldots, l_L)$$

and the background log-likelihood density function as

$$f'_{background}(l_1, l_2, \ldots, l_L).$$

Using the process described in previous section, we can compute

$$f'_{target}(l_1, l_2, \ldots, l_L)$$

and

$$f'_{background}(l_1, l_2, \ldots, l_L)$$

for each sample. The pixel can then be said to belong to an object if

$$f'_{target}(l_1, l_2, \ldots, l_L) > C' \times f'_{background}(l_1, l_2, \ldots, l_L)$$

where, once again, C' is equivalent to a threshold. If the density functions are assumed to be Gaussian then the pixel can be classified as belonging to part of a target if

$$(X-M'_b)^T \Sigma'^{-1}_b (X-M'_b) - (X-M'_t)^T \Sigma'^{-1}_t (X-M'_t) + \ln\left\{\frac{|\Sigma'_b|}{|\Sigma'_t|}\right\} < 2ln(C) \quad (3.2)$$

Where M'_t, Σ'_t, M'_b, Σ'_b are the mean vectors and covariance matrices for the log-likelihood target and background densities, respectively. Note that inequality (3.2) is the functional expression of a quadratic classifier. Once again, if the pixel values are set to the value produced by the left hand side of the inequality (3.2), then the optimal value for C' can be found by thresholding the image so formed.

It is informative to compare the multi-line detection scheme described in this report with the techniques for multiline detection discussed in [KaYo87]. The first technique presented in that report did not break the multi-line detection process into the two step process used here. Instead, it increased the dimensionality of the sample vector to include more data from the multiple scan lines. Using this scheme, if the dimension of the data vector for each line is M and L scan lines are used in the detection process, then the dimensionality of the multiline data vector would be $Dim = M \times L$. Obviously, this process can become computationally infeasible very quickly as the single-line dimensionality or the number of scan lines increases.

The second extension to multiline detection presented in that report used a two-part process similar to the one described here. However, instead of using a quadratic classifier to combine the output of the scan lines, as the current method does, the previous scheme added the log-likelihood values for the scan lines being combined. If the sum of the values was less than zero, the detector would indicate the presence of a target. This combination process was viewed as a weighted-voting scheme where the weights of the votes were proportional to the decision's confidence. To relate this process to the current combination scheme, it should be noted that the summation process can be viewed as classifying the feature vectors (composed of the log-likelihood values from the single-line detector) with a linear classifier. It is possible to view the process as a linear classification because the summation can be expressed as $S_{total} = W^t X$ where W^t is a M dimensional row vector

whose elements are all 1's and X is the data vector. Thus, if S_{total} was negative, the detector would assert the presence of a target; otherwise, it would deny a target's presence. Since the linear classifier can be viewed as a special case of the quadratic classifier (a quadratic classifier degenerates to a linear classifier if the covariance matrices of all classes are equal), the current scheme using a quadratic classifier can be viewed as an extension to the previous voting scheme. Note that the current scheme can be viewed as a further generalization of the previous scheme because the previous scheme used a fixed threshold of zero where the current scheme allows an optimal threshold to be set.

Since we are comparing the current experiment with those performed in the past, the differences between the data in the new experiment and the data in the old experiments should be explored. The data obtained for the past experiments was very preliminary and had some unusual properties that could be exploited by target detectors and segmenters. For example, valid data was produced by the sensor for only a few *ambiguity intervals*. If the sensor did not detect any object in this valid range, it would output high variance noise. Thus, the technicians at the data collection sight had to tune the sensor for every target scanned to guarantee that the target would fall within the valid range interval. Although it would seem that the sensor providing random measurements for some pixels would hinder target detection and segmentation, it actually made both processes much easier (the technicians tunning the system can be though of as a preprocessing step with perfect knowledge). Past segmenters and detectors were able to take advantage of the variance of out of band signals by labeling any group of pixels with low variance as a target and all other pixels as background. Since the current data has valid AM data for all pixels, this technique is not available with the current data set.

3.2.4.6. Experimental Procedure

Three sets of experiments were run to determine the performance of the detector when the number of scan lines used in the detection process was varied. A different method of building the data vector was used in each experimental set. The performance of the detector was determined in each experimental set when 1, 2, 3, 4, 5, 10, 15, 20 and 25 scan lines were used in the detection process. Furthermore, the performance of the detector was determined when both horizontal and vertical raster-scanning was used to scan the image.

In the first experimental set, no preprocessing was applied to the image before the data vectors were built. In this experiment, 51 dimensional data vectors were used for the single-line detector when horizontal scanning was used; 38 dimensional data vectors were used for vertical scanning. However, when horizontal scanning was used, only 50 (37 for vertical scanning) of the data vector's values were taken from the LADAR resolved-range image; the final value for the data vectors was set equal to the LADAR's *return amplitude*. The experiment in which no preprocessing was applied to the image will be referred to as

the *raw-data* experiment. Some preprocessing was applied to the data in the second experimental set. The pixel values in this set were set equal to the resolved-range value modulo 1875. This process was meant to recover, to the greatest extent possible, the raw AM LADAR data. Besides this preprocessing, this experiment was equivalent to the first. That is, the data vectors were 51 (38) dimensional with 50 (37) values representing the resolved-range and one value representing the return-amplitude. This experiment will be referred to as the *pseudo-AM* experiment. The data vectors used in the final experiment were built differently then those used in the first two experiments. This experiment was designed to mimic the variance-based segmenters that have been applied to previous LADAR data sets. To mimic these segmenters, the pixel values were set to the resolved-range modulo 1875 as in the pseudo-AM experiment. However, the data vectors were 5 dimensional in this experiment and were built from the values of the pixel and its two immediate neighbors on each side. It was hoped that the pseudo-AM values for the on-target pixels would have a lower variance than those off-target; the detector should then be able to distinguish between the on-target and off-target pixels via their variances. This experiment will be referred to as the *variance* experiment.

In both the single-line and multi-line detection procedures, a simple post-processing scheme was applied to the image after the pixels were classified by the detector. This post-processing scheme deleted all detections that were only a single pixel long in order to lower the detector's false alarm rate.

In the following experiments, any connected group of pixels classified as belonging to a target is considered to be a single detection. (A group of pixels on the same scan line are connected if all of them are labeled as target pixels and each of them is adjacent to at least one other pixel in the group.) If any pixel of a detection touches the target then the target is labeled as found and the detection is said to be correct. If more than one detection touches a single target stripe, then the target stripe is only counted as being detected once. Since the detections are defined in this manner, the probability of detection can be defined as

$$P_{detect} = \frac{T_{target}}{T_{true}}$$

where T_{target} is the number of target stripes touched by at least one detection and T_{true} is the total number of target stripes present in the ground-truth image. Likewise, the false alarm rate can be defined as

$$P_{false} = \frac{D_{background}}{D_{total}}$$

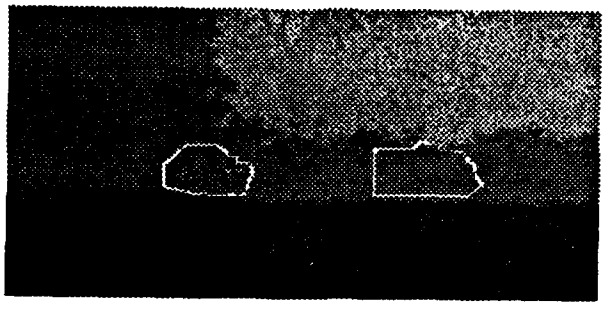
where $D_{background}$ is the number of detections that do not touch a target stripe, and D_{total} is the total number of detections found in the image. Note that this defines the false alarm rate to be the percentage of target detections that are incorrect.

The data used to determine the detector's performance consisted of the four typical LADAR images described in Section 3.1.2; these images are shown in Figure 3.2.23. The targets in the images were hand-segmented to provide ground truth values for the detector training process. The borders of the targets are shown in Figure 3.2.23 as white outlines. Note that the results of the experiments presented here are better than could be expected in the real world because the detector was trained and run on the same data set.

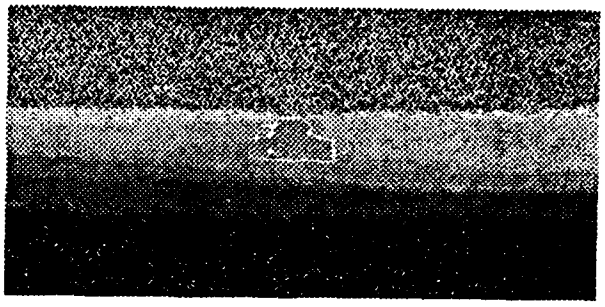
3.2.4.7. Experimental Results

Figures 3.2.24, 3.2.25 and 3.2.26 show the log-likelihood images produced by the single-line detector for the raw, the pseudo-AM and the variance experiments, respectively. Pixels in these images with negative log-likelihood values (shown as the light pixels here) are pixels that the detector hypothesized as belonging to part of a target. As indicated by these images, the single-line detector performed very poorly on the LADAR data. Figures 3.2.27, 3.2.28 and 3.2.29 show the detector statistics for the three experiments; they show the probability of detection and false alarm rates for a range of thresholds. The probability of detection is indicated in these figures with a solid line and the false alarm rate is indicated with a dashed line. Although the three detectors performed poorly, we believe that the statistics reported are biased by the large amount of background area. Since there are about two orders of magnitude more background pixels than target pixels, a small percentage of background pixels misclassified as belonging to a target will result in a large false alarm rate. Note that in each of these experiments, the false alarm rate when vertical scanning was used was slightly lower than when horizontal scanning was used. We believe that this effect is due to the nonhomogeneity of the horizontal scan lines that made it difficult to model the samples with a single density function. When horizontal scanning is used to scan the image, the properties of the scan lines differed significantly with their placement on the images. For example, scan lines near the top of the images consisted almost entirely of "sky" pixels; these pixels typically had very small return-amplitude values and took random range values. Scan lines from the center of the image on down usually produced progressively smaller range values as the position of the scan line moved down the image. Conversely, when vertical scan lines were used to scan the image, all scan lines exhibited the same properties regardless of their position in the image. As the scan moved down the scan line from the top, most scan lines started with "sky" pixels and progressed to valid range pixels. Furthermore, for multi-line detection, since targets usually are wider than the are tall, vertical scan lines can be spaced farther apart than horizontal scan lines can and still provide the same number of scan-lines on target.

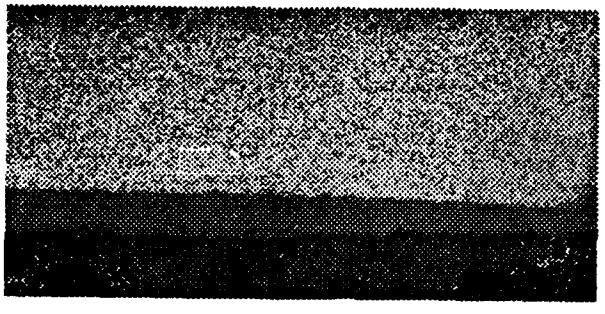
Figures 3.2.30 - 3.2.35 show the probability of detection and false alarm rates for the three experimental sets when multiple scan lines were used to detect the targets. These figures show the detection statistics for a range of thresholds. (The statistics shown in these figures for large thresholds (> 200) is meaningless. If the detection with the



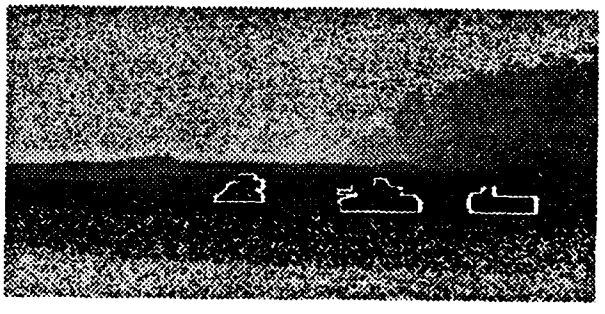
c10m1



c14ta3



c17m1



c17m3

Figure 3.2.23 This figure shows the unprocessed data used in the three experiments. The outlines of the hand-segmented targets used to train the detector are shown in white.

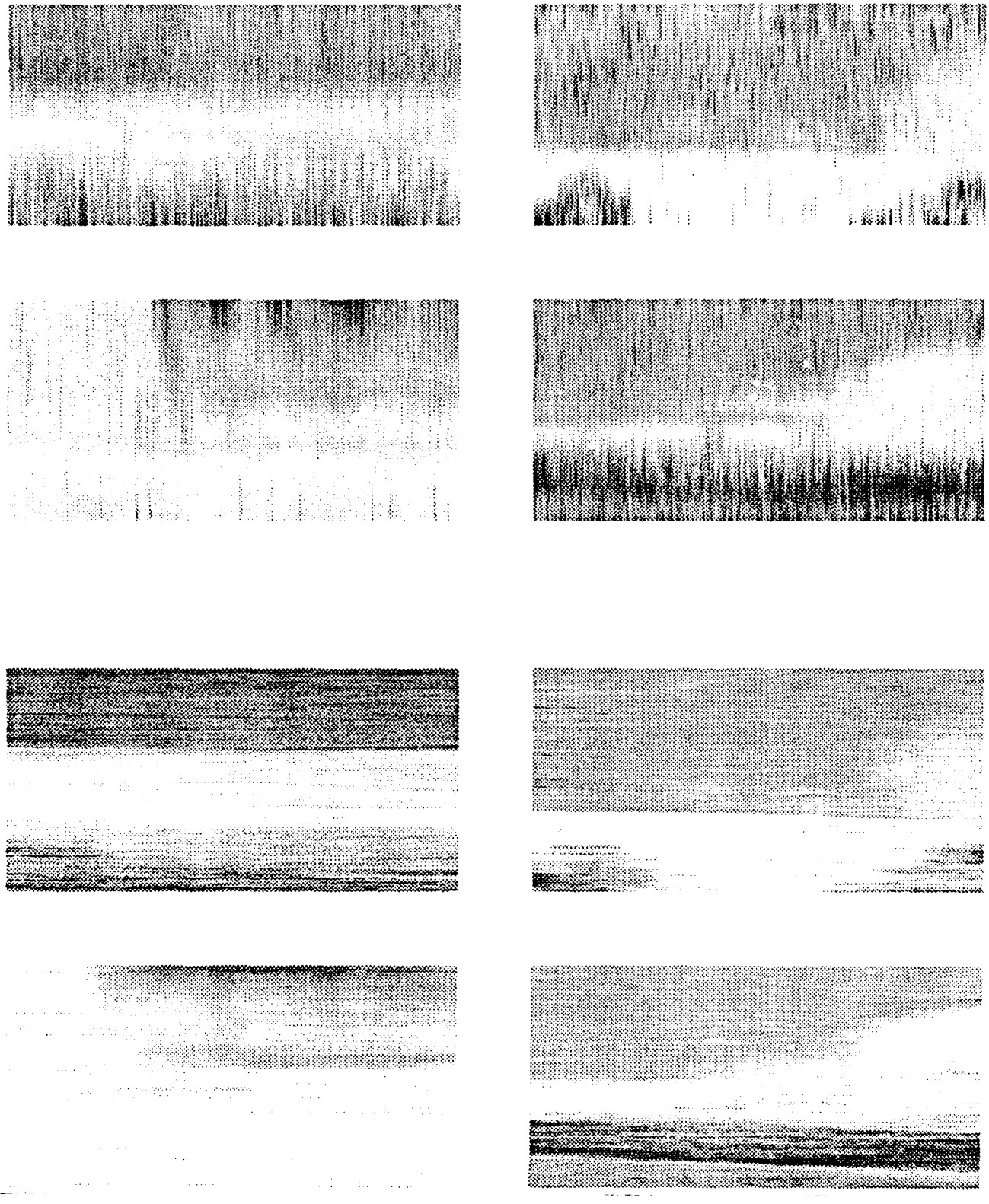


Figure 3.2.24 This figure shows the log-likelihood images produced by the detector when presented with *raw-data*. The top four images are the output when vertical scanning is used and the bottom four show the output when horizontal scanning is used.

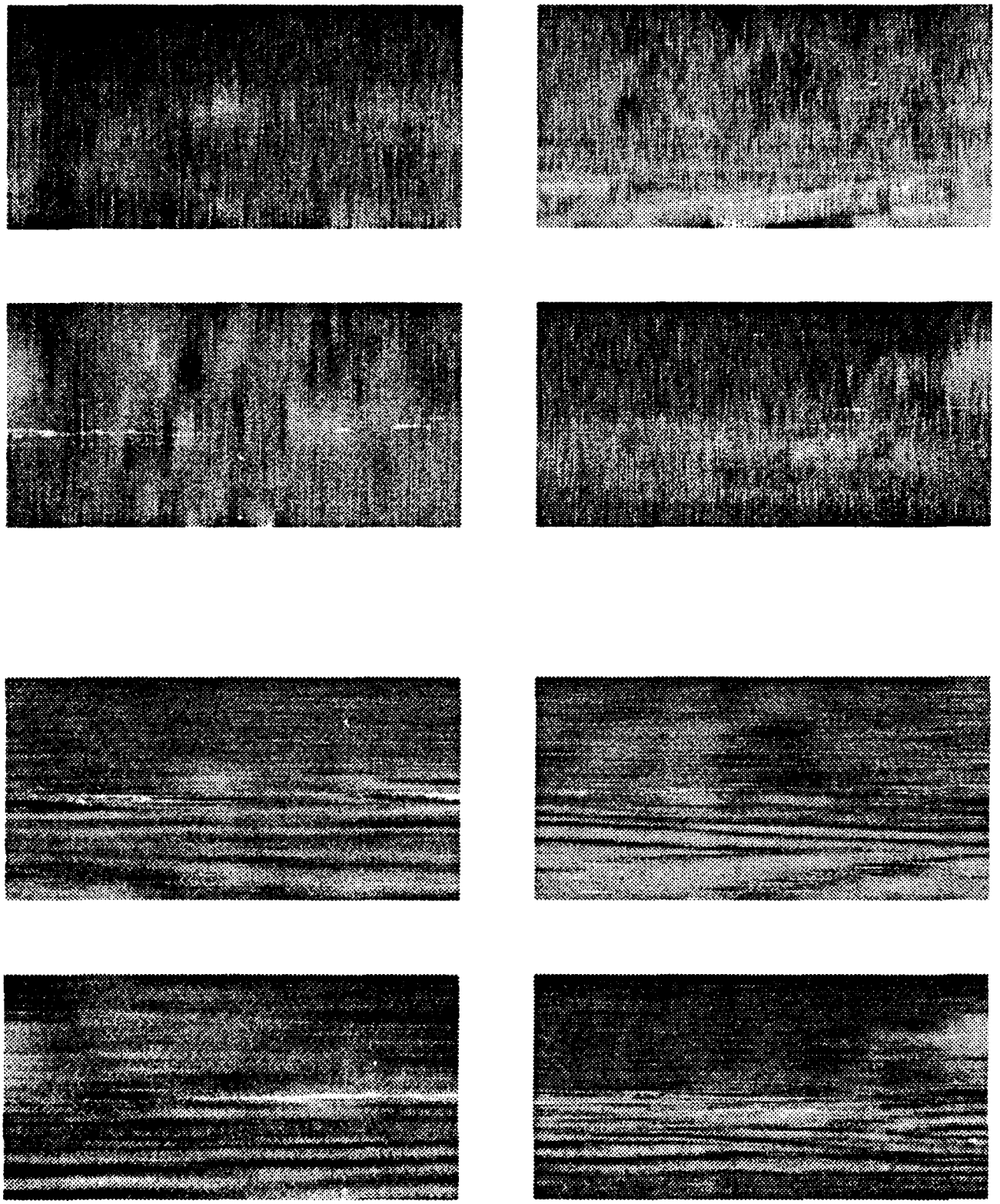


Figure 3.2.25 This figure shows the log-likelihood images produced by the detector when presented with *pseudo-AM* data. The top four images are the output when vertical scanning is used and the bottom four show the output when horizontal scanning is used.

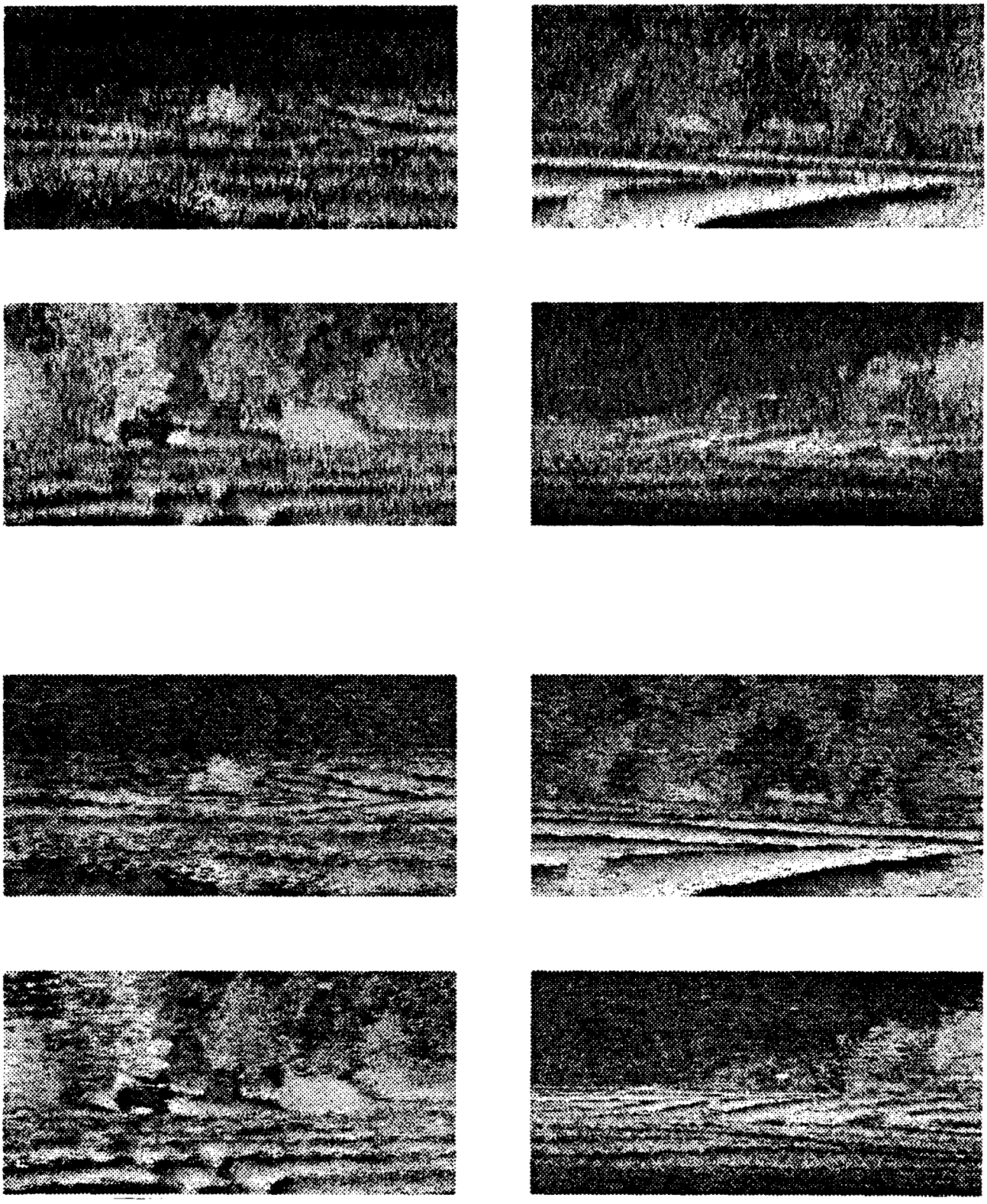
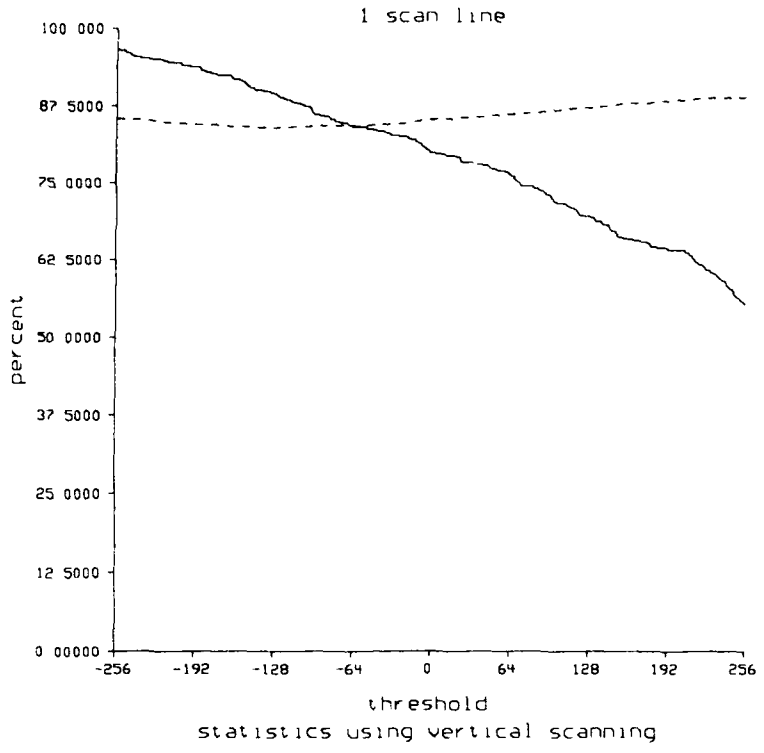


Figure 3.2.26 This figure shows the log-likelihood images produced by the detector when presented with *variance* data. The top four images are the output when vertical scanning is used and the bottom four show the output when horizontal scanning is used.



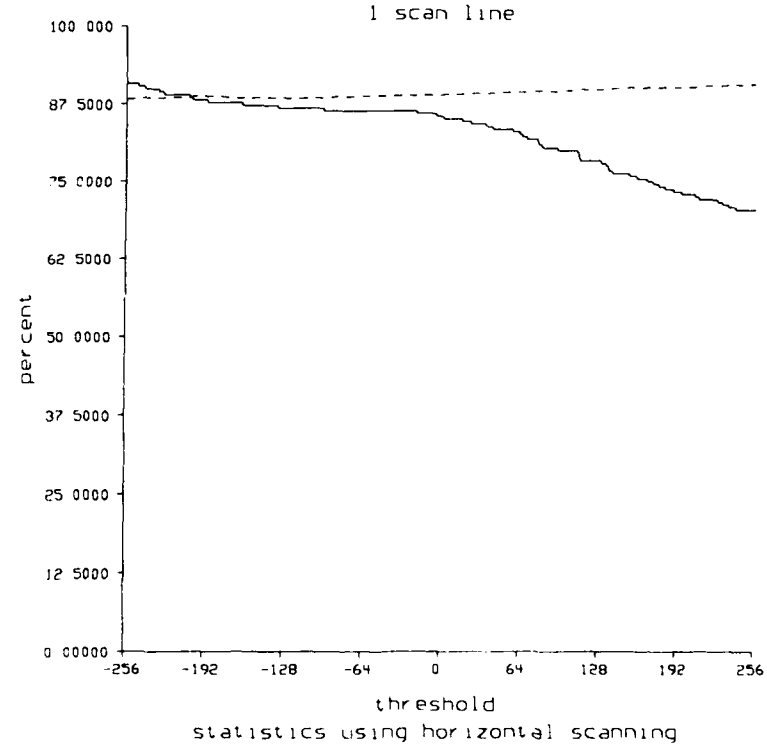


Figure 3.2.27 This figure shows the probability of detection (solid line) and false alarm rates (dashed line) for the *raw-data* experiment using a single scan line. The top and bottom graphs show the statistics for vertical and horizontal scanning, respectively.

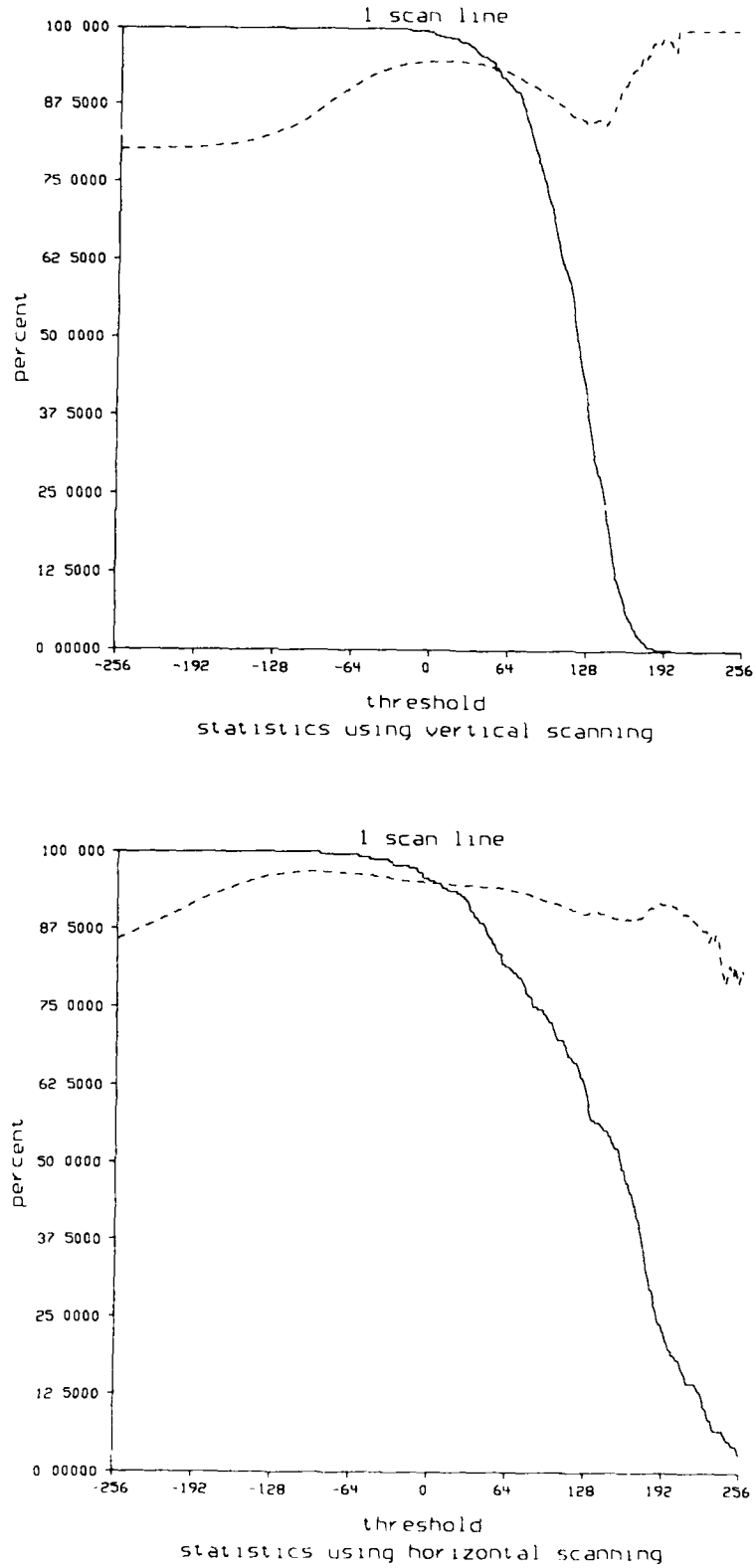
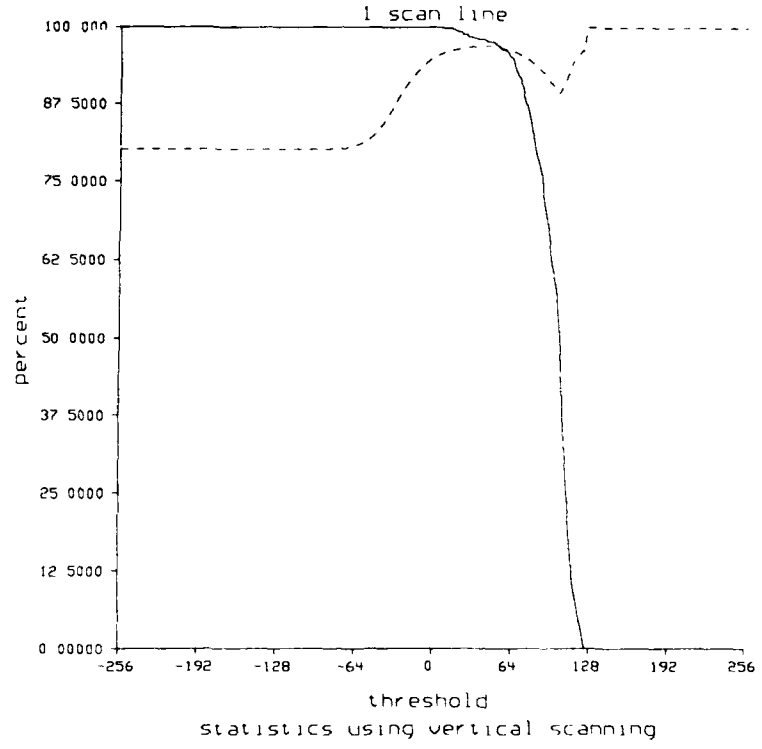


Figure 3.2.28 This figure shows the probability of detection (solid line) and false alarm rates (dashed line) for the *pseudo-AM* experiment using a single scan line. The top and bottom graphs show the statistics for vertical and horizontal scanning, respectively.



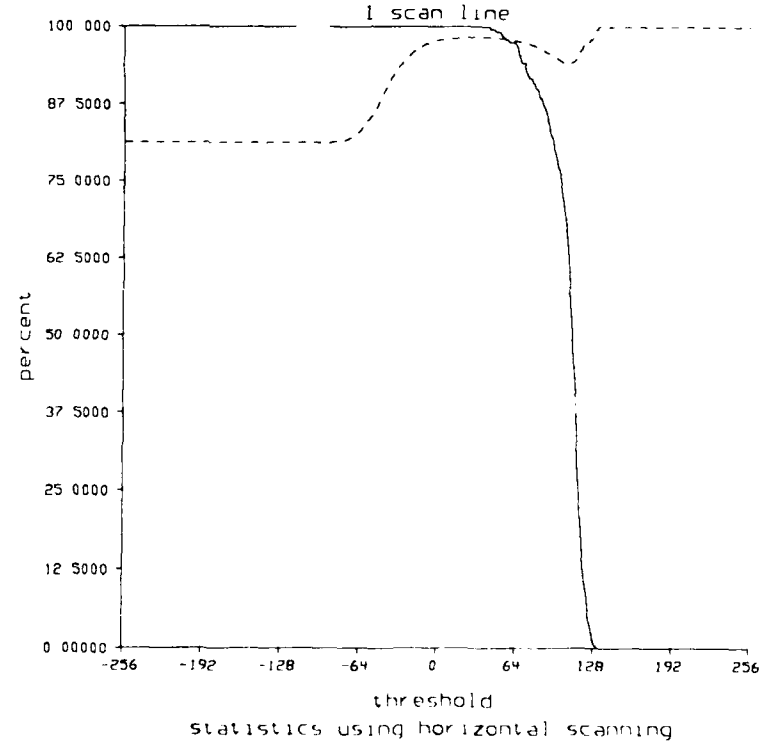


Figure 3.2.29 This figure shows the probability of detection (solid line) and false alarm rates (dashed line) for the *variance* experiment using a single scan line. The top and bottom graphs show the statistics for vertical and horizontal scanning, respectively.

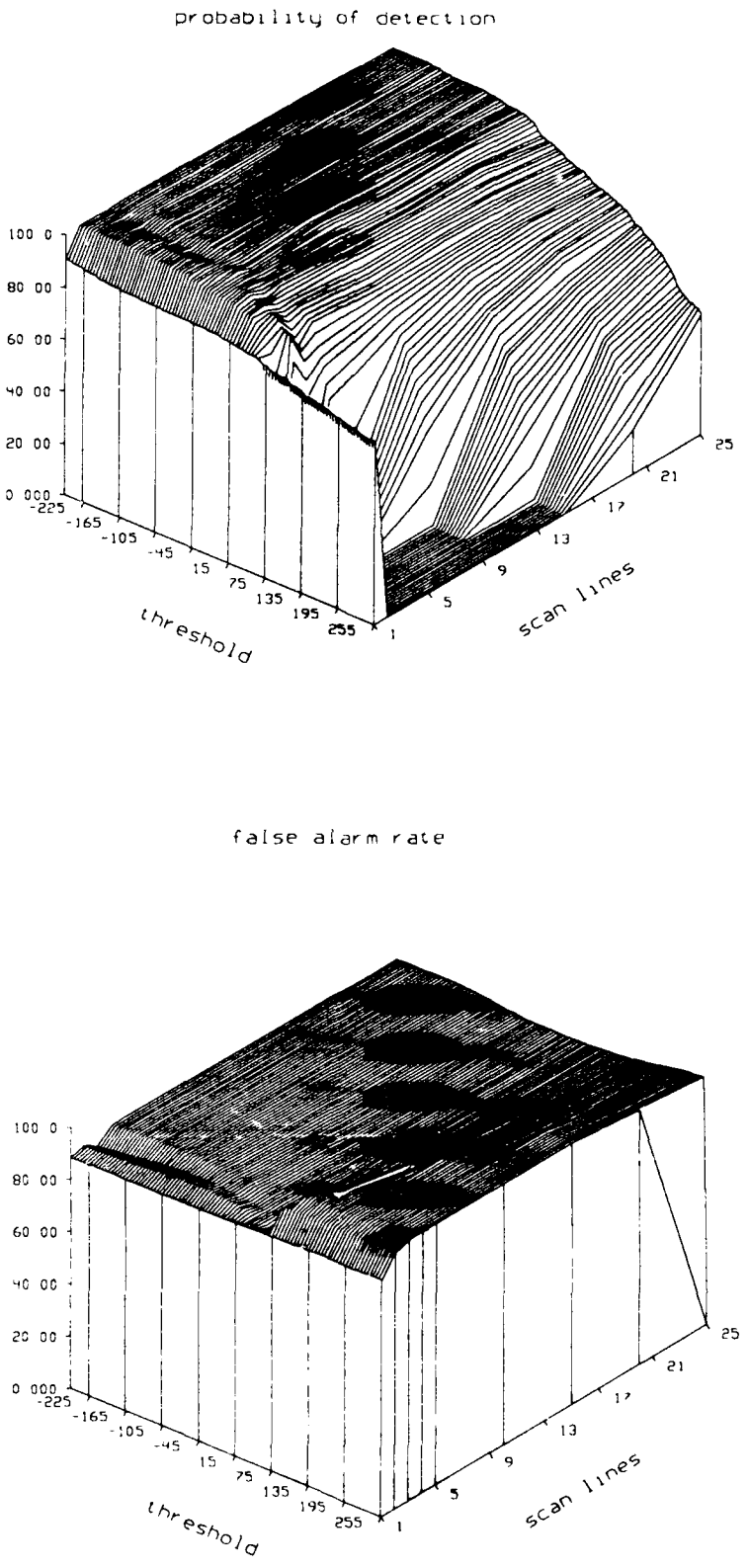


Figure 3.2.30 This figure shows the probability of detection (top graph) and false alarm rates (bottom graph) for the *raw-data* experiment using horizontal scanning. Both graphs show the statistics for a number of scan lines and a range of thresholds.

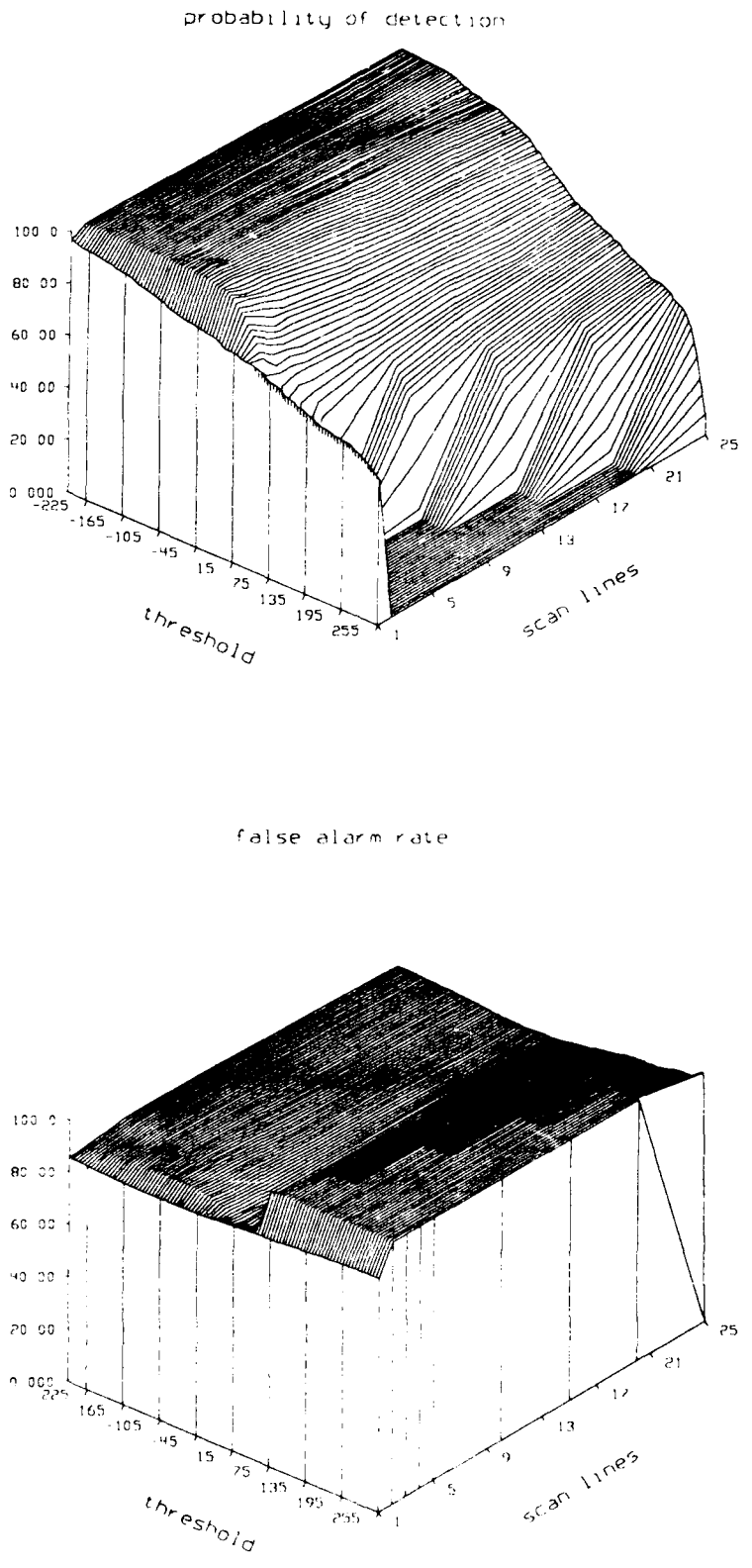


Figure 3.2.31 This figure shows the probability of detection (top graph) and false alarm rates (bottom graph) for the *raw-data* experiment using vertical scanning. Both graphs show the statistics for a number of scan lines and a range of thresholds.

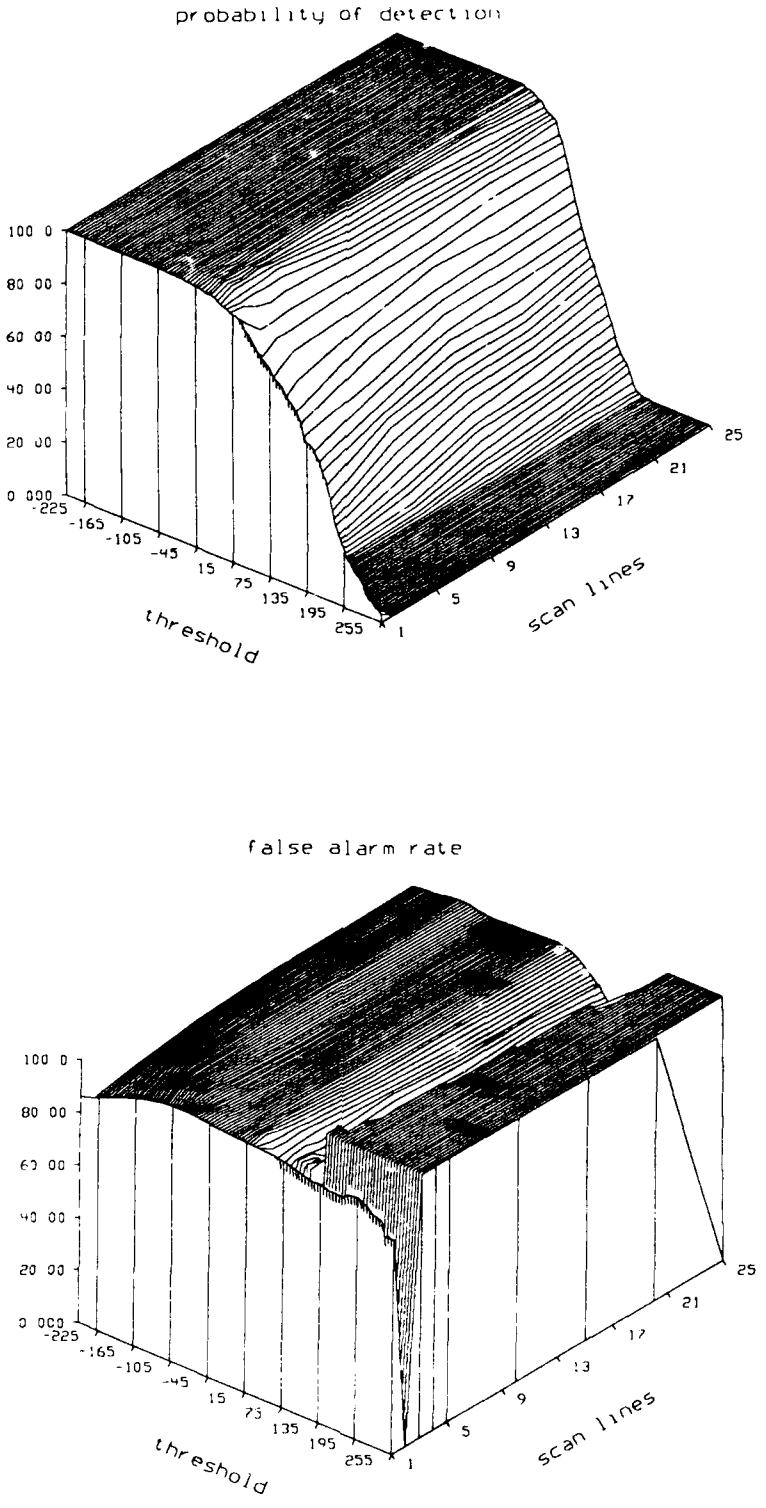


Figure 3.2.32 This figure shows the probability of detection (top graph) and false alarm rates (bottom graph) for the *pseudo-AM* experiment using horizontal scanning. Both graphs show the statistics for a number of scan lines and a range of thresholds.

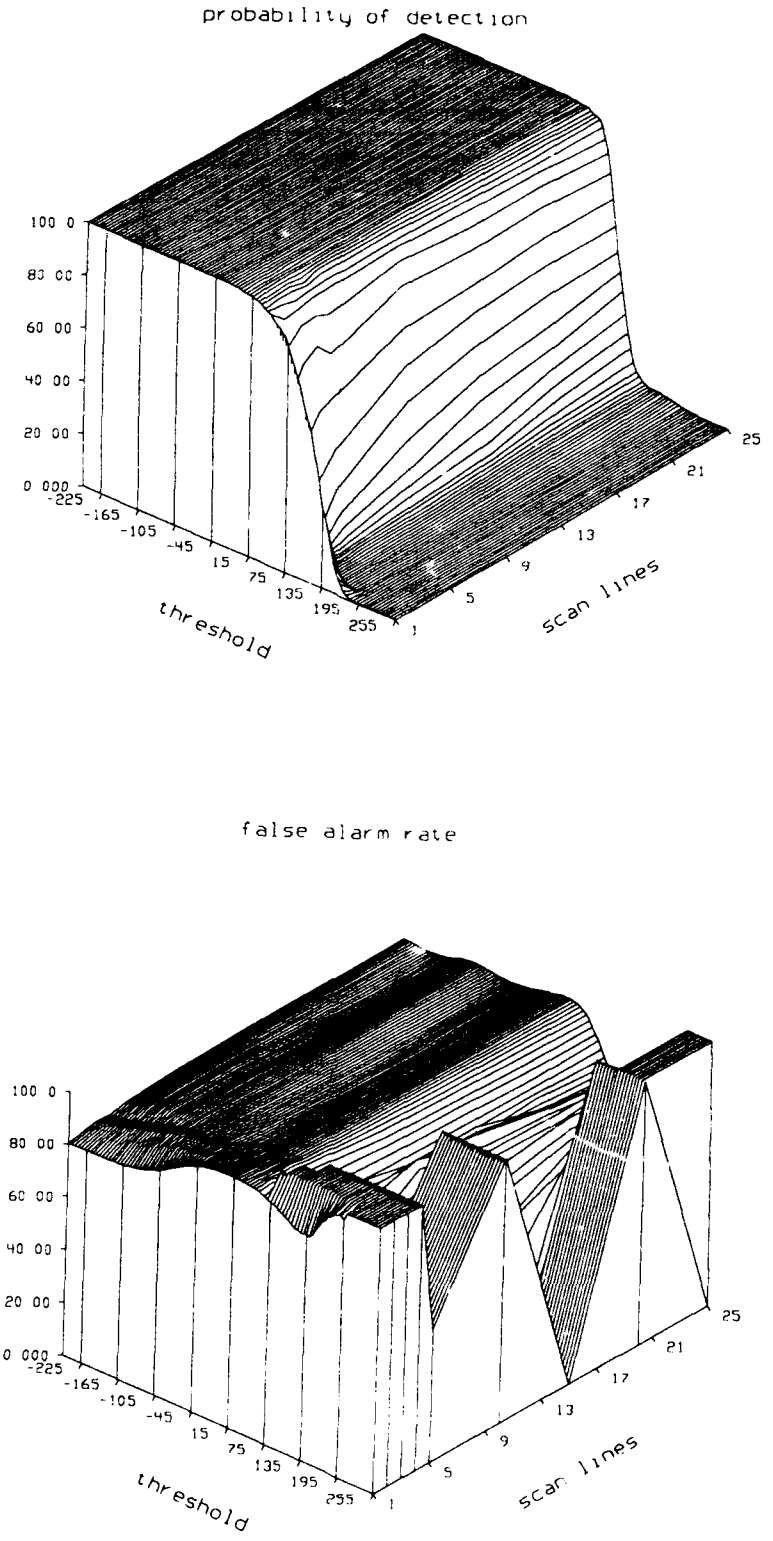


Figure 3.2.33 This figure shows the probability of detection (top graph) and false alarm rates (bottom graph) for the *pseudo-AM* experiment using vertical scanning. Both graphs show the statistics for a number of scan lines and a range of thresholds.

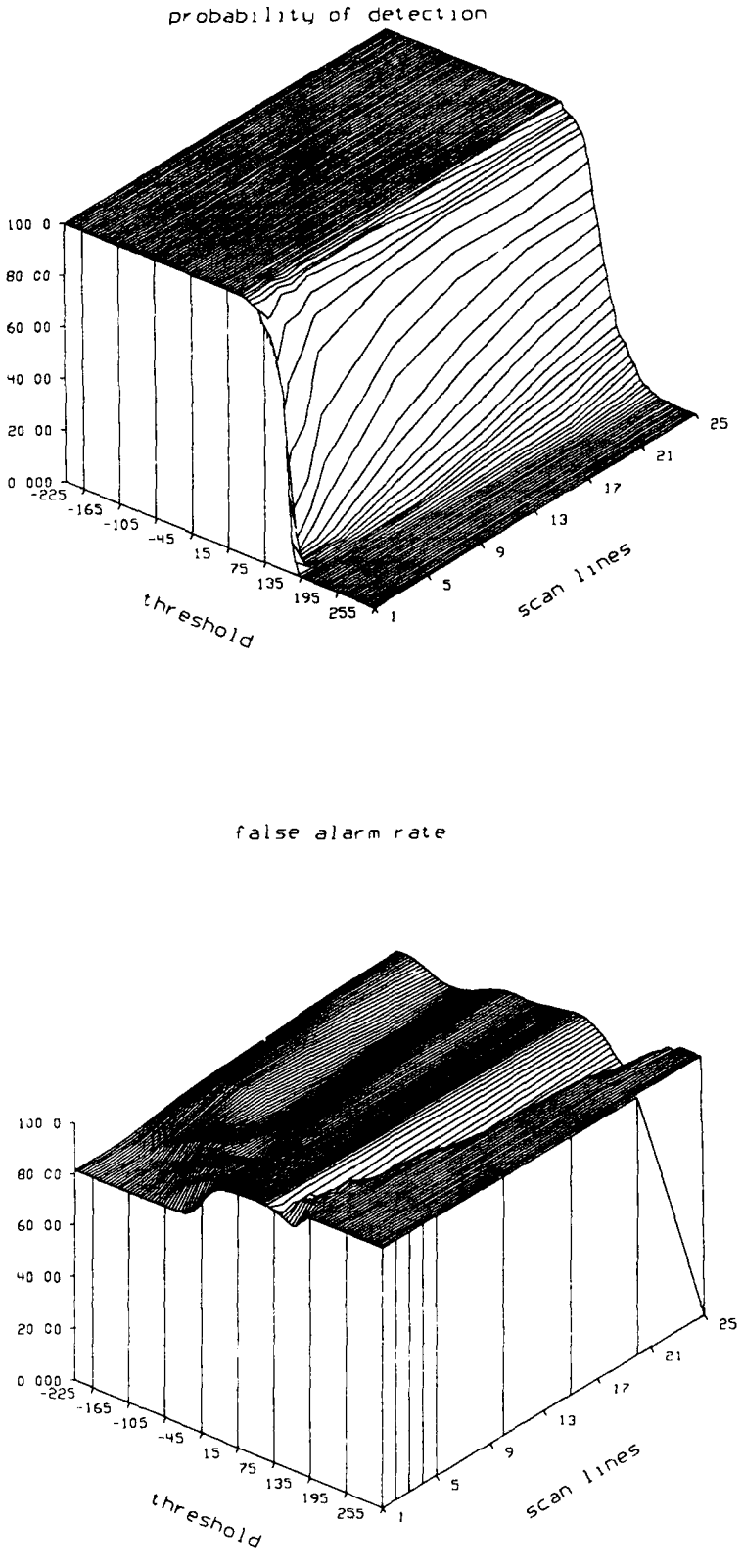


Figure 3.2.34 This figure shows the probability of detection (top graph) and false alarm rates (bottom graph) for the *variance* experiment using horizontal scanning. Both graphs show the statistics for a number of scan lines and a range of thresholds.

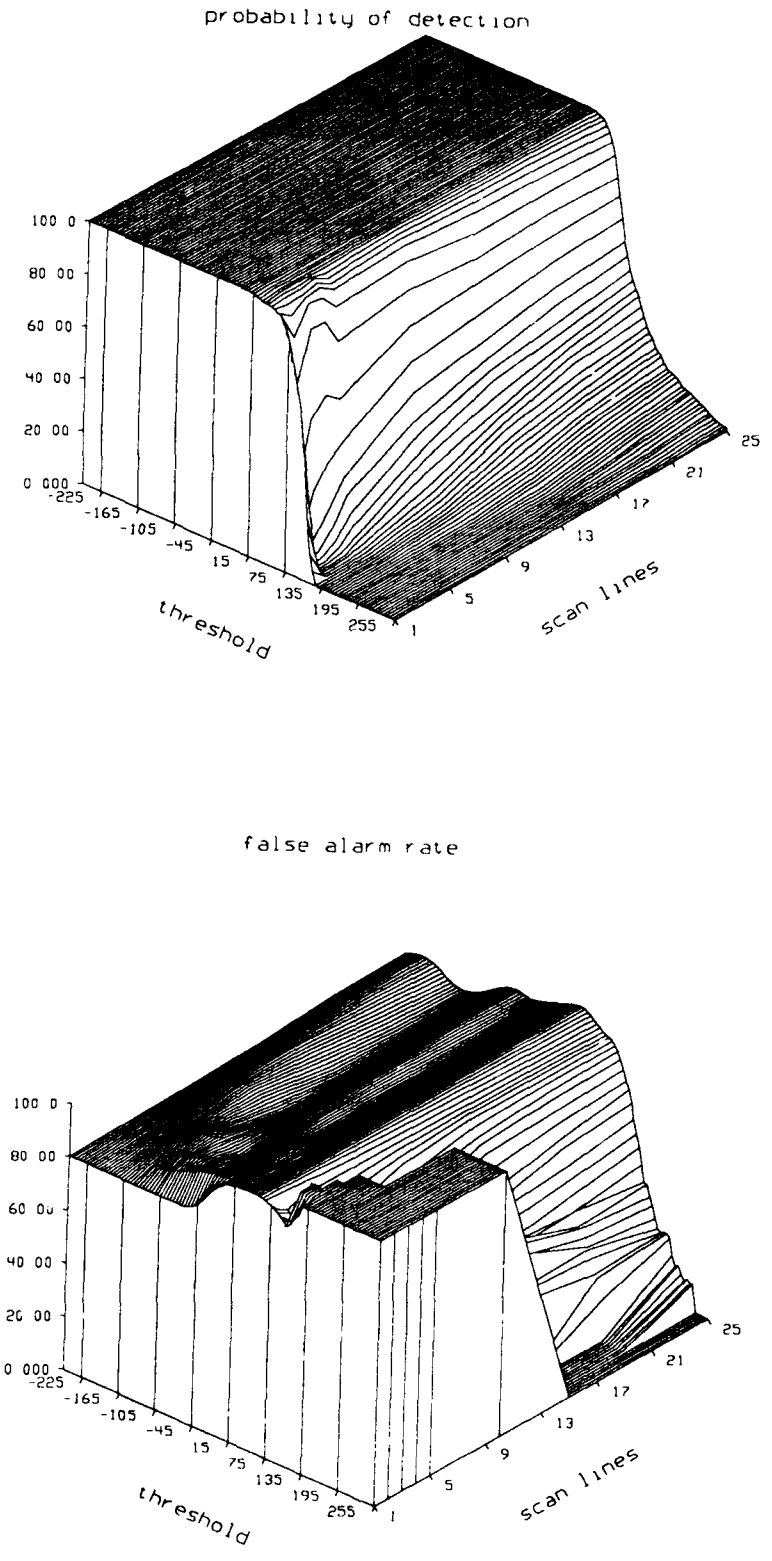


Figure 3.2.35 This figure shows the probability of detection (top graph) and false alarm rates (bottom graph) for the *variance* experiment using vertical scanning. Both graphs show the statistics for a number of scan lines and a range of thresholds.

strongest belief was correct then these values indicate that there is a 0% false alarm rate for the large thresholds; otherwise, the values indicate that there is a 100% false alarm rate for the large thresholds. In either case, the correctness of the detection with the largest belief is insignificant.) As can be seen by the graphs, only a very subtle improvement was realized when an increased number of scan lines was used to detect the targets. For some of the experiments, it was easier to select a valid detection threshold when a large number of scan lines were used to detect the targets. When only a small number of scan lines were used to detect the targets, there was only a small number of thresholds where the detector did not classify all pixels as belonging to part of the target or classify all pixels as belonging to part of the background. The number of valid detection thresholds in the transition region between these two cases grew as the number of scan lines increased. Figures 3.2.36, 3.2.37, and 3.2.38 show the log-likelihood images for the three experiments when 25 lines were used to scan the image.

3.2.4.8. Discussion

The results produced by the statistical detector for all three experiments is disappointing. The false alarm rate in all three experiments, due to the large area of the background, was large enough to render the detector unusable. We are also disappointed with the inability of the multi-line detector to improve upon the results of the single-line detector. We believe that the output of the single-line detector was so poor that there was not enough information in the log-likelihood images that it produced to enable the multi-line detector to improve the performance. Because of the poor performance of the detector, we do not believe that there is much promise for the development of a reliable detector based on the simple statistical scheme presented here; thus, we believe that resources would be better spent investigating other avenues for detecting tactical targets with a limited amount of LADAR data.

3.3. LOW LEVEL PROCESSING OF LADAR DATA

This section describes our progress in the low level processing of LADAR data which will feed the geometric methods in Section 3.4. As proposed in [KaYo88], we have moved beyond our initial low level edge detection and component labeling scheme for surface extraction in favor of a more noise immune region growing approach. We have also implemented two optional preprocessing steps to help diminish the effects of the noise so prevalent in current LADAR imagery. Processing results illustrating the performance of our new algorithms are presented for both synthetic and actual LADAR data. The algorithms are also evaluated more rigorously via a newly defined feature set for surfaces and several performance measures.

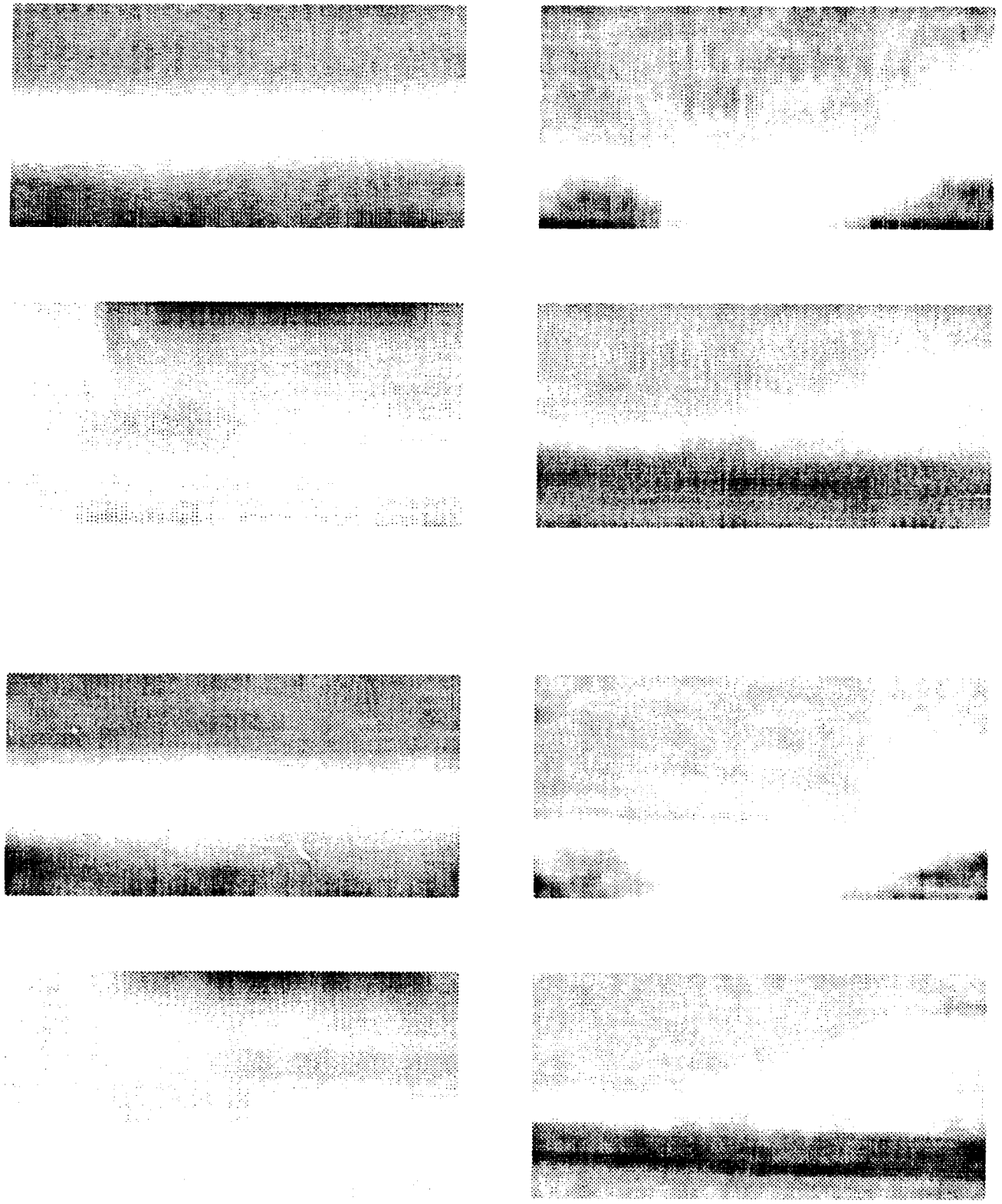


Figure 3.2.36 This figure shows the *raw-data* log-likelihood images produced by the multi-line detector when 25 lines were used to scan the images. The top four images are the output when vertical scanning is used and the bottom four show the output when horizontal scanning is used.

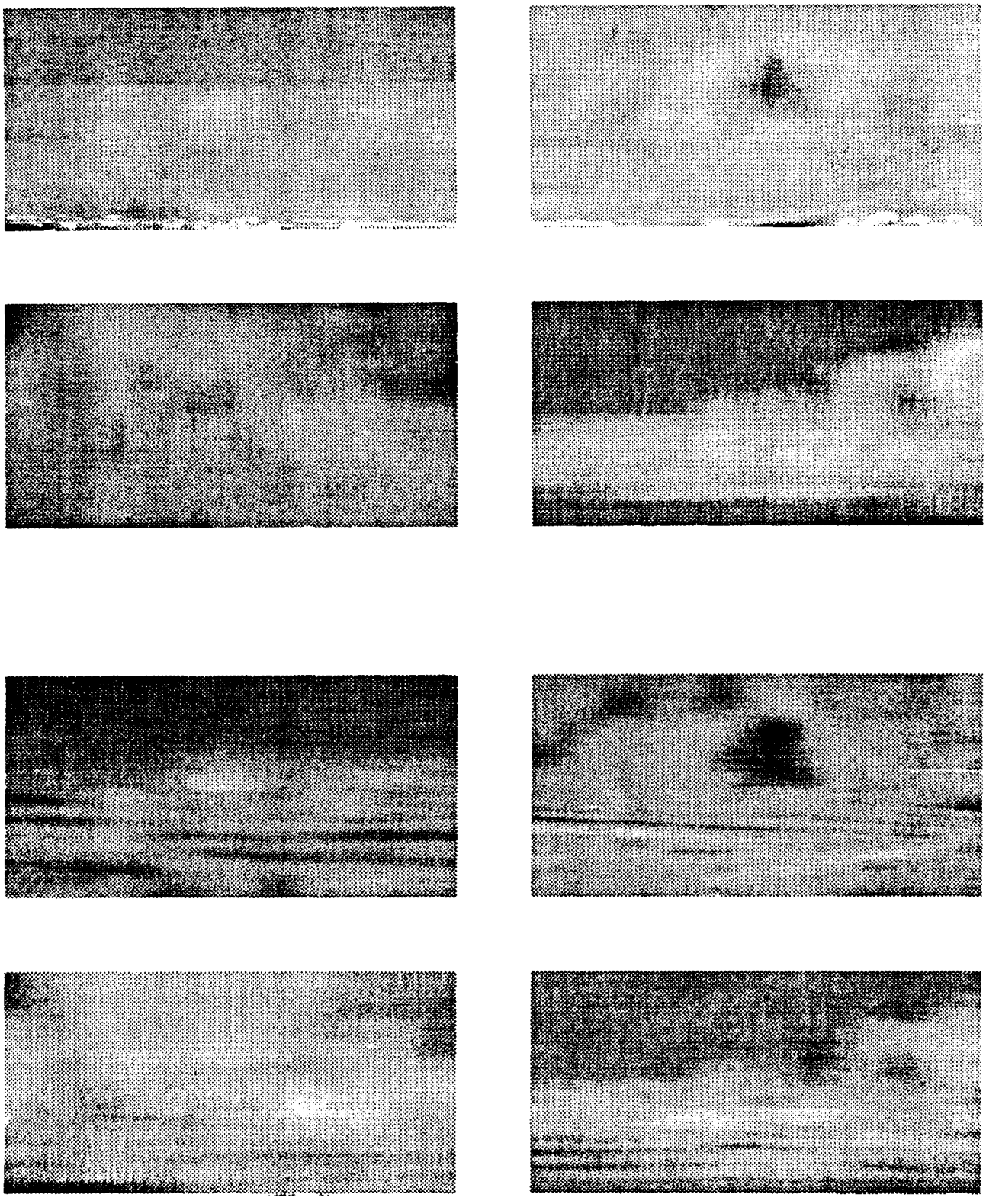


Figure 3.2.37 This figure shows the *pseudo-AM* log-likelihood images produced by the multi-line detector when 25 lines were used to scan the images. The top four images are the output when vertical scanning is used and the bottom four show the output when horizontal scanning is used.

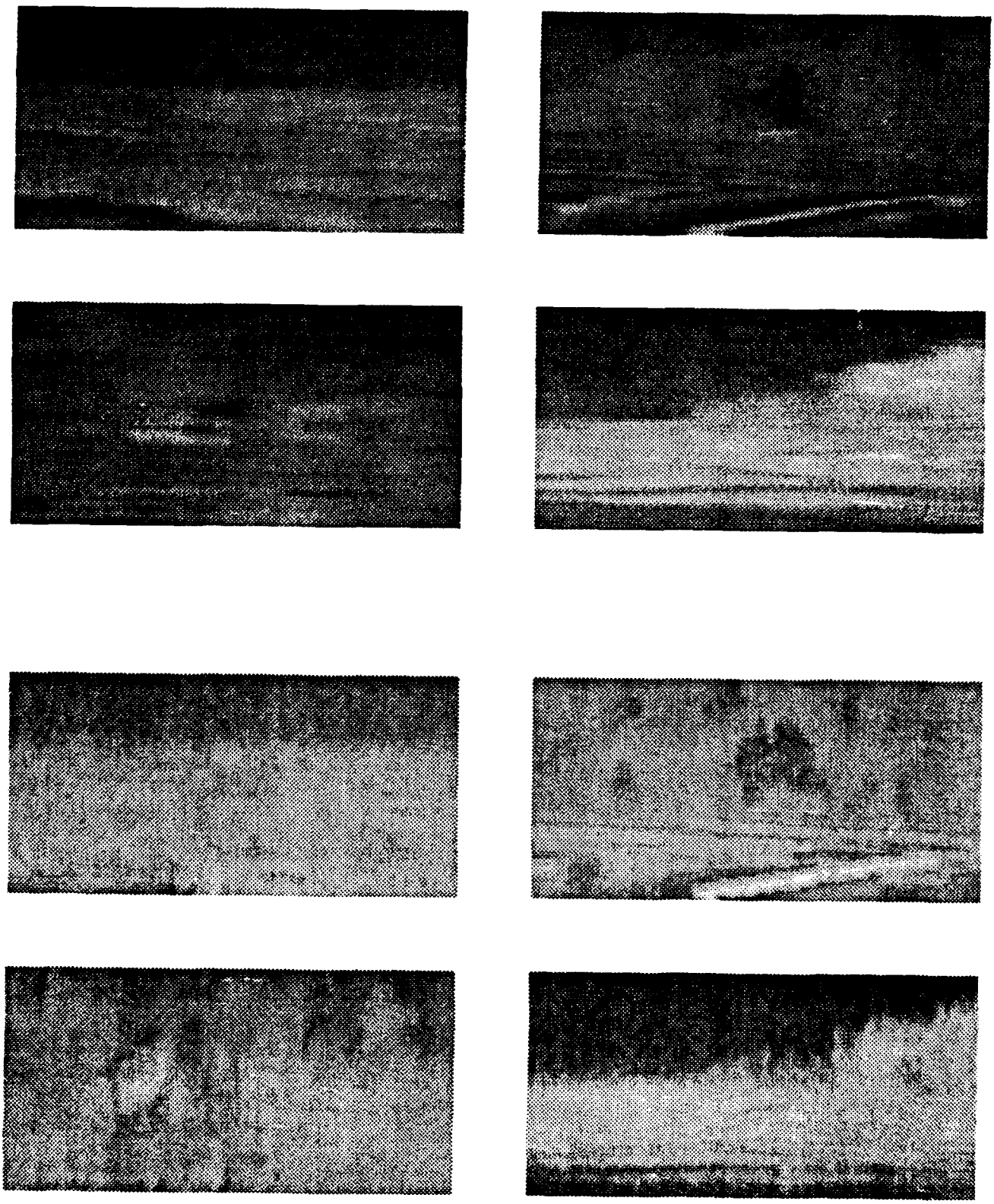


Figure 3.2.38 This figure shows the *variance* log-likelihood images produced by the multi-line detector when 25 lines were used to scan the images. The top four images are the output when vertical scanning is used and the bottom four show the output when horizontal scanning is used.

3.3.1. Low Level LADAR Range Data Processing

We are developing the special purpose low level software necessary to process LADAR imagery. Thus far we have implemented the surface segmentation scheme described in Section 3.4.2.1.1. This section addresses some of the difficulties encountered when processing LADAR data and presents the algorithms and techniques we have designed to overcome them. Determination of Surface Normals

Aside from the (x,y,z) locations of pixels, surface normals are the most important information we can get out of a range image. It is important to do the best job possible when determining surface normals because most subsequent processing steps use them. Since they are based on first order derivatives of the range map, however, their computation is susceptible to the effects of quantization error, dropouts, sparseness of data, and noise.

Figure 3.3.1 is a range image of an M60A1 tank which was generated using our Electronic Terrain Board Model (see Section 4). The resolution of the range image is comparable to a LADAR image taken at 500 meters, in the sense that the sampling in the x and y directions is similar to that of LADAR at that range. However, our simulated tank image has greated resolution in the z direction (the full dynamic range of gray scale values is used on target). The coordinate system convention we have adopted for range images is that the image sits in the *x,y-plane*, with the origin in the upper-left-hand corner, the *x-axis* pointing from left to right, and the *y-axis* pointing from top to bottom. The positive *z-axis* must be the cross product of the *x-* and *y-axes*, and so points into the image.

We computed surface normals for the tank image by fitting symmetric planar patches to the data as described in Section 3.3.3.1. The (x,y,z) components of a surface normal are easily expressed in terms of the coefficients of the planar patch fit equation $z = ax + by + c$ as: $(S_x,S_y,S_z) = (a,b,-1)$. Figures 3.3.2 and 3.3.3 illustrate the (x,y,z) components of the surface normals computed using 3 by 3 and 5 by 5 windows, respectively. The darker the pixel in the images, the larger the particular component of its surface normal. The wavy noise in the component images is due to the quantization error of PADL-generated 8-bit integer range images. While the effects of having only 256 available gray values in a range image are not apparent to the naked eye, moderately sized window operators can detect the step from one quantization level to the next as one moves along an oblique surface. Using windows that are too small can cause these steps to appear as false edges, as demonstrated by the 3 by 3 windows of Figure 3.3.2. Using larger windows reduces the effect, but can make actual edges appear blurry, as seen in Figure 3.3.3. If windows that are too large are used, the edges may become transitional surfaces between the actual surfaces they separate.

We have devised a way to remove the effects of quantization error and some other noise without producing fat edges or false surfaces. The surface normals are determined in a two step process [KaCh87, ChKa88]. The first step fits symmetric planar patches as

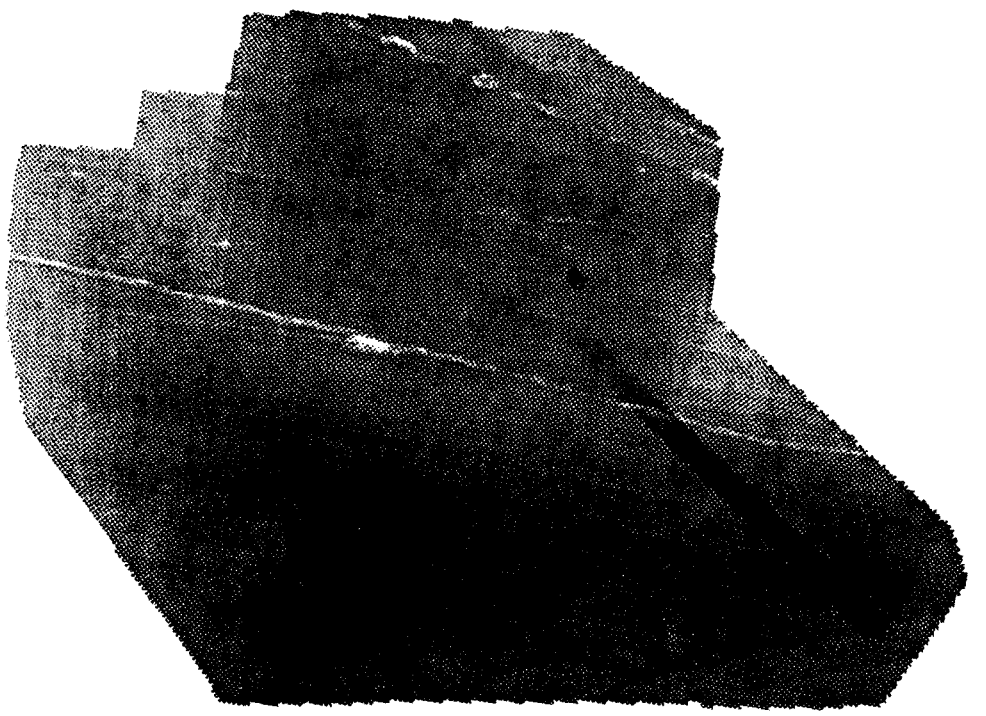
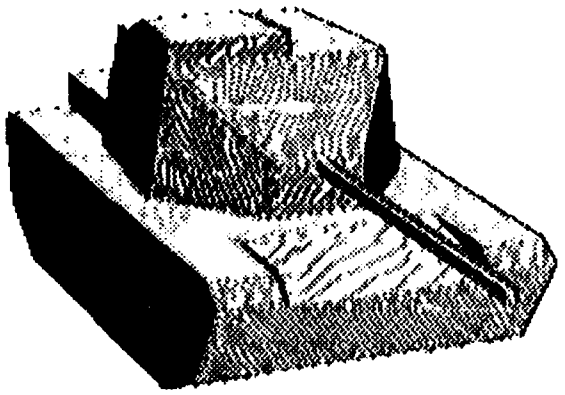
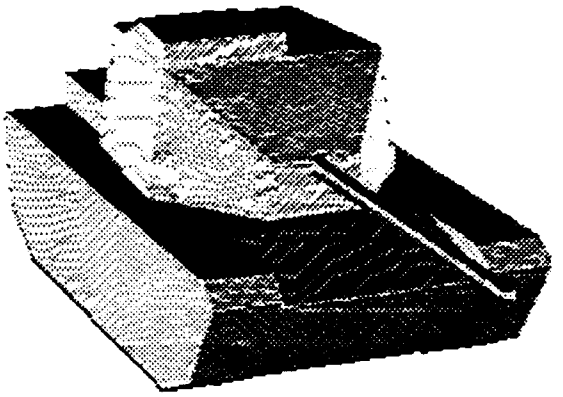


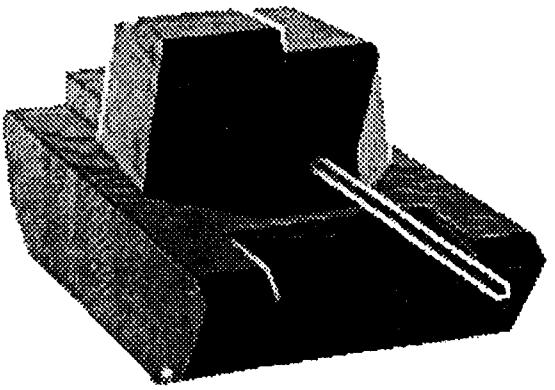
Figure 3.3.1 PADL simulated range image of an M60A1 tank.



(a) X Component

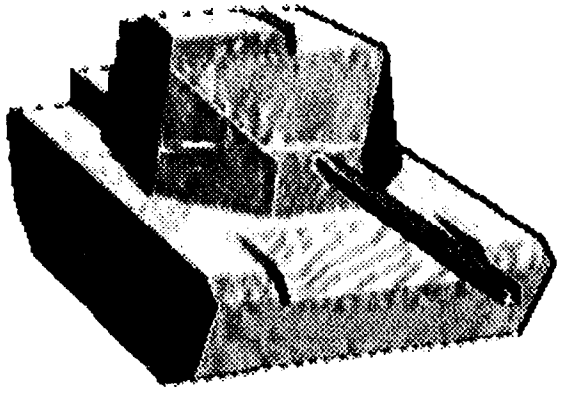


(b) Y Component

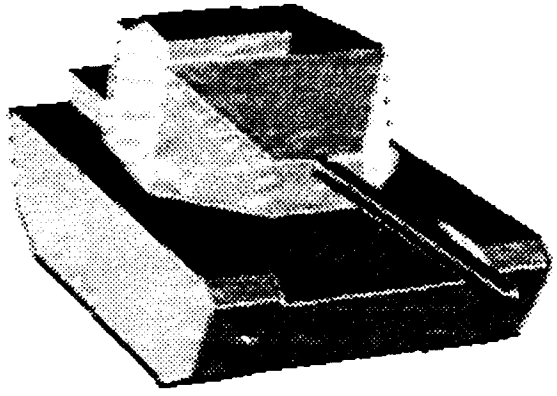


(c) Z Component

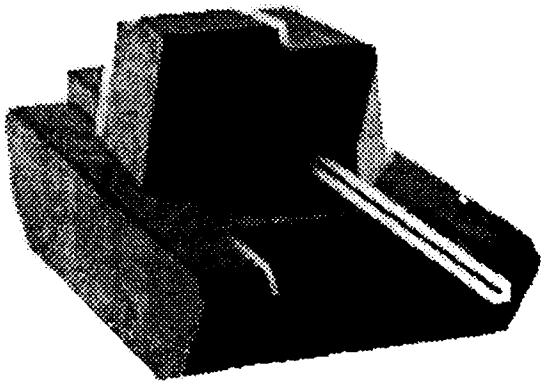
Figure 3.3.2 Surface normal components using 3 by 3 windows.



(a) X Component



(b) Y Component



(c) Z Component

Figure 3.3.3 Surface normal components using 5 by 5 windows.

before, and computes the mean square fit error for each patch. The window size used is large enough to remove the effects of noise. We found that 9 by 9 windows work well for targets at 500 meter resolution. During the second step another pass is made over the image, examining the fit error of pixels within the processing window used to compute the surface normal of the current pixel. These pixels contributed to the computation of the current pixel's planar patch, and conversely, the current pixel contributed to the computation of theirs. We may therefore conclude that if the fitted planar patch for one of these neighboring pixels has smaller fit error e_n than that of the current pixel e_c, it is reasonable to consider the current pixel to be part of that better fitting patch (since it contributed to it), and to assign the same surface normal to the current pixel. Since pixels in the same processing window could actually be any distance away from the current pixel in 3-D space, we normalize their fit error by this distance when we make the comparison:

$$e_{cn} < e_c \ ,$$

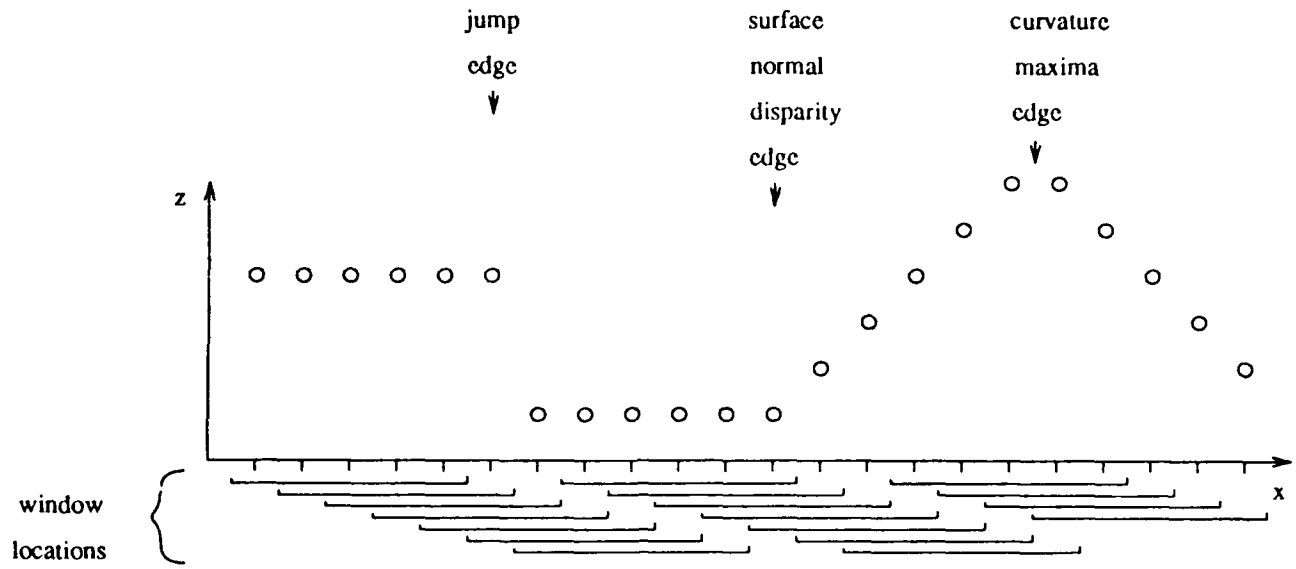
where

$$e_{cn} = e_n(\sqrt{(x_c - x_n)^2 + (y_c - y_n)^2 + (z_c - z_n)^2 + 1})$$

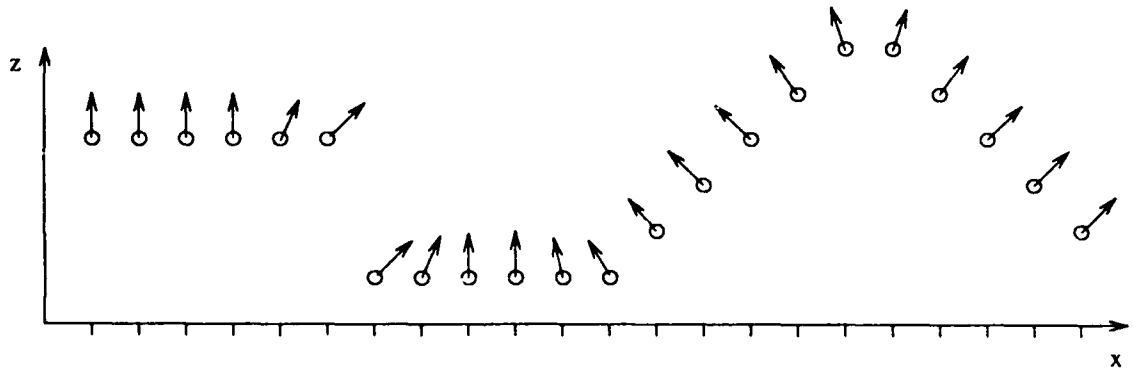
is the neighbor's error normalized by its distance. This prevents pixels from being assigned to surfaces which are a substantial distance away from them, and makes the reassignment process local.

Figure 3.3.4 is a 2-D example illustrating the two step process. Figure 3.3.4(a) shows a range map of four surfaces surfaces separated by the three types of edges we are interested in detecting (see Section 3.4.2.1.1 and Section 3.3.1.2). The bars below the *x-axis* indicate the consecutive positions of a 5 pixel wide processing window as it moves across the data. At each window position a minimal mean square error linear fit is computed for the sample points, corresponding to fitting a planar patch in the 3-D case. The vector in the *x,z-plane* orthogonal to the fitted line (corresponding to a 3-D surface normal) is associated with the pixel in the center of the window, as is the fit error e_c. Figure 3.3.4(b) illustrates the "normals" for these fitted "patches." Figure 3.3.4(c) shows the "surface normal" assignments made during the second pass over the data by examining the fit error e_{cn} of pixels within the processing window used to calculate the current "normal" and fit error e_c. Remember that e_{cn} is a neighbor's fit error normalized by its distance away from the current pixel (the one in the center of the window). Note that the "normals" of Figure 3.3.4(c) are much better than those of (b), and will produce better results when used in later processing.

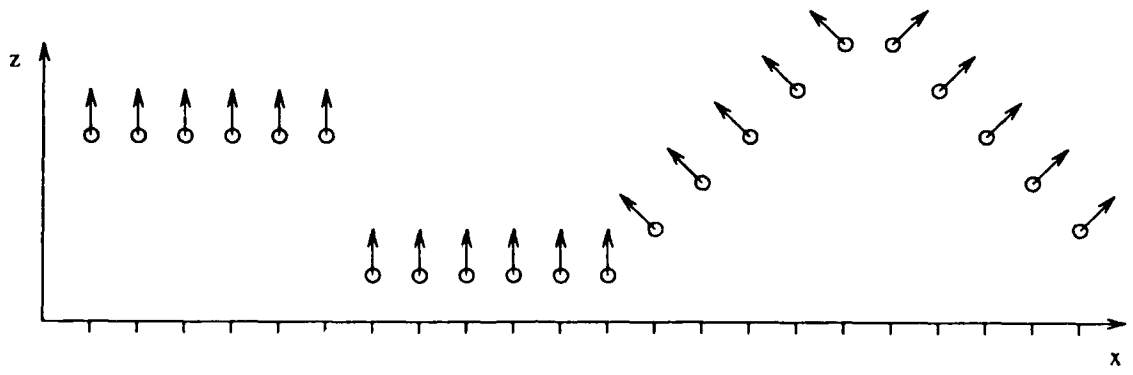
The above procedure has a two-fold effect. First, it performs a kind of smoothing, producing more uniform surfaces and eliminating noise. Second, it automatically does edge thinning, yielding surfaces with maximal area. Figure 3.3.5 shows the surface normal components computed for the range image of Figure 3.3.1 using a 9 by 9 window. Now that we have the best surface normals possible without extensive preprocessing, our later



(a) Range Map and 5 Pixel Wide Windows

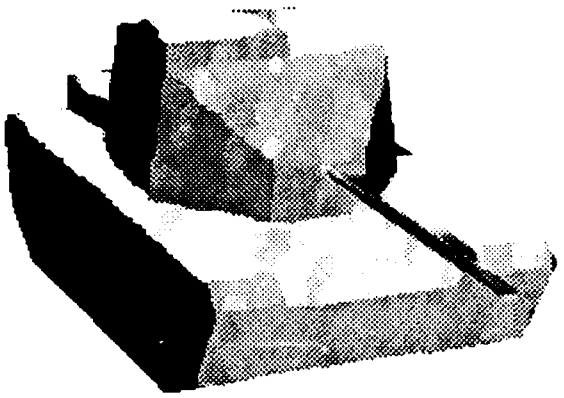


(b) Normals For Fitted Patches

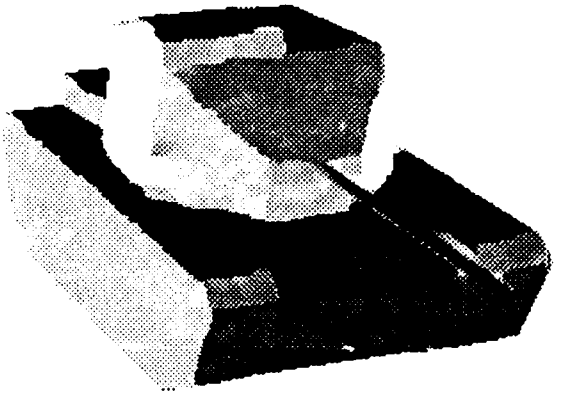


(c) Assigned Normals After Examining Fit Error

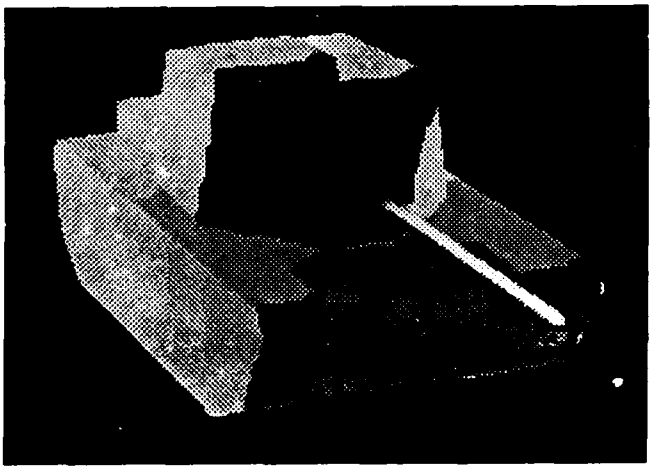
Figure 3.3.4 Computation of surface normals.



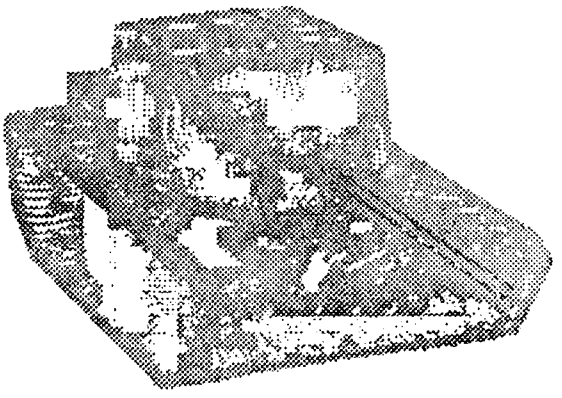
(a) X Component



(b) Y Component



(c) Z Component



(d) Modified Range Pixels

Figure 3.3.5 (a)-(c) Surface normal components using 9 by 9 windows and reassignment to patches with minimal error. (d) Range pixels modified to improve curvature calculation.

processing should also generate better results.

3.3.1.1. Curvature Computation

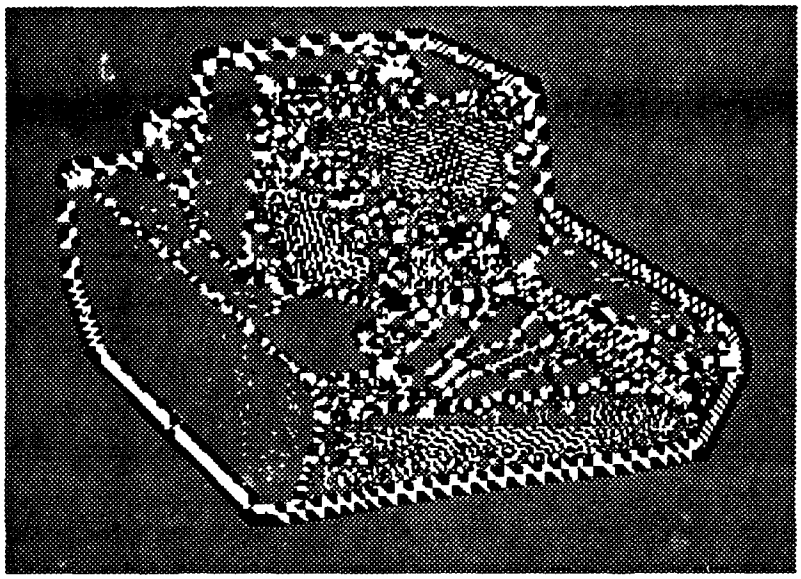
If we were to estimate curvature by merely convolving the range map with the window operators described in [BeJa85, BeJa86, YaKa86a], we would not be taking advantage of the noise removal performed by our surface normal determination technique. Since curvature computation requires second order derivatives of the range map, the estimate is already very sensitive to noise. In order to get better results, a modified range map is created as surface normal assignment is performed during the second step of the aforementioned technique. If the e_{cn} of a neighboring pixel is less than e_c, then as the neighboring surface normal is assigned to the current pixel, the current pixel is also moved in the modified range map to its projected position on the better fitting planar patch. This modified range map is then convolved with curvature window operators which are the same size as the surface normal computation windows.

Figure 3.3.5(d) shows the range pixels which were moved in the modified range map of the tank image. White pixels were not moved at all, gray pixels were moved a negligible amount, and pixels in the gray to black range (like those near the edges of the gun barrel) are darker if they were moved farther. The images of Figure 3.3.6 show the signs of the Gaussian and mean curvatures obtained by convolving the modified range map with 9 by 9 window operators. Gray indicates roughly zero curvature (planar), white is positive curvature (convex), and black is negative curvature (concave). Note that even with the steps taken to provide a better range map, curvature is still fairly noisy even for synthetic images due to the fact that it is a second order operation.

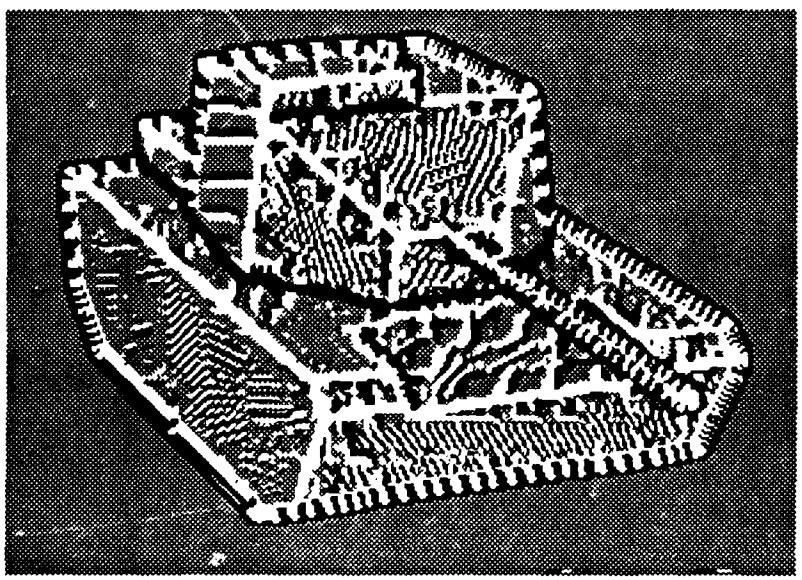
3.3.1.2. Edge Detection and Surface Labeling

The next task is to find the edges between surfaces, and then label the individual surfaces. The types of edges to be found include jump edges, curvature maxima edges, and surface normal disparity edges, as described below. The edge labels retain information about the pixel's specific edge type, which is useful during high level processing of the scene.

The most obvious edges occur where 3-D distance from a point to one of its neighbors is greater than some threshold. A pixel meeting this criteria is labeled as a *jump edge.* In order to prevent every pixel on an oblique surface from being labeled as an edge, just the ones at the front and back, the range discontinuity to the neighbor on the opposite side of the one under question must be significantly less. Figure 3.3.7(a) shows jump edges detected for the tank image. Both coarse and fine thresholds are used. A coarse jump edge would separate the top of a target from background, while a finer threshold allows detection of jumps resulting from one surface of the object occluding another, such as the gun barrel or hatch occluding a portion of the turret.

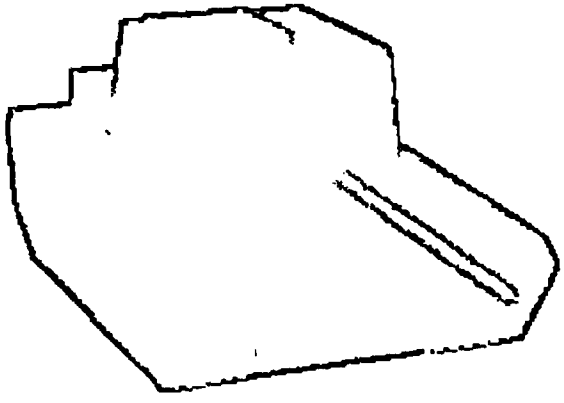


(a) Gaussian Curvature

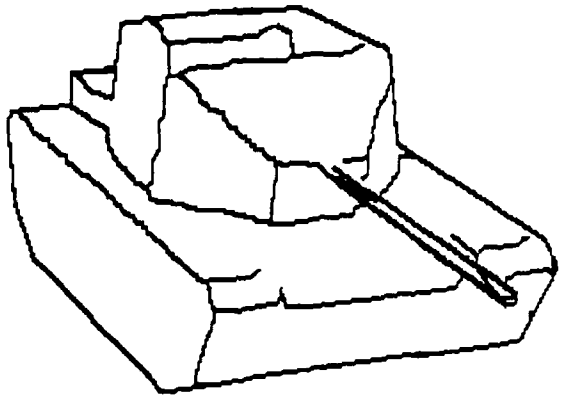


(b) Mean Curvature

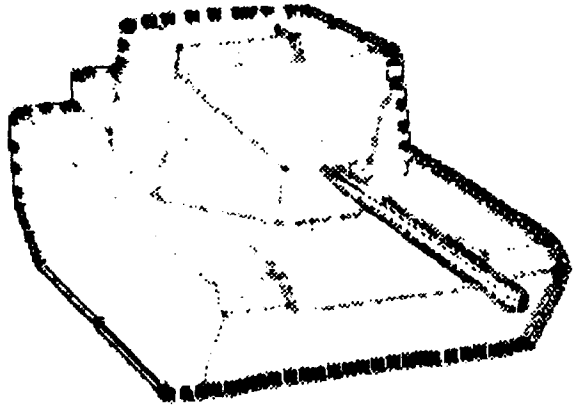
Figure 3.3.6 Signs of curvatures calculated by convolving modified range map with 9 by 9 window operators. Gray is zero curvature (planar), black is negative curvature (concave), and white is positive curvature (convex).



(a) Jump Edges



(b) Surface Normal Disparity Edges



(c) Curvature Edges

Figure 3.3.7 Detected edges.

The next most obvious boundary between surfaces is formed by points where surface normals differ significantly in orientation. These are known as *surface normal disparity edges*. We use a threshold value of 25 degrees difference in orientation to identify these pixels. Figure 3.3.7(b) is an example surface normal disparity edge image. The results are very good because of good surface normal determination.

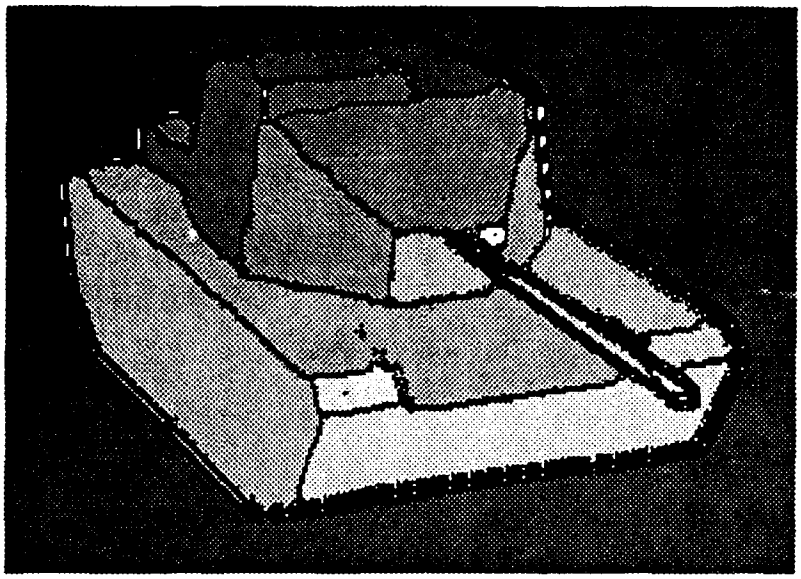
An improvement in the performance of edge segmentation can sometimes be gained by considering curvature information. Thresholds corresponding to some minimum allowable radii of curvature are set, and if the magnitude of the mean of Gaussian curvatures exceed these thresholds, the pixel is labelled as *curvature maxima edge*. Whether the edge is concave or convex may be determined by looking at the sign of the curvature. Figure 3.3.7(c) illustrates curvature edges for a minimum allowable radius of a tenth of a meter for both Gaussian and mean curvature. These edges are poorer than the previous types because curvature is generally noisy. However, if one looks back at the sign of the mean curvature image in Figure 3.3.6, one can immediately tell which edges are convex and concave.

The remaining pixels are now given a label corresponding to what surface they belong to via a connected component labeling algorithm. Figure 3.3.8 shows the resulting labels when curvature information is and isn't taken advantage of. In these images, each surface is given a unique gray value. Note that even though the curvature edges are not as clean, they do help fill in the gaps between other edges which lead to incorrect labeling.

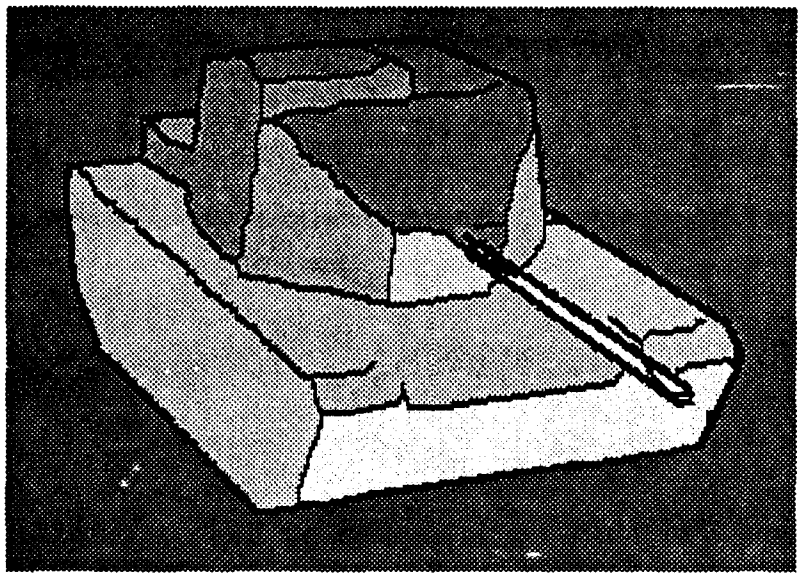
3.3.1.3. Results on Noisy Images

The M60A1 tank image we have been working with was downsampled by a factor of four to obtain an image roughly equivalent in resolution to a LADAR image taken at two kilometers (same x and y resolution, but finer z resolution as discussed in the beginning of this section). The image was then degraded with noise as described in Section 4.1.3 using two dropout rates and three different Gaussian variances. Variances of 1, 3, and 7 were used with a dropout probability of 0.005, and a Gaussian variance of 3 was used with a dropout probability of 0.05. The cleanest LADAR images found when determining the noise characteristics of the actual data had a Gaussian variance of 7 for their gray values and dropout rate of 0.005. Note that we have a higher resolution in the z direction (in our gray values), and that a Gaussian variance of 7 in gray values for us corresponds to an error of 0.29 meters, which is pretty high for the size of the targets and the kind of processing we hope to do. For completeness, a variance of 3 for us corresponds to 0.125 meters, and a 1 gray value variance corresponds to 0.04 meters.

Figure 3.3.9 contains the original degraded downsampled range images, while Figure 3.3.10 shows the results of processing them using 5 by 5 processing windows and no curvature information (it is too noisy). Higher level processing (Section 3.4) should be able to handle the low level results of Figure 3.3.10(a) & (b) with no problem, would fail

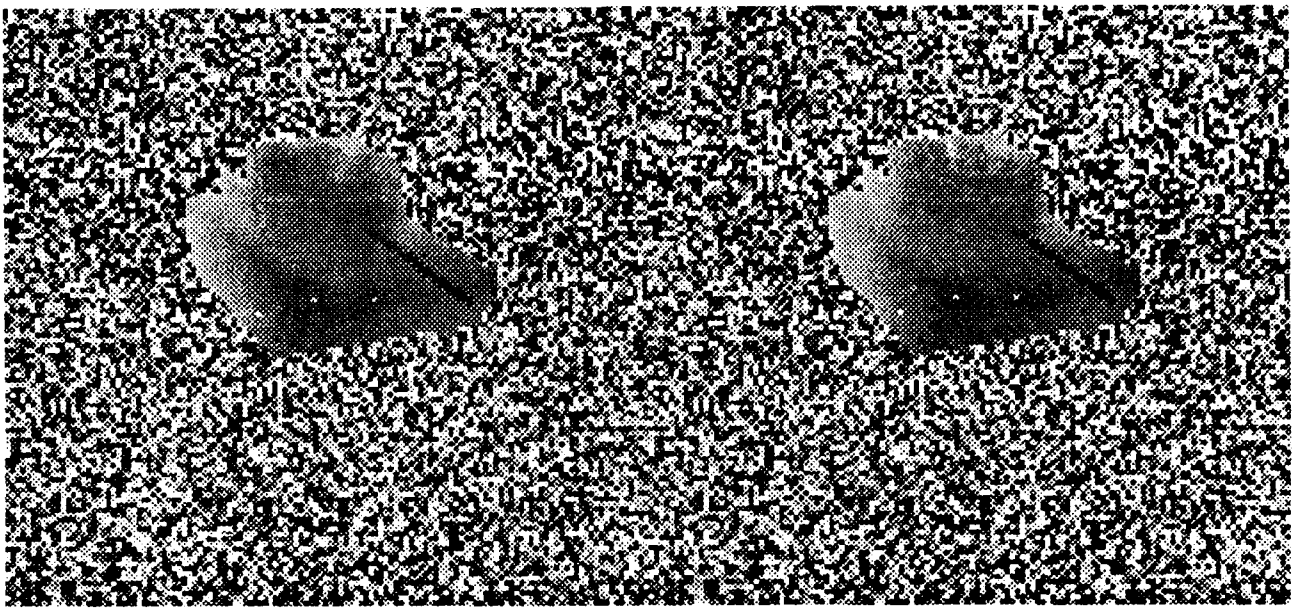


(a) Using Curvature



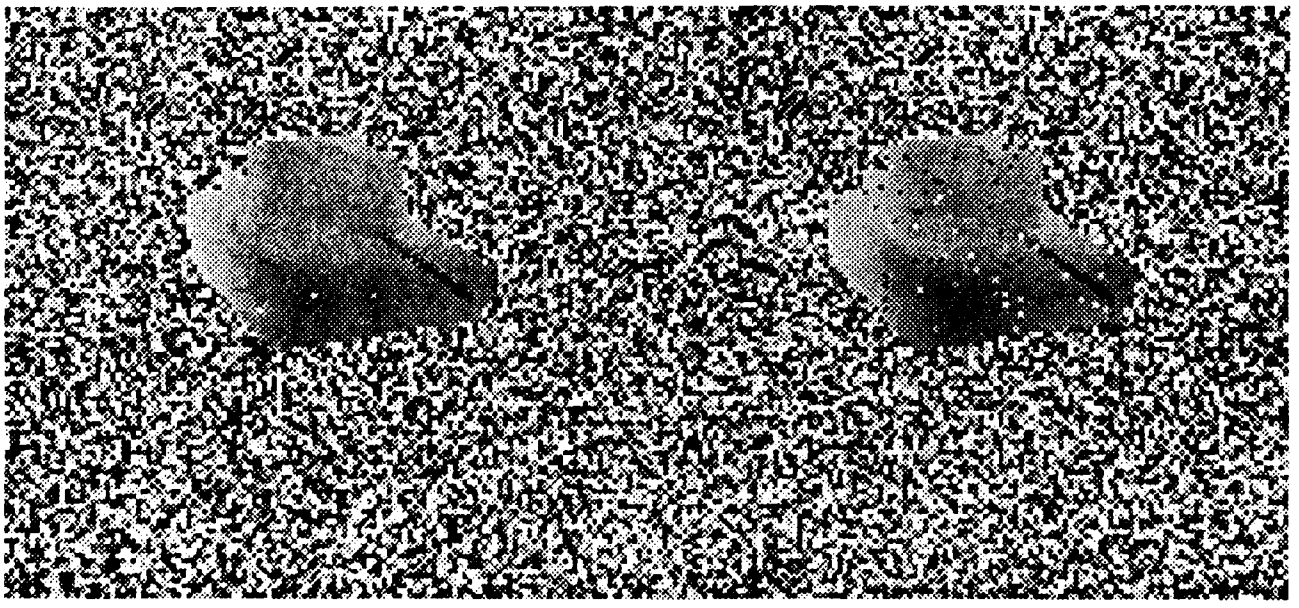
(b) Without Curvature

Figure 3.3.8 Surface label images.



(a) Var = 1, Prob. Dropout = 0.005

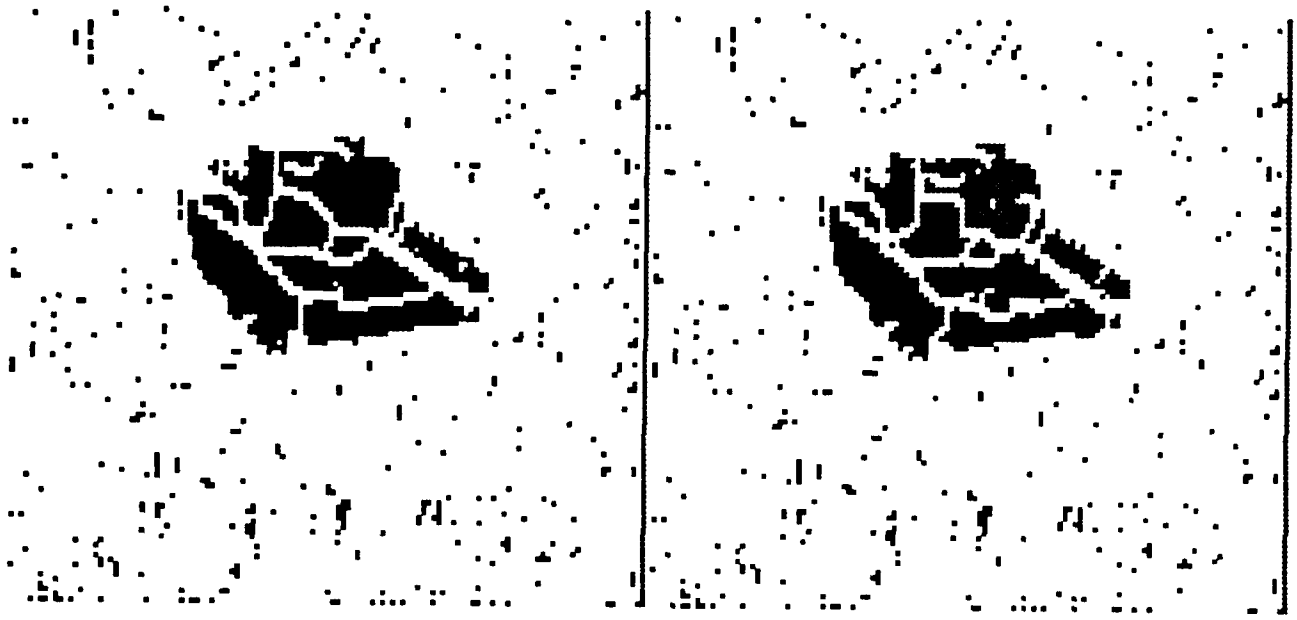
(b) Var = 3,Prob. Dropout = 0.005



(c) Var = 7, Prob. Dropout = 0.005

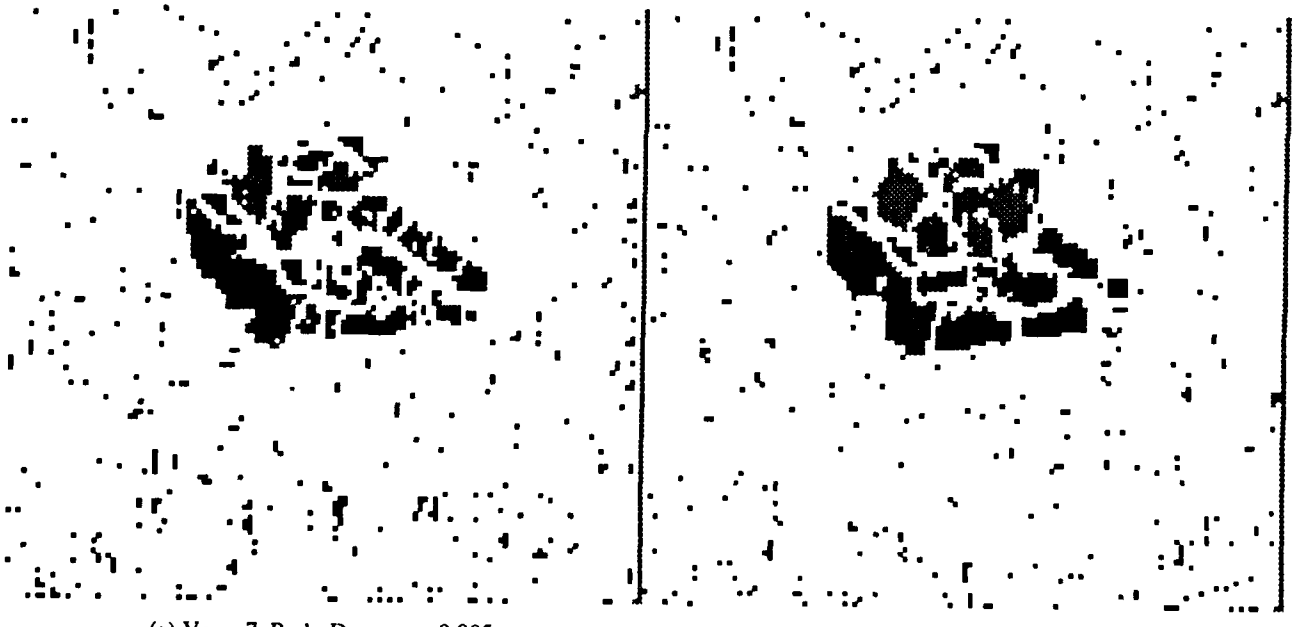
(d) Var = 3, Prob. Dropout = 0.05

Figure 3.3.9 Original degraded range images.



(a) Var = 1, Prob. Dropout = 0.005

(b) Var = 3, Prob. Dropout = 0.005



(c) Var = 7, Prob. Dropout = 0.005

(d) Var = 3, Prob. Dropout = 0.05

Figure 3.3.10 Labeled surfaces for degraded images using 5 by 5 processing windows and no curvature information.

miserably for (c), and may or may not be able to make sense out of (d).

3.3.1.4. Conclusions and Future Work

Our algorithms and techniques work well for reasonable range data. Additional preprocessing may be necessary to rid actual LADAR of noise, and we might not be able to use a geometric approach if later LADAR data does not prove to have the resolution necessary to provide adequate surface information. If this is found to be the case, we would be forced to use the 2-D, silhouette-based approach we have been using in our processing of FLIR data. The only advantage of using LADAR would be better silhouette segmentation immune to the variance found in FLIR and other reflectance imagery.

As far as surface segmentation goes, a region growing scheme may fare better than our present edge detection/surface labeling scheme. A region growing approach would not be susceptible to gaps in edges, since it does not depend on them explicitly. With this exploration of region growing we will also try using normal curvature instead of Gaussian and mean curvature. Since normal curvature uses surface normal information directly, it will be able to take advantage of our improved surface normal routines and should not be as noisy as Gaussian and mean curvature. Also, since normal curvature is defined between two points it gives information regarding edge direction, whereas Gaussian and mean curvatures only provide magnitudes with no directional information.

In the future, our low level routines will be extended to calculate the surface attributes and relations necessary for our high level processing, once we have determined what these attributed and relations are.

3.3.2. New Low Level Processing Scheme

We initiated our research effort by applying our existing general 3-D vision software to LADAR data. As described in [KaYo88, CroKa87], this scheme entailed the generation of *(x,y,z)* locations (the *range map*), computation of surface normals and curvature, labeling of edge pixels, and labeling of surface regions, followed by determination of surface attributes and relations and invocation of domain-specific classification rules. Surface extraction by first detecting edges and then performing connected component labeling on the remainder of the image proved to be too sensitive to the high variance noise and large number of dropouts present in actual LADAR data. This large amount of noise also made Gaussian and mean curvature information, which is derived from the second order derivatives of the range map, relatively useless.

Figure 3.3.11 illustrates our new approach to surface extraction. As before, the first step is to convert the range image into a range map of *(x,y,z)* locations. This is followed by two optional range map preprocessing steps for noise removal, as indicated by the dashed bubbles in the figure. These will be presented in the next section. Surface normals are then computed using the planar patch fitting, minimal error assignment algorithm

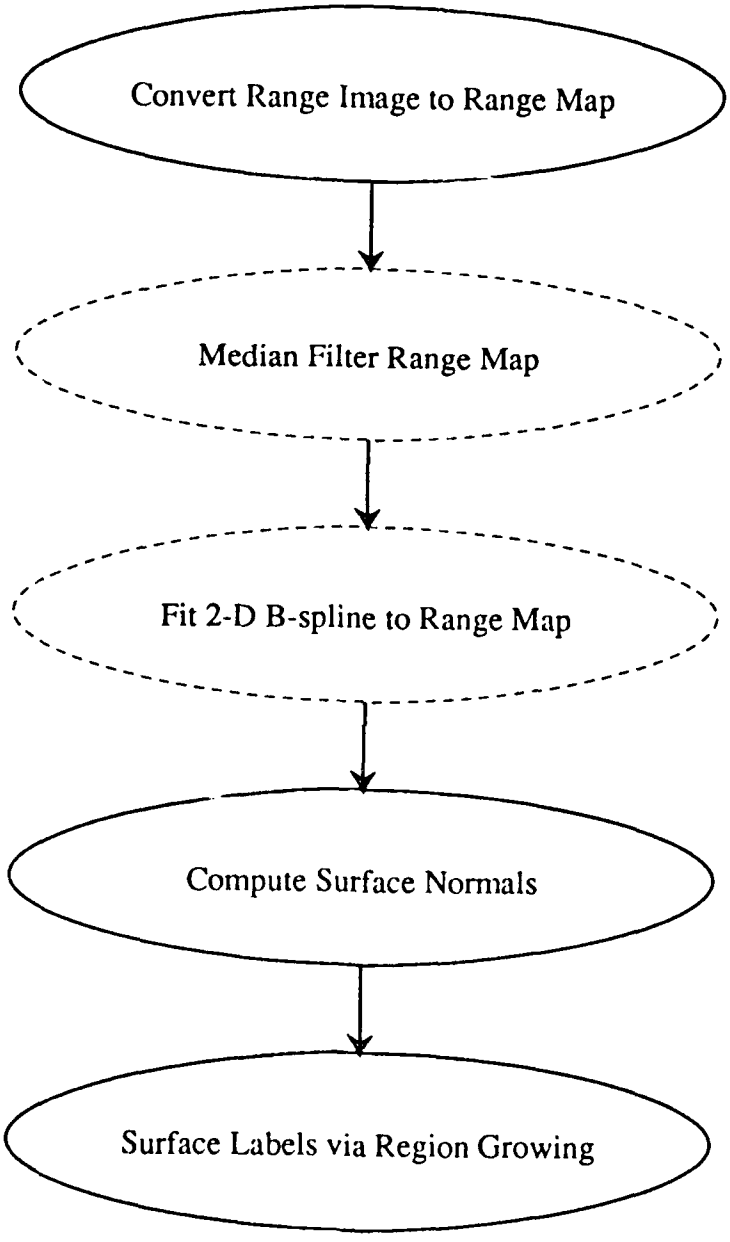


Figure 3.3.11 Extraction of surfaces from range images.

discussed in [KaYo88, KaCh87, ChKa88]. Finally, surface labels are assigned by a region growing algorithm to be discussed in detail in Section 3.3.2.2. As before, surface extraction is to be followed by determination of surface attributes and relations, and finally object classification via the high level routines.

3.3.2.1. Preprocessing

The study of the noise characteristics of the 1986 A.P. Hill LADAR data reported in [KaYo88] indicates that the FM component of the resolved range data may be off as much as 9 meters on a target at 1400 meters, and that the AM portion is corrupted by Gaussian noise with a typical standard deviation of 0.75 meters. Various dropout rates are also present in the imagery. Until a more rigorous noise model is found for the sensor, proven general purpose methods of removing noise must be relied upon.

One such method is the simple application of a 2-D median filter to the range map. Because the output of the median filter at a given location could be any of the pixel values within the processing window, there was some concern that the median filter may destroy some of the fine surface information present in the original range map. A simple experiment, presented in Section 3.3.2.3, was carried out in order to determine how median filtering the range map affects the results of later processing steps. Based on that experiment and processing results for synthetic and actual LADAR images, we found that a 3 by 3 median filter improved the ability of our programs to successfully extract surfaces. It is therefore included as an optional preprocessing step in our overall scheme.

A second approach to handling noisy data involves fitting surfaces to it. We have already shown how fitting planar patches can be useful for surface normal computation. However, real surfaces in real data are seldom planar. We therefore propose the fitting of higher order surfaces to the data. Yang and Kak [YaKa86a, YaKa86b] have shown how 2-D B-splines may be used to fit a bicubic surface patch. A bicubic patch is of third order, guaranteeing continuity of the first-order and second-order derivatives, and so usually yields the smoothest fit to the data points. 2-D B-splines are fit to successive 4 by 4 windows of the range map. The fit does not necessarily pass through the 16 data values, but merely uses them as control points. At a given window position the fit is computed for the four points in the center of the window. Continuity up to the second order derivative is guaranteed between the current patch and the one computed for the next window position. See [FaPr79] for a discussion of the merits of splines over other interpolation schemes.

While 2-D B-spline interpolation definitely yields a smooth range map, it does not remove the effects of dropouts. Single noise pixels which have values that differ greatly from the true distance to the surface can cause large distortions in the fitted surface. It is therefore desirable to remove dropouts via median filtering or median-based range bin correction (MBRBC) [KaYo88] before fitting B-splines. The fitting of 2-D B-splines is included as an optional step in the preprocessing of the range map.

Besides the using the median filter and 2-D B-splines, we have also tried processing the AM component of the data alone and the "range bin corrected" AM/FM resolved range data (3 by 3 MBRBC in [KaYo88]). The AM component of the data is obtained by taking mod 1875 of the original AM/FM resolved range. The magnitude of the noise in this data is considerably smaller than in the combined AM/FM. The MBRBC data is basically a median filter of the FM component of the data, and so is an attempt to remove the larger noise component from the combined AM/FM.

3.3.2.2. Region Growing Approach

We have adopted a region growing algorithm for labeling surfaces in our range map after preprocessing and surface normal computation. There are three criteria which must be satisfied by a neighboring pixel b at (x,y,z) location $\vec{r}_b$ in order for it to be included in the same region as the current pixel a at $\vec{r}_a$. First, the neighbor's distance from the current pixel must be less than a specified threshold *dst_thr*. Second, the angle between the neighbor's unit surface normal $\hat{n}_b$ and the current pixel's unit normal $\hat{n}_a$ cannot exceed *ang_thr*. Finally, the normal curvature defined between the neighboring pixel and the current one must be less than *crv_thr*. The expression we use for normal curvature is

$$\kappa_n = \frac{2\sin(\arccos(\hat{n}_a \cdot \hat{n}_b)/2)}{|\vec{r}_a - \vec{r}_b|}$$

which is merely the sine of half the angle between the surface normals divided by half the distance between the two pixels. This expression for normal curvature has been found to be less sensitive to noise than the mean and Gaussian curvatures used previously [ChKa88]. This is due in part to the fact that the explicit computation of the second order derivatives of the range map is not necessary in order to determine the normal curvature.

If all three of the above criteria are met, the neighboring pixel is considered to be part of the region to which the current pixel belongs. The 8-neighbors of the current pixel are considered for annexation to the current region, resulting in 8-connected surfaces. Our implementation of the region growing algorithm is a two-pass labeling scheme which is more efficient than the usual recursive implementation. The algorithm also discards surfaces made up of fewer pixels than the processing window used in surface normal computation. For example, if a 5 by 5 processing window was used to compute the surface normals, then surfaces containing less than 25 pixels would be considered too small and discarded. The following section presents results obtained by this new low level for surface extraction. Of course, the criteria used to evaluate the processing results is the examination of how well the extracted surfaces compare with those present in our models.

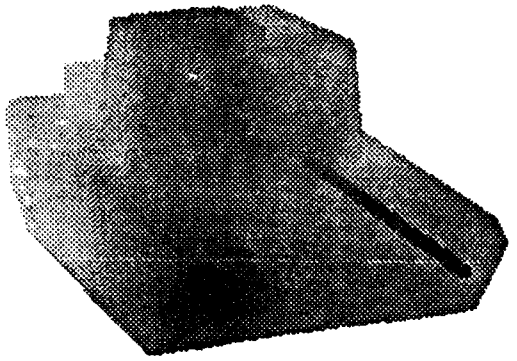
3.3.2.3. Results on Synthetic Data

As mentioned in Section 3.3.2.1, an experiment was run on "clean" synthetic data in order to determine the effects of median filtering range maps using different sized processing windows. Figure 3.3.12 shows the original PADL synthetic LADAR image and difference images for 3 by 3, 5 by 5, and 7 by 7 median filter processing windows. The darkness of a difference image pixel is directly related to the difference between the gray level of that pixel in the original image and in the median filtered image, thereby illustrating which pixels were affected and by how much. Figure 3.3.13 shows the surfaces extracted for each of the four images. Each surface is assigned a random color so that it is easy to differentiate from its neighbors. The white blotches are made up of surfaces which were discarded because they were too small. The processing parameters and results are summarized in Table 3.3.1. Note that the larger 5 by 5 and 7 by 7 median filters destroyed some of the edge information by smoothing out discontinuities, causing adjacent surfaces to become merged together. The 3 by 3 median filter produced results comparable to those obtained with no preprocessing, and so appears acceptable for noise removal.

Figures 3.3.14 through 3.3.17 show the processing results for synthetic imagery with various amounts of noise. In each of these four figures (a) is the original range image, (b) is the result obtained using the old edge detection/component labeling method of surface extraction and no preprocessing, and (c)-(f) are the results obtained with the new region growing scheme and various amounts of preprocessing. No preprocessing was done for (c), B-splines alone were used for (d), median filter only for (e), and (f) illustrates median filtering followed by fitting of B-splines. The results and processing parameters are summarized in Table 3.3.2. Compare (b) & (c), the results for the old and new low level schemes with no noise removal. The only case where they are significantly different is Figure 3.3.16, the additive noise with the highest Gaussian standard deviation. Here the region growing approach proved to be superior to the edge detection/component labeling scheme. Also note that preprocessing always helped extract surfaces ((d) and (e) are always better than (b) and (c)), but it is also possible to over-preprocess, as indicated by the merged surfaces when both median filtering and B-spline interpolation are applied to images with low noise. It is interesting to note that for the Figure 3.3.17, the one with the highest dropout rate, the application of both median filtering and B-splines produced the best segmentation of *any* of those attempted on the noisy synthetic data.

3.3.2.4. Results on 1987 A.P. Hill LADAR Data

We also tried our surface extraction algorithms on actual LADAR data. Combined AM/FM resolved range images, 3 by 3 MBRBC AM/FM images, and mod 1875 AM component images were processed. The results for subimages containing single targets are presented in Figures 3.3.18 through 3.3.26. Part (a) of each of these figures is the original range image, part (b) is the result with no preprocessing, (c) used B-splines alone, (d) only



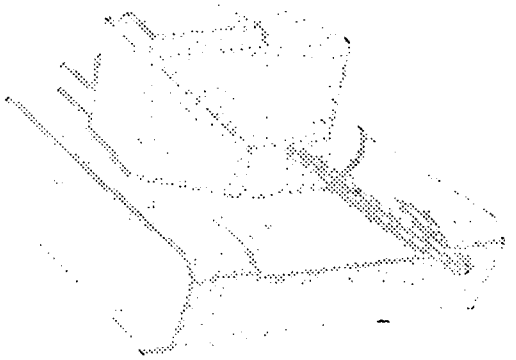
(a) m60



(b) diff.3

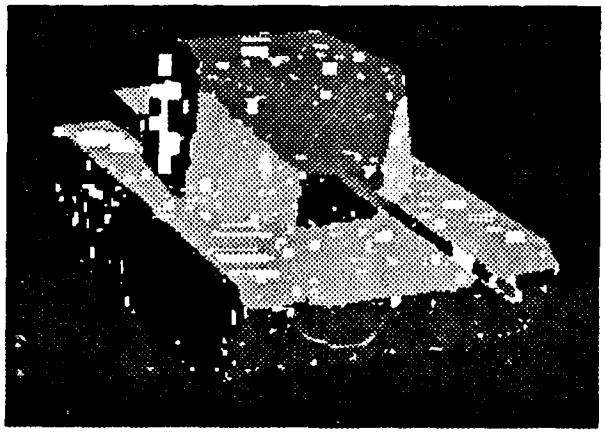


(c) diff.5

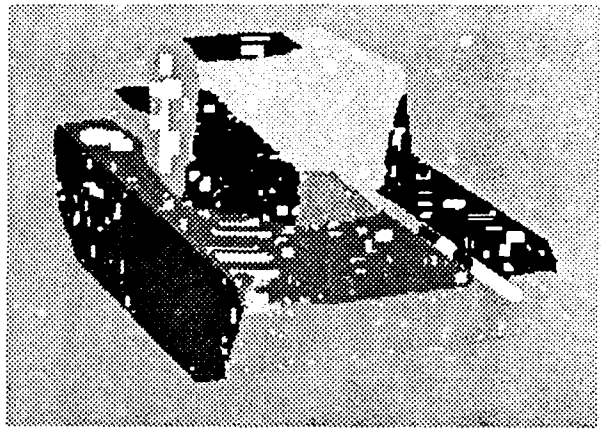


(d) diff.7

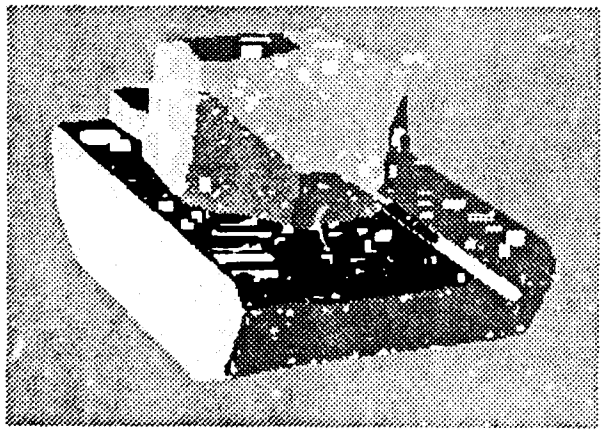
Figure 3.3.12 Effects of median filtering range images: (a) original PADL range image (b) difference between original and 3 by 3 median (c) 5 by 5 median (d) 7 by 7 median.



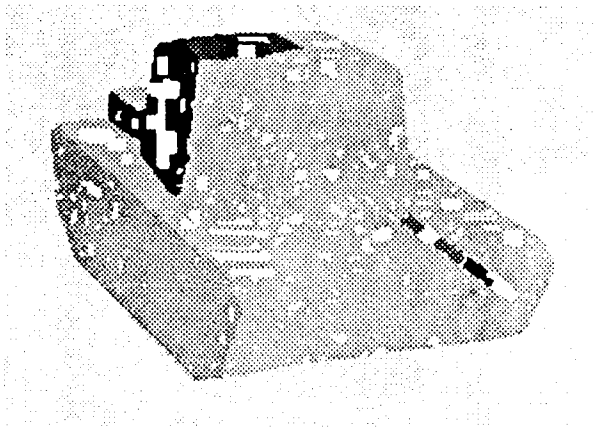
(a) m60.7x7.nomed



(b) m60.7x7.med3



(c) m60.7x7.med5



(d) m60.7x7.med7

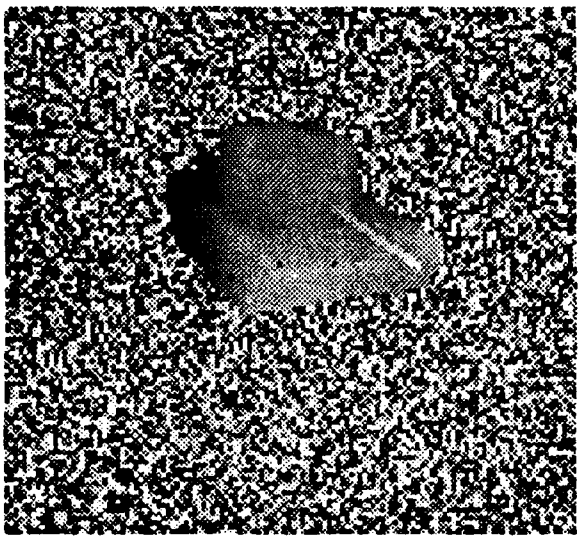
Figure 3.3.13 Effects of median filtering range images: surfaces extracted for (a) original PADL range image (b) 3 by 3 median (c) 5 by 5 median (d) 7 by 7 median.

Table 3.3.1 Summary of processing results for median filtered PADL images.

Processing Results for Median Filtered Synthetic Range Images						
Filename	Preprocessing		Thresholds			Number of surfaces
	Median Filter?	Fit 2-D B-splines?	distance (meters)	angle (degrees)	curvature (1/meters)	
m60.7x7.nomed	no	no	0.5	20	5	17
m60.7x7.med3	3x3	no	0.5	20	5	15
m60.7x7.med5	5x5	no	0.5	20	5	13
m60.7x7.med7	7x7	no	0.5	20	5	9

Table 3.3.2 Summary of processing results for noisy PADL images.

Processing Results for Noisy Synthetic Range Images						
Filename	Preprocessing		Thresholds			Number of surfaces
	Median Filter?	Fit 2-D B-splines?	distance (meters)	angle (degrees)	curvature (1/meters)	
sm_m60.n1.l	no	no	0.5	20	5	9
sm_m60.n1.lb	no	yes	0.5	20	5	10
sm_m60.n1.lm	yes	no	0.5	20	5	35
sm_m60.n1.lmb	yes	yes	0.5	20	5	71
sm_m60.n2.l	no	no	0.5	20	5	8
sm_m60.n2.lb	no	yes	0.5	20	5	10
sr _m60.n2.lm	yes	no	0.5	20	5	35
sm_m60.n2.lmb	yes	yes	0.5	20	5	72
sm_m60.n3.l	no	no	0.5	20	5	9
sm_m60.n3.lb	no	yes	0.5	20	5	15
sm_m60.n3.lm	yes	no	0.5	20	5	35
sm_m60.n3.lmb	yes	yes	0.5	20	5	73
sm_m60.n4.l	no	no	0.5	20	5	10
sm_m60.n4.lb	no	yes	0.5	20	5	11
sm_m60.n4.lm	yes	no	0.5	20	5	35
sm_m60.n4.lmb	yes	yes	0.5	20	5	68



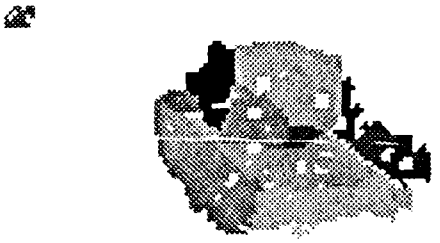
(a) sm_m60.n1



(b) sm_m60.n1.lbl



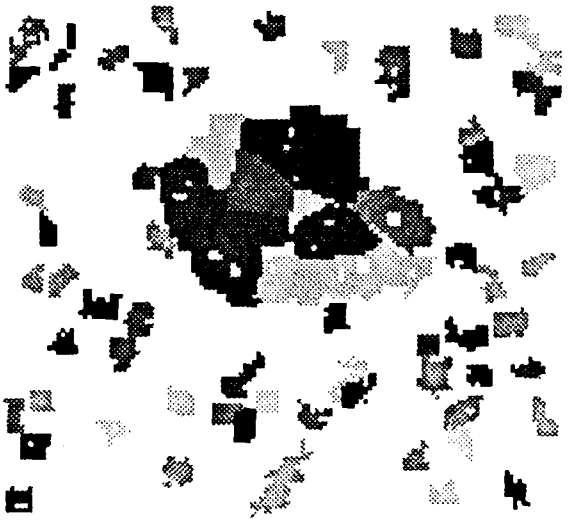
(c) sm_m60.n1.l



(d) sm_m60.n1.lb

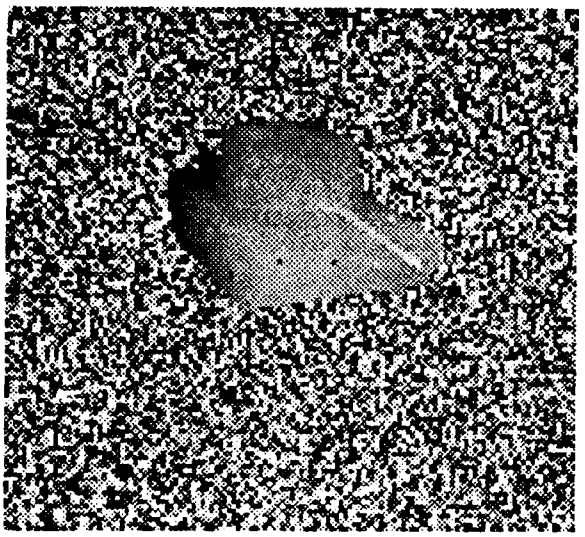


(e) sm_m60.n1.lm



(f) sm_m60.n1.lmb

Figure 3.3.14 Results of processing noisy PADL image with Gaussian standard deviation = 1 and dropout probability = 0.005 using a 5 by 5 processing window and various preprocessing.



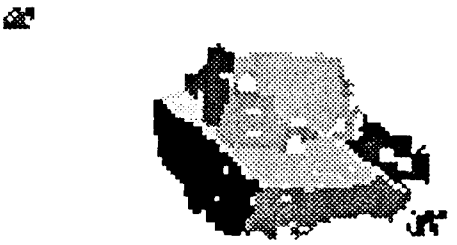
(a) sm_m60.n2



(b) sm_m60.n2.lbl



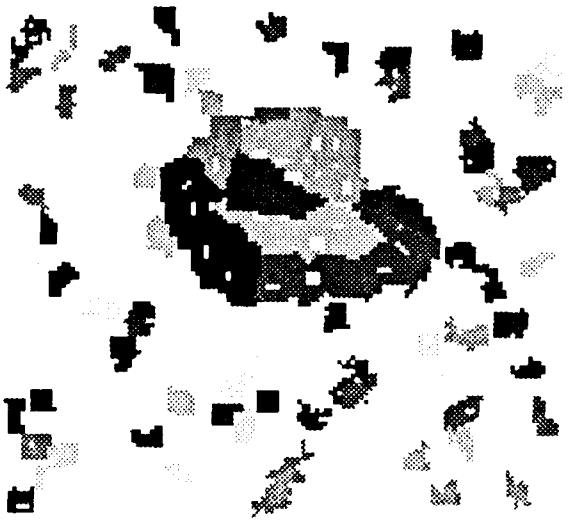
(c) sm_m60.n2.l



(d) sm_m60.n2.lb

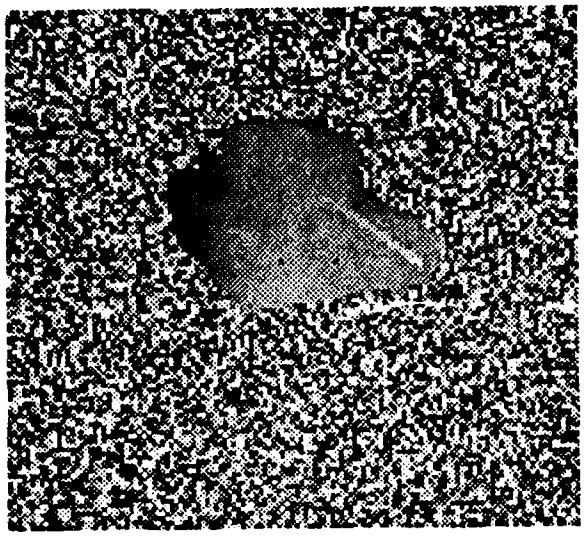


(e) sm_m60.n2.lm

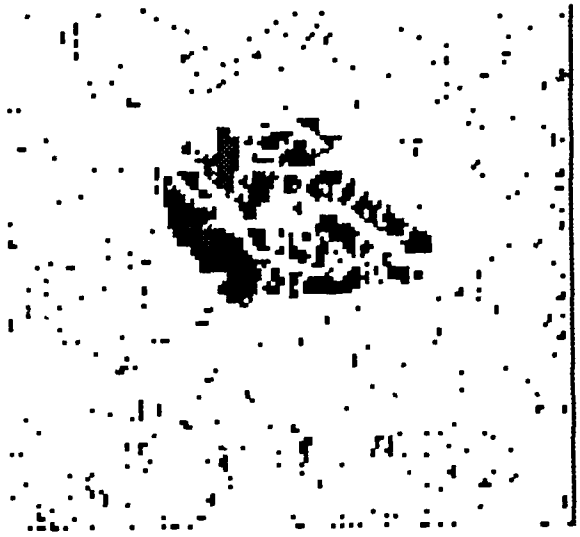


(f) sm_m60.n2.lmb

Figure 3.3.15 Results of processing noisy PADL image with Gaussian standard deviation = 3 and dropout probability = 0.005 using a 5 by 5 processing window and various preprocessing.



(a) sm_m60.n3



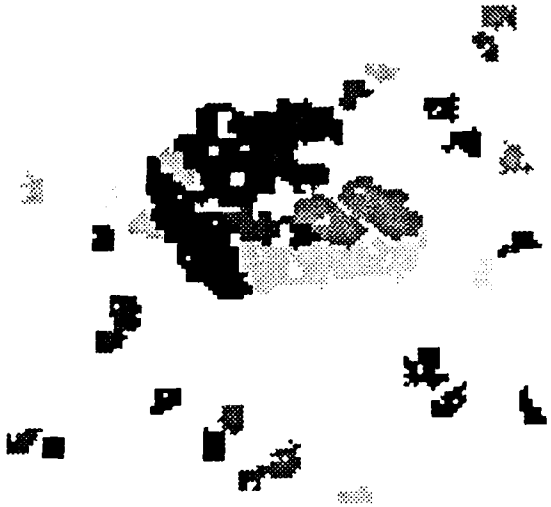
(b) sm_m60.n3.lbl



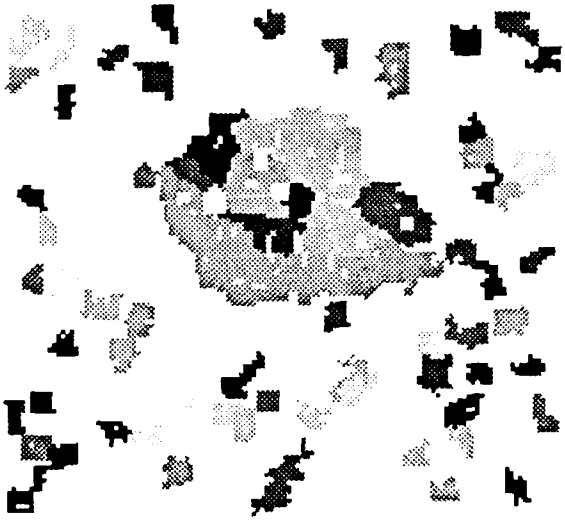
(c) sm_m60.n3.l



(d) sm_m60.n3.lb

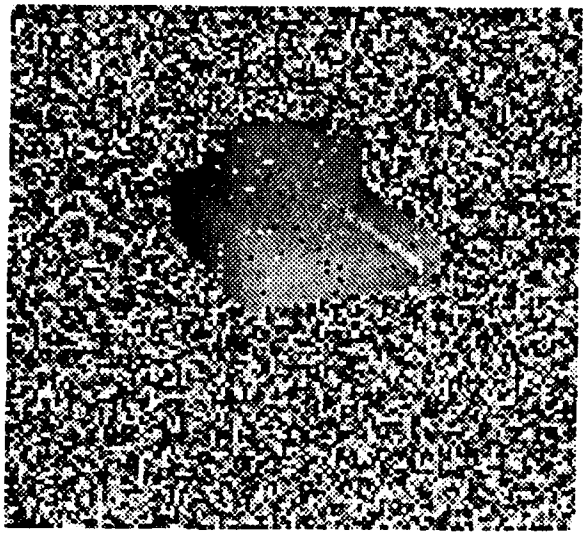


(e) sm_m60.n3.lm

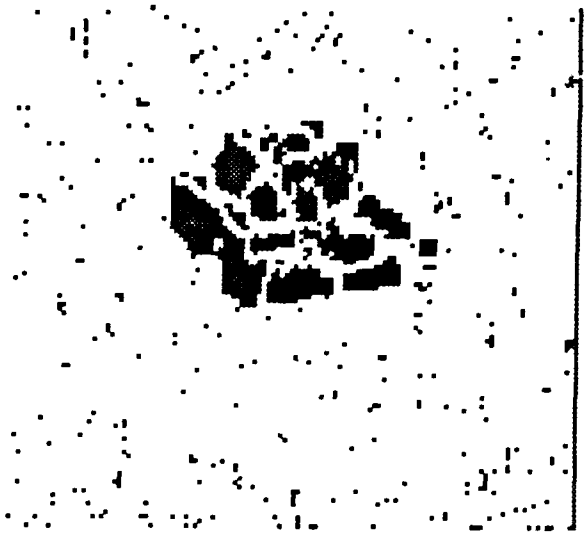


(f) sm_m60.n3.lmb

Figure 3.3.16 Results of processing noisy PADL image with Gaussian standard deviation = 7 and dropout probability = 0.005 using a 5 by 5 processing window and various preprocessing.



(a) sm_m60.n4



(b) sm_m60.n4.lbl



(c) sm_m60.n4.l



(d) sm_m60.n4.lb



(e) sm_m60.n4.lm



(f) sm_m60.n4.lmb

Figure 3.3.17 Results of processing noisy PADL image with Gaussian standard deviation = 3 and dropout probability = 0.05 using a 5 by 5 processing window and various preprocessing.

applied a median filter, and (e) is median filtering followed by fitting B-splines. Files c10_apc and c10_tank were extracted from image c10m1, while c17_apc was extracted from c17m1. Both of the larger images were processed in their entirety, and the results are presented in Figures 3.3.27 through 3.3.36. Part (a) of these figures leal with the combined AM/FM resolved range image, while part (b) shows results obtained by processing the AM component alone. The c10 targets are at a range of 1020m, while c17 targets are 1700m away. Both sets of images were processed using 7 by 7 processing windows for surface normal computation. Table 3.3.3 summarizes the results for the resolved range LADAR images, Table 3.3.4 shows MBRBC corrected AM/FM image results, and Table 3.3.5 contains AM component image statistics.

The resolved range images tended to be too noisy and produced poor results. The MBRBC images did better, producing good results for broad side views of the APCs after additional preprocessing, but poor results for the more complex tank at a 45 degree aspect angle. However, the mod 1875 images containing only the AM component produced good results after a little preprocessing. For example, Figure 3.3.21 shows how the turret and front and side surfaces of a tank were extracted. Median filtering by itself and followed by fitting B-splines proved most effective, while B-splines alone did a poor job because of the problem with dropouts. While mod 1875 images produced good results, the fact that there is periodic wrap around from black to white can cause problems. For instance, the front of c10_apc.AM is barely touching a wrap around point and so has lots of white spot "noise" due to the normal variance of the data. While the median filter cleared most of this up, there is still a hole in the final result caused by what remained.

3.3.2.5. Surface Feature Set & Algorithm Evaluation Measures

In this section we define a preliminary surface feature set in order to bridge the gap between the low and high level processing in our geometric approach, as well as to provide a rigorous means of evaluating our surface extraction algorithms. Our set includes:

(1) *pixel count*, $|S|$, the number of pixels on surface S.

(2) *surface area*, A_S, in square meters.

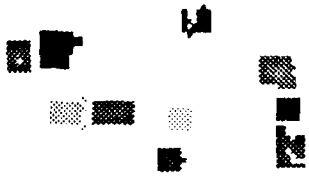
(3) *bounding box*, $B_S = \left[\vec{b}_{\min}, \vec{b}_{\max}\right]_S$, defining the 3-D extent of the surface via its minimum and maximum x, y, and z coordinate values.

(4) *centroid* vector $\vec{r}_S$ for the surface, which may be considered to be its "location." It is computed by averaging the (x,y,z) locations $\vec{r}_p$ over the pixels p comprising the surface S.

(5) *unit normal* vector to the surface, $\hat{n}_S$, computed by averaging the normals of all pixels belonging to the surface. It may be considered as the surface "orientation."



(a) c10_apc AMFM.i



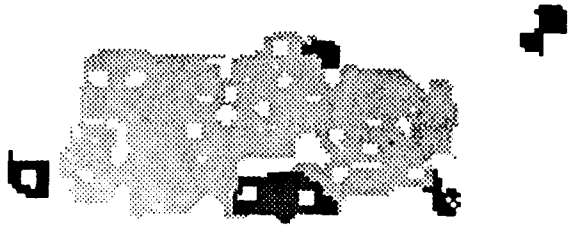
(b) c10_apc.AMFM.l



(c) c10_apc.AMFM.lb



(d) c10_apc.AMFM.lm

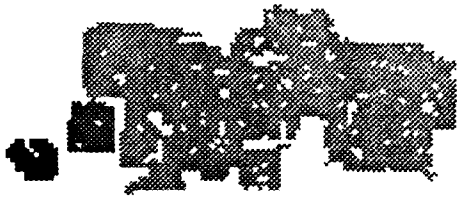


(e) c10_apc.AMFM.lmb

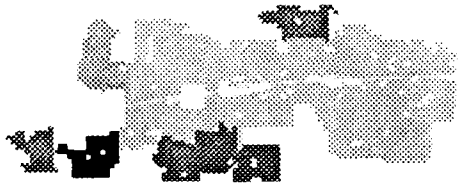
Figure 3.3.18 Results of processing resolved range image of an apc at 1020m.



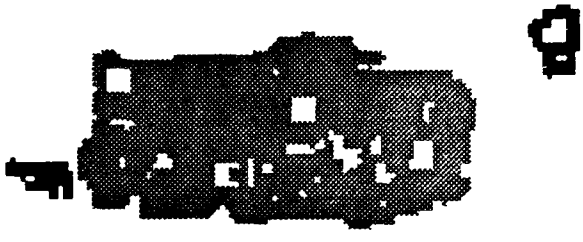
(a) c10_apc.RBC.i



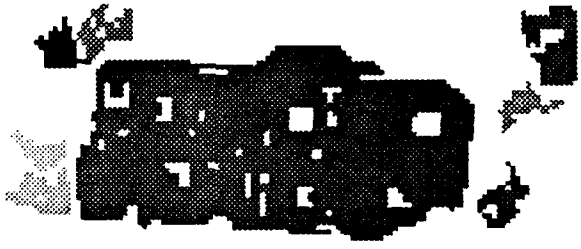
(b) c10_apc.RBC.l



(c) c10_apc.RBC.lb

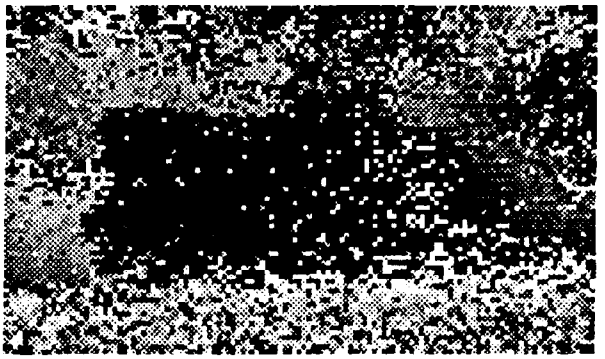


(d) c10_apc.RBC.lm



(e) c10 apc.RBC.lmb

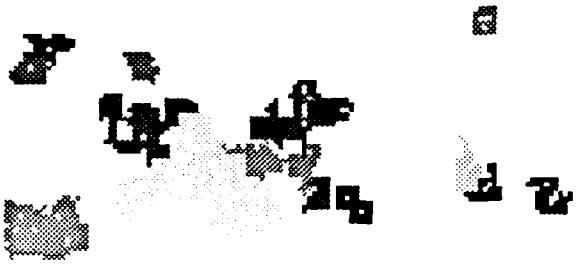
Figure 3.3.19 Results of processing MBRBC range image of an apc at 1020m.



(a) c10_apc.AM.i



(b) c10_apc.AM.l



(c) c10_apc.AM.lb



(d) c10_apc.AM.lm



(e) c10_apc.AM.lmb

Figure 3.3.20 Results of processing AM component range image of an apc at 1020m.



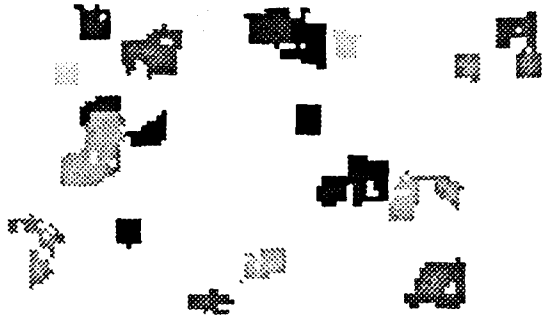
(a) c10_tank.AMFM.i

(b) c10_tank.AMFM.l

(c) c10_tank.AMFM.lb

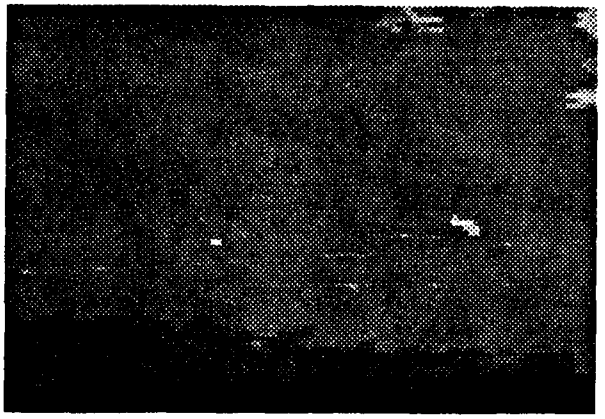


(d) c10_tank.AMFM.lm

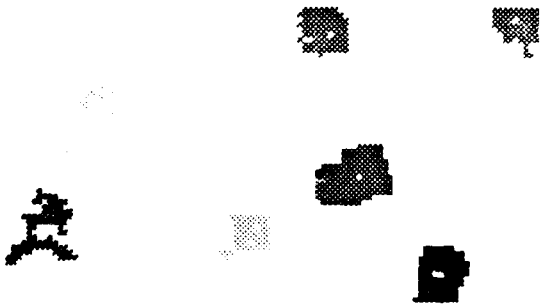


(e) c10_tank.AMFM.lmb

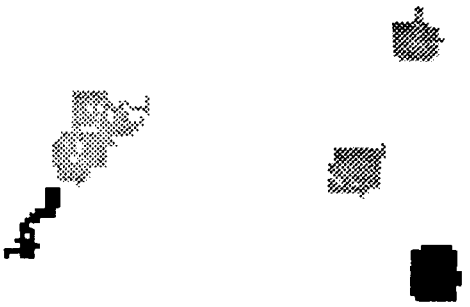
Figure 3.3.21 Results of processing resolved range image of a tank at 1020m.



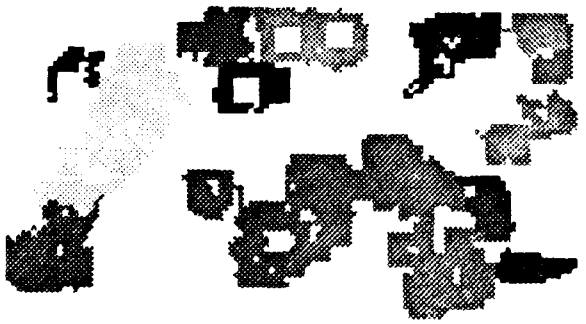
(a) c10_tank.RBC.i



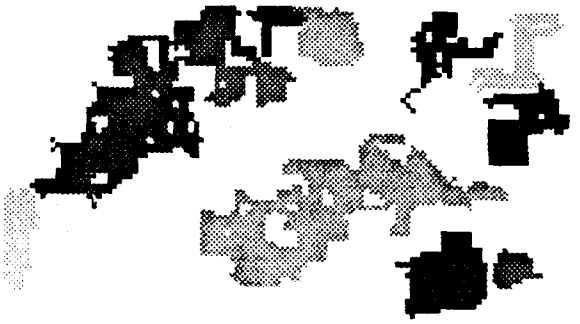
(b) c10_tank.RBC.l



(c) c10_tank.RBC.lb

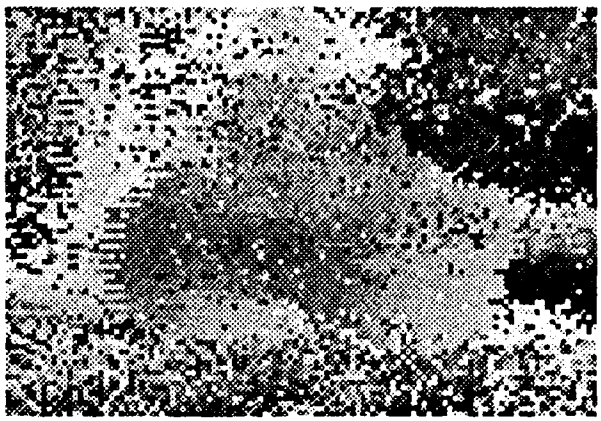


(d) c10_tank.RBC.lm



(e) c10_tank.RBC.lmb

Figure 3.3.22 Results of processing MBRBC range image of a tank at 1020m.



(a) c10_tank.AM.i



(b) c10_tank.AM.l



(c) c10_tank.AM.lb

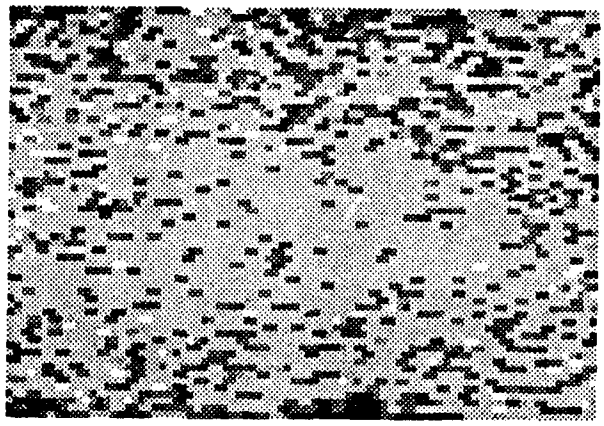


(d) c10_tank.AM.lm



(e) c10_tank.AM.lmb

Figure 3.3.23 Results of processing AM component range image of a tank at 1020m.



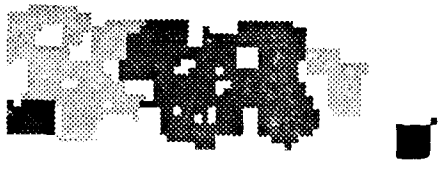
(a) c17_apc.AMFM.i



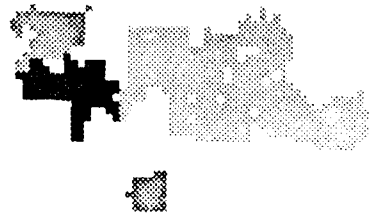
(b) c17_apc.AMFM.l



(c) c17_apc.AMFM.lb

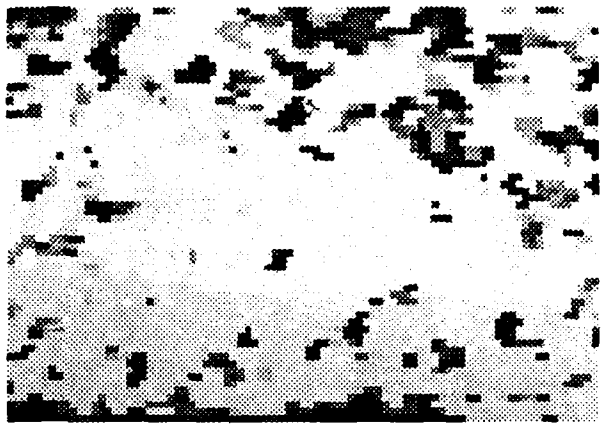


(d) c17_apc.AMFM.lm



(e) c17_apc.AMFM.lmb

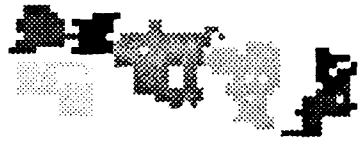
Figure 3.3.24 Results of processing resolved range image of an apc at 1700m.



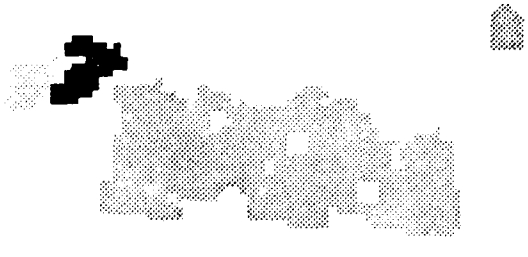
(a) c17_apc.RBC.i



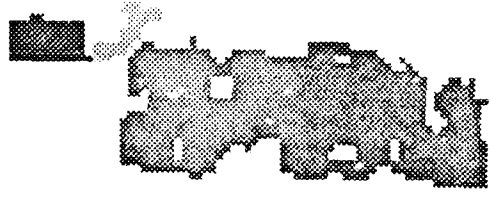
(b) c17_apc.RBC.l



(c) c17_apc.RBC.lb

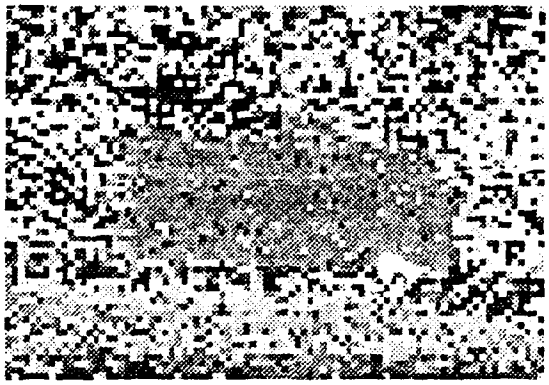


(d) c17_apc.RBC.lm



(e) c17_apc.RBC.lmb

Figure 3.3.25 Results of processing MBRBC range image of an apc at 1700m.



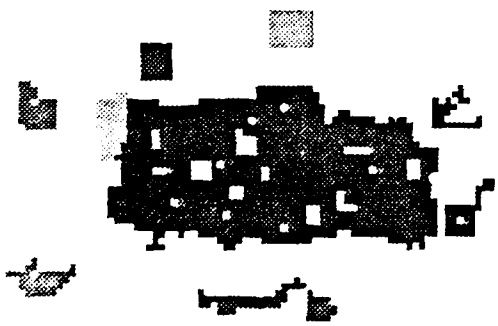
(a) c17_apc.AM.i



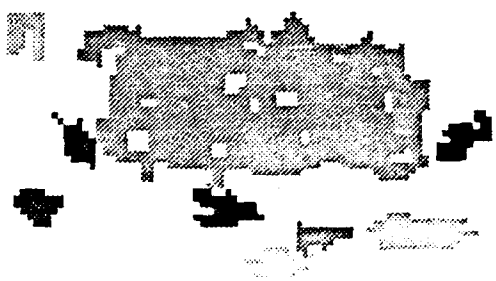
(b) c17_apc.AM.l



(c) c17_apc.AM.lb



(d) c17_apc.AM.lm



(e) c17_apc.AM.lmb

Fig .re 3.3.26 Results of processing AM component range image of an apc at 1700m.

Table 3.3.3 Summary of processing results for AM/FM resolved range LADAR images.

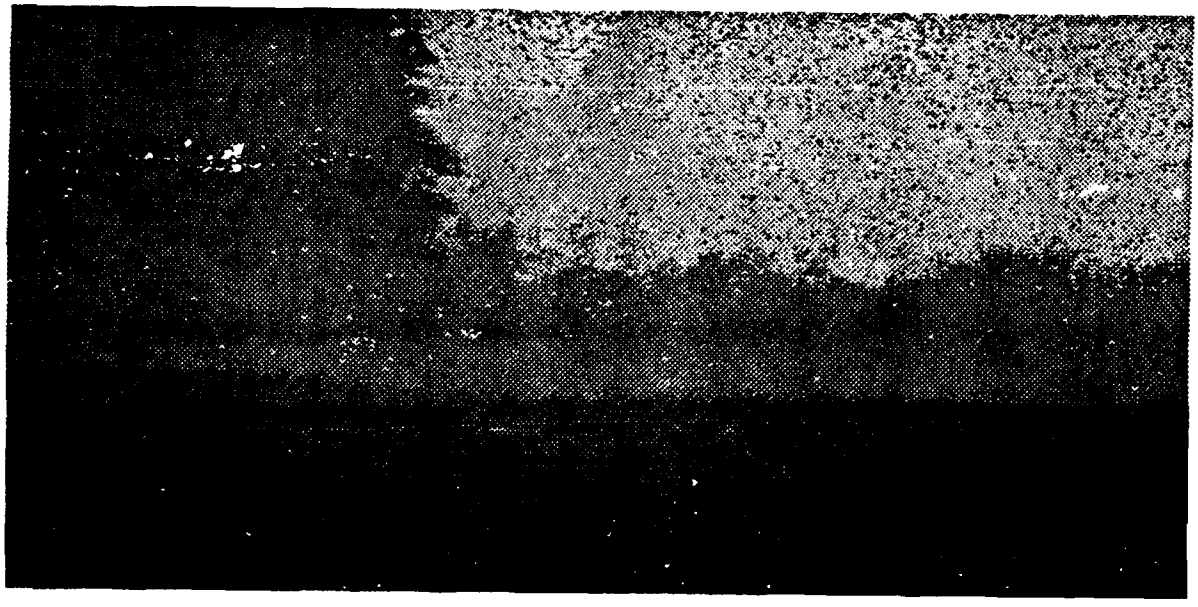
Processing Results for Actual LADAR Range Images						
Filename	Preprocessing		Thresholds			Number of surfaces
	Median Filter?	Fit 2-D B-splines?	distance (meters)	angle (degrees)	curvature (1/meters)	
c10.AMFM.l	no	no	0.3	60	20	2
c10.AMFM.lb	no	yes	0.3	60	20	8
c10.AMFM.lm	yes	no	0.3	60	20	82
c10.AMFM.lmb	yes	yes	0.3	60	20	98
c17.AMFM.l	no	no	0.3	60	20	4
c17.AMFM.lb	no	yes	0.3	60	20	1
c17.AMFM.lm	yes	no	0.3	60	20	39
c17.AMFM.lmb	yes	yes	0.3	60	20	43
c10_apc.AMFM.l	no	no	0.3	60	20	10
c10_apc.AMFM.lb	no	yes	0.3	60	20	5
c10_apc.AMFM.lm	yes	no	0.3	60	20	8
c10_apc.AMFM.lmb	yes	yes	0.3	60	20	6
c10_tank.AMFM.l	no	no	0.3	60	20	1
c10_tank.AMFM.lb	no	yes	0.3	60	20	0
c10_tank.AMFM.lm	yes	no	0.3	60	20	31
c10_tank.AMFM.lmb	yes	yes	0.3	60	20	22
c17_apc.AMFM.l	no	no	0.3	60	20	1
c17_apc.AMFM.lb	no	yes	0.3	60	20	1
c17_apc.AMFM.lm	yes	no	0.3	60	20	5
c17_apc.AMFM.lmb	yes	yes	0.3	60	20	4

Table 3.3.4 Summary of processing results for 3x3 MBRBC of combined AM/FM LADAR images.

Processing Results for MBRBC LADAR Range Images						
Filename	Preprocessing		Thresholds			Number of surfaces
	Median Filter?	Fit 2-D B-splines?	distance (meters)	angle (degrees)	curvature (1/meters)	
c10_apc.RBC.l	no	no	0.3	60	20	3
c10_apc.RBC.lb	no	yes	0.3	60	20	6
c10_apc.RBC.lm	yes	no	0.3	60	20	4
c10_apc.RBC.lmb	yes	yes	0.3	60	20	8
c10_tank.RBC.l	no	no	0.3	60	20	7
c10_tank.RBC.lb	no	yes	0.3	60	20	5
c10_tank.RBC.lm	yes	no	0.3	60	20	13
c10_tank.RBC.lmb	yes	yes	0.3	60	20	15
c17_apc.RBC.l	no	no	0.3	60	20	7
c17_apc.RBC.lb	no	yes	0.3	60	20	8
c17_apc.RBC.lm	yes	no	0.3	60	20	5
c17_apc.RBC.lmb	yes	yes	0.3	60	20	4

Table 3.3.5 Summary of processing results for AM component of LADAR range images.

Processing Results for Mod 1875 LADAR Range Images						
Filename	Preprocessing		Thresholds			Number of surfaces
	Median Filter?	Fit 2-D B-splines?	distance (meters)	angle (degrees)	curvature (1/meters)	
c10.AM.l	no	no	0.3	60	20	57
c10.AM.lb	no	yes	0.3	60	20	110
c10.AM.lm	yes	no	0.3	60	20	241
c10.AM.lmb	yes	yes	0.3	60	20	249
c17.AM.l	no	no	0.3	60	20	22
c17.AM.lb	no	yes	0.3	60	20	30
c17.AM.lm	yes	no	0.3	60	20	145
c17.AM.lmb	yes	yes	0.3	60	20	164
c10_apc.AM.l	no	no	0.3	60	20	10
c10_apc.AM.lb	no	yes	0.3	60	20	15
c10_apc.AM.lm	yes	no	0.3	60	20	27
c10_apc.AM.lmb	yes	yes	0.3	60	20	25
c10_tank.AM.l	no	no	0.3	60	20	25
c10_tank.AM.lb	no	yes	0.3	60	20	23
c10_tank.AM.lm	yes	no	0.3	60	20	44
c10_tank.AM.lmb	yes	yes	0.3	60	20	32
c17_apc.AM.l	no	no	0.3	60	20	10
c17_apc.AM.lb	no	yes	0.3	60	20	7
c17_apc.AM.lm	yes	no	0.3	60	20	9
c17_apc.AM.lmb	yes	yes	0.3	60	20	13



(a) c10.AMFM



(b) c10.AM

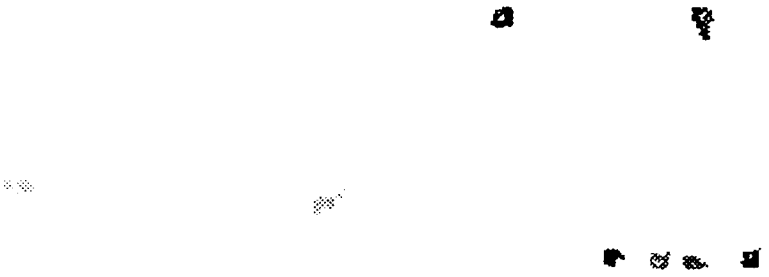
Figure 3.3.27 1020m LADAR images (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c10.AMFM.l

(b) c10.AM.l

Figure 3.3.28 Processing results for 1020m LADAR images, no preprocessing (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c10.AMFM.lb

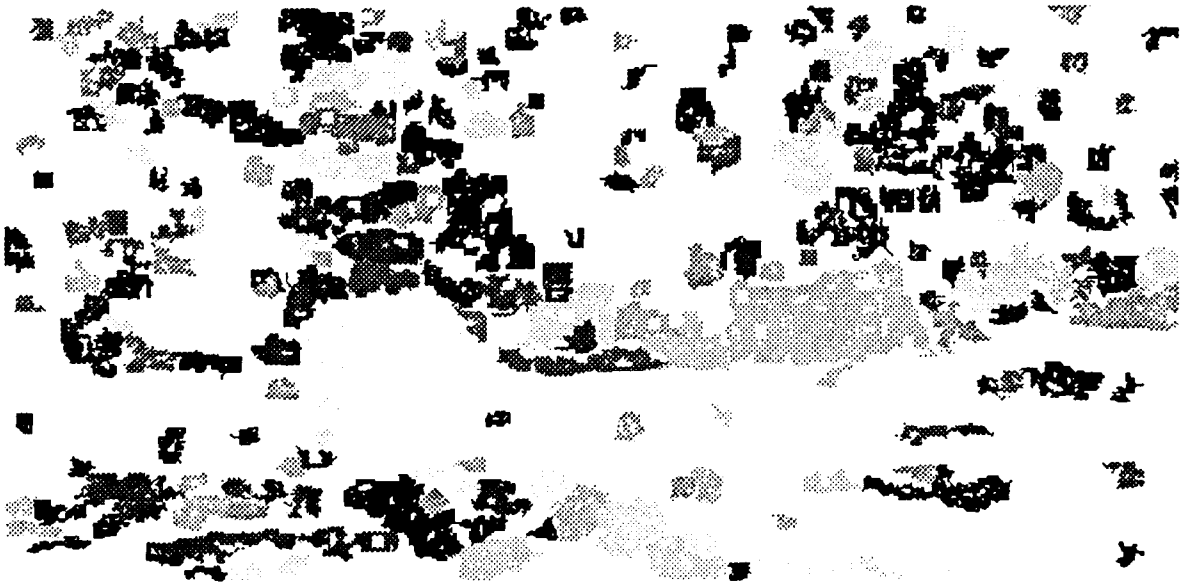


(b) c10.AM.lb

Figure 3.3.29 Processing results for 1020m LADAR images, B-splines only (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c10.AMFM.lm

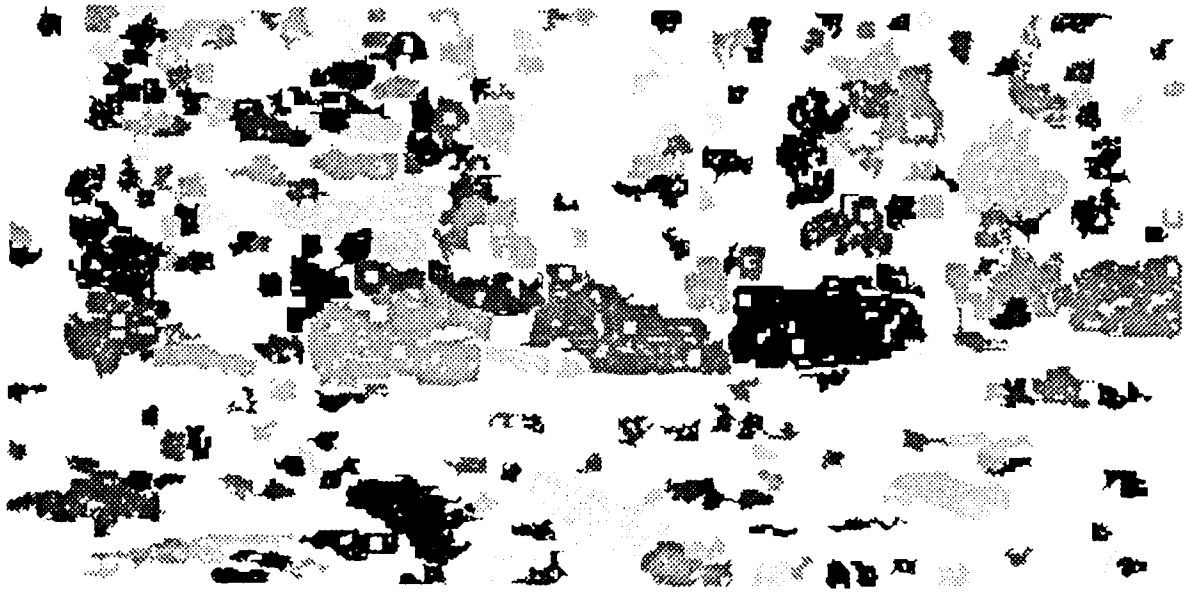


(b) c10.AM.lm

Figure 3.3.30 Processing results for 1020m LADAR images, median filter only (a) AM/FM resolved range (b) mod 1875 AM component.

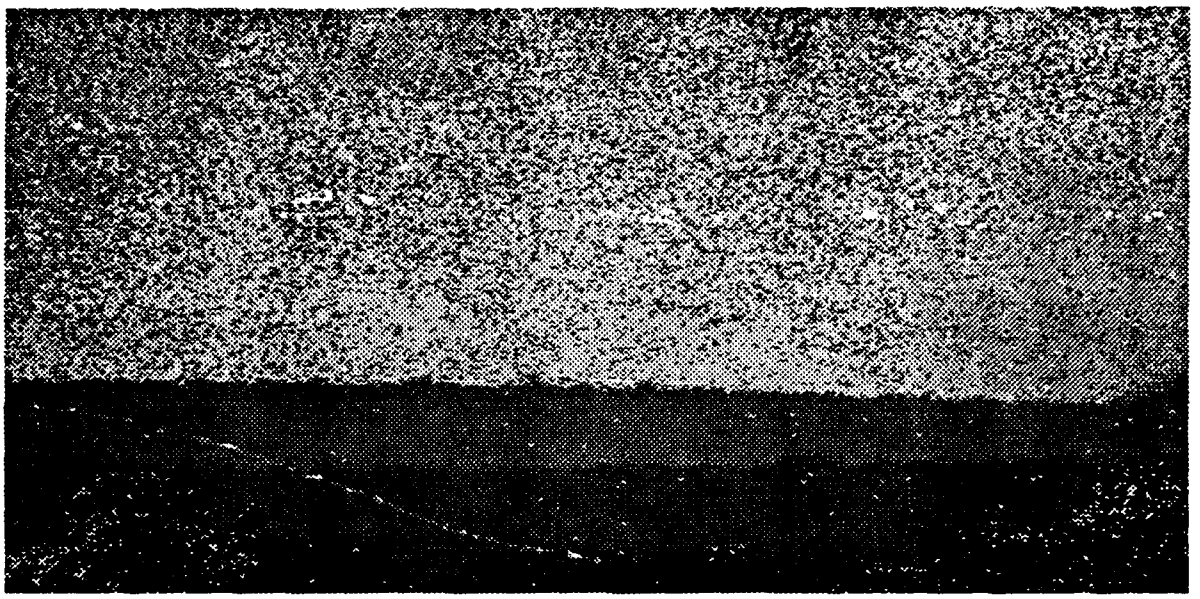


(a) c10.AMFM.lmb

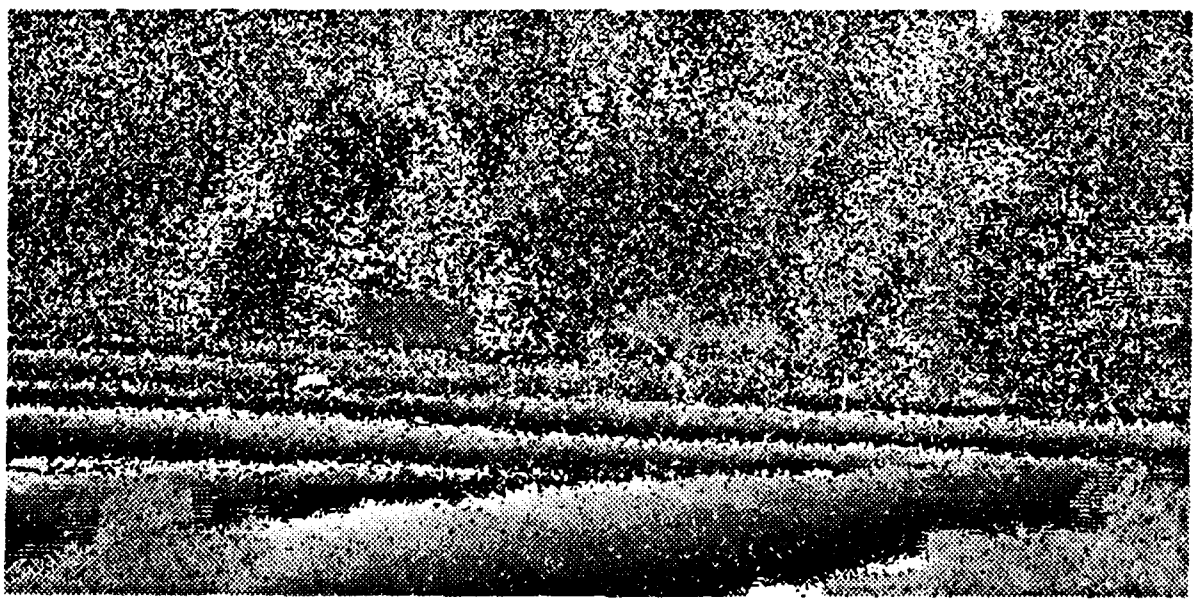


(b) c10.AM.lmb

Figure 3.3.31 Processing results for 1020m LADAR images, median filter & B-splines (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c17.AMFM



(b) c17.AM

Figure 3.3.32 1700m LADAR images (a) AM-FM resolved range (b) mod 1875 AM component.



(a) c17.AMFM.1

(b) c17.AM.1

Figure 3.3.33 Processing results for 1700m LADAR images, no preprocessing (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c17.AMFM.lb

(b) c17.AM.lb

Figure 3.3.34 Processing results for 1700m LADAR images, B-splines only (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c17.AMFM.lm



(b) c17.AM.lm

Figure 3.3.35 Processing results for 1700m LADAR images, median filter only (a) AM/FM resolved range (b) mod 1875 AM component.



(a) c17.AMFM.lmb



(b) c17.AM.lmb

Figure 3.3.36 Processing results for 1700m LADAR images, median filter & B-splines (a) AM/FM resolved range (b) mod 1875 AM component.

(6) *local planar fit mean square error, E_l.* When surface normals are computed for each pixel by fitting a planar patch to the data points within the processing window, the mean square fit error of the patch is also computed and associated with the pixel. The average of this fit error taken over the pixels comprising a surface is called its local planar fit error. It is a measure of how well the surface was represented by its composite patches.

(7) *global planar fit mean square error, E_g,* measured between the pixels of a surface S and the plane specified by its location (centroid $\vec{r}_s$) and orientation (unit normal $\hat{n}_s$). This feature is indicative of the overall planarity of the surface.

(8) *noise of location, L_n,* is computed by averaging over pixels $p \in S$ whose 4-neighbors are all also in surface S, the distance in meters between each pixel location $\vec{r}_p$ and the centroid of its 4-neighbors $\vec{r}_4$.

(9) *noise of orientation, O_n,* is the average taken over pixels $p \in S$ of the angle in degrees between the individual pixel's surface normal $\hat{n}_p$ and the average normal $\hat{n}_s$ of surface S.

Features 1-5 yield basic geometric properties of the surfaces, while 6-9 indicate image quality by measuring noise. Features 6 and 7 also provide a limited idea of surface shape by measuring local and global planarity. Surface relations (e.g. region adjacency) and additional features (e.g. perimeter) will be implemented in the future.

We now define several measures which will prove useful in evaluating our processing algorithms. In order to determine how successfully we have extracted a surface from an image, we must measure how it differs from the ground truth. This ground truth information is obtained from the target model after it has been transformed to coincide with the position and orientation of the target in the image. Each extracted target surface is compared with its corresponding model surface, checking both its position and orientation. We may therefore define *locational disparity*

$$L_d = \frac{|\vec{r}_s - \vec{r}_m|}{\bar{z}}$$

as the distance between the extracted surface location, specified by its centroid $\vec{r}_s$, and the model surface location, specified by its centroid $\vec{r}_m$, normalized by the range of the surface $\bar{z}$. We take range $\bar{z}$ to be the z- component of $\vec{r}_m$. *Orientation disparity* may also be defined as

$$O_d = \frac{1 - \hat{n}_s \cdot \hat{n}_m}{2}$$

where $\hat{n}_s$ is the extracted surface's unit normal, specifying its orientation, $\hat{n}_m$ specifies the model surface's orientation, and $O_d \in [0,1]$ indicates the disparity between them. We also

define *model fit error*

$$E_f = \frac{\sum_{p \in S} (\vec{r}_p \cdot \hat{n}_m - \vec{r}_m \cdot \hat{n}_m)^2}{|S|}$$

as the mean square fit error between the *(x,y,z)* locations $\vec{r}_p$ of the pixels p in the extracted surface S and the plane specified by the corresponding model surface's location $\vec{r}_m$ and orientation $\hat{n}_m$. E_f not only determines how well our extracted surface corresponds to the model surface, but also may be used as a measure of the noise present in the image.

3.3.2.6. Algorithm Evaluation Experiment

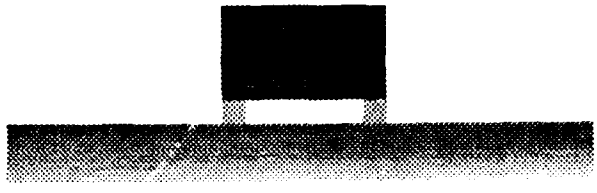
In order to evaluate the performance of our algorithms, we use as test images the set of range images corresponding to the shaded images in Figure 3.3.37. These high resolution range images were generated using the new ETBM built upon the TWIN Solid Modeling Package, and contained no additive noise. The images are shown in increasing order of complexity. Figure 3.3.37(a) is an M113 APC placed on a flat ground plane and viewed from broadside. Figure 3.3.37(b) is an M113 on a flat ground plane viewed from head on. Figure 3.3.37(c) is the more complex M60A1 tank placed on a gently rolling ground plane and viewed obliquely. Figure 3.3.37(d) contains a handful of relatively complex targets at random aspects placed on rugged terrain.

At the same time TWIN generates a range image, it also generates two ground truth data files. The first of these two files contains a ground truth surface label image, where the value of each pixel in the image is the label of the model surface to which it belongs. This information is illustrated in Figures 3.3.38 through 3.3.41. The second file contains the ground truth values of geometric features $|M|$, A_m, B_m, $\vec{r}_m$, and $\hat{n}_m$ for each model surface M.

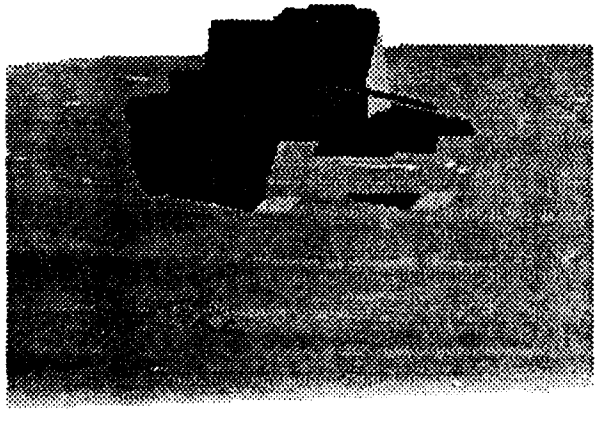
Our surface extraction routines were run on this test set. Table 3.3.6a and 3.3.6b summarize the surface features computed for the M113 APC of Figure 3.3.37(b), and serve as an example of feature computation. Note that only surfaces comprising the target are listed (8 surfaces were found in the entire image), and that the surface labels listed in the table are *extracted* surface labels (S_i's), and do not correspond to the *model* surface numbers (M_i's) in Figure 3.3.39. Once surface features are computed, we are ready to evaluate the processing results by first computing L_d, O_d, and E_f for each target surface extracted, and then determining which model surfaces went undetected. In order to compute the three performance measures for extracted surface S, we must first match it to the appropriate model surface M. This is accomplished by using the x and y components of $\vec{r}_s$ to index into the model surface label image. This approach is valid as long as the projections of our object surfaces onto the image plane have convex borders, and no projected surface completely surrounds another projected surface. This is true for our test data set.



(a) m113-90.shade



(b) m113.shade

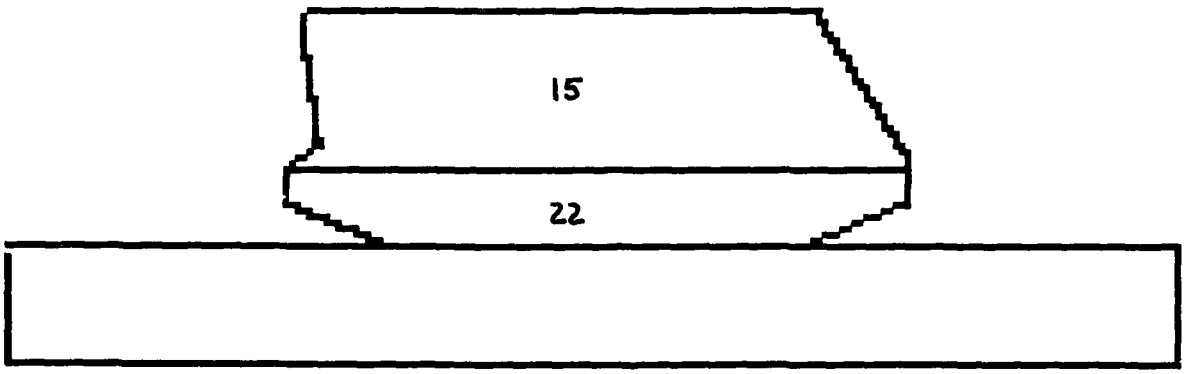


(c) m60a1.shade



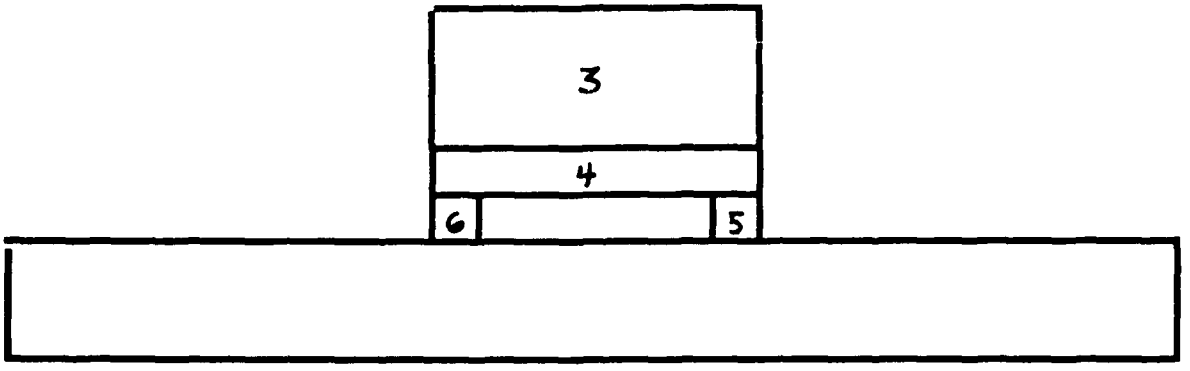
(d) complex.shade

Figure 3.3.37 Test images for surface extraction algorithm evaluation experiment.



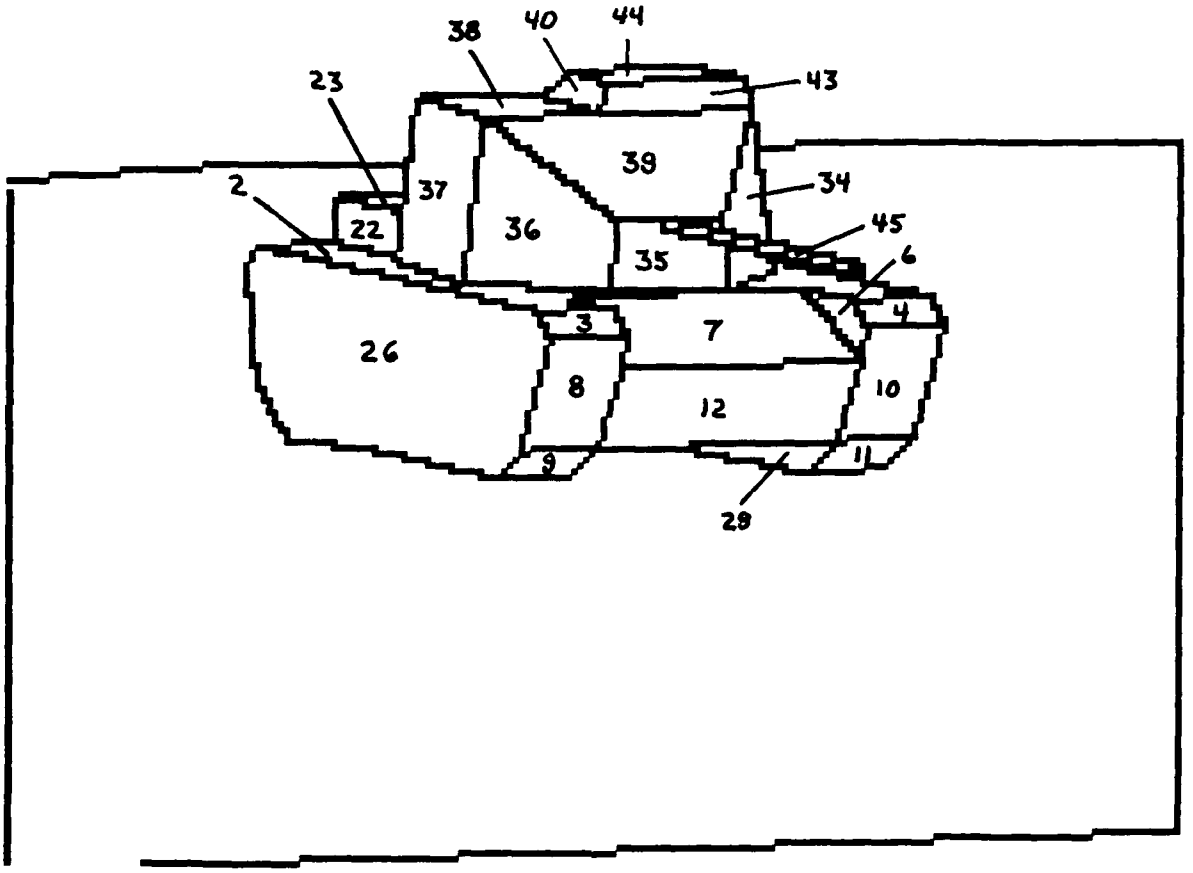
m113-90.edges

Figure 3.3.38 Model surface labels for M113 apc broadside view test image.



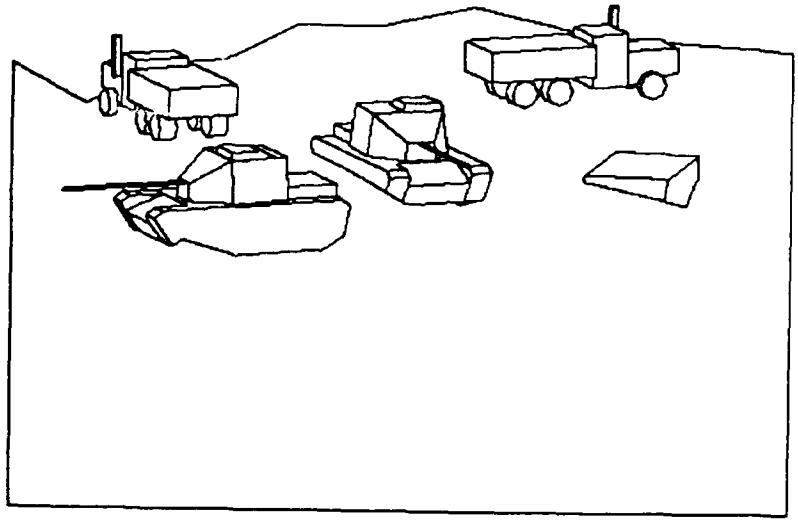
m113.edges

Figure 3.3.39 Model surface labels for M113 apc head-on view test image.

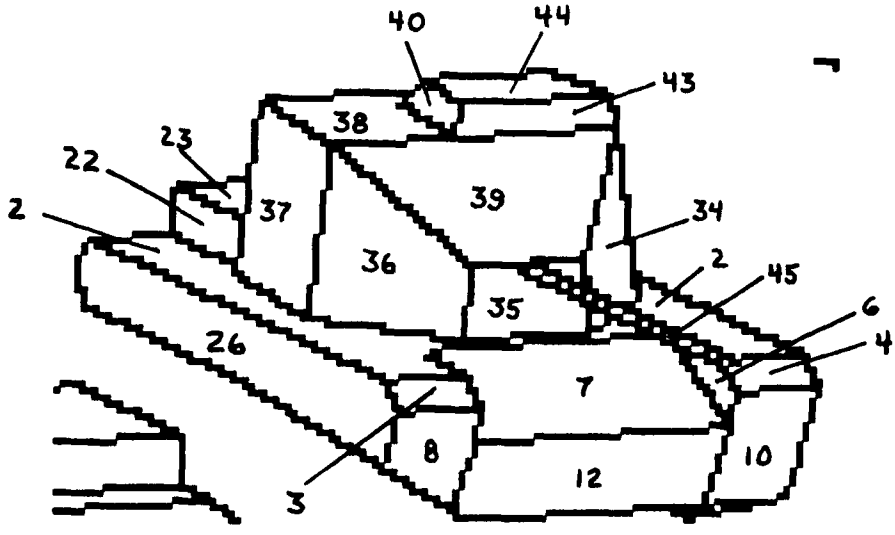


m60a1.edges

Figure 3.3.40 Model surface labels for M60A1 tank test image.

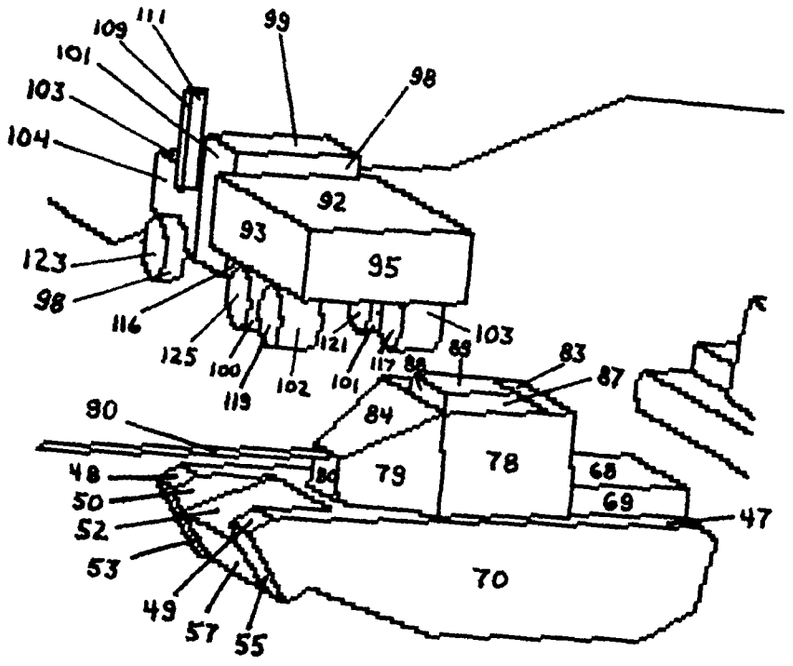


(a) complex.edges

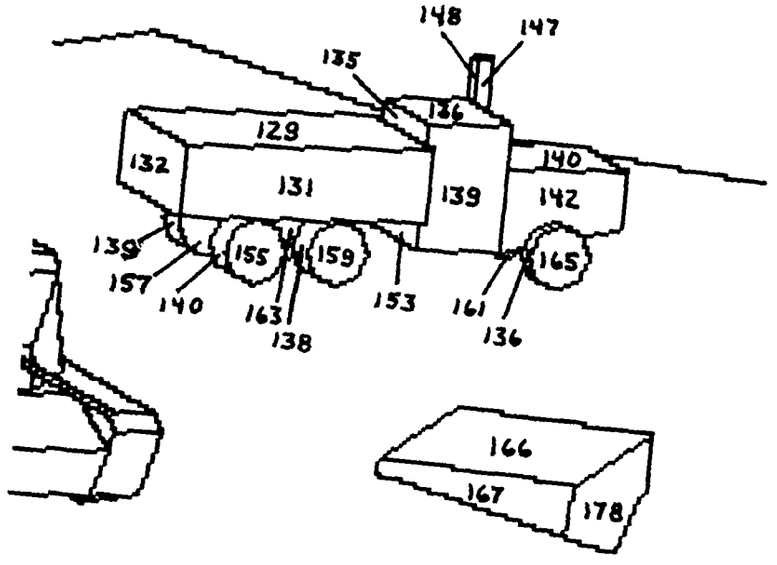


(b) complex.sub1

Figure 3.3.41 Model surface labels for complex, multiple target test image.



(c) complex.sub2



(d) complex.sub3

Figure 3.3.41 continued.

Table 3.3.6a Summary of "Geometric" Surface Features for Figure 3.3.37(b).

Extracted Target Surface Features - Part 1					
S	$\|S\|$	A_s	B_s	$\vec{r}_s$	$\hat{n}_s$
S3	64	0.312819	$\begin{bmatrix} 6.05 & 6.4 \\ 4.35 & 4.7 \\ 1007.35 & 1008.17 \end{bmatrix}$	$\begin{bmatrix} 6.225 \\ 4.525 \\ 1007.76 \end{bmatrix}$	$\begin{bmatrix} -9.63622e-06 \\ 0.920119 \\ -0.39164 \end{bmatrix}$
S5	64	0.312819	$\begin{bmatrix} 3.65 & 4 \\ 4.35 & 4.7 \\ 1007.35 & 1008.17 \end{bmatrix}$	$\begin{bmatrix} 3.825 \\ 4.525 \\ 1007.76 \end{bmatrix}$	$\begin{bmatrix} -9.63622e-06 \\ 0.920119 \\ -0.39164 \end{bmatrix}$
S6	448	0.962511	$\begin{bmatrix} 3.65 & 6.4 \\ 3.95 & 4.3 \\ 1007.35 & 1007.35 \end{bmatrix}$	$\begin{bmatrix} 5.025 \\ 4.125 \\ 1007.35 \end{bmatrix}$	$\begin{bmatrix} 4.86298e-05 \\ -0.000134476 \\ -1 \end{bmatrix}$
S7	1344	3.72941	$\begin{bmatrix} 3.65 & 6.4 \\ 2.75 & 3.9 \\ 1007.38 & 1008.1 \end{bmatrix}$	$\begin{bmatrix} 5.025 \\ 3.325 \\ 1007.74 \end{bmatrix}$	$\begin{bmatrix} 2.78105e-05 \\ -0.529884 \\ -0.84807 \end{bmatrix}$

Table 3.3.6b Summary of "Image Quality" Surface Features for Figure 3.3.37(b).

Extracted Target Surface Features - Part 2				
S	E_l	E_g	L_n	O_n
S3	1.37836e-08	5.34058e-05	5.72205e-06	0.0252654
S5	1.37836e-08	4.81606e-05	5.72205e-06	0.0252654
S6	3.35276e-08	0.00399944	3.02451e-05	0
S7	3.40066e-08	0.00217102	3.84194e-05	0.0219154

Once all extracted surfaces have been matched, the remaining unmatched model surfaces are flagged as being undetected. Tables 3.3.7a-d contain the evaluation measures computed for our test set and a list of undetected surfaces. Overall, the results are very good. All processing errors were caused by one of the following cases:

i) surfaces whose height or width or both were on the order of the processing window dimensions. This condition results in higher E_f due to merged surfaces or moved pixels in the modified range map. The modified range map becomes a factor because the number of border pixels (where most of the moving takes place in clean imagery) is close to the total number of pixels on the surface for small surfaces. Although E_f increases for problems caused by small surfaces, O_d and L_d remain small and good estimates of $\hat{n}_m$ and $\vec{r}_m$ are obtained.

ii) small surfaces generated where objects meet ground plane. This problem occurred infrequently and depended on the slope of the ground patch, the processing window size, and *ang_thr*. It is easily recognized since multiple extracted surfaces are matched to the same model surface.

iii) overdetailed object models. In several cases, the target models indicate separate surfaces where no true edge in the rendered image exists, either because of a lack of sufficient jump discontinuity or surface normal disparity. For instance, the distinction between surfaces 15 and 22 in the broadside view of the M113 is overdetailed since no sensible edge exists between them. High L_d and low O_d and E_f help identify an object model problem.

3.3.2.7. Conclusions and Future Work

In conclusion, we see that region growing is an appropriate approach for surface extraction for actual LADAR data. As far as preprocessing is concerned, it is apparent that median filtering is a mandatory step. We now have in place the tools needed to evaluate whether or not the fitting of B-splines after median filtering improves the quality of our results, and whether the modified range map helps. We have seen that median-based range bin correction provided only marginal improvement. An examination of the noise in the 1987 A.P. Hill data will allow us to improve our sensor noise model and aid in the selection of better preprocessing methods.

While our 2-pass implementation of region growing is more efficient, it may be more beneficial to use a standard recursive region growing procedure that can keep track of the aggregate surface normal for the surface it is currently working on. A neighboring pixel would be annexed only if it satisfied an additional fourth constraint specifying that its surface normal point in the same relative direction as the aggregate surface normal. This would allow the routine to eventually stop growing a surface past an edge blurred by preprocessing or obscured by noise.

Table 3.3.7a Summary of Evaluation Measures for Figure 3.3.37(a).

Evaluation Measures for M113 APC Broadside View				
S	M	O_d	L_d	E_f
S3	M15	0	0.000200208	0
Undetected Model Surfaces: 22				

Table 3.3.7b Summary of Evaluation Measures for Figure 3.3.37(b).

Evaluation Measures for M113 APC Head-on View				
S	M	O_d	L_d	E_f
S3	M5	0.00037545	0.00156411	2.16653
S5	M6	0.00037545	0.00156411	2.16653
S6	M4	0	0.000797031	0
S7	M3	0	0.000819187	0.179751
Undetected Model Surfaces: none				

Table 3.3.7c Summary of Evaluation Measures for Figure 3.3.37(c).

Evaluation Measures for M60A1 Tank Oblique View				
S	M	O_d	L_d	E_f
S43	M29	0.00325307	0.00273358	0.00235579
S44	M29	0.371073	0.00112623	0.0238056
S45	M26	0.10029	0.00112859	0.0559099
S62	M12	0.00124243	0.00060535	0.200608
S65	M3	0	0.0019762	2.5064
S67	M4	0	0.00217118	3.03086
S68	M7	0.00130308	0.00190204	2.93268
S77	M34	0.00249407	0.00330262	0.086582
S78	M26	0	0.00135086	0.0022219
S83	M35	0.00226185	0.00323787	0.102651
S99	M34	0.00244948	0.00386266	0.0882066
S100	M36	0	0.00358067	0.0131025
S101	M37	0.000524372	0.00414599	0.213152
S102	M39	0	0.00465199	13.1433
S103	M43	0.00219193	0.0058059	0.27924
S104	M40	0.00175837	0.00586515	0.687514
Undetected Model Surfaces: 2 6 8 9 10 11 22 23 38 44 45 46				

Table 3.3.7d Summary of Evaluation Measures for Figure 3.3.37(d).

Evaluation Measures for Multiple Target Image				
S	M	O_d	L_d	E_f
S30	M52	0.402822	0.00452281	13.1401
S31	M57	0	0.00540025	12.5729
S38	M50	0.00154132	0.00367961	0.391368
S43	M52	0.00128183	0.00396174	13.3717
S47	M80	8.86619e-05	0.00280539	0.0336571
S52	M68	0.00182018	0.00218754	4.50247
S53	M167	0.0209292	0.00265962	1.40193
S54	M12	0	0.00283225	4.67118
S56	M79	0	0.00273736	0.0378982
S58	M4	0.00313321	0.00127112	1.08883
S61	M178	0.00272781	0.00247508	0.00905435
S62	M2	0.0273056	0.00254277	0.0597504
S63	M2	0.00180414	0.00204464	0.0552833
S64	M70	0.00508827	0.00504109	0.880138
S65	M2	0.0287021	0.00178398	0.0628618
S66	M84	0	0.00137135	1.28281
S67	M166	0.00180656	0.000942693	0.860649
S68	M88	0	0.000457099	0.00132486
S69	M89	0.0018284	0.000171174	0.0309367
S70	M35	0.00247163	0.000110612	0.003139
S71	M26	0.0839526	0.00166672	0.223233
S73	M7	0.00395471	0.00166675	2.6894
S78	M102	0.853339	0.00145082	1.62057
S80	M103	0.857945	0.00167102	2.25506
S84	M117	0	0.00135749	0.0119419
S85	M121	0	0.00166185	0.0214856
S86	M102	0.492381	0.00178154	2.49839
S88	M103	0.526422	0.00199583	3.20104
S90	M36	0.00309047	0.000338387	0.000281739
S91	M34	0.00208801	0.000562763	0.00423172
S92	M100	0.598341	0.00223298	4.27224

Table 3.3.7d continued.

Evaluation Measures for Multiple Target Image (cont.)				
S	M	O_d	L_d	E_f
S96	M98	0.853361	0.00289243	7.45927
S97	M39	0.00050348	0.00143146	0.997297
S98	M37	0.00150022	0.00128127	0.0117693
S103	M43	0.00158155	0.00270667	0.216993
S104	M38	0.00181633	0.00274001	7.43183
S105	M40	0.0025796	0.00285629	0.0941295
S106	M44	0.00193286	0.00314221	9.66038
S107	M98	0.496242	0.00324824	9.35153
S108	M123	0	0.00268549	0.0393614
S109	M95	0.00153363	0.0028051	0.231066
S111	M140	0.49757	0.00344057	10.1276
S112	M157	0.0167855	0.00405866	0.602413
S114	M138	0.48768	0.00361747	11.2488
S115	M153	0	0.00469583	0.119958
S116	M93	0	0.00305897	0.0580622
S119	M92	0.00180712	0.00453183	20.8193
S122	M98	0.695915	0.00505758	14.1323
S124	M104	0.000269055	0.00396092	0.0613279
S127	M99	0.0400915	0.00520749	19.4405
S132	M111	0.00310796	0.00565898	0.947669
S133	M132	0	0.00567215	0.174822
S134	M140	0.96009	0.00679415	37.2564
S135	M129	0.00180465	0.00692986	48.5121
S136	M131	0.00159964	0.00558704	1.18194
S138	M135	0.303765	0.00622982	26.0463
S139	M136	0.959486	0.0058219	22.9939
S141	M147	0.00285903	0.00828083	1.92179

Undetected Model Surfaces:

3 6 8 10 11 22 23 29 45 46 47 48 49 53 55 69 78
83 87 90 91 101 107 109 116 119 125 139 142 144
148 155 159 161 163 165 185

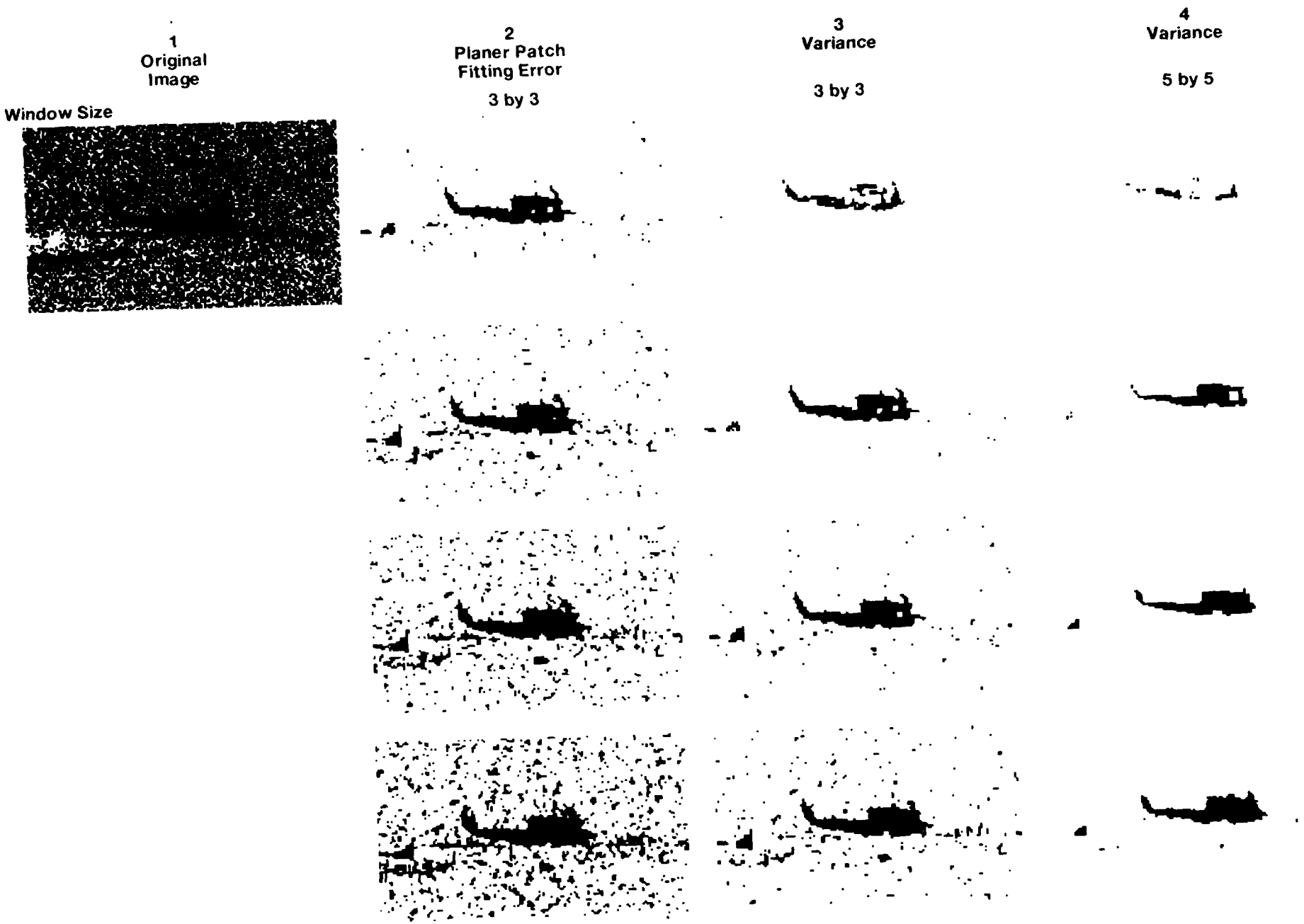


Figure 3.3.42 Output of segmenters for target ap1.32403.

Our evaluation experiment has validated the use of our surface extraction approach. Except for a few incidences which were explainable by three special cases, very low values for our evaluation measures were obtained on clean, truthed data. This indicates that our algorithms are able to accurately extract surfaces present in LADAR imagery and do not introduce errors which may corrupt later processing. As has been seen, the measures provide a means to evaluate our models and the quality of our data as well as our surface extraction algorithms. We are encouraged by the results obtained thus far, and feel that the pursuit of our current approach holds much promise.

3.3.3. A Study of Five Laser Radar Range Data Segmenters

After the targets are detected in an image, the images need to be segmented to separate the targets from each other and from the background. In this section we examine five different range data segmenters to identify their strong and weak points in segmenting laser radar range data. Although detection might be possible using single scan lines, the segmenters used here assume the sensor recorded the targets in imaging mode giving the same vertical resolution as horizontal.

Each of the segmenters was tested in the following way. First, the six range images selected from the A. P. Hill data set described in Section 3.1.1 were used. The images, shown in Figures 3.1.1-3.1.6, were chosen to include a variety of targets, ranges, field of view, clutter, and occlusion. Each segmenter was run on each image without any preprocessing (no filtering, etc) or postprocessing (like region merging or finding largest regions). All the segmenters tested required setting a threshold, so instead of trying to pick a threshold (which might influence the performance of the algorithm), we show the results using various thresholds. Figures 3.3.42-3.3.47 show the outputs of each of the segmenters for various thresholds, and each of the given input images. Since the thresholds are based on different features for each of the different segmenters it is meaningless to try to compare images on the same rows in these figures. Instead these figures are to show the sensitivity of each segmenter to threshold selection, and the effects of a misplaced threshold for each segmenter individually. The image in column 1 is the original unprocessed LADAR image. The images in columns 2-4 and 8 are the output of the given segmenter with white representing background and black being the target. The images in columns 5-7 have white as the background and the inverted image (black is white and white is black , and light gray is dark gray, etc) is the target. The following section discusses each of the segmenters.

3.3.3.1. Planar Patch Fitting Error Segmenter

The motivation behind planar patch fitting is that objects to be segmented are planar (or can be approximated by planes) and the background and clutter are not planar. Planar patch fitting segmentation was presented in [KaYo87], however there were a couple of

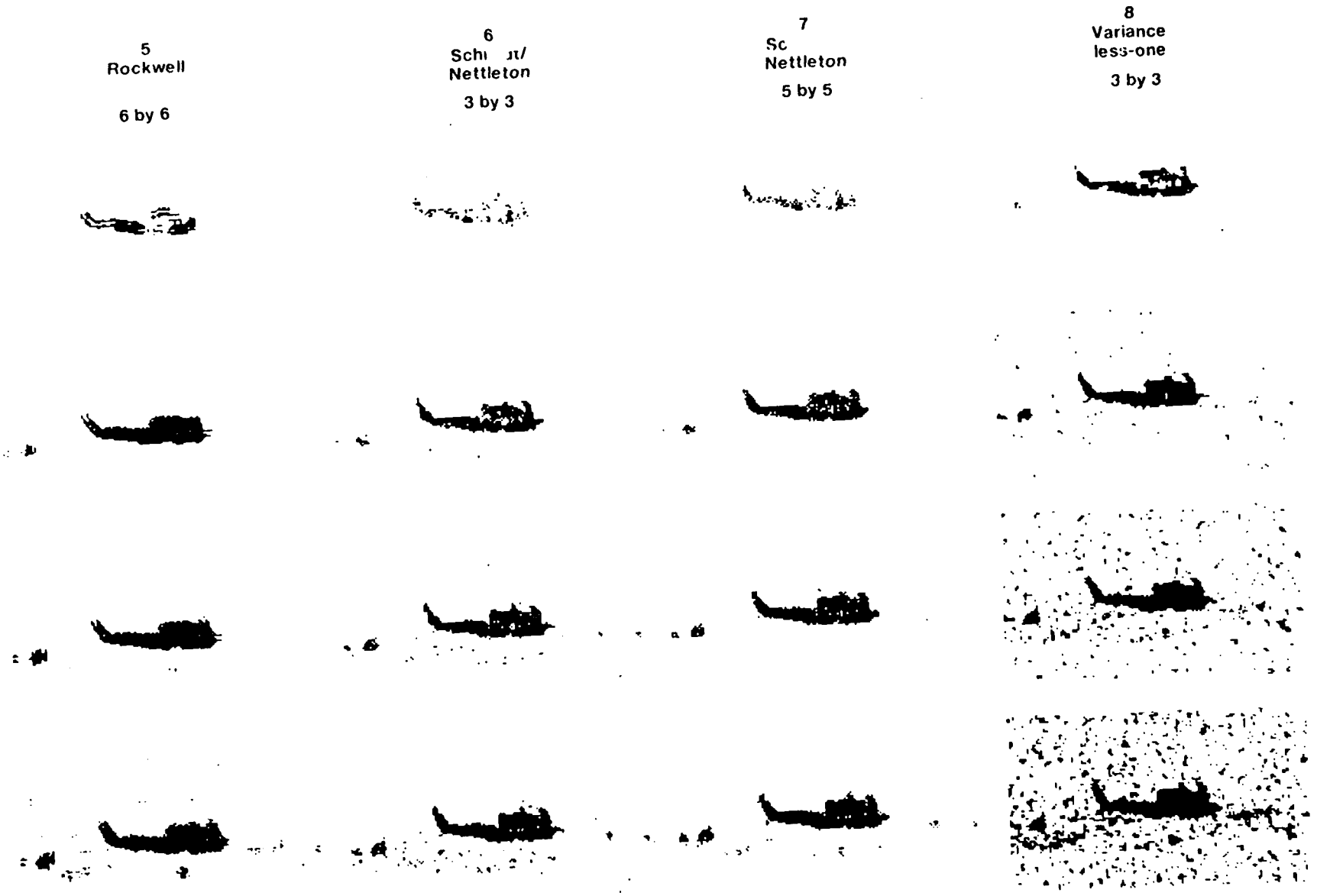


Figure 3.3.42 (continued)

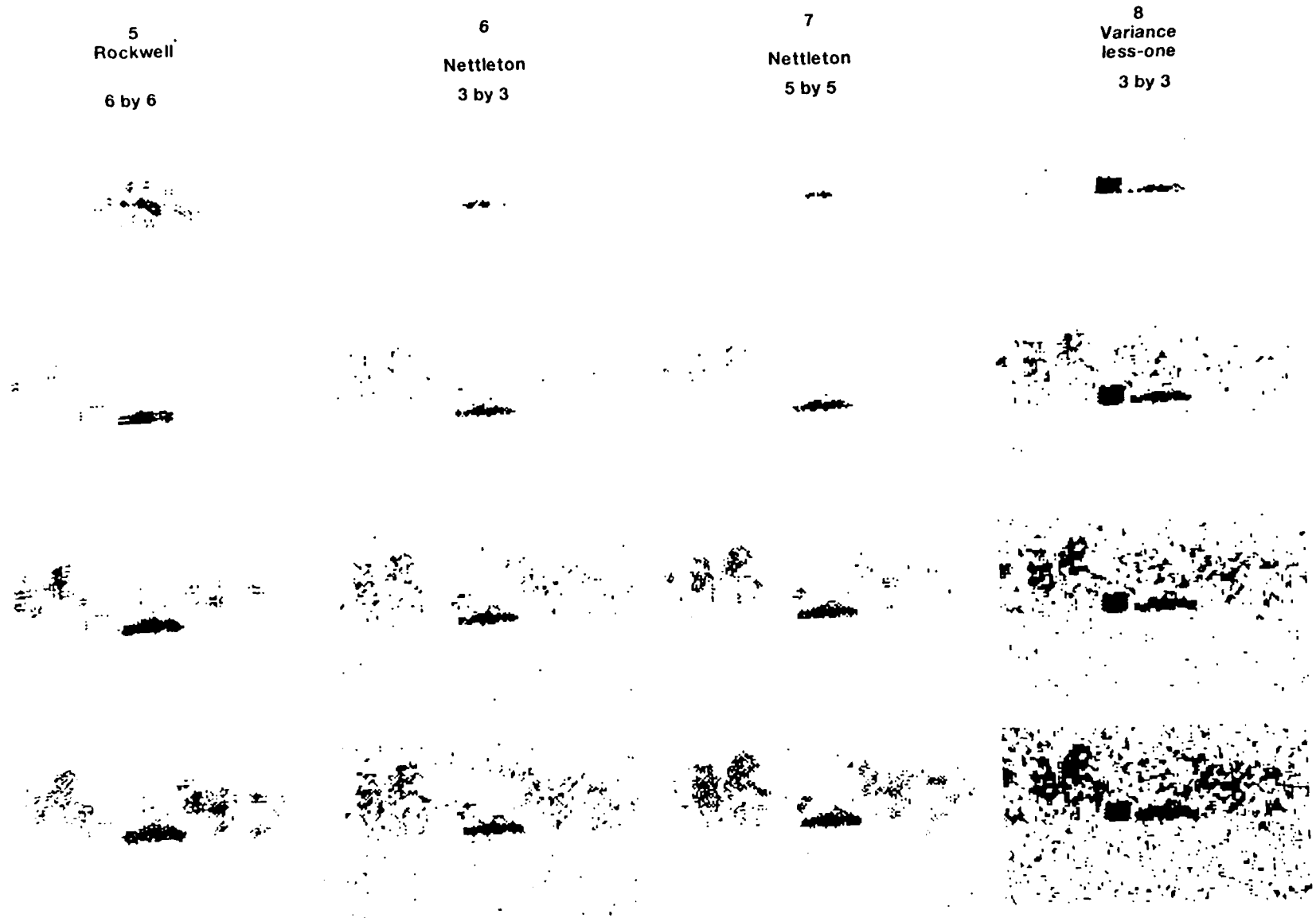


Figure 3.3.43 (continued)

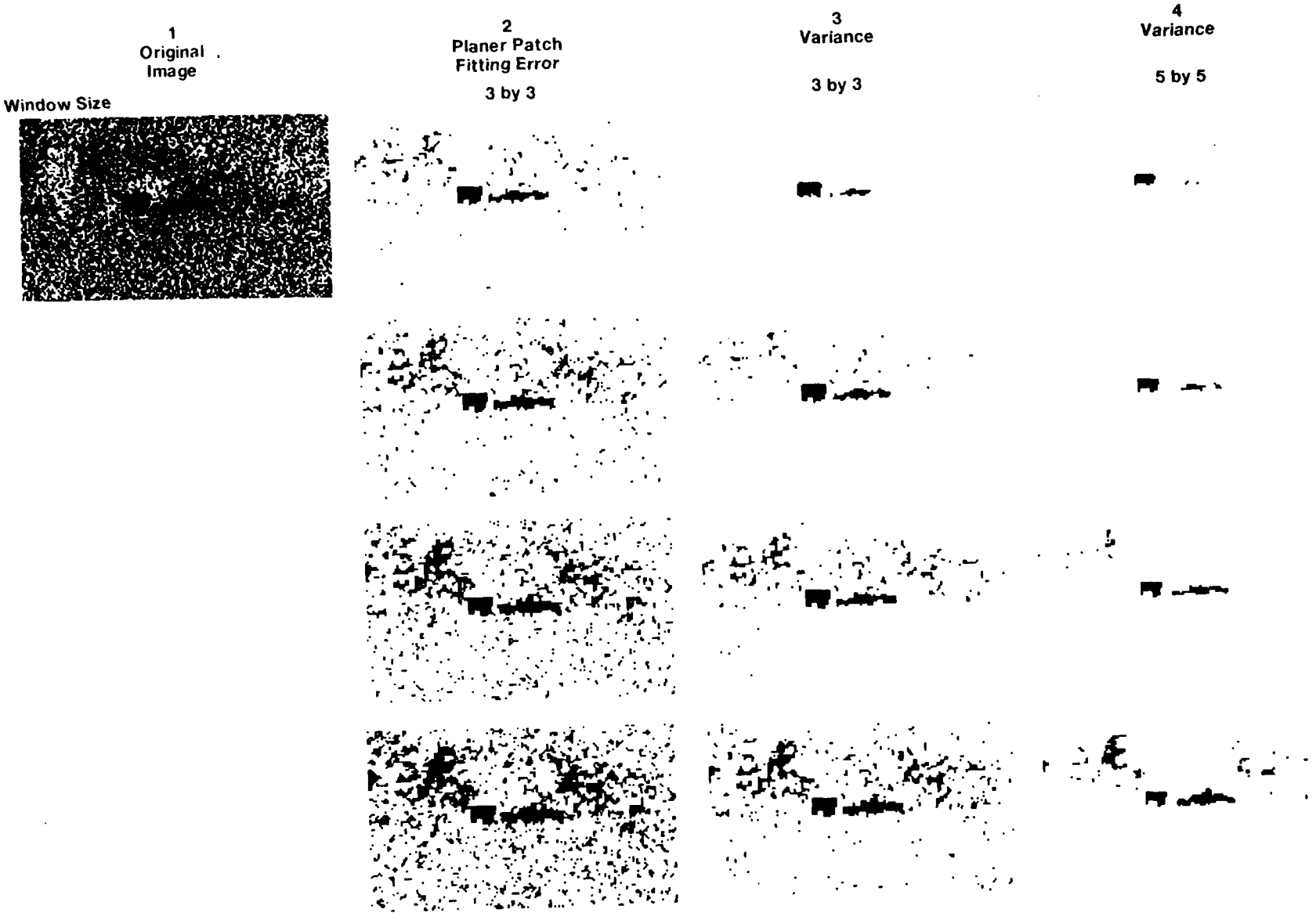


Figure 3.3.43 Output of segmenters for target ap1.32411.

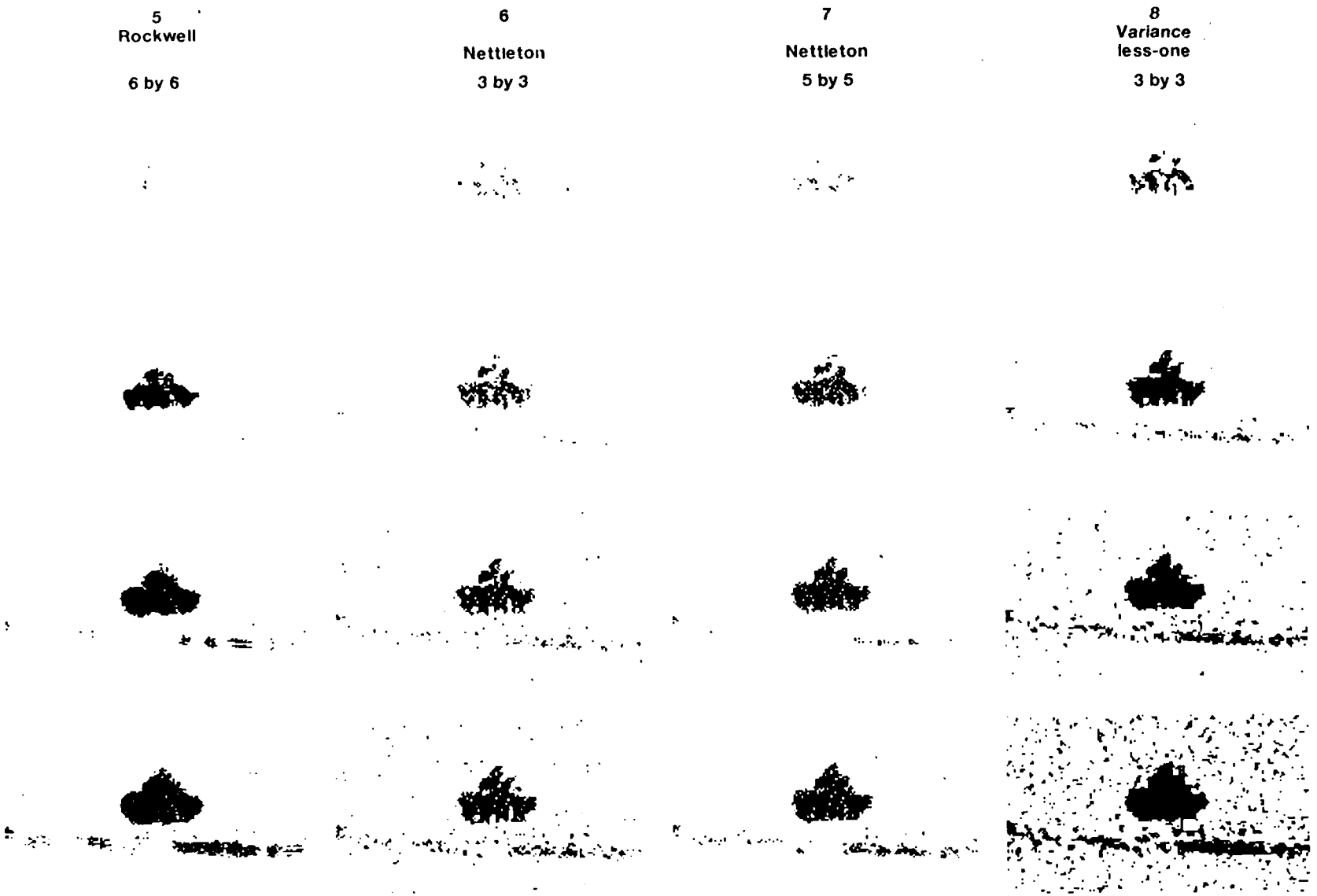


Figure 3.3.44 (continued)

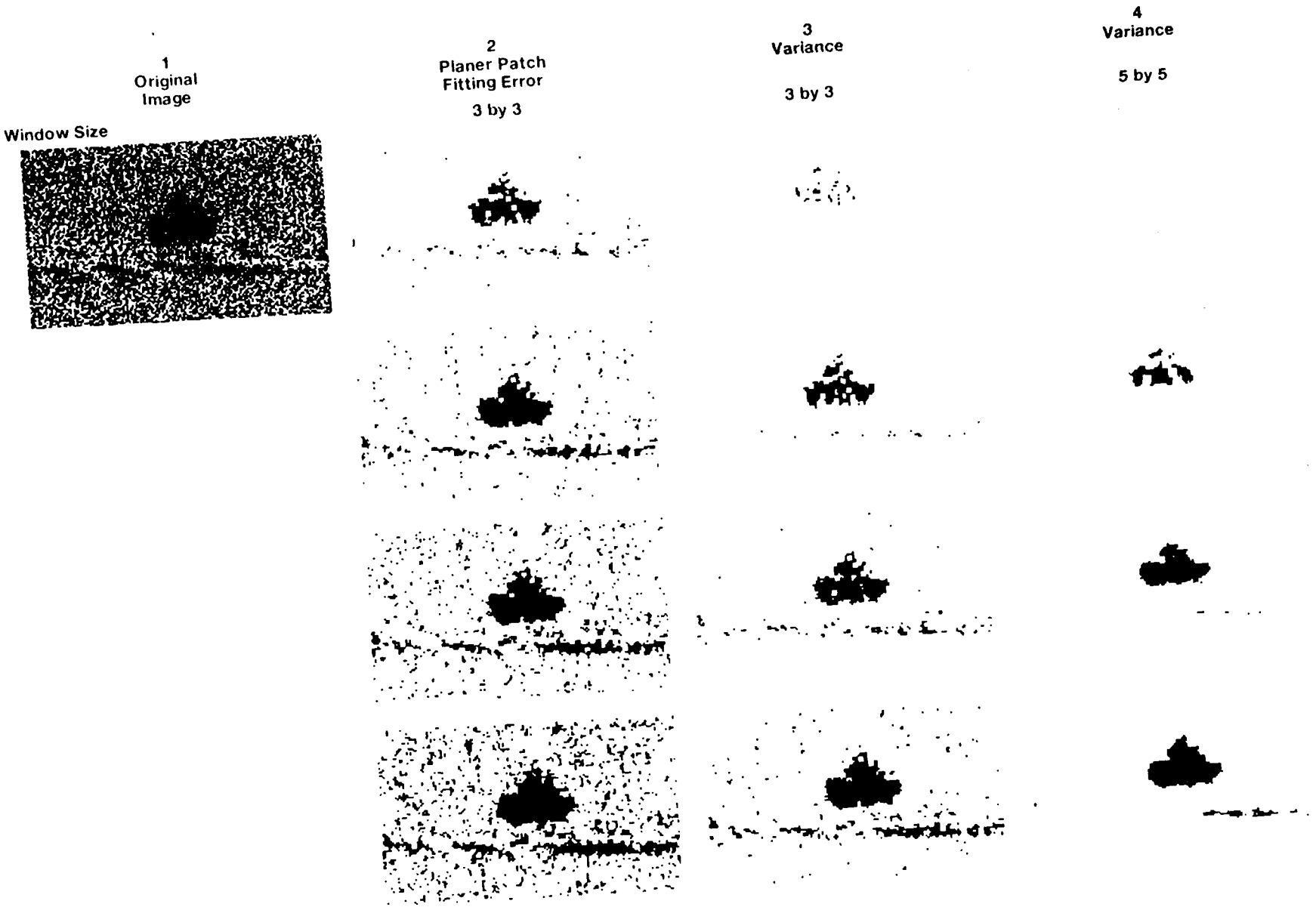


Figure 3.3.44 Output of segmenters for target ap1.32504.

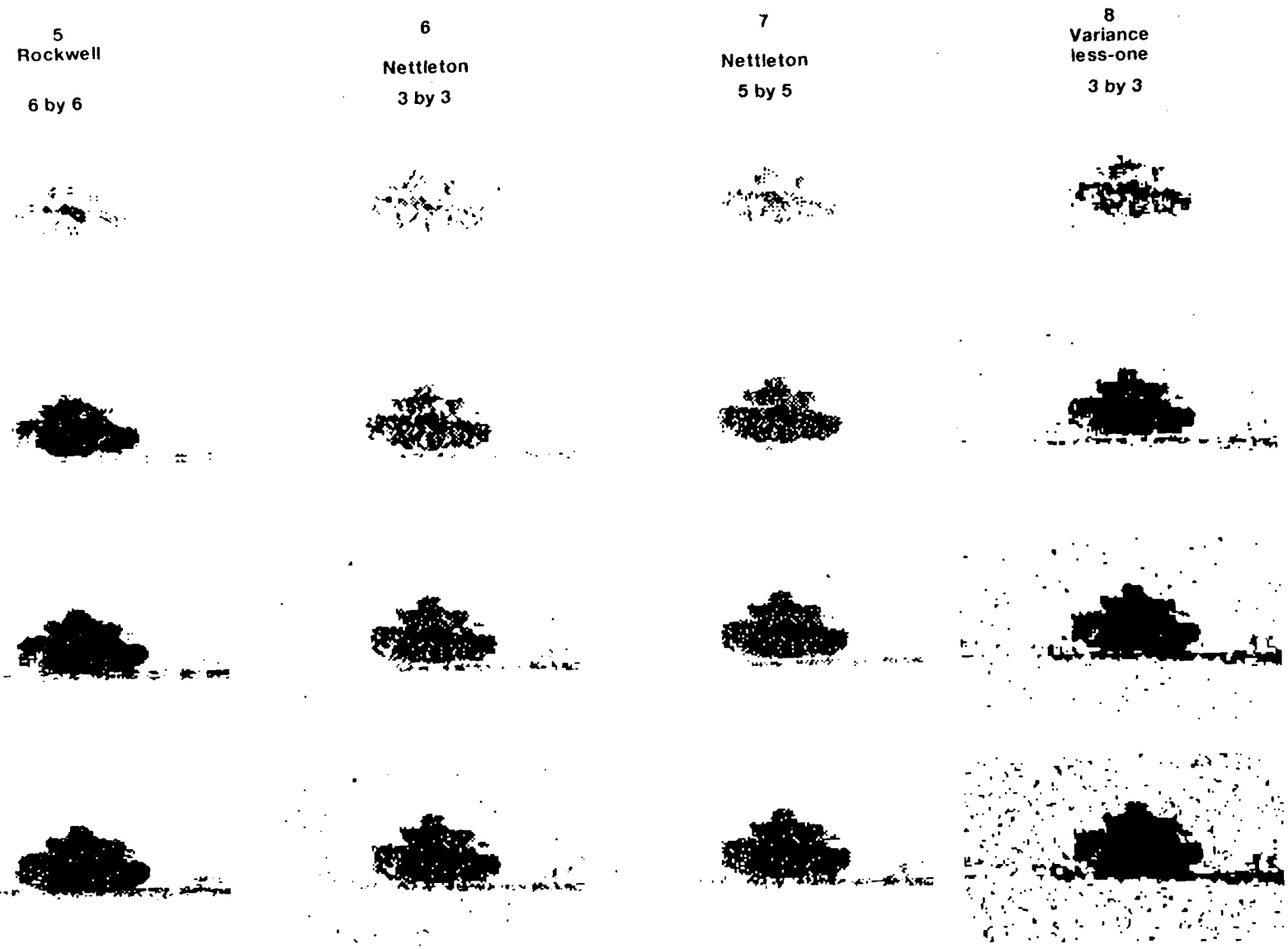


Figure 3.3.45 (continued)

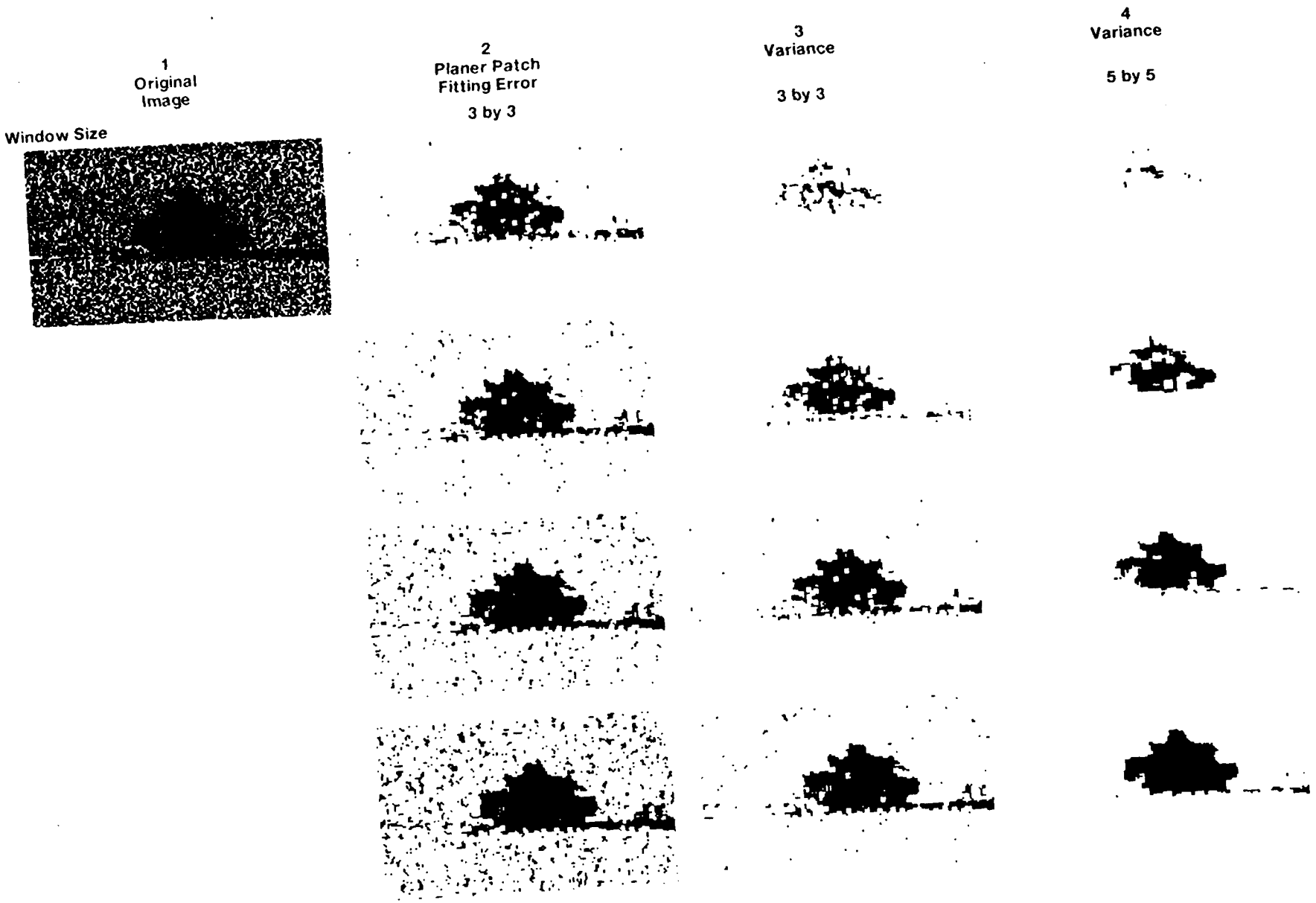


Figure 3.3.45 Output of segmenters for target ap1.32633.

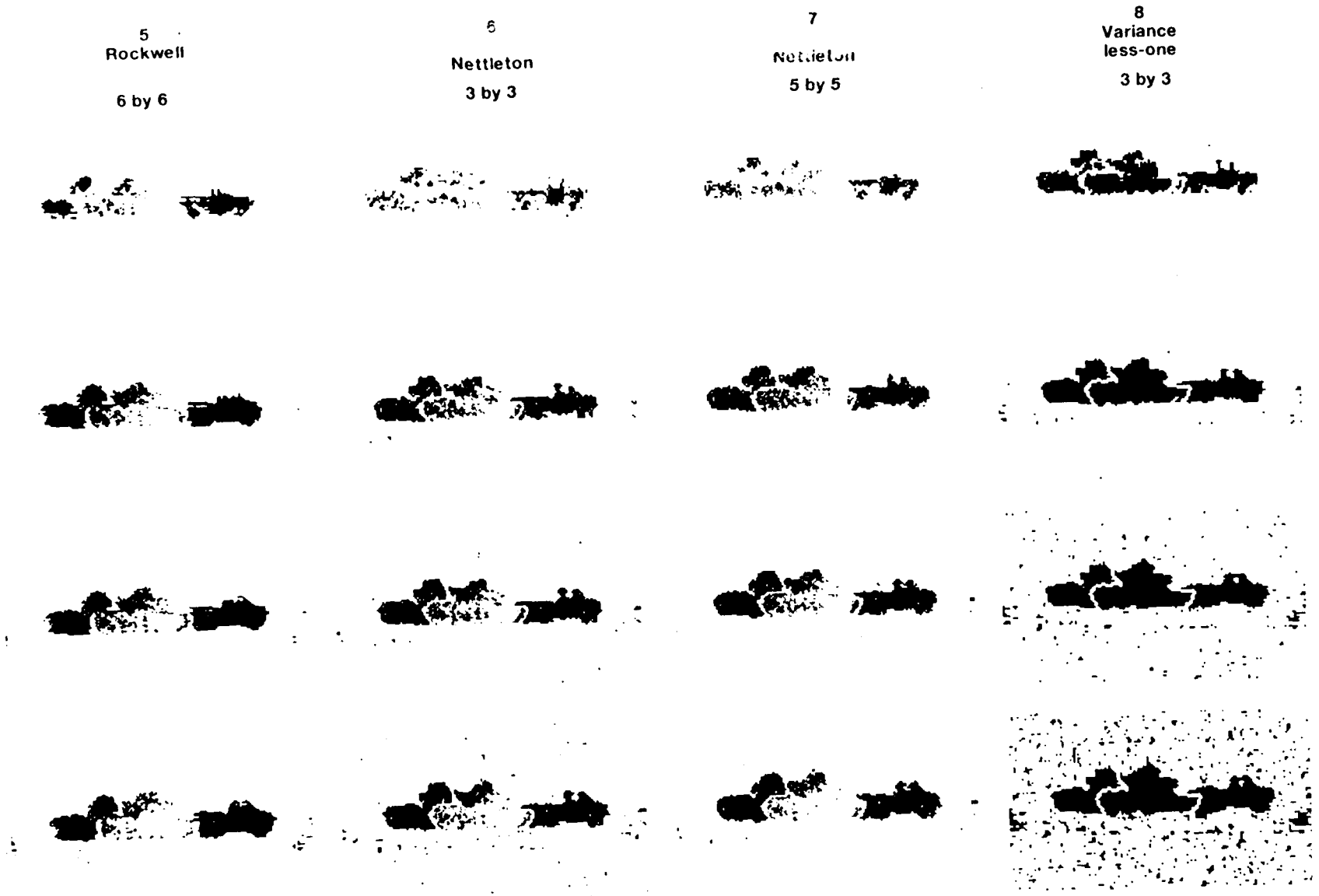


Figure 3.3.46 (continued)

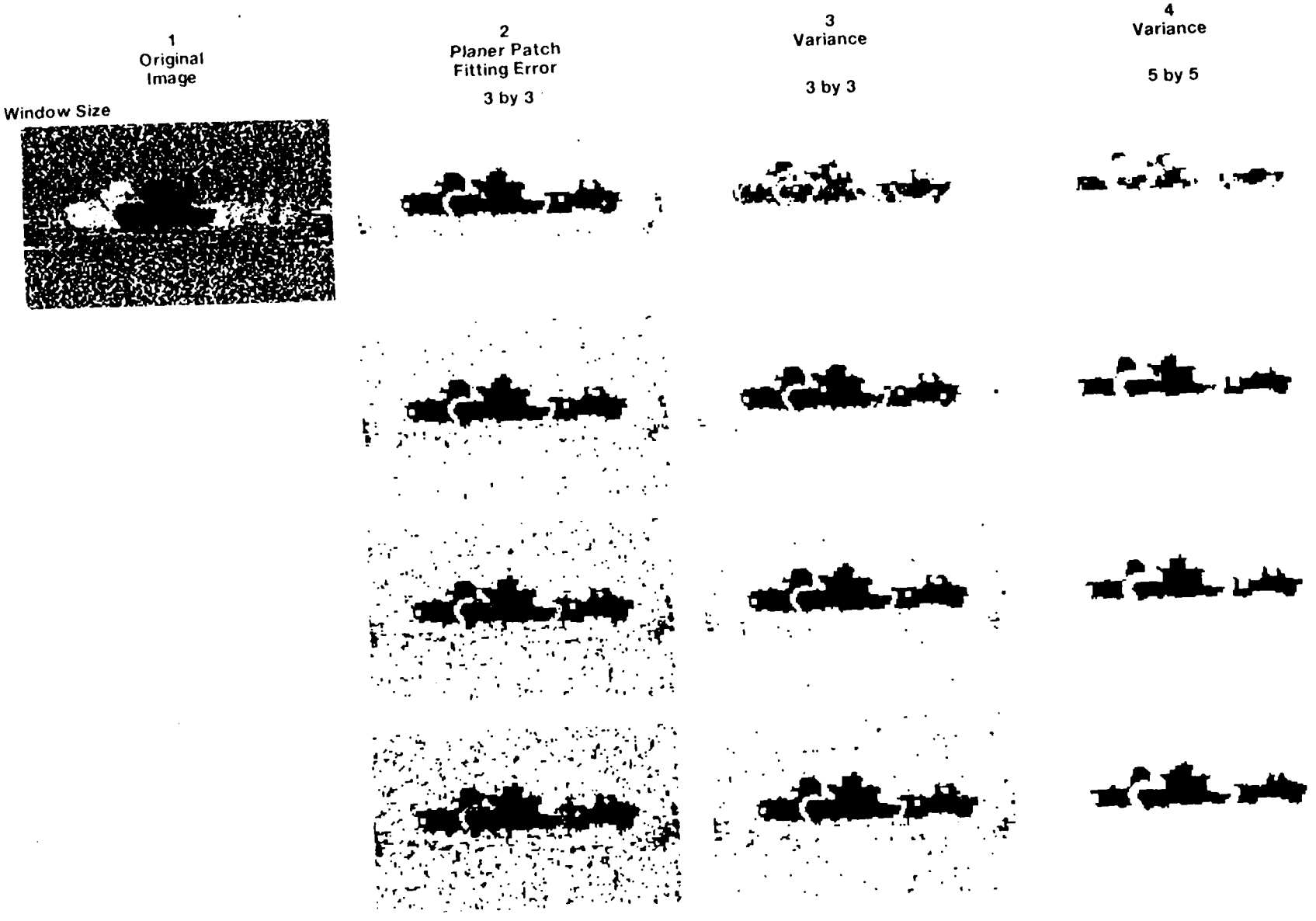


Figure 3.3.46 Output of segmenters for target ap1.32837.

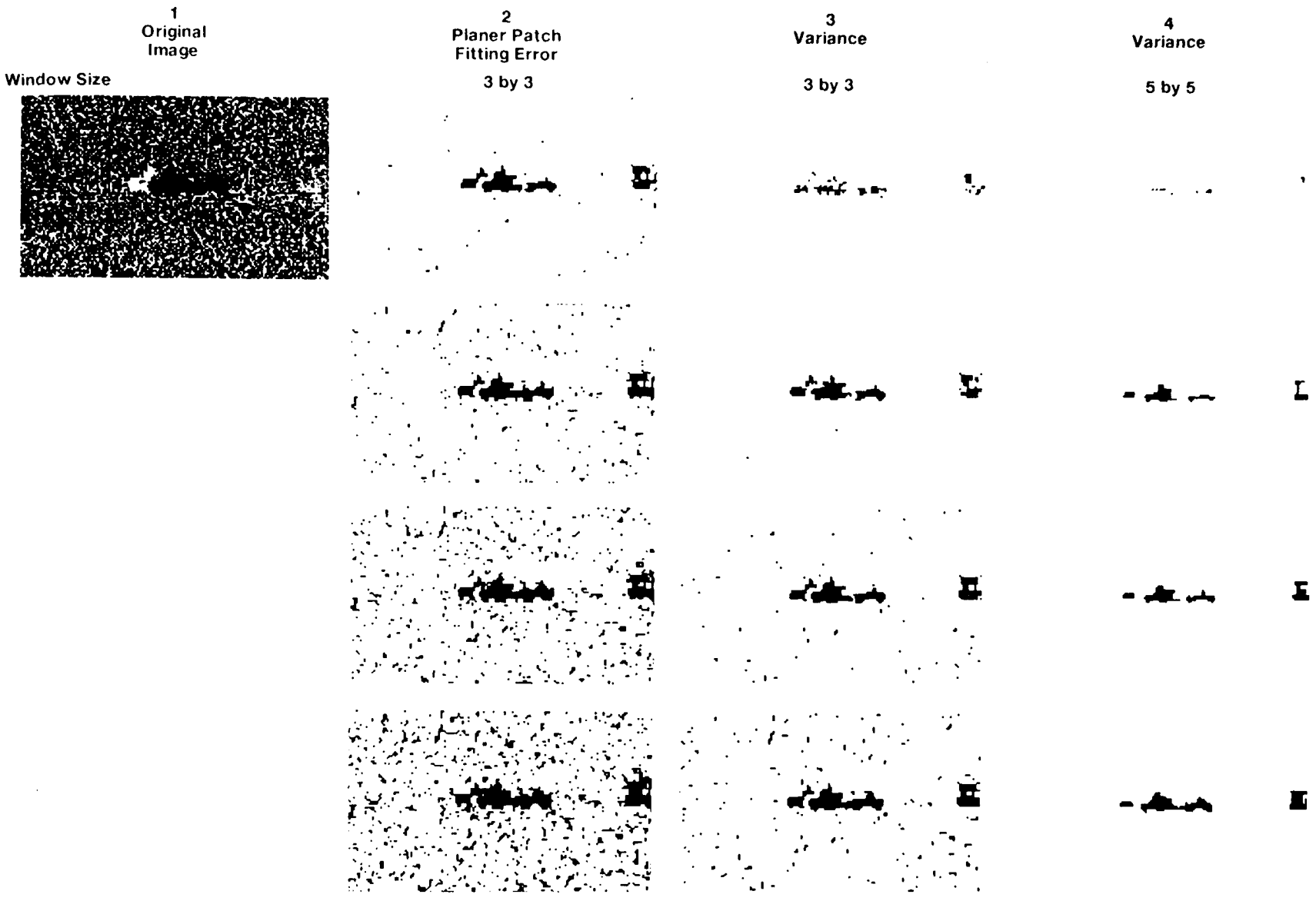


Figure 3.3.47 Output of segmenters for target ap1.32839.

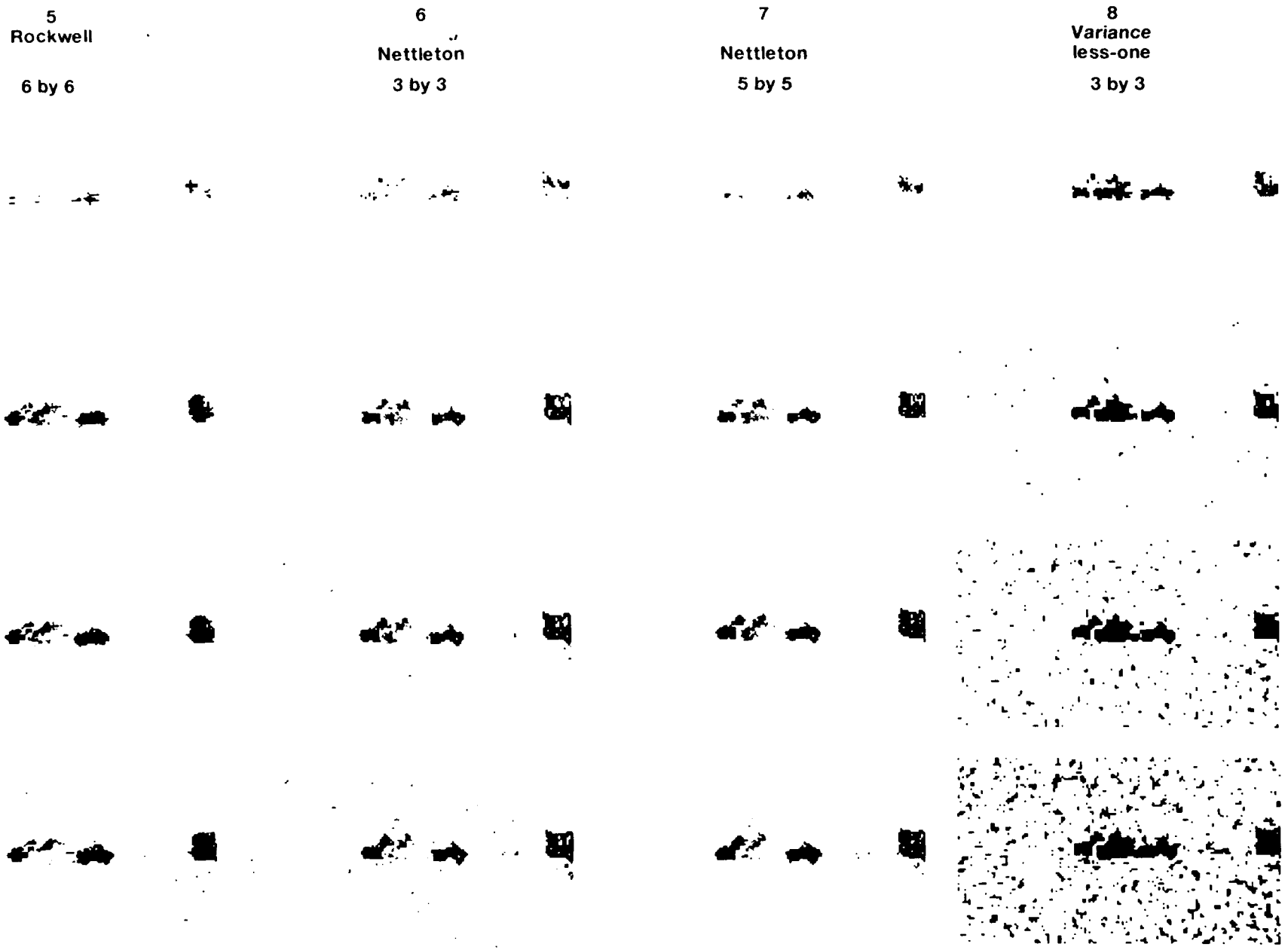


Figure 3.3.47 (continued)

typographical errors in the equations that set the partial derivative to zero, so the derivation is repeated here. In addition some simplifications are shown that can be used when the window is symmetric.

3.3.3.1.1. Derivation of Planar Patch Fitting

Planar patches are fitted as follows: First denote an M by N range image by $z(x,y)$ where (x,y) belongs to a finite region $D = \{ (x,y) : 0 \le x \le M-1, 0 \le y \le N-1 \}$ in the xy plane. Next partition the plane into overlapping m by n windows by defining a region of vertices consisting of $V = \{(u,v) : 0 \le u \le M-m+1, 0 \le v \le N-n+1\}$ as shown is Figure 3.3.48. For each vertex (u,v) in V associate a window of size m by n whose upper left corner starts at (u,v). The exhaustive enumeration of the vertex set covers the entire image and provides a convenient structure for labeling all the windows as illustrated in Figure 3.3.49.

Since the planar fit over each window is similar, we will discuss fitting a plane over a single window. A plane is described by the set of all points $\{(x,y,z) : z = ax + by + c \}$. The fitting plane is determined by minimizing the error square criterion, that is determine (a,b,c) to minimize ε where,

$$\varepsilon = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (z(x_i,y_j) - ax_i - by_j - c)^2$$

Taking partial derivatives of ε with respect to a, b, and c,

$$\frac{\partial \varepsilon}{\partial a} = -2 \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_i \, (z(x_i,y_j) - ax_i - by_j - c)$$

$$\frac{\partial \varepsilon}{\partial b} = -2 \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} y_j \, (z(x_i,y_j) - ax_i - by_j - c)$$

$$\frac{\partial \varepsilon}{\partial c} = -2 \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (z(x_i,y_j) - ax_i - by_j - c)$$

and setting the partial derivatives to zero, we obtain

$$a\, n \sum_{i=0}^{m-1} x_i^2 + b \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_i y_j + c\, n \sum_{i=0}^{m-1} x_i = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_i z(x_i,y_j) \qquad (2.1)$$

$$a \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_i y_j + b\, m \sum_{j=0}^{n-1} y_j^2 + c\, m \sum_{j=0}^{n-1} y_i = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} y_j z(x_i,y_j) \qquad (2.2)$$

$$a\, n \sum_{i=0}^{m-1} x_i + b\, m \sum_{j=0}^{n-1} y_j + c\, mn = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} z(x_i,y_j) \qquad (2.3)$$

a linear system of equations to be solved for (a,b,c) in terms of the range data within the window. This process is repeated for each window in the image.

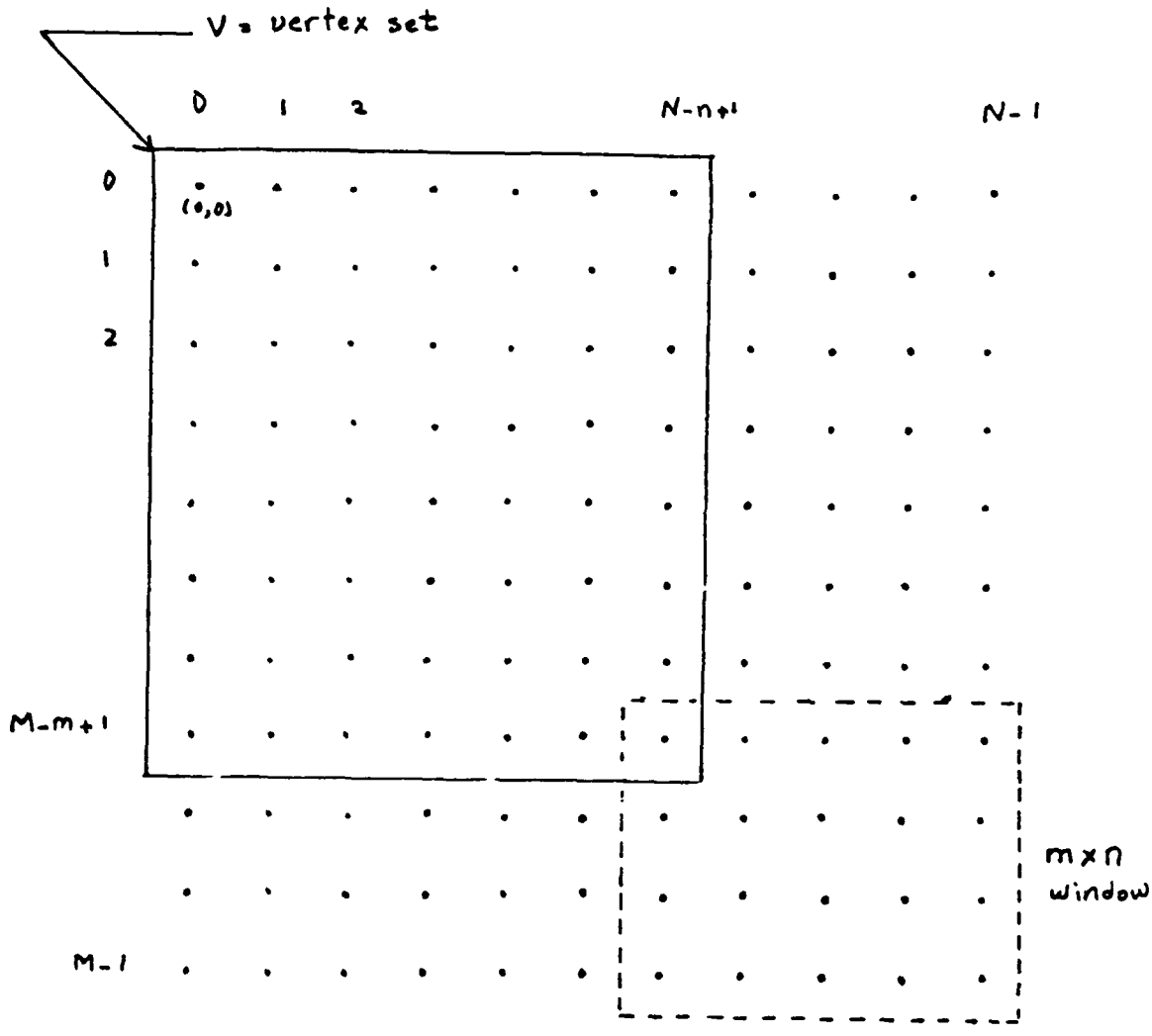


Figure 3.3.48 Region of vertices identifying all overlapping windows.

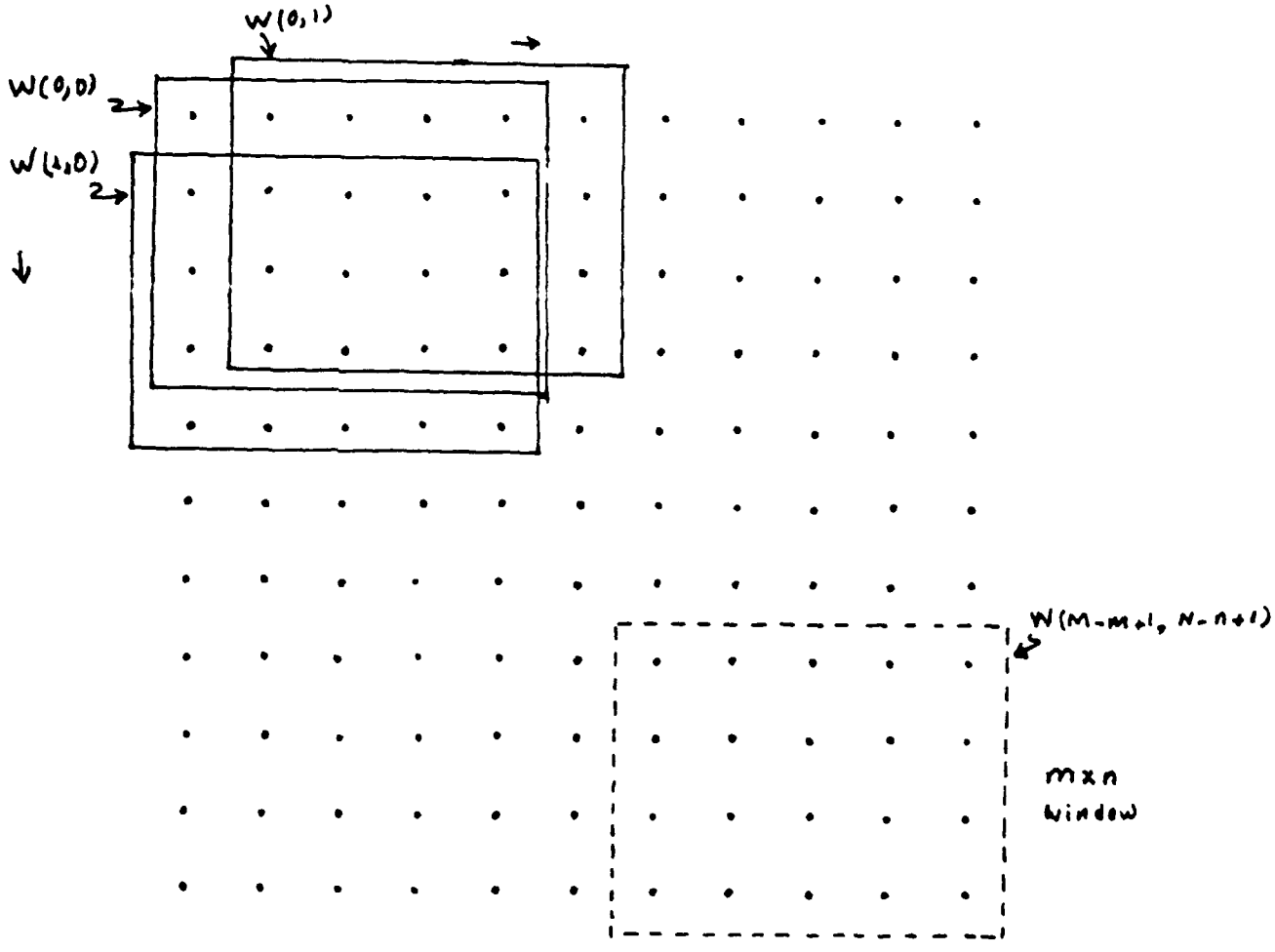


Figure 3.3.49 The correspondence between vertices and windows.

3.3.3.1.2. Symmetric Patches

If the window is symmetric (i.e. x_i=(...,−2,−1,0,1,2,...) and y_j=(...,−2,−1,0,1,2,...)) the following simplifications can be used:

$$\sum_{i=0}^{m-1} x_i = 0$$

$$\sum_{j=0}^{n-1} y_j = 0$$

$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i y_j = 0$$

Applying these to equation 2.1 gives:

$$an\sum_{i=0}^{m-1} x_i^2 = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i\, z(x_i,y_j)$$

$$a = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i\, z(x_i,y_j)}{n\sum_{i=0}^{m-1} x_i^2} \tag{2.4}$$

Applying these to equation 2.2 gives:

$$bm\sum_{j=0}^{n-1} y_j^2 = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} y_j\, z(x_i,y_j)$$

$$b = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} y_j\, z(x_i,y_j)}{m\sum_{j=0}^{n-1} y_j^2} \tag{2.5}$$

And finally applying these to equation 2.3 gives:

$$c = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} z(x_i,y_j)}{mn} \tag{2.6}$$

Therefore a, b, and c can be solved for directly without any matrix inversion.

3.3.3.1.3. Fitting Error Segmentation

The targets in column 2 of Figures 3.3.42-3.3.47 were segmented by finding the planar patches for an image and eliminating those pixels that are in the center of each patch with large fitting error. This method is clearly able to locate all the targets over a wide range of thresholds. This is not so much a result of the method being a good one than it is that the data was non-planar except where the targets are located. Table 3.3.8 gives

comments on each of the images. Although the images show that this method works well on the range data, it does require a number of computations for each pixel. Assuming a symmetric m by n window, each window (and therefore each pixel since we are using overlaying windows) requires $2mn$ multiplications, $3mn$ additions, and 3 divisions to compute a, b, and c, plus $3mn$ subtractions and $3mn$ multiplications to compute the error.* Finally an additional subtraction is needed to compare the error to the threshold.

Table 3.3.8 Comments on planar patch fitting error segmentation results.

Target	Comments
ap1.32403	At a range of 1km the curves of the helicopter look like planes therefore planes fit well to the target.
ap1.32411	The square building (clutter) to the left of the M60A2 was nicely segmented since the planes fit to it too. The classifier will have to be used to eliminate this clutter from being considered a target. The trees in the background will fit planes if too large of a threshold is used.
ap1.32504 ap1.32633	Both the different apparent sizes of the M60A2 were easily found. Dropouts caused many holes in the target if the fit error threshold is too small. Some of the ground shows up in these images. We expect to see more ground showing up in the new data. If this is the case, the orientation of the plane may have to be used to help in segmenting planar targets from planar ground.
ap1.32837 ap1.32839	The 3 by 3 window size causes at least a 3 pixel gap between the front target and the occluded targets. This could be a problem if there are only a few pixels on target. The Rockwell segmenter uses a novel approach to get around this problem.

The variance method, presented in the next section, is able to fit patches to planes, but it uses fewer computations.

3.3.3.2. Variance Window Method

Suppose we still want to find planar surfaces, but we know they are always parallel to the viewing plane. If this is the case, a and b in Equations 2.4 and 2.5 would always be zero and c in Equation 2.6 would be the distance to the middle of the plane. This distance is simply the average over the window which we will denote as:

* These operation counts do not take into account the additions and multiplications that might be needed to index into a two dimensional array.

$$\bar{z} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} z(x_i, y_j)$$

The error equation then becomes:

$$\varepsilon = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (z(x_i, y_j) - \bar{z})^2$$

which is simply the variance of the pixels in the window.

The variance method is the planar patch fitting error method except the planar patch is always parallel to the viewing plane. Column 3 of Figures 3.3.42-3.3.47 show the output using this method with a 3 by 3 window and column 4 is the output using a 5 by 5 window.

The 3 by 3 window results appear to be as good as those of the planar patch fitting error method with the same sized window. Using a larger window makes the variance method less sensitive to noise, therefore the 5 by 5 window is able to segment out the targets and have fewer false segmentations on the background. As expected, the larger window also results in more pixels on the edges of the targets being missed. If the threshold is too low, large (5 by 5) holes are left in the target.

The advantage of this method is that it requires fewer computations than the planar patch fitting error method. A total of $2mn+1$ addition/subtractions, 2 multiplications, 2 divisions, and 1 square root * are required for each pixel.

3.3.3.3. Rockwell Segmenter

When working with window operators, one likes to use as large a window as possible to decrease the sensitivity to noise. We saw this effect in the previous section where both 3x3 and 5x5 windows were used. The larger window picked up fewer false segmentations on the background. The tradeoff with large windows is that pixels near the edge of the target are lost and very small targets may be completely missed, or large window-sized holes are left in the target when there is a noise spike. The 3 by 3 window in the variance method left a 3 pixel wide gap between the overlapping targets in *ap1.32837* and *ap1.32839*. while the 5 by 5 window left a 5 pixel gap.

The Algorithm Development group of Rockwell International has come up with a novel solution to this problem. They use a large 6 by 6 window, but compute the variance in eight 3 by 6 windows around a given pixel. If any of the windows has a variance below a given threshold, the pixel is classified as the target. Figure 3.3.50 shows the eight windows. The idea is that the pixel might be on an edge whose orientation is unknown. If this edge passes through the pixel, one of the eight windows will lie completely on the target

* The square root is not needed if the threshold is squared

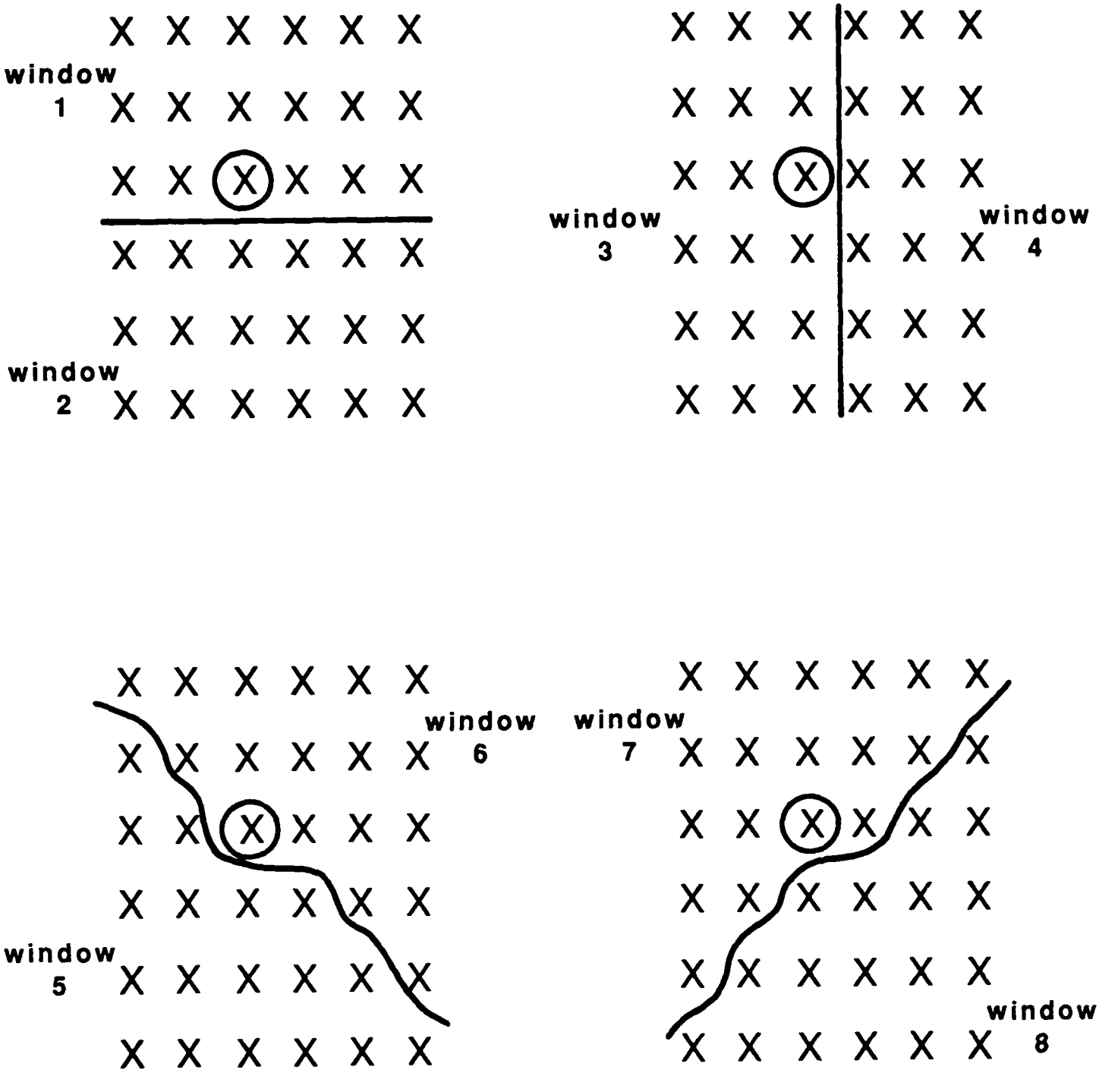


Figure 3.3.50 The eight windows around a given pixel used in the Rockwell Segmenter.

even if the other half of the window is background. If any one of the windows falls on a planar surface, the pixel is classified as planar (i.e. a target).

Column 5 of Figures 3.3.42-3.3.47 show the output of the Rockwell segmenter. Targets *ap1.32837* and *ap1.32839* show that it is able to extract overlapping targets without leaving a gap between them.

The price that is paid of such good segmentation is the number of computations. For an m by n window, the variance must be computed over each of the eight orientations. Each of the subwindow has an area of $\frac{1}{2}mn$. A brute force method requires $8(\frac{1}{2}2mn+1)$ addition/subtractions, 1 multiplication, 16 divisions, and 8 square roots. The Rockwell implementation takes note of the overlapping windows and is able to reduce the computations to $4mn+40$ addition/subtractions, 1 multiplication, 16 divisions, and 1 square root.

3.3.3.4. Nettleton Method

If speed is important, the following algorithm by John Nettleton of the Center for Night Vision and Electro-Optics performs well with very few computations. His approach is to examine the value of the pixel in the center of the window and compare it to the values of all the pixels around it. If enough pixels are close enough in value to the center pixel, the center pixel is classified as target. Typically for a 3 by 3 window, 6 neighbors in the window must be within the threshold. For a 5 by 5 window 15 neighbors must be close enough.

Columns 6 and 7 of Figures 3.3.42-3.3.47 show the results for a 3 by 3 window and a 5 by 5 window. This method does better than any of the other methods in separating the overlapping targets in *ap1.32837* and *ap1.32839*. It leaves just single pixels between the targets.

Unfortunately, since the center pixel is the basis for comparison, it is very sensitive to dropouts and noise spikes. Notice that even with the largest threshold, *ap1.32633* (Figure 3.3.45) still has many holes in it. A simple improvement would be to use the average value of the window instead of the center value.

The method is very fast since it requires only an addition and a subtraction to find the upper and lower ranges and mn subtractions to compute the differences between the center pixel and all the other pixels in the window. If the mean value were used instead of the center values, the cost would be an additional mn additions, and a division. Both methods require an additional mn subtractions to compare the pixel differences to a threshold.

3.3.3.5. Variance-Less-One Method

One of the deficiencies of the planar patch fitting methods is that it is sensitive to noise spikes. For example, Figure 3.3.51 shows the six sample images we have been working with, the variance of the image for a 3 by 3 window, and the histograms for the

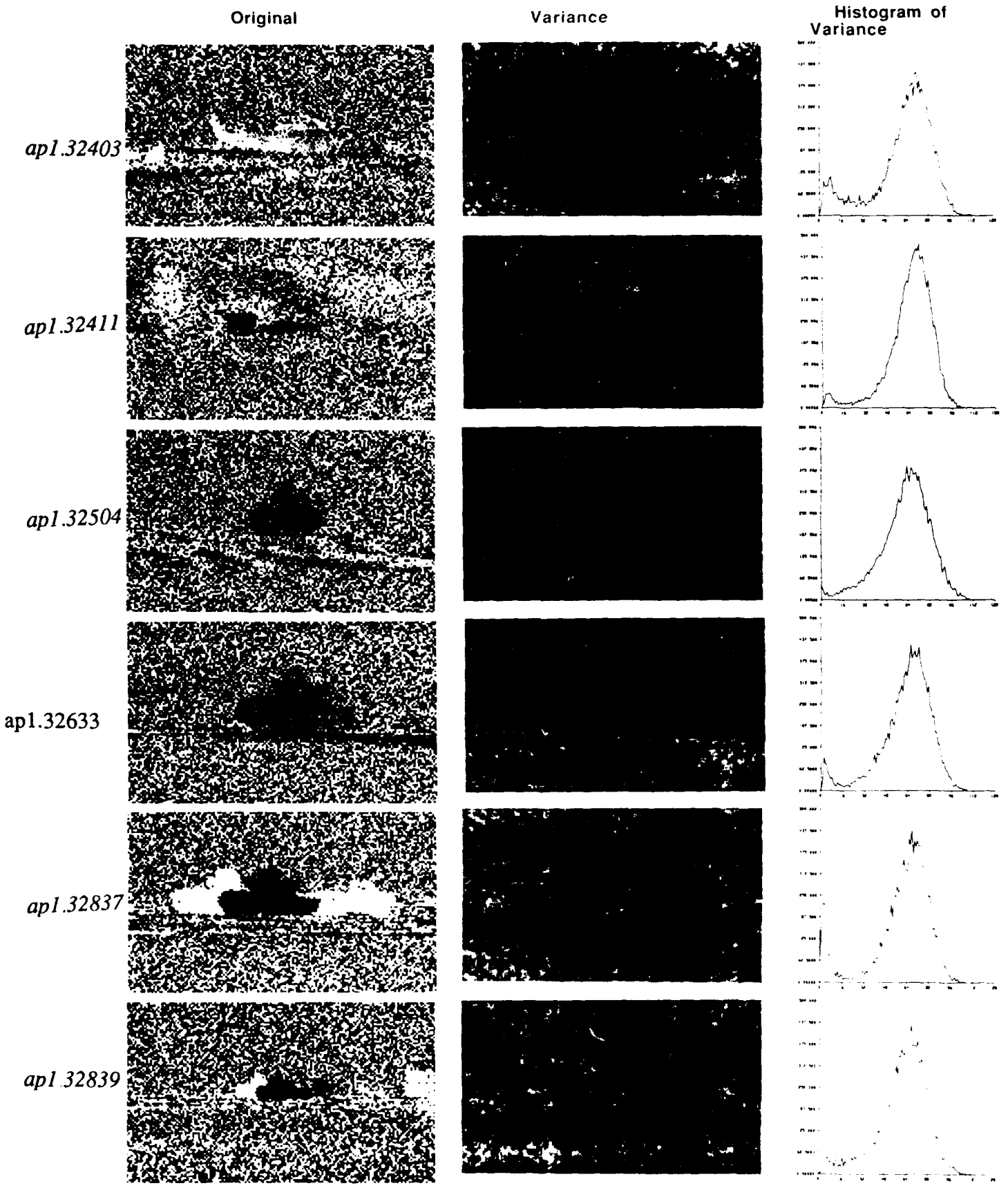


Figure 3.3.51 The six test images from Figures 3.1.1-3.1.6 and their variance images using a 3 by 3 window. The histograms are of the variance images.

variance. There is a dropout near the front of the leftmost tank in *ap1.32837*. This dropout caused the variances in the nine windows which contain it to be larger than the variances of the surrounding windows. This shows up on the variance image as a 3 by 3 square which is brighter than the surrounding pixels. This effect can be seen on the other targets as well. In particular, *ap1.32633* has a large number of dropouts near the lower left part of the target.

The variance-less-one method overcomes this problem by examining all the pixels in a given window and ignoring the pixel whose value is farthest from the mean of the window when computing the variance for that window. Figure 3.3.52 shows the variance-less-one images for the same images as in Figure 3.3.51. There are fewer "bright squares" caused by the noise spikes. The histograms show that, as expected, the variance is lower when the most extreme point in each window is omitted. In general the histograms appear to be more bimodal, and the segmented images in column 8 of Figures 3.3.42-3.3.47 show the targets have fewer holes in them and that a lower threshold can be used and still segment the entire target.

The computational complexity is the same as the variance method, except two additional subtractions are needed to remove the given pixel from the variance calculation.

3.3.3.6. Conclusions

Five different laser radar range image segmenters were tested and all five performed well on the images they were tested on. The image test set c tained a variety of targets, ranges, clutter, and occlusion.

The Nettleton segmenter was very sensitive to noise spikes and dropouts because it compares all the pixels in a given window to the center pixel. If the center pixel is a noise spike on a target, for example, none of the pixels around will be close enough so it will be classified as a background pixel. An advantage of this technique is that the effects of a noise spike are confined to a single pixel.

The planar fit, variance, and Rockwell methods are not as sensitive to noise spikes as the Nettleton method, however the effect of a spike will show up in all the windows which contain the spike.

Our variance-less-one method overcomes this problem by finding the pixel which is farthest from the mean of a given window and computing the variance without that pixel. The new method works well with the ladar images we currently have since most windows will contain one or no noise spikes. If an image has many spikes in a small area (like *ap1.32633*) a given window may contain more than one spike. This method will only remove the largest spike, leaving any others to affect the variance calculation. An improvement to this method would be to ignore all spikes that deviate significantly from the mean. This is a common technique used in SAR image processing.

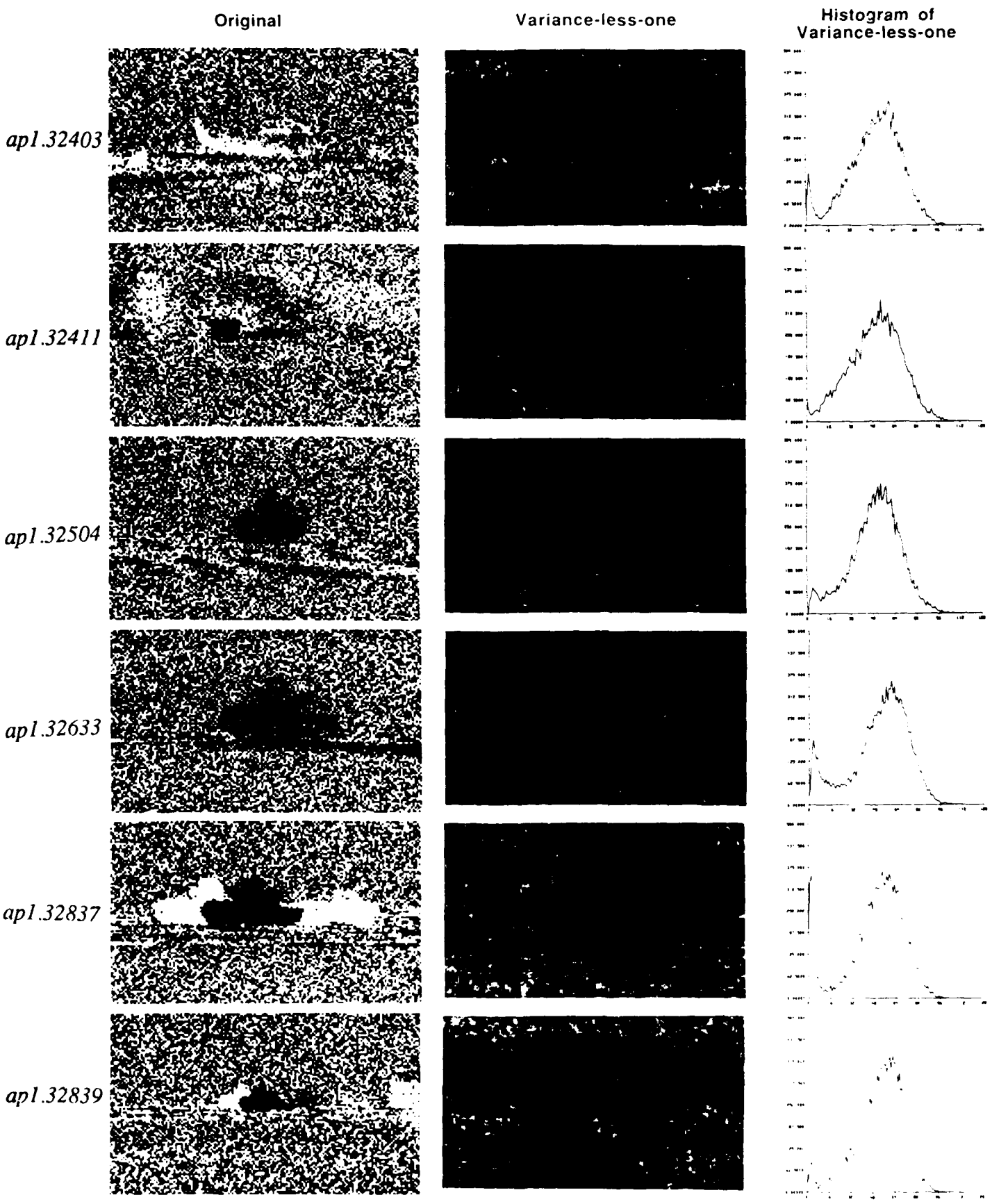


Figure 3.3.52 The six test images from Figures 3.1.1-3.1.6 and their variance-*less-one* images using a 3 by 3 window. The histograms are of the variance-*less-one* images.

Table 3.3.9 summarizes the computational complexities of the various methods. The Nettleton method is certainly the fastest method, with the variance, variance-less-one, Rockwell, and fitting error, methods following in order. Since the dwell time for the sensor is 80 μs [Rayt] and current signal processor chips can perform 16 bit multiplies [TI85] in 200 ns or faster, any of these methods should be able to keep up with the sensor if implemented on such a chip.

Table 3.3.9 Summary of computational complexities of the LADAR segmenters.

Method	Additions/ Subtractions	Multiplications	Divisions	Square Roots
Fit Error	6mn+3	5mn	3	
Variance	2mn+1	2	2	1
Rockwell (brute force)	8mn+8	1	16	8
Rockwell (fast)	4mn+40	1	16	1
Nettleton (center value)	2mn+2		1	
Nettleton (average)	3mn+2		1	
Variance-less-one	2mn+3	2	2	1

3.3.4. Results of Classifying the 1986 A.P. Hill Laser Range Data

The final step in the ATR process, after detection and segmentation, is to classify the targets. For this experiment a set of 26 - M60A2 targets and 26 - 5 ton trucks were selected as show in Figure 3.3.53 and 3.3.54. The following sections discuss how the classification was performed and what the results were.

3.3.4.1. Target Segmentation

The 52 targets were segmented using the planar fitting error segmenter presented in Section 3.3.3.1. The fitting error threshold was set by using the automatic threshold selector which simply found the the fitting error histogram and set the threshold at the point where second derivative was zero. Figures 3.3.55 and 3.3.56 show the targets after segmentation.

3.3.4.2. Feature Extraction

Although many features have been proposed for characterizing FLIR data (over twenty features were used in the experiments in [KaYo87]), much less work has been done for range data [BeJa85]. Since the segmenter was extracting well defined silhouettes, we

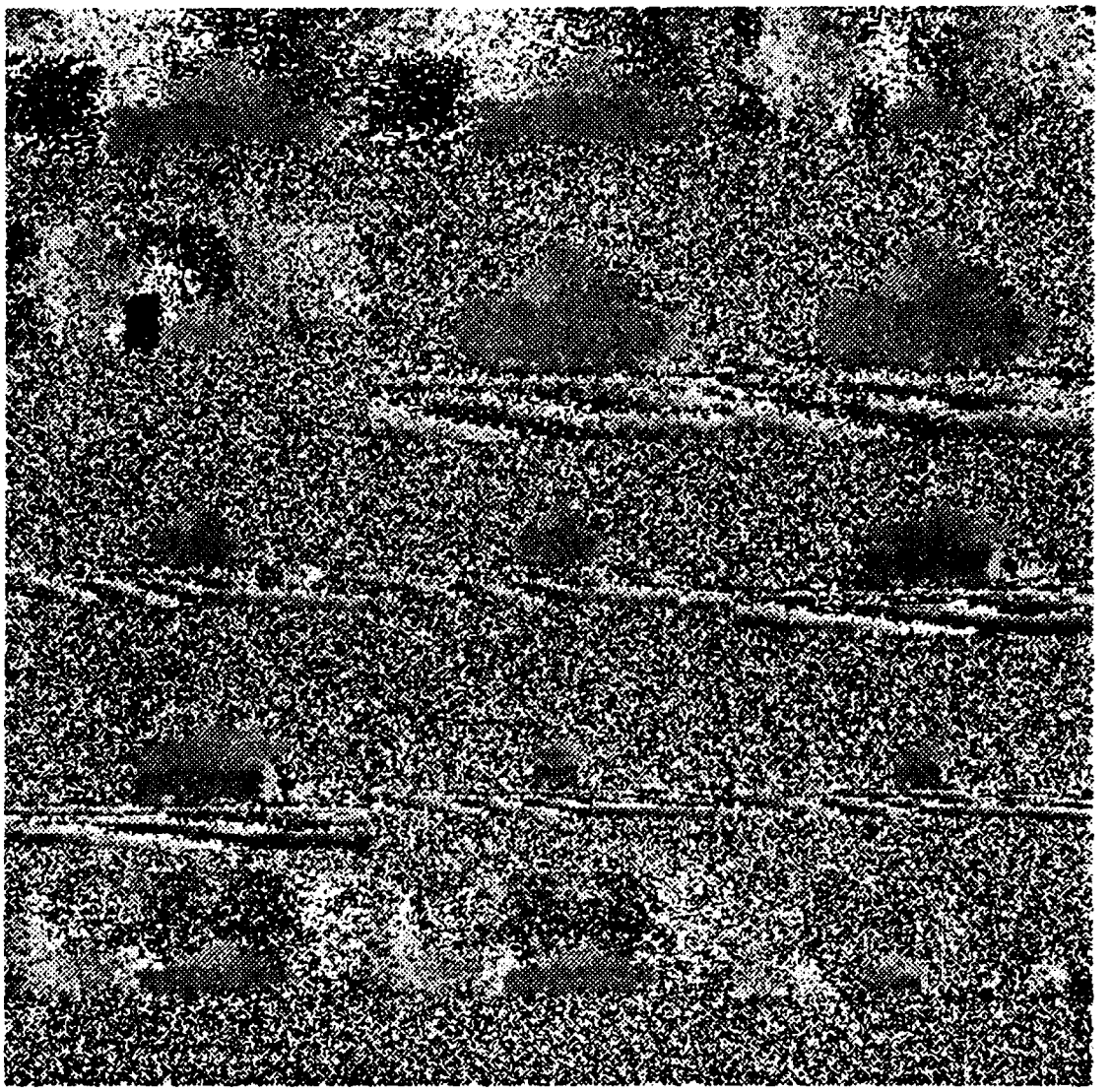


Figure 3.3.53 M60A2 targets used in Laser Radar classification experiment.

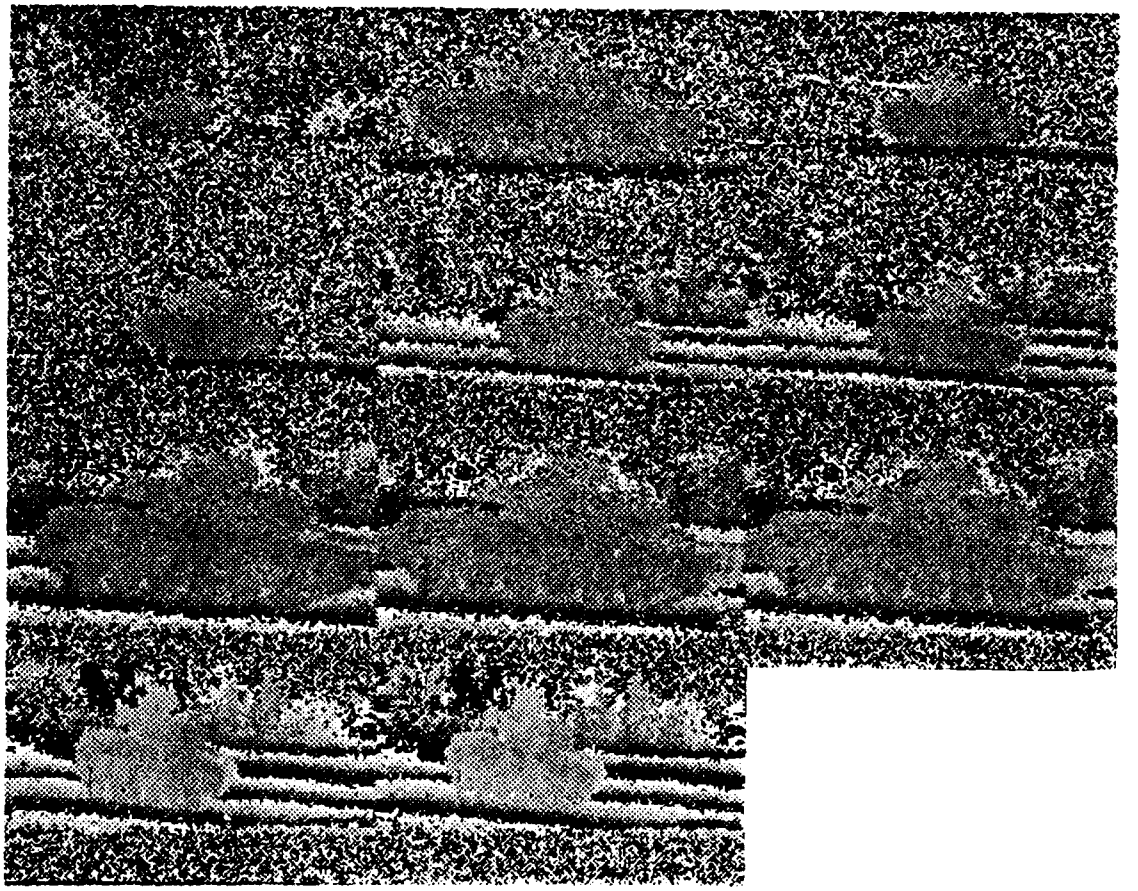


Figure 3.3.53 (continued)

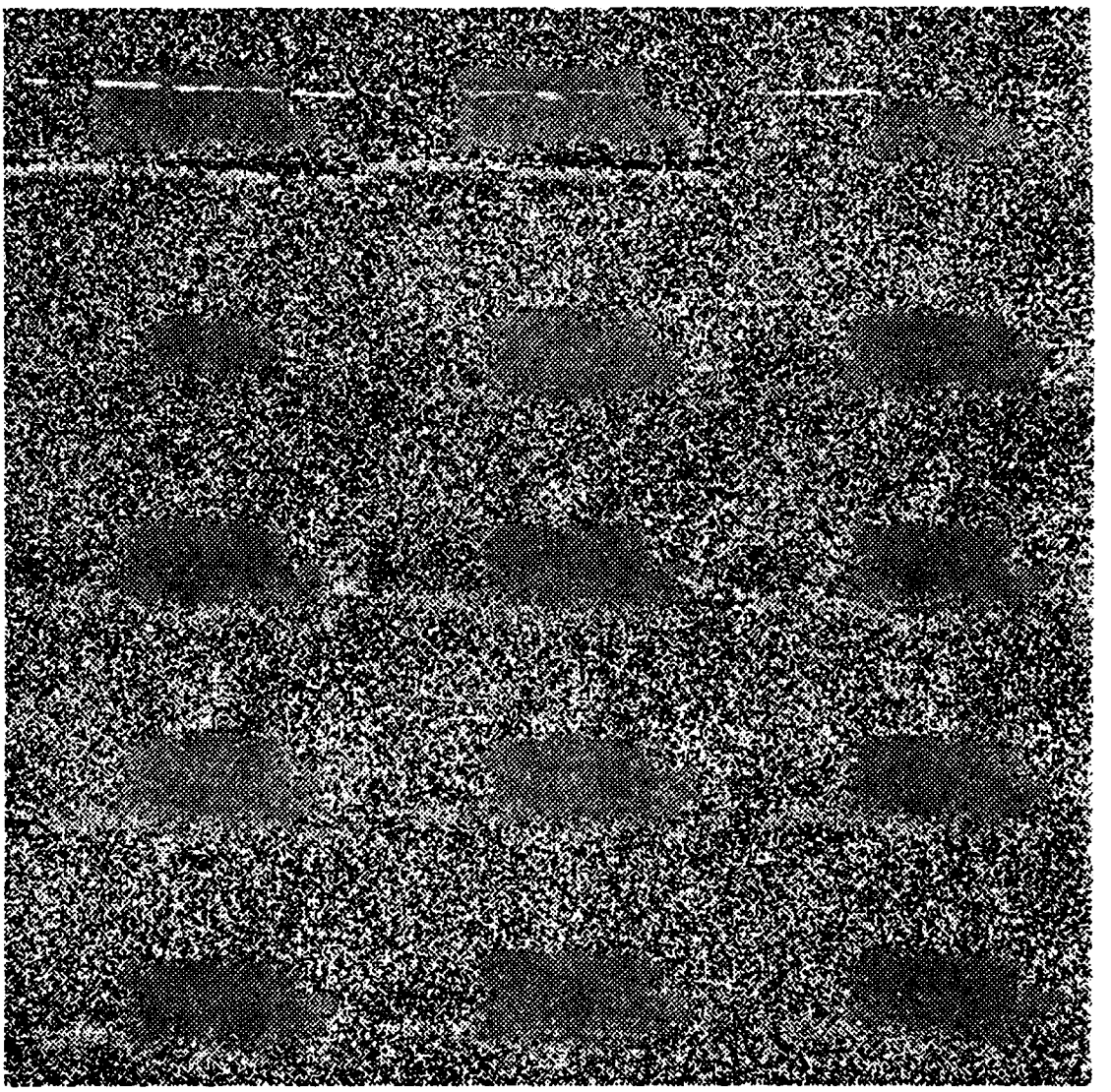


Figure 3.3.54 Five ton truck targets used in Laser Radar classification experiment.

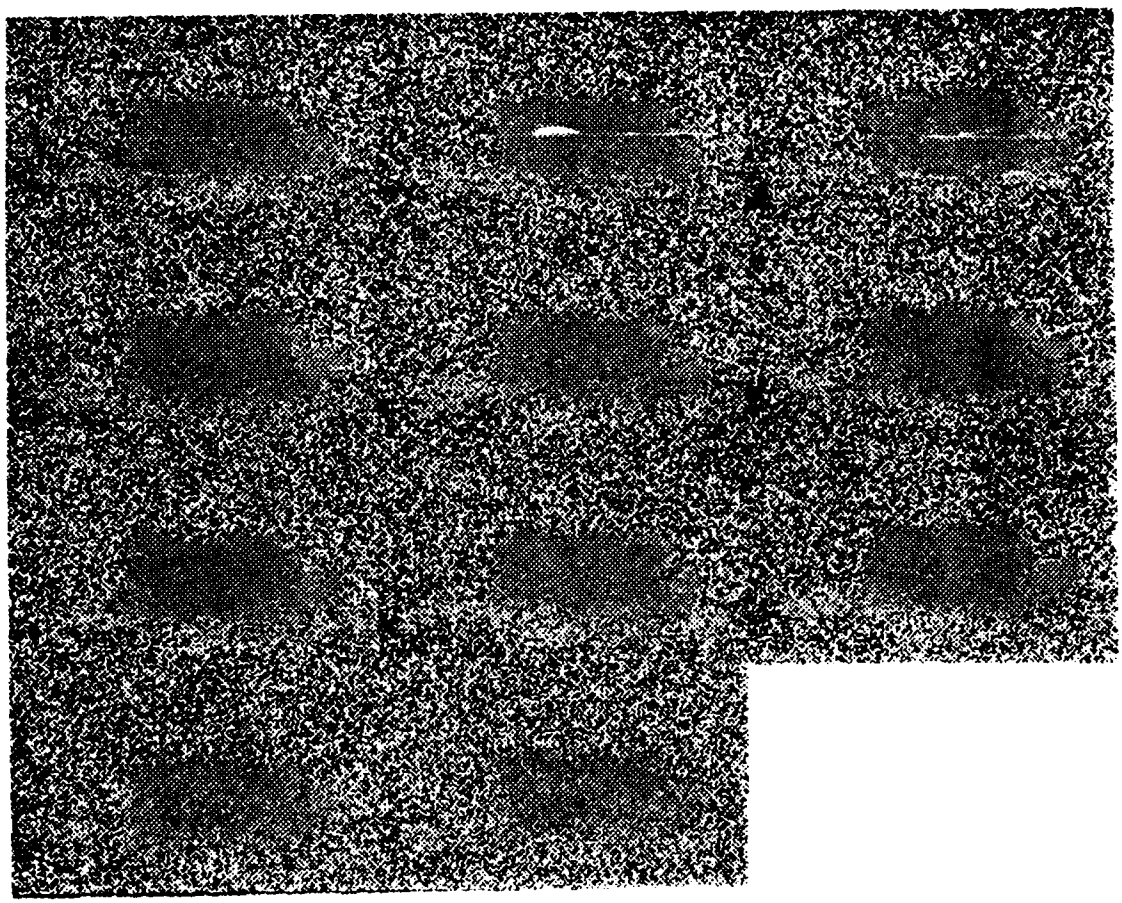


Figure 3.3.54 (continued)

m60a2.1.seg

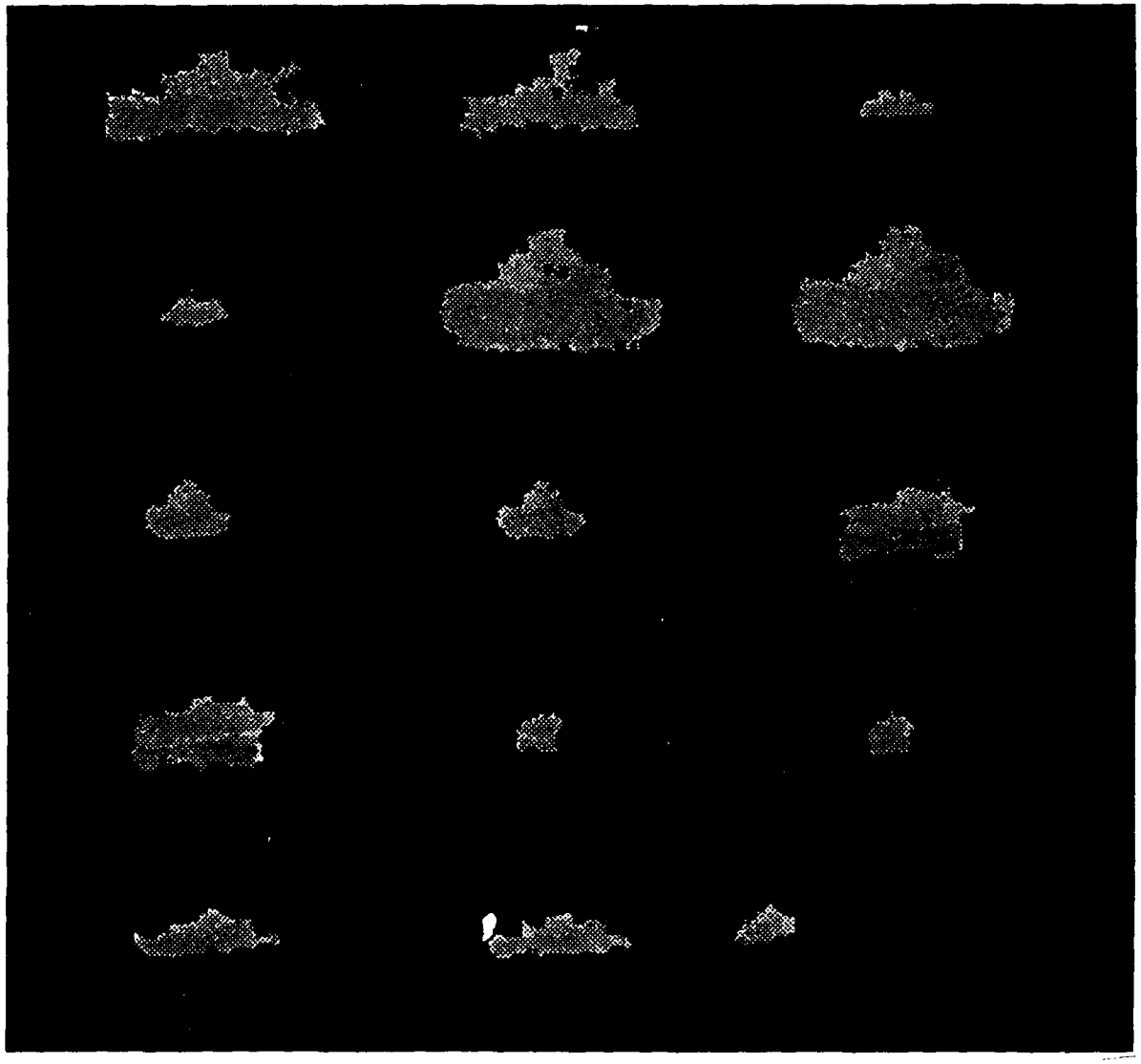


Figure 3.3.55 Segmented M60A2 targets used in the Laser Radar classification experiment.

m60a2.2.seg

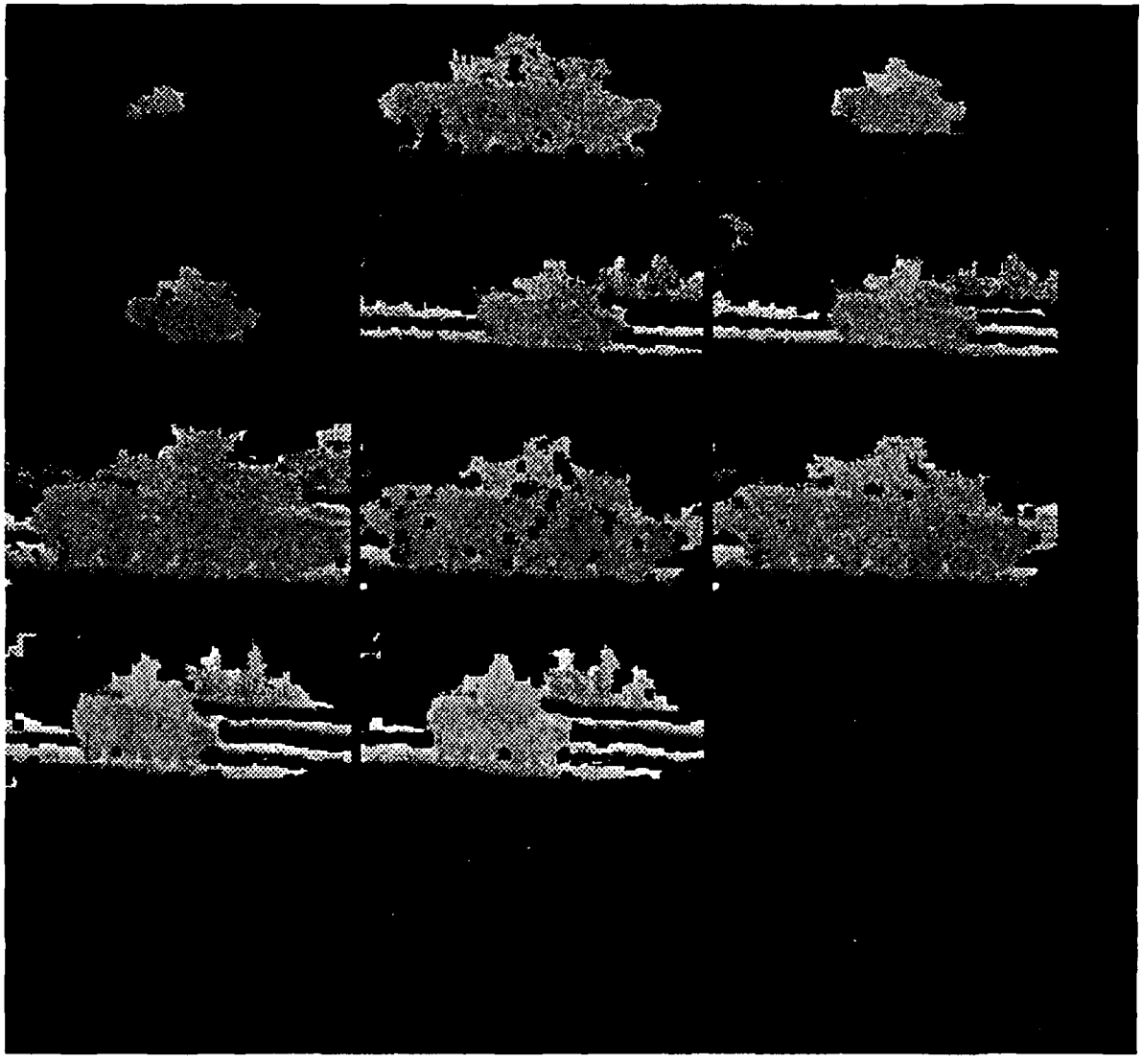


Figure 3.3.55 (continued)

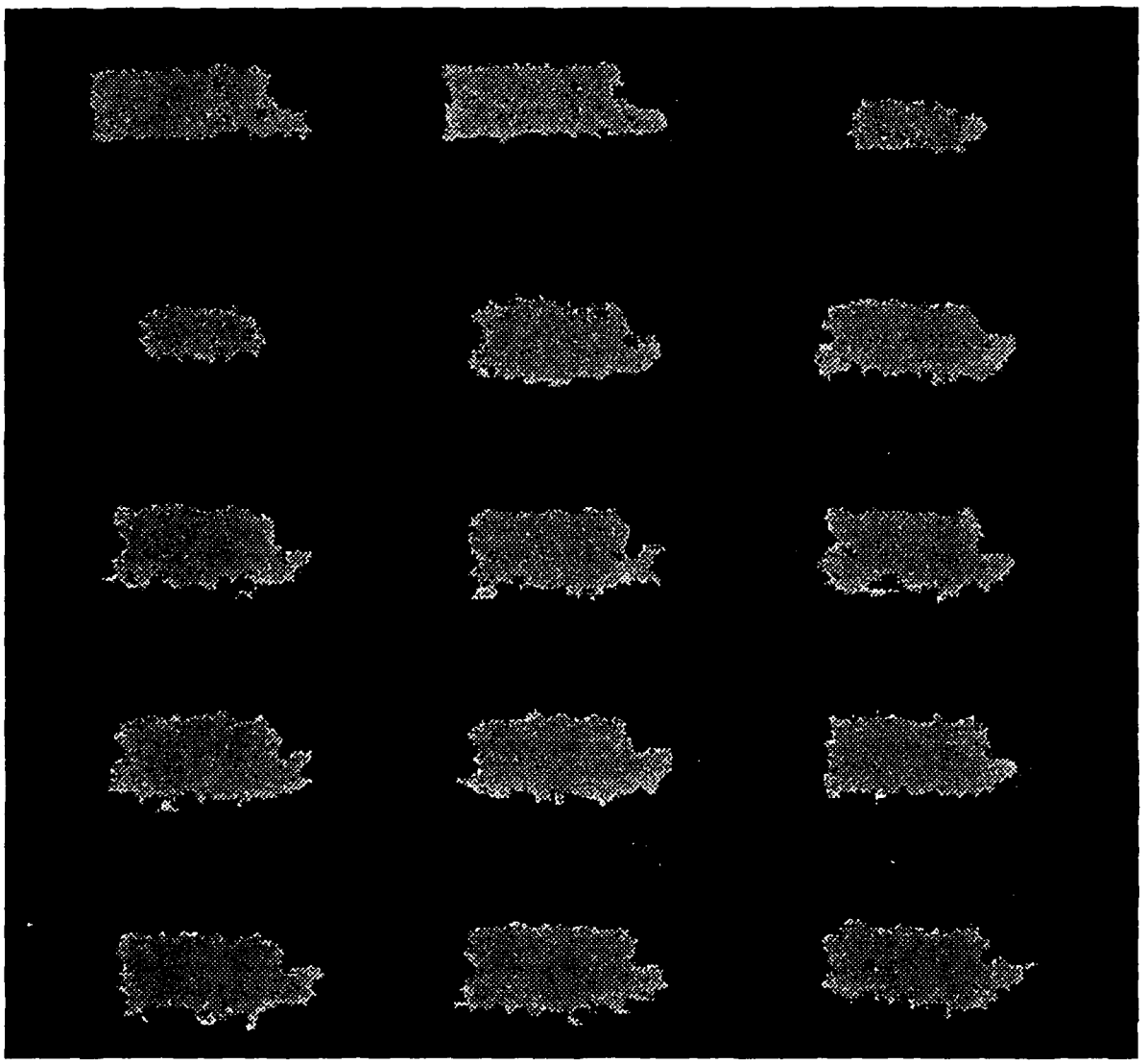


Figure 3.3.56 Segmented five ton truck targets used in the Laser Radar classification experiment.

5tt.2.seg

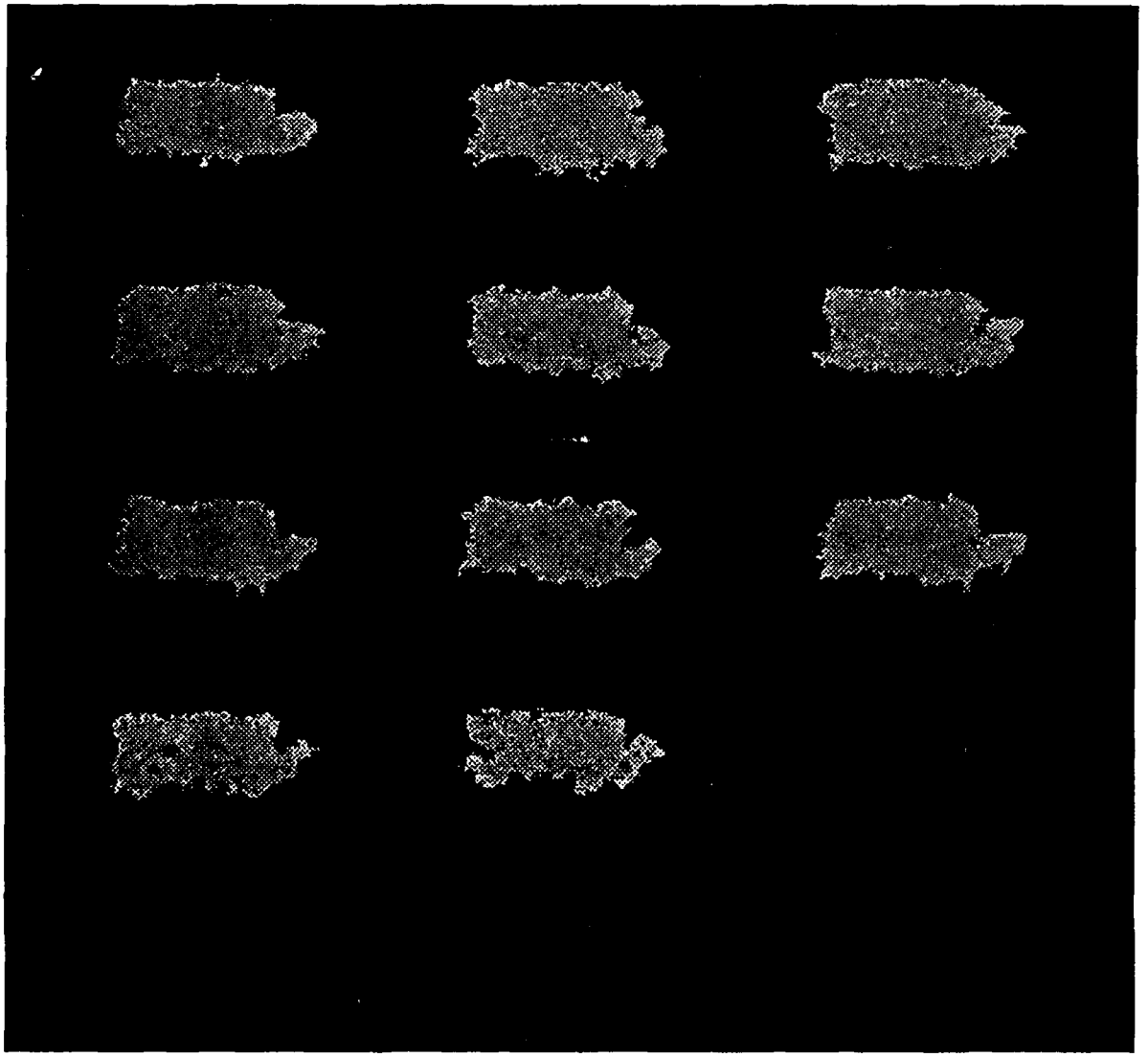


Figure 3.3.56 (continued)

decided to use the same silhouette based features that were used with the FLIR data. Tables 3.3.10 and 3.3.11 show the two features sets that were used.

Table 3.3.10 Shape features used in Laser Radar classification experiment.

Feature number in Section 2.1.1.2	Feature Name
7	Area
10	Height to width ratio
12	Rectangularity
13	Width to height ratio
18	Height squared over area

Table 3.3.11 Moment features used in Laser Radar classification experiment.

Moment	Equation Number in [KaYo87]
ϕ_1	A.10
ϕ_2	A.11
ϕ_3	A.12
ϕ_4	A.13
ϕ_5	A.14

Note that no range information used in the features set, just the outline data.†

3.3.4.3. Classification Results

The classification experiments like those used on the FLIR data in [KaYo87], were run on the segmented data. Table 3.3.12 shows the upper and lower bound estimates on the classification of the ladar data.

Table 3.3.12 Classification results on Laser Radar range data.

Feature Set	Lower Bound	Upper Bound
Shape	1.9%	5.7%
Moments	1.9%	1.9%

† i.e. the moment functions were computed using the value one where the target was and the value 0 where the background was.

These results show an improvement over the FLIR data. Remember that the targets ranged from 1 to 3 km which is the same range as the FLIR data. However, note that there were only two target classes and all the targets were viewed from the same view (side view), and from the same aspect angle (0 degrees).

3.3.4.4. Conclusions

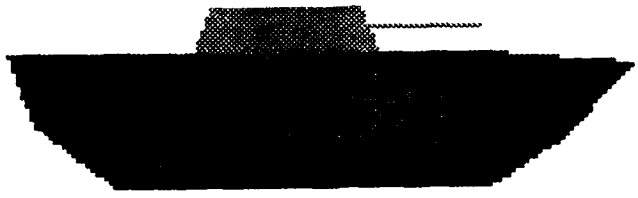
With such a limited set of data we feel that no definite conclusions can be drawn from this experiment. The results do show that when more LADAR data arrives we will be ready to conduct experiments with a more relistic selection of targets, ranges, clutter, etc.

3.4. HIGH LEVEL LADAR PROCESSING

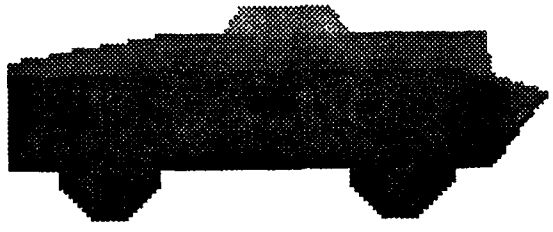
3.4.1. Production Systems for Target Recognition

This section presents two different high level solutions to the problem of object recognition from LADAR images. Our long term goal is to have an autonomous software system which will be able to recognize a given set of targets whenever these targets are present in LADAR range images in any orientation, combination, and number. The system should also be able to recognize partially obscured targets, and to generate multiple hypotheses with confidence values where appropriate.

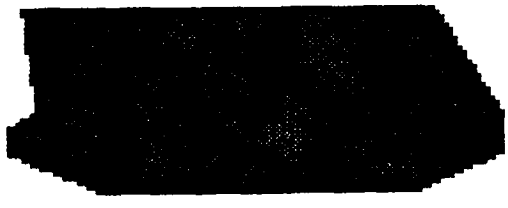
During this first series of experiments, the production systems will have to recognize four classes of targets: *BMP, BRDM2, M113* (all three of which are armored personnel carriers), and the *M60A1* tank. As an initial constraint we will only be considering the single target aspect illustrated in Figure 3.4.1. These targets may appear in any number and any combination in a given range image, and may have missing or noisy surfaces. It is assumed that low level processing has already segmented out the surfaces in the scene and computed their various attributes and relationships. Surface attributes may include location, orientation, dimensions, surface area, mean curvature, and planar patch fitting error. Among the relationships found between surfaces might be adjacency, the kind of edges separating them, and whether they have a convex or concave relationship. All of this information is output to a file in a format that may be understood by the expert systems. The data provided as input to the production systems in this section was hand extracted from model information and has had some error added to it in order to illustrate uncertainty reasoning and recovery from low level errors. Of course, future versions of our software will have to handle targets extracted from actual LADAR data at all possible aspects.



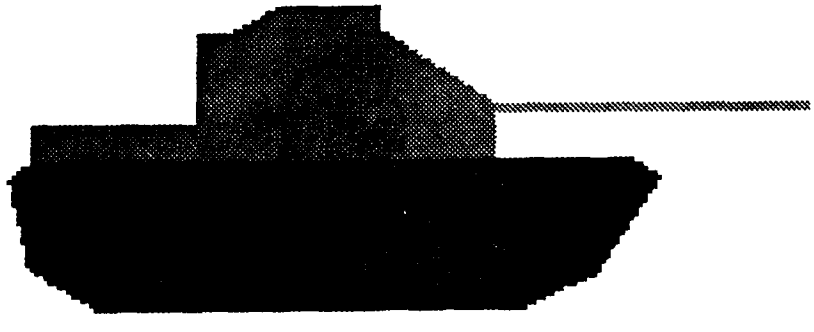
(a) BMP apc



(b) BRDM2 apc



(c) M113 apc



(d) M60A1 tank

Figure 3.4.1 Target aspect to be recognized by high level processing.

3.4.1.1. A Top-Down Goal-Driven Approach

The first target recognition expert system we developed is written in PROLOG. The source for the expert system shell may be found in Appendix E.1. Instead of merely expressing our rules in the usual conclusion-if-condition form PROLOG uses, the expert system shell is used to provide a more flexible system. A shell allows a user-friendly interface, makes it possible to use any format for expressing rules, keeps track of why the current line of reasoning is being pursued, provides for propagation of belief, and enables the system to show how a conclusion was reached, all of which are not done automatically by PROLOG alone.

3.4.1.1.1. Facts and Rules for Solving Goals

Our system design is strongly influenced by the language it is implemented in, reflecting the top-down, goal-driven nature of PROLOG. First, surface attributes and relationships found by the low level processing routines are read in by the expert system and are considered to be *facts*. Figure 3.4.2 is an example of such low level input to the system. Next, the user specifies a goal like

```
target isa m113
```

and the system uses *facts* and *rules* to determine its confidence in the goal. *Rules* have the format:

```
rulename :  if
            condition
      then
            conclusion
      with
             strength (N,S).
```

Condition is a possibly compound new goal to solve to determine the belief in the current goal *conclusion*. A goal is said to be *compound* if it is a disjunction or conjunction of subgoals. The *strength* clause contains factors used by the uncertainty reasoning scheme.

The predicate **explore** (see Appendix E.1) solves a goal by first checking if it has been asserted as fact, then seeing if there are any directly applicable rules which may be used to solve it. A rule is applicable if an instantiation of its *conclusion* matches the current goal, and the rule is used by exploring the *condition* goal and propagating the evidence from this exploration using the Prospector model as described below. If **explore** cannot find a matching fact or rule and the current goal is compound, the subgoals are explored and their results are combined to determine the confidence in the entire original goal. Finally, if the answer still cannot be determined, the *a priori* probability of the goal is used.

```
%%
%% Low-level output for range image containing BMP apc.
%%

%
%       Surface Attributes
%
%               attr_location( <surface id>, <x location>, <y location> )
%               attr_hwa( <surface id>, <height>, <width>, <area> )
%               attr_fit( <surface id>, <fit error> )
%
                attr_location(1, 2.3, 1.75).
                attr_hwa(1, 0.5, 0.4, 0.2).
                attr_fit(1, 250).
                attr_location(2, 3.0, 1.75).
                attr_hwa(2, 0.5, 0.4, 0.2).
                attr_fit(2, 342).
                attr_location(3, 3.7, 1.75).
                attr_hwa(3, 0.5, 0.4, 0.2).
                attr_fit(3, 501).
                attr_location(4, 4.6, 1.75).
                attr_hwa(4, 0.07, 1.224, 0.08568).
                attr_fit(4, 186).
                attr_location(5, 1.35, 1.35).
                attr_hwa(5, 0.4, 2.9, 0.58).
                attr_fit(5, 603).
                attr_location(6, 4.0, 1.25).
                attr_hwa(6, 0.5, 6.5, 2.67).
                attr_fit(6, 23).
                attr_location(7, 3.0, 0.5).
                attr_hwa(7, 1.0, 6.0, 5.5).
                attr_fit(7, 438).

%
%       Surface Relations
%
%               rel_adjacent( <surface id>, <surface id>, <edge type> )
%
                rel_adjacent(1, 2, snd).
                rel_adjacent(2, 3, snd).
                rel_adjacent(3, 4, jmp).
                rel_adjacent(5, 6, snd).
                rel_adjacent(1, 5, jmp).
                rel_adjacent(3, 6, jmp).
                rel_adjacent(5, 7, snd).
                rel_adjacent(6, 7, snd).
```

Figure 3.4.2 Example BMP surface attributes and relationships found by low level processing and input to our first expert system as facts.

3.4.1.1.2. The Prospector Model of Uncertainty Reasoning

The Prospector model [Bratko, Duda] is used to propagate the evidence from the next generation goal *condition* to the confidence in the parent goal *conclusion*, and makes use of the *strength* factors **N** and **S**. **N** ranges between 0 and 1 and tells how *necessary* the *condition* is for the *conclusion*; if the *condition* is false then the lower **N** is the less likely the *conclusion* is. **S** tells how *sufficient* the *condition* is for the *conclusion* and takes values greater than 1; if the *condition* is true then the higher **S** is the more likely the *conclusion* is. Figure 3.4.3 illustrates how probability is propagated using these strength factors, and how evidence is combined for compound goals. Note that posterior probability of the next generation goal *condition*, which is found using a fact or another rule, is used to determine the multiplier **M** via the graph. This multiplier is then used to change the odds of the parent goal *conclusion* in light of the current evidence. The following discussion describes how the *a priori* probabilities are derived and how the system rules work.

3.4.1.1.3. Recognition via Decomposition and Evidence Accumulation

Figure 3.3.4 illustrates the names given to the surfaces composing each target. The system tries to find evidence for a target by looking for its constituent parts. At the first level these are turrets, decks, and tracks. These are decomposed further until the surface level is reached. Figure 3.4.5 shows the breakdown of our four targets. Let us call a node at any level of the tree, including the leaves, a *construct*. The rules reflect the structure of the objects, and the system will generate object goals and break them down the same way the object is decomposed into component constructs. The system uses the rules to discover the object structure as it moves down the decomposition tree. On the way down the tree, the surface identities in this structure remain uninstantiated. Once the leaves of the tree are reached, the system uses the facts asserted by the low level processing to find surfaces meeting stated attribute requirements. A surface is acceptable if its attribute is within 2 percent of the required value. The system now "unwinds", moving back up the tree with these instantiations for surfaces. Consistency and adjacency checking is performed by the rules during this "unwinding" process, and the Prospector model is used to combine the acquired evidence.

The *a priori* probabilities are assigned using the decomposition tree. Since there are four possible targets at the top level, each is assigned an equal *a priori* probability of 0.25. The remaining construct probabilities are determined by breaking down the top level probabilities. Each sibling construct gets an equal share of its parent's probability, as also illustrated in Figure 3.4.5. These *a priori* probabilities are asserted into the database along with the rules. See Appendix E.2 for a complete list of the rules and probabilities for our targets.

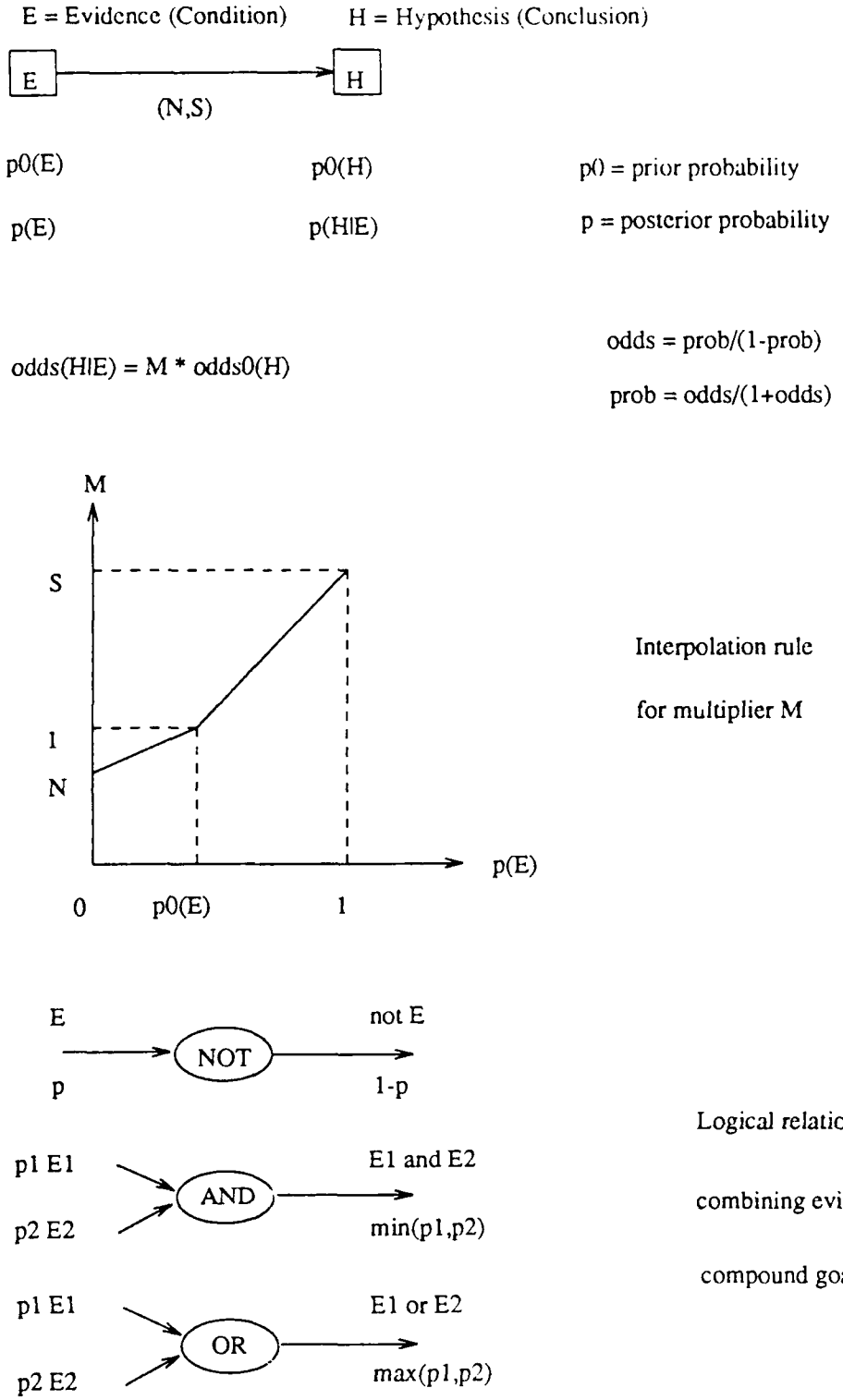
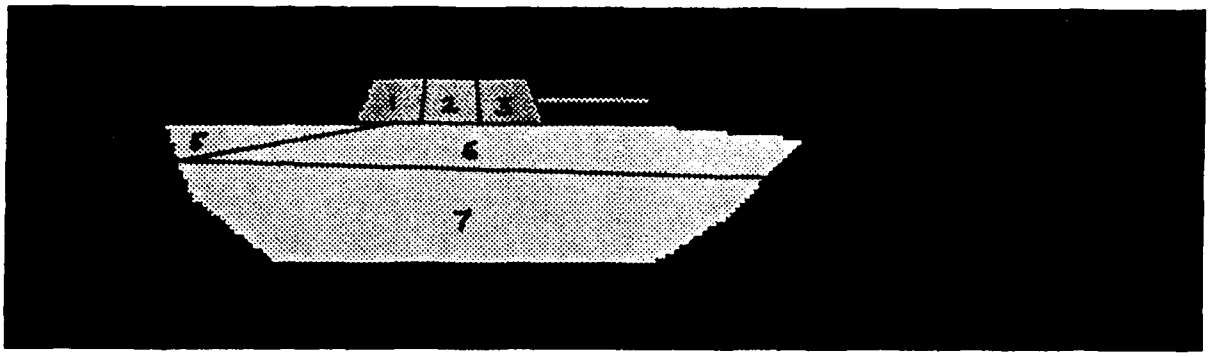


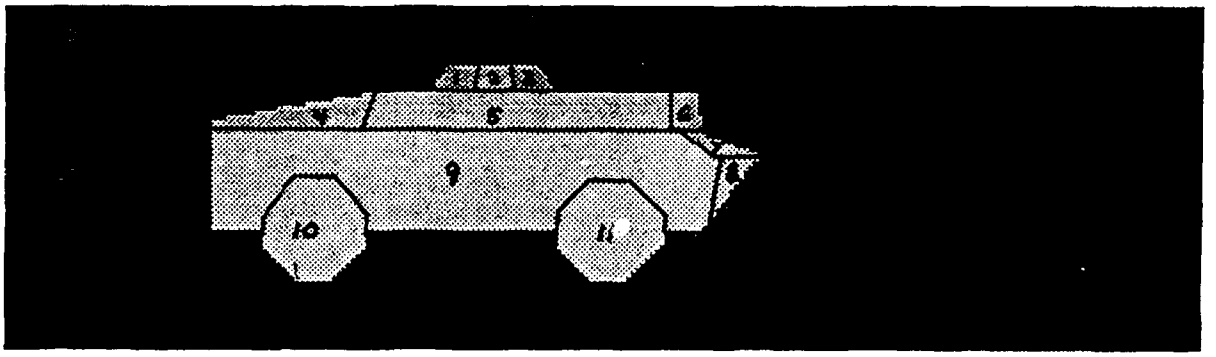
Figure 3.4.3 How evidence is propagated using the Prospector model.

BMP apc



1 : medium_dome_panel
2 : medium_dome_panel
3 : medium_dome_panel
4 : small_gunbarrel
5 : med_rear_deck
6 : med_fore_deck
7 : medium_track

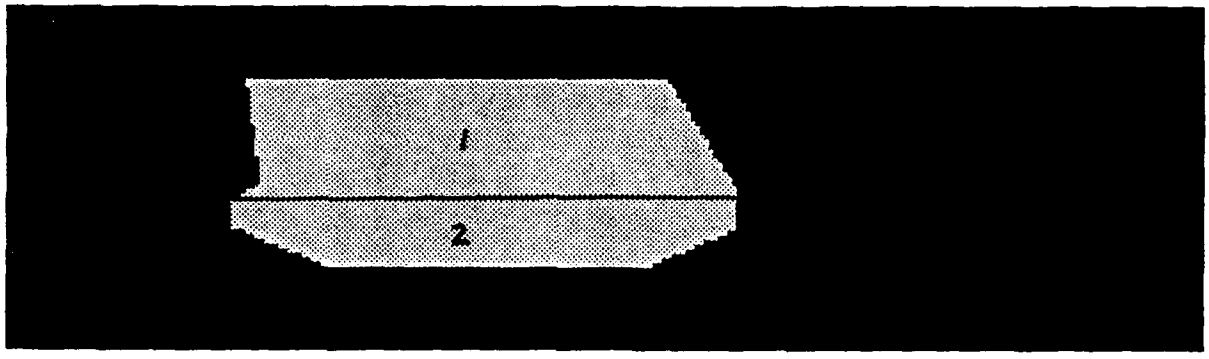
BRDM2 apc



1 : small_dome_panel
2 : small_dome_panel
3 : small_dome_panel
4 : lg_rear_deck
5 : lg_mid_deck
6 : lg_fore_deck
7 : top_front
8 : bottom_front
9 : med_side
10 : wheel
11 : wheel

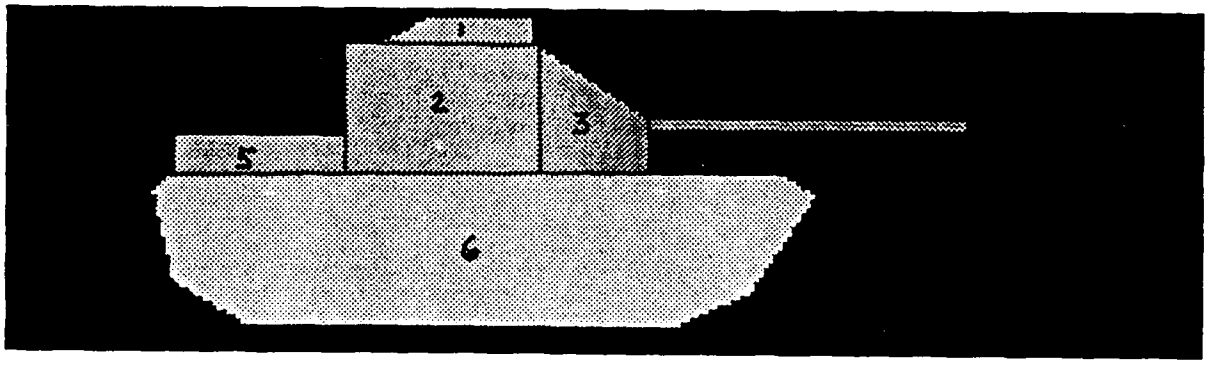
Figure 3.4.4 Identities of surfaces composing the targets.

M113 apc




```
1 : large_side
2 : small_track
```

M60A1 tank




```
1 : hatch
2 : turret_body
3 : turret_front
4 : large_gunbarrel
5 : small_deck
6 : large_track
```

Figure 3.4.4 continued.

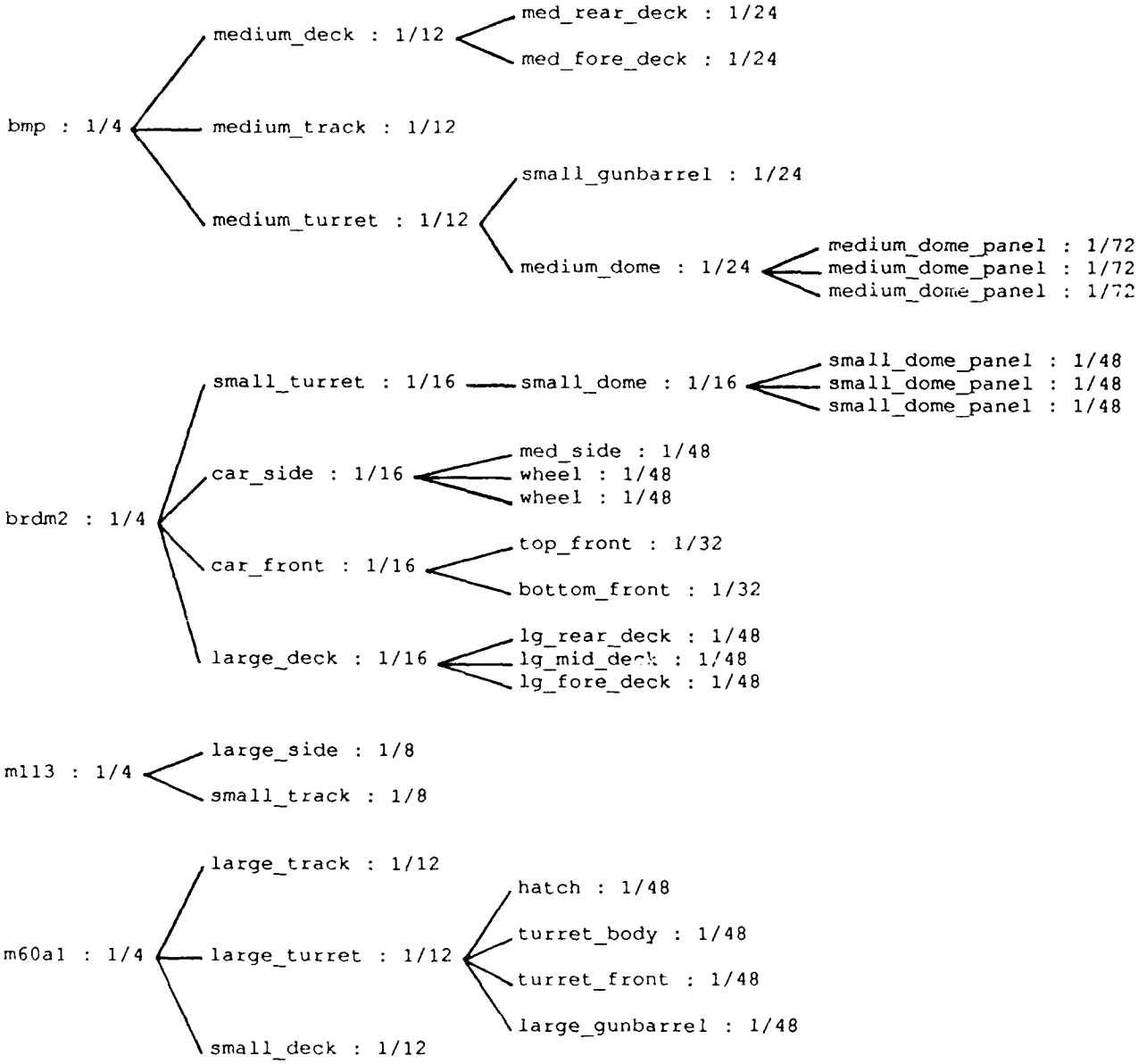


Figure 3.4.5 Tree showing object decompositions and a priori probabilities.

3.4.1.1.4. Default Reasoning

If the system cannot find a surface with the required attributes, we do not want PROLOG to fail. Instead, we want to build up a lower confidence in our hypothesized target class. For this reason, attribute value facts are assigned an *a priori* probability of 0.00001. This allows the system to continue the reasoning process even though a required surface is not present or was corrupted due to noise or some low level processing error. Later, we can tell that a given surface was not found because it remained uninstantiated.

The relative importance of an object component or surface is captured in the *strength* factors of the rule responsible for finding it. If a construct is relatively unimportant or often hard to find (e.g. a gun barrel), **N** should be set close to 1 so that the confidence in the entire object will not be severely penalized if it is not present. If a construct is important (e.g. a turret), a high **S** value will ensure that if it is found the confidence in the target class will increase greatly, and a low **N** value will greatly decrease the confidence if it is not found.

3.4.1.1.5. An Example and Some Conclusions

Figure 3.4.6 is an example session with the expert system. Once inside PROLOG, the first step is to read in the expert system shell, its utility routines, and the file containing the rules and *a priori* probabilities. Next, the file containing the surface attributes and relationships found by the low level processing is loaded. In this case, the input file is simulated information for a range image containing an *M113* apc. The expert system shell is asked whether an *M113* is present, and it responds with a confidence value (probability) of 0.998499. It is also able to show how it arrived at this conclusion. When asked if an *M60A1* tank is present, the program responds with a confidence value of 0.004831.

We discovered some serious problems with the high level approach, which made implementing it very educational. The primary source of trouble is that it is top-down. Since the user obviously doesn't know what is in the range image (if he did, he probably wouldn't be wasting his time talking to our software), he is not in a position to ask the best questions and probably shouldn't even be consulted. On the other hand, the computer doesn't know what is in the range image either, and so must try all possibilities. This gives us an unreasonably large search space and, unfortunately, there is no way to prune it using our approach. Once we have found a tank we cannot exclude the possibility of there being an apc in the image as well, and we don't even know if we have found all of the tanks that are there. The system also is not smart enough to find the solution with the highest confidence value, but rather returns the first solution it finds. Finding the most probable solution would require searching the entire space, which would take an unacceptable amount of time.

There are also complications arising from the fact that we allow a surface to remain uninstantiated in order to handle low level processing errors, that we never really force

```
% prolog
C-Prolog version 1.5
| ?- [expert, utilities, target_rules].
expert consulted 5144 bytes 1.5 sec.
utilities consulted 1100 bytes 0.400001 sec.
target_rules consulted 12588 bytes 3.46667 sec.

yes
| ?- ['m113.surfaces'].
m113.surfaces consulted 280 bytes 0.133334 sec.

yes
| ?- expert.

Question, please:
|: target isa m113.

target isa m113 : 0.998499
Would you like to see how? yes.

target isa m113 : 0.998499 was derived by rule3 from
     large_side(1) and small_track(2) and m113_adjacencies(1,2) : 0.998602 was derived from
          large_side(1) : 0.998602 was derived by rule16 from
               elevation(1,1.35) and height(1,1.3) and width(1,5.3) and area(1,6.2) and planar(1) : 1 was derived from
                    elevation(1,1.35) : 1 was found as a fact
                    and
                    height(1,1.3) and width(1,5.3) and area(1,6.2) and planar(1) : 1 was derived from
                         height(1,1.3) : 1 was found as a fact
                         and
                         width(1,5.3) and area(1,6.2) and planar(1) : 1 was derived from
                              width(1,5.3) : 1 was found as a fact
                              and
                              area(1,6.2) and planar(1) : 1 was derived from
                                   area(1,6.2) : 1 was found as a fact
                                   and
                                   planar(1) : 1 was found as a fact
          and
          small_track(2) and m113_adjacencies(1,2) : 0.9993 was derived from
               small_track(2) : 0.9993 was derived by rule11 from
                    elevation(2,0.35) and height(2,0.7) and width(2,5.3) and area(2,3.08) and planar(2) : 1 was derived from
                         elevation(2,0.35) : 1 was found as a fact
                         and
                         height(2,0.7) and width(2,5.3) and area(2,3.08) and planar(2) : 1 was derived from
                              height(2,0.7) : 1 was found as a fact
                              and
                              width(2,5.3) and area(2,3.08) and planar(2) : 1 was derived from
                                   width(2,5.3) : 1 was found as a fact
                                   and
                                   area(2,3.08) and planar(2) : 1 was derived from
                                        area(2,3.08) : 1 was found as a fact
                                        and
                                        planar(2) : 1 was found as a fact
               and
               m113_adjacencies(1,2) : 0.9993 was derived by rule19 from
                    adjacent(1,2,jmp) : 1 was found as a fact

More solutions?  no.

yes
| ?- expert.

Question, please:
|: target isa m60a1.

target isa m60a1 : 0.00483114
Would you like to see how? no.

More solutions?  no.

yes
| ?-
[ Prolog execution halted ]
%
```

Figure 3.4.6 Example session with first expert system.

things to fail since we are using uncertainty reasoning, and that PROLOG backtracks to the most recent choice it made in order to explore other possibilities. When we look for other solutions via backtracking, the system will return all of the possible permutations of the current set of surfaces with subsets of them uninstantiated before proceeding to a new set of surfaces making up the next target. In the same way, it is also difficult to make the system recover from errors in early choices. The system tends to propagate a low confidence value and continue working if an adjacency constraint is not met, instead of backtracking and making a proper choice for a surface.

Besides gross inefficiencies arising from its top-down approach, the system also suffers from parameters that are difficult to set. Strength factors are determined heuristically through experience in using the system. The effect of the factors is not local; tweaking the factors in one rule affects the interaction with sibling rules, and side effects propagate up and across the tree. This makes the system hard to adjust, and slight structural reconfigurations of the system require an extensive amount of work. Keeping all of these problems in mind, we had far more success developing the expert system described in the next section.

3.4.1.2. A Data-Driven Bottom-Up Approach

The second target recognition expert system we developed is written in OPS5 [BroFar]. Like the first one, it reflects the character of the programming language it is implemented in. OPS5 is a data-driven language, and is amenable to the bottom-up approach we wish to experiment with. We found the natural structure of OPS5 rules and its flow of control to be sufficient for our purposes, and so we did not build a shell on top of it. At this point we wish to note that the OPS5 production system programming language is implemented in LISP and runs in the LISP environment.

3.4.1.2.1. Data Objects, Rules, and the Inference Engine

Before we delve into the details of the expert system, a few words about OPS production system architecture. A data store, called *working memory*, serves as a global database of symbols representing facts and assertions about the problem. The data are instances of *objects*, which may represent either physical objects (or facts) related to the domain of application or conceptual objects (such as goals) related to the problem solving strategy. An instantiated data object is called a *working memory element*. Figure 3.4.7 contains the declarations of the data object classes used by the expert system. The capitalized word is the name of the object class, and is followed by attribute names associated with the class. The classes *Start* and *Phase* represent conceptual objects used in flow of control, while *World, Surface, Adjacent,* and *Construct* represent physical objects or facts.

A set of rules constitutes an OPS program, and resides in the *production memory*. Rule definitions have the format:

```
(literalize Start)      ; Element class for initialization.

(literalize Phase
        description     ; expand, hypothesize, build, or clean
                        ;       expand : fill in missing attributes & relations
                        ;       hypothesize : propose surface identities
                        ;       build : construct higher level objects
                        ;       clean : perform garbage collection
        status          ; active, finished
                                                                        )

(literalize World
        classes         ; number of object classes
        threshold       ; confidence threshold
                                                                        )

(literalize Surface
        id              ; unique number > 0 identifying surface
        x_loc           ; x-coordinate of surface location
        y_loc           ; y-coordinate of surface location
        height          ; surface "height" (x-coordinate span)
        width           ; surface "width" (y-coordinate span)
        depth           ; surface "depth" (z-coordinate span)
        area            ; surface area
        h_to_w          ; height to width ratio
        fit_error       ; mean planar patch fitting error
                                                                        )

(literalize Adjacent
        edge_type       ; type of boundary between two surfaces
                        ;       jmp : jump edge due to range discontinuity
                        ;       crv : curvature edge
                        ;       snd : surface normal disparity edge
        first           ; first surface
        second          ; second surface
                                                                        )

(vector-attribute surfaces)
(literalize Construct
        type            ; the name of the construct (e.g. bmp, small_turret,
                        ;       gun_barrel) or flag value "output"
        confidence      ; amount of belief in existence of construct
        surfaces        ; ordered list of surfaces in construct
                                                                        )
```

Figure 3.4.7 Declarations of working memory element classes.

```
(p rulename
       ( condition element 1 )
        ...
       ( condition element n )
-->
       ( action 1 )
        ...
       ( action m )          )
```

The *condition* part of the rule is a list of element templates which are matched against the contents of working memory. The *action* part of a rule is a list of instructions which may modify the working and production memories. The complete set of rules for our system is listed in Appendix F.

The third component of the OPS production system architecture is the *inference engine*. It must determine which rules are relevant to a given working memory configuration and choose one to execute. This selection or control strategy is sometimes called *conflict resolution*. Figure 3.4.8 illustrates the production system architecture used by OPS. The inference engine iterates through a cycle of three action states. In the first state, *MATCH*, the machine finds all of the rules that are satisfied by the current contents of the working memory. The rule matchings that are found are all candidates for execution, and are known collectively as the *conflict set*. The same rule may appear in the conflict set several times if it is satisfied by different sets of working memory elements. The *SELECT* state applies some predetermined selection strategy to determine which rules in the conflict set will actually be executed. These rules are then fired in the *EXECUTE* state, which usually produces some change in production and/or working memory. Control then cycles back to the *MATCH* state. The program terminates normally when the conflict set is empty.

In this system, control is based on frequent re-evaluation of the data states, not on any static control structure of the program. It therefore uses a data-driven philosophy. This works well with the bottom-up approach. The program has been designed to operated in *phases*. Each phase has its own particular goals and uses a disjoint subset of the system rules to accomplish them. Figure 3.4.9 contains the flow of control production rules which perform the phase transitions. The currently active phase is kept track of using the *Phase* working memory element. We will now describe the individual phases in detail.

3.4.1.2.2. Phase 1: Fill in Missing Attributes and Relations

The purpose of Phase 1 is to overcome several deficiency of OPS, including the inability of OPS to compute items on the left hand (condition) side of rules and the lack of an easy way to deal with reflexive relations. During this "expand" phase surface attributes

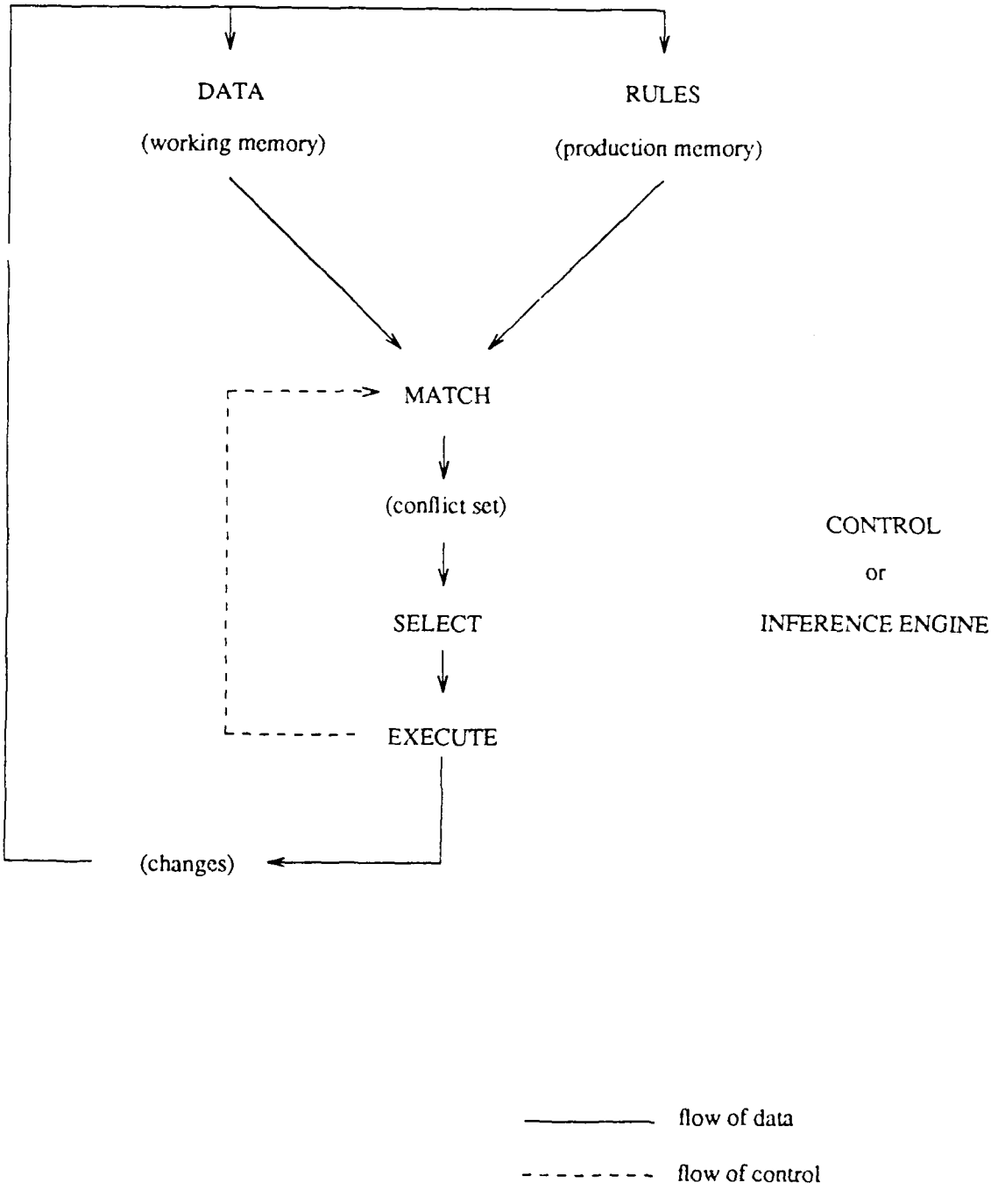


Figure 3.4.8 OPS5 production system architecture.

```
;; Flow of Control Production Rules

(p Phase-Finished                                                    ; 001
   { (Phase ^status active) <phase> }
-->
   (modify <phase> ^status finished)
                                                                   ) ; 001

(p Expand-to-Hypothesize                                             ; 002
   { (Phase ^description expand ^status finished) <phase> }
-->
   (modify <phase> ^description hypothesize ^status active)
                                                                   ) ; 002

(p Hypothesize-to-Build                                              ; 003
   { (Phase ^description hypothesize ^status finished) <phase> }
-->
   (modify <phase> ^description build ^status active)
                                                                   ) ; 003

(p Build-to-Clean_Up                                                 ; 004
   { (Phase ^description build ^status finished) <phase> }
-->
   (modify <phase> ^description clean ^status active)
                                                                   ) ; 004

(p Output_Results                                                    ; 005
   { (Phase ^description clean ^status finished) <phase> }
-->
   (remove <phase>)
                                                                   ) ; 005
```

Figure 3.4.9 Flow of control production rules performing phase transitions.

and relationships not explicitly represented are calculated and filled in. *Height-to-width ratio*, an attribute which could be easily computed from the *height* and *width* attributes and checked on the fly if another programming language were used, here must be explicitly stored if it is to be checked in the precondition part of a rule. In order to handle the relation *Adjacent* without resorting to an excessive number of rules, it was decided that **adjacent(B,A)** would be asserted into working memory for every occurrence of **adjacent(A,B)**.

While building up higher level constructs from surfaces, as will be described later, this system uses a default surface, with id number 0, in place of any surface which may not have been found by the low level processing. Since it is not known at this stage of the processing which surface or surfaces the default surface will have to substitute for, Phase 1 also asserts the adjacency of surface 0 with every known surface. Once again we have tried to keep the rules simple and few in number at the expense of working memory use. This should allow the production system to remain lucid while taking full advantage of the efficient Rete pattern matcher used by OPS. Figure 3.4.10 contains example Phase 1 rules.

3.4.1.2.3. Phase 2: Hypothesize Surface Identities

In Phase 2 the expert system attempts to make hypotheses about the identities of surfaces based on their attributes. The surface names used are the same as those used by our first expert system, and are given in Figure 3.4.4. If the attributes of a surface are within the ranges specified by the preconditions of a hypothesis-generating rule, that surface is asserted as a *Construct* consisting of the single surface. Say there are **n** classes in the particular domain in which we are working and that the *a priori* probability of the single-surface construct which we are hypothesizing is **c**, then the confidence in our construct using the given surface is **n*c**. The reason for this will be given in the description of the next phase. The *a priori* probability for a given construct was asserted by the initialization rule (*Initialize*, rule number *999*, see Figure 3.4.11), and it can be easily recognized because it consists of the default surface (id number 0).

A "clean up" rule that is active at the end of this phase deserves special mention. This rule (*H-Clean-Up*, rule number *222*) removes the default constructs for which hypothetically valid surfaces have been found. This is to avoid the use of the default surface when valid surfaces exist, and thereby prevent the building up of a large number of false hypothetical constructs, each representing a different permutation of where the default surface could have been used. The only problem with this temporary fix is that if two instances of a target class occur in one image, one complete and the other with a missing surface, the default is no longer around to contribute to the construction of the incomplete target. The robust and necessary solution to this nontrivial problem is to allow the system to build up the constructs using the default surface, and then have rules that recognize and discard these "more general" instantiations of the construct when "more specific"

```
;; Phase 1 Production Rules : Fill in missing surface attributes & relations

(p Expand-H_to_W                                                    ; 101
   (Phase ^description expand ^status active)
   { (Surface ^h_to_w nil ^height <h> ^width <w>) <s> }
-->
   (modify <s> ^h_to_w (compute <h> // <w>) )
                                                                    ) ; 101

(p Expand-Adjacent_1                                                ; 102
   (Phase ^description expand ^status active)
 - (Adjacent ^first 0 ^second 0)
-->
   (make Adjacent ^edge_type jmp ^first 0 ^second 0)
   (make Adjacent ^edge_type snd ^first 0 ^second 0)
   (make Adjacent ^edge_type crv ^first 0 ^second 0)
                                                                    ) ; 102

(p Expand-Adjacent_2                                                ; 103
   (Phase ^description expand ^status active)
   (Surface ^id { <s>  <> nil  <> 0 })
 - (Adjacent ^first <s> ^second 0)
-->
   (make Adjacent ^edge_type jmp ^first <s> ^second 0)
   (make Adjacent ^edge_type snd ^first <s> ^second 0)
   (make Adjacent ^edge_type crv ^first <s> ^second 0)
                                                                    ) ; 103

(p Expand-Adjacent_3                                                ; 104
   (Phase ^description expand ^status active)
   (Adjacent ^edge_type <type> ^first <s1> ^second <s2>)
 - (Adjacent ^edge_type <type> ^first <s2> ^second <s1>)
-->
   (make Adjacent ^edge_type <type> ^first <s2> ^second <s1>)
                                                                    ) ; 104
```

Figure 3.4.10 Example Phase 1 production rules.

```
;; Rule to Initialize Working Memory
;;      When an element of class Start enters working memory,
;;      this rule initializes the database of a priori probabilities.

(p Initialize                                                          ; 999
   { <initialize> (Start) }
-->
   (remove <initialize>)
   (make Phase ^description expand ^status active)
   (make World ^classes 4 ^threshold 0.5)
   (make Construct ^type bmp                ^confidence 0.25    )
   (make Construct ^type medium_track       ^confidence 0.08333 ^surfaces 0)
   (make Construct ^type medium_deck        ^confidence 0.08333 )
   (make Construct ^type med_rear_deck      ^confidence 0.04166 ^surfaces 0)
   (make Construct ^type med_fore_deck      ^confidence 0.04166 ^surfaces 0)
   (make Construct ^type medium_turret      ^confidence 0.08333 )
   (make Construct ^type small_gunbarrel    ^confidence 0.04166 ^surfaces 0)
   (make Construct ^type medium_dome        ^confidence 0.04166 )
   (make Construct ^type medium_dome_panel  ^confidence 0.01389 ^surfaces 0)
   (make Construct ^type brdm2              ^confidence 0.25    )
   (make Construct ^type small_turret       ^confidence 0.0625  )
   (make Construct ^type small_dome         ^confidence 0.0625  )
   (make Construct ^type small_dome_panel   ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type large_deck         ^confidence 0.0625  )
   (make Construct ^type lg_rear_deck       ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type lg_mid_deck        ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type lg_fore_deck       ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type car_side           ^confidence 0.0625  )
   (make Construct ^type med_side           ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type wheel              ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type car_front          ^confidence 0.0625  )
   (make Construct ^type top_front          ^confidence 0.03125 ^surfaces 0)
   (make Construct ^type bottom_front       ^confid· ce 0.03125 ^surfaces 0)
   (make Construct ^type m113               ^confidence 0.25    )
   (make Construct ^type large_side         ^confidence 0.125   ^surfaces 0)
   (make Construct ^type small_track        ^confidence 0.125   ^surfaces 0)
   (make Construct ^type m60a1              ^confidence 0.25    )
   (make Construct ^type large_turret       ^confidence 0.08333 )
   (make Construct ^type hatch              ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type turret_body        ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type turret_front       ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type large_gunbarrel    ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type small_deck         ^confidence 0.08333 ^surfaces 0)
   (make Construct ^type large_track        ^confidence 0.08333 ^surfaces 0)
                                                                    ) ; 999

;; Make initialization class element
(make Start)
```

Figure 3.4.11 Production rule that initializes working memory by asserting the a priori probabilities of surfaces and higher level constructs.

ones exist. Example Phase 2 production rules appear in Figure 3.4.12.

3.4.1.2.4. Phase 3: Build Higher Level Constructs

This is a very important phase, the one in which we will address the assignment of *a priori* probabilities and uncertainty propagation. Phase 3 uses the surface constructs hypothesized in Phase 2 to build higher level constructs such as turrets and decks and, eventually, complete targets. Adjacencies are checked as these constructs are put together, and the default surface may be used in place on one that the low level processing may have missed. Figure 3.4.13 contains example Phase 3 production rules.

Since there are four possible targets, the *a priori* probability of each object class is 0.25, and would in general be the reciprocal of the number of classes (**1/n**). Figure 3.4.5 is a diagram showing how each of the four targets is broken down into smaller constructs. The fraction following each construct node in the tree is the *a priori* probability of that construct, and is equal to the sum of the *a priori* probabilities of its immediate children. This is also how evidence is propagated up the tree. The confidence value for a higher level construct is the sum of the confidence values for its constituent parts. In this particular probability assignment, all constituent constructs (those which are not complete targets by themselves) were given "equal" weight in that all sibling constructs contribute an equal amount of confidence to their parent construct. The weights of these siblings could be shifted to emphasize the relative importance or unimportance of a given part, so long as the sum of their confidence values remained the same and the effect is propagated down the tree (each sibling construct's confidence must still equal the sum of its immediate children's confidences). Figure 3.4.14 is an example of such a weighted tree.

At the lowest level of the tree, surfaces may provide positive evidence for the existence of a construct, but the lack of a surface cannot disprove the existence of a construct, since the surface may have been missed because of noise or some other error in low level processing. If a surface is missing, the default surface is automatically used in its place with the correct *a priori* probability. Notice that if a target is built up entirely of default surfaces, the highest confidence we can have in that target is its *a priori* value, which is precisely what will be computed. However, if all the constituent surfaces were found in the image, their respective *a priori* probabilities would have been multiplied by the number of classes **n** to obtain new confidence values (see Phase 2 description), which would result in a confidence value of 1.0 for the whole target. The more surfaces we find (or, in the unequally weighted construct case, the higher the number of "important" surfaces found), the higher our confidence in the composite target.

```
;; Phase 2 Production Rules : Hypothesize Surface Identities

(p H-turret_body                                                    ; 217
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type turret_body ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 2.254  < 2.346 }    ; elevation 2.3 +- 2%
                ^height         { > 1.372  < 1.428 }    ; height 1.4 +- 2%
                ^width          { > 1.96  < 2.04 }      ; width 2.0 +- 2%
                ^area           { > 2.744  < 2.856 }    ; area 2.8 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type turret_body
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 217

(p H-large_gunbarrel                                                ; 219
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type large_gunbarrel ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 2.107  < 2.193 }    ; elevation 2.15 +- 2%
                ^area           { > 0.4175  < 0.4345 }  ; area 0.426 +- 2%
                ^h_to_w         { > 0.02303  < 0.02397 }; h_to_w 0.0235 +- 2%
                                                        )
-->
   (make Construct      ^type large_gunbarrel
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 219

(p H-Clean_Up                                                       ; 222
   (Phase ^description hypothesize ^status finished)
   { <prior> (Construct ^type <t> ^surfaces 0) }
   (Construct ^type <t> ^surfaces { <> 0 } )
-->
   (remove <prior>)
                                                                    ) ; 222
```

Figure 3.4.12 Example Phase 2 production rules.

```
;; Phase 3 Production Rules : Build Higher Level Constructs

(p Build-m60a1                                                          ; 312
   (Phase ^description build ^status active)
   (Construct    ^type large_turret       ^confidence <c1>
                 ^surfaces {<s1> <> nil} <s2> <s3> <s6>)
   (Construct    ^type small_deck         ^confidence <c2>
                 ^surfaces <s4>)
   (Construct    ^type large_track        ^confidence <c3>
                 ^surfaces <s5>)
   (Adjacent ^first <s2> ^second <s4> ^edge_type jmp)
   (Adjacent ^first <s2> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s3> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s4> ^second <s5> ^edge_type jmp)
-->
   (make Construct       ^type m60a1
                         ^confidence (compute <c1> + <c2> + <c3>)
                         ^surfaces <s1> <s2> <s3> <s4> <s5> <s6>)
                                                                        ) ; 312

(p B-large_turret                                                       ; 313
   (Phase ^description build ^status active)
   (Construct    ^type hatch              ^confidence <c1>
                 ^surfaces <s1>)
   (Construct    ^type turret_body        ^confidence <c2>
                 ^surfaces <s2>)
   (Construct    ^type turret_front       ^confidence <c3>
                 ^surfaces <s3>)
   (Construct    ^type large_gunbarrel    ^confidence <c4>
                 ^surfaces <s4>)
   (Adjacent ^first <s1> ^second <s2> ^edge_type jmp)
   (Adjacent ^first <s2> ^second <s3> ^edge_type snd)
   (Adjacent ^first <s3> ^second <s4> ^edge_type jmp)
-->
   (make Construct       ^type large_turret
                         ^confidence (compute <c1> + <c2> + <c3> + <c4>)
                         ^surfaces <s1> <s2> <s3> <s4>)
                                                                        ) ; 313
```

Figure 3.4.13 Example Phase 3 production rules.

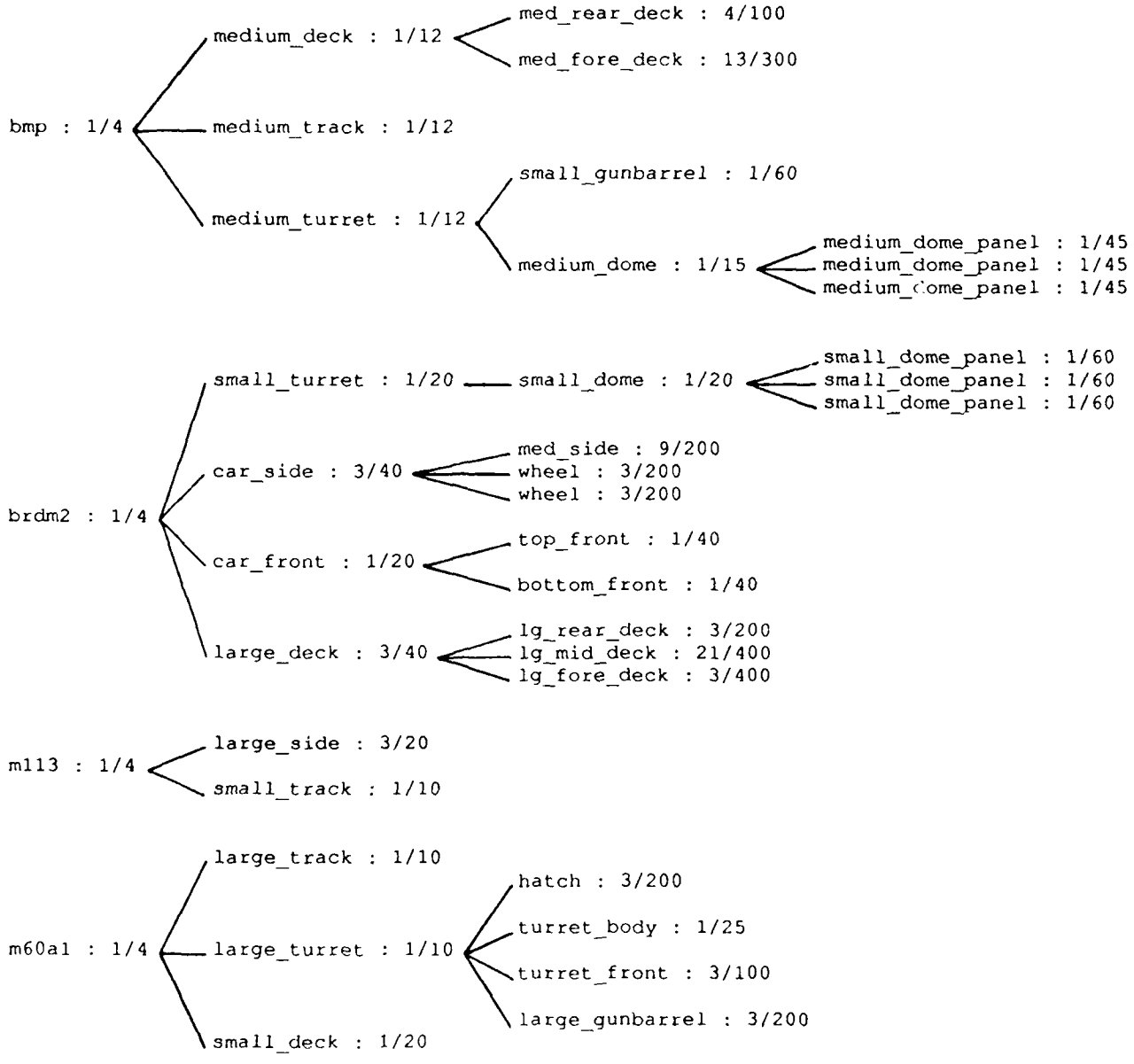


Figure 3.4.14 A priori probabilities with constructs weighted according to their relative importance.

3.4.1.2.5. Phases 4 & 5: Garbage Collection and Output of Results

At this point we have done the hard part of our processing. In Phase 4 we throw away those constructs whose confidence is less than our likelihood threshold, and then clean up working memory. We are done with surface attribute and relationship information at this point, so all of that excess baggage may be discarded. All that should remain in working memory are constructs we are confident in and the *World* element describing our problem domain. Figure 3.4.15 contains example Phase 4 production rules. Phase 5 presents our results by writing out each construct, our confidence in it, and the list of surfaces making it up. The output rules are listed in Figure 3.4.16.

3.4.1.2.6. An Example

Figure 3.4.17 shows a script session of the program in action. Once inside LISP, the first step is to load OPS5, our data object declarations, the system rules, and the initialization rule. We then load in surface data found by our low level processing. The file used in this example contains simulated data for three targets, and is much like that in Figure 3.4.17. The first target is a *BRDM2* apc with a missing surface, the second is a complete *M60A1* tank, and the third is another *M60A1* whose gun barrel has been broken up into two parallel parts by low level processing. The program successfully finds the apc, substituting default surface 0 for the missing surface. The complete tank is easily found with high confidence, and two possibilities are given for the second tank, each using one of the possible gunbarrels (surfaces 23 & 24). All other objects and constructs had confidences that were too low to consider. The command (*wm*) examines the contents of working memory, and shows that only the element defining the number of object classes and the confidence threshold remains.

3.4.1.2.7. Conclusions

We are very pleased with our results using this approach, and it is the one we will build on in the future. The data-driven philosophy has proven itself superior to any kind of top-down approach. Our second expert system is much a cleaner program because the architecture and techniques used arise naturally when one looks at the problem from the proper perspective. Aside from suffering from a few problems with OPS, the system performs very efficiently. It deals with missing surfaces and low level processing errors in a simple, noncontrived fashion, and it easily finds all targets present in the image. The method of target decomposition into constructs is a beautifully simple idea, and it reflects how humans perceive complex objects. The definition of constructs and assignment of weighted *a priori* probabilities can be done quickly through a user interface for an object modeller, and from there rules can be generated automatically.

```
;; Phase 4 Production Rules : Perform Garbage Collection

(p Clean_Up-Adjacencies                                          ; 401
   (Phase ^description clean ^status active)
   { <adj> (Adjacent) }
-->
   (remove <adj>)
                                                                 ) ; 401

(p Clean_Up-Surfaces                                             ; 402
   (Phase ^description clean ^status active)
   { <surf> (Surface) }
-->
   (remove <surf>)
                                                                 ) ; 402

(p Clean_Up-Prior                                                ; 403
   (Phase ^description clean ^status active)
   { <prior> (Construct ^surfaces nil) }
-->
   (remove <prior>)
                                                                 ) ; 403

(p Clean_Up-Unlikely                                             ; 404
   (Phase ^description clean ^status active)
   (World ^threshold <thresh>)
   { <unlikely> (Construct ^confidence {<= <thresh>}) }
-->
   (remove <unlikely>)
                                                                 ) ; 404
```

Figure 3.4.15 Example Phase 4 production rules.

```
;; Output Results

(p Output-Type_and_Confidence                                          ; 501
   { <object> (Construct ^type { <t> <> output } ^confidence <c>) }
 - (Phase)
-->
   (write (crlf) | Object class : | <t> (crlf))
   (write |Confidence : | <c> (crlf))
   (write |Surfaces : |)
   (modify <object> ^type output ^confidence nil)
                                                                       ) ; 501

(p Output-Surfaces                                                     ; 502
   { <object> (Construct ^type output ^surfaces { <s> <> nil }) }
 - (Phase)
-->
   (write (rjust 4) <s>)
   (bind <first-surface> (litval surfaces))
   (bind <second-surface> (compute <first-surface> + 1))
   (modify <object> ^surfaces (substr <object> <second-surface> inf) nil)
                                                                       ) ; 502

(p Output-Complete                                                     ; 503
   { <object> (Construct ^type output ^surfaces nil) }
 - (Phase)
-->
   (write (crlf))
   (remove <object>)
                                                                       ) ; 503
```

Figure 3.4.16 Output Results production rules.

```
% lisp
Franz Lisp, Opus 43.1 [vax dec/bsd.1]
(C) Copyright 1985,1986,1987 Franz Inc., Alameda Ca.
=> (load 'startup)
;; Loading file "startup"
;; Fast loading file "/usr/lib/lisp/ops5.o"
;; Loading file "declarations"
;; Loading file "rules"
**************************************************;; Loading file "make"
*t
=> (load 'test.all)
;; Loading file "test.all"
t
=> (run)

 Object class :  brdm2
 Confidence :  0.93739
 Surfaces :    3   2   1   4   5   0   7   8   9  10  11

 Object class :  m60a1
 Confidence :  0.99992
 Surfaces :   12  13  14  15  16  17

 Object class :  m60a1
 Confidence :  0.99992
 Surfaces :   18  19  20  21  22  23

 Object class :  m60a1
 Confidence :  0.99992
 Surfaces :   18  19  20  21  22  24

end -- no production true
 52 productions (614 // 1137 nodes)
 520 firings (721 rhs actions)
 156 mean working memory size (284 maximum)
 93 mean conflict set size (292 maximum)
 476 mean token memory size (933 maximum)
nil
=> (wm)

57:  (World    ^classes 4    ^threshold 0.5)nil
=> (exit)
%
```

Figure 3.4.17 Example session with OPS5 expert system.

```
;;
;; Low-level output for range image containing BMP apc.
;;

;       Surface Attributes
        (make Surface ^id 1 ^x_loc 2.3  ^y_loc 1.75
              ^height 0.5  ^width 0.4   ^area 0.2
              ^fit_error 250)
        (make Surface ^id 2 ^x_loc 3.0  ^y_loc 1.75
              ^height 0.5  ^width 0.4   ^area 0.2
              ^fit_error 342)
        (make Surface ^id 3 ^x_loc 3.7  ^y_loc 1.75
              ^height 0.5  ^width 0.4   ^area 0.2
              ^fit_error 501)
        (make Surface ^id 4 ^x_loc 4.6  ^y_loc 1.75
              ^height 0.07 ^width 1.224 ^area 0.08568
              ^fit_error 186)
        (make Surface ^id 5 ^x_loc 1.35 ^y_loc 1.35
              ^height 0.4  ^width 2.9   ^area 0.58
              ^fit_error 603)
        (make Surface ^id 6 ^x_loc 4.0  ^y_loc 1.25
              ^height 0.5  ^width 6.5   ^area 2.67
              ^fit_error 23)
        (make Surface ^id 7 ^x_loc 3.0  ^y_loc 0.5
              ^height 1.0  ^width 6.0   ^area 5.5
              ^fit_error 438)

;       Surface Relations
        (make Adjacent ^first 1 ^second 2 ^edge_type snd)
        (make Adjacent ^first 2 ^second 3 ^edge_type snd)
        (make Adjacent ^first 3 ^second 4 ^edge_type jmp)
        (make Adjacent ^first 5 ^second 6 ^edge_type snd)
        (make Adjacent ^first 1 ^second 5 ^edge_type jmp)
        (make Adjacent ^first 3 ^second 6 ^edge_type jmp)
        (make Adjacent ^first 5 ^second 7 ^edge_type snd)
        (make Adjacent ^first 6 ^second 7 ^edge_type snd)
```

Figure 3.4.18 Example low level input to our expert system.

3.4.1.3. Future Work

Encouraged by our results, we have many new ideas to investigate. The first path we wish to explore is the automatic generation of all possible target aspects from model information. Once we can generate aspects automatically, the next task will be generation of rules to recognize each of these aspects. We will also be trying to determine which attributes are important for the recognition of specific object component surfaces. The attributes and relations used may depend on the particular target aspect to be recognized, and once they are known we will be able to extend our low level processing routines and bridge the gap to our expert system. We will also need to examine the unique properties of LADAR data, including a study of its noise characteristics, and keep these in mind as we explore the above ideas. The problem of determining the range at which the geometric approach breaks down, forcing the use of the 2-D silhouette approach, also needs to be addressed.

3.4.2. A Multi-Resolution Data Structure for Model-Based Geometric Reasoning

The availability of LADAR sensor data brings with it the promise of using geometric techniques to improve target recognition. Much of the work that has been preformed in geometric reasoning has been in the area of robot vision [Besl88]. In many ways robot vision is an easier domain to work in because the environment is much more controlled than the domain of LADAR sensing. In a robot cell, the perceived size of an object can be controlled to be almost constant. However, the perceived size of a target in a LADAR image can vary by more than a magnitude of ten. (i.e. the target can be closer than 500 meters, or further than 5000 meters.) Such a wide range of possible distances to the target indicates that there is also a wide range in the on-target resolution that can be used to classify the target.

This section presents a novel multi-resolution data structure (called a *multi-resolution aspect graph*) for model based geometric classification of targets. Given a 3D solid model of a target it can automatically produce a hierarchical representation of the target ranging from very low resolution to high resolution. The low resolution representation excludes all structural information about a target except its silhouette. Such information is useful in classifying targets that are farthest from the sensor. The high resolution information can give details about the relative location, size, orientation, etc, of the all the surfaces and edges on the target. Such information is useful in generating hypotheses to verify the identity of the target.

Section 3.4.2.1 surveys a few geometric methods which have been used on LADAR data and shows how the multi-resolution aspect graph can enhance their performances. Section 3.4.2.2 describes *aspect graphs* which are a data representation that provide a method of selecting viewpoints of a target which see the same features of a given target. Section 3.4.2.3 presents the **multi-resolution aspect graph** (MRAG) which is a new data

structure that considers the resolution of the LADAR sensor when selecting which views see the same features.

3.4.2.1. Geometric Based Target Recognition

There are numerous methods for using the geometric information for object recognition [Besl88]. Some approaches were presented earlier in this section and [VeWi87] presents another approach, all are designed specifically for LADAR data. The following paragraphs give a brief summary of each method.

3.4.2.1.1. Surface Segmentation Approach

The Surface Segmentation method presented in Section 6.1.1 of [KaYo88] classifies a target based on the type, shape, relative location, orientation, area, etc of the surfaces which comprise the target. Surfaces types include *planer, cylindrical, spherical*, and *unknown*. The surface shapes include *irregular, trapezoid, rectangle*, and *square*. A bottom-up rule based system groups surfaces which are nonconcavely adjacent and nearly coplanar into objects, then higher level rules attempt to match these objects to known objects.

The main disadvantage to this system is that *very* detailed surface information is needed. It may not be possible to measure the curvature of a surface that is more than one kilometer away. Many surfaces must be found on a given target before it can be accurately classified. The current LADAR sensors may be able to deliver such information for targets at less than a kilometer away, but for targets at four to five kilometers, two or three planer surfaces, at best, will be able to be identified.

3.4.2.1.2. Goal-Driven Approach

The Goal-Driven Top-Down approach presented in Section 3.4.1.1 assumes the area and location of all the surfaces of a target can be located. With this information it uses a goal driven approach to match the surfaces to a given target using the Prospector Model of Uncertainty Reasoning [Bratko, Duda]. The result of the match is a confidence value showing how well the surfaces matched the model. This process is repeated for each possible target in the image and the target is classified as the model with the highest confidence.

Although this system can handle missing surfaces, it still depends on the bulk of the surfaces being found before it has significant confidence in its match. In addition, there currently is no automatic method for building the probability tree (See Figure 3.4.5). This tree must be rebuilt if the list of candidate targets changes. It also needs a hypothesis generating system to select which target models to attempt to match to the unknown surfaces, otherwise it must compare the unknown surfaces to every known model.

3.4.2.1.3. Data-Driven Approach

The Data-Driven Bottom-Up approach presented in Section 3.4.1.2 also assumes that most of the surfaces of an unknown target can be located and certain features of these targets can be measured. Instead of trying to prove the surfaces are from a certain target as the Goal-Driven method did, the data-driven method has a library of all possible surfaces which can can be seen based on the knowledge of all the possible targets which can be seen. It then attempts to match each unknown surface to the surfaces in the library and build up to the target.

This system like the others relies on being able to identify most of the visible surfaces on the target. Although it can use default reasoning if a couple of the surfaces are missing, if the bulk of the surfaces cannot be found, this method may fail.

3.4.2.1.4. Appearance Models

The Appearance Model approach [VeWi87] is like the Surface Segmentation method from Section 6.1.1 of [KaYo87] in that it must identify various parts (and their relative locations) of an unknown target in order to identify the entire target. Figure 3.4.19 shows the appearance model of a vehicle which can be a tank, a Howitzer, or an APC. Each of the possible targets are then broken down into smaller objects. The tank, for example, is classified by finding the tank gun beside the tank turret which has the tank body below it and the tank antenna above it.

The major drawback of this system is it must be able to identify fine features such as the tank gun and antenna in order to identify the tank. In much of the current LADAR data only the tank body (and maybe the turret) can be found . The inability to find the other features would hinder the Appearance Model approach. The other drawback is that there is no automatic method to create the appearance model.

3.4.2.1.5. Summary of Geometric Approaches

Each of these systems relies heavily on the ability to accurately identify several features (edges, surfaces, or entire parts of the target such as the turret or main gun) of the target. The advantage of such an approach is that if the desired features can be measured, the system can make good use of the information to classify the target. However, current technology sensors can report such information about targets at ranges of a kilometer or less. However, LADAR systems need to be able to classify targets that are at least four to five kilometers away. Although future sensors will most likely be able to deliver detailed data at such distances, such data is not available today. The systems discussed above could fail to classify the target if the bulk of the features cannot be identified. The multi-resolution aspect graph could be used to enhance the above systems by: 1) providing an automatic means of generating the probability trees needed by both the Top-Down and Bottom-Up approaches, and 2) automatically generating hypothesis (goals) for the Top-

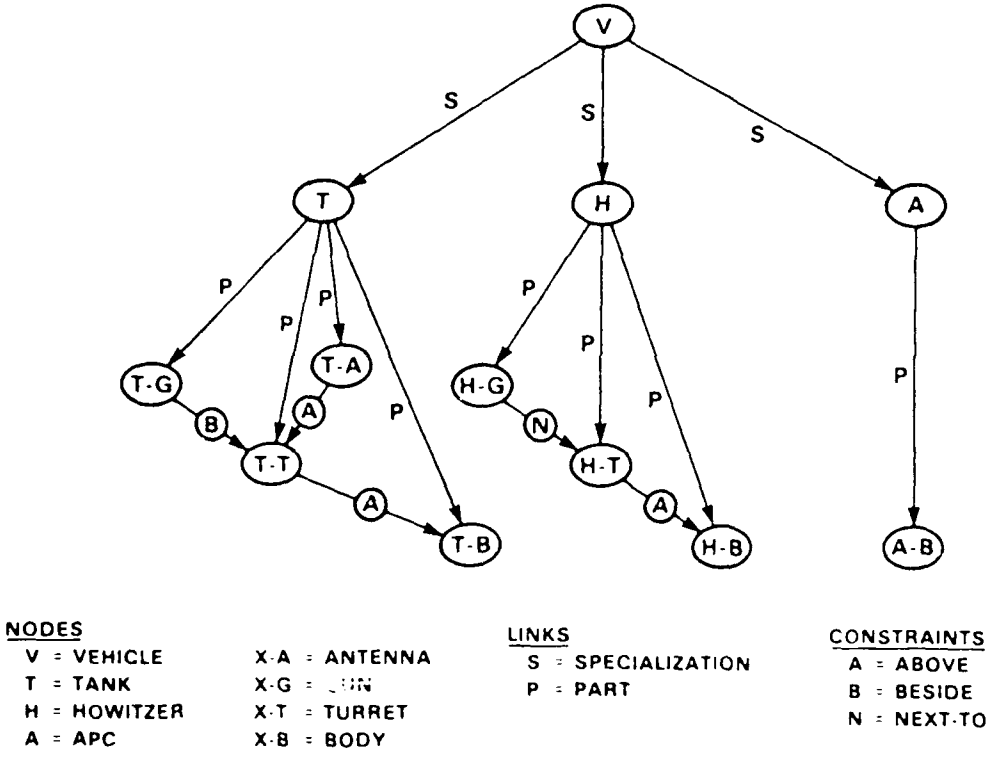


Figure 3.4.19 Region-Based appearance model of a vehicle which can be a Tank, Howitzer, or an APC. (From [VeWi87]).

Down system. Section 3.4.2.2 gives some background information and Section 3.4.2.3 presents the actual structure.

3.4.2.2. Aspect Graphs

The appearance of a target varies greatly with the point from which it is viewed. Although the overall appearance of a target will change as the target is simply rotated about its axis through 90° of rotation, the visibility of geometric features (such as surfaces and edges) does not change greatly from one view to another. (The characteristics of features may change as the target turns, but the features usually will not completely appear or disappear). Such an observation has lead to the creation of aspect graphs.

An aspect graph [KoDo76] characterizes the possible viewpoints from which a target can be viewed by grouping viewpoints that see the same features into equivalence classes. A node in the aspect graph corresponds to all the viewpoints that can observe the same features. Aspect graphs can be generated analytically or by exhaustively examining the object. When generating aspect graphs exhaustively the object is centered within a tessellated *viewing sphere* (with between 60 and 80 tessels [HuKa88, HaHe87]) and the geometric model is viewed from each of the tessels. The visible features in each tessel are recorded and the tessels which view the same features are grouped together. Since the LADAR sensor is currently ground based, our aspect graphs are generated from TWIN models by using a *viewing cylinder*. (i.e. only the tessels corresponding to ground level views are used.) Figure 3.4.20 shows an M113 as viewed from 32 tessels one meter above ground level at a range of one kilometer and a resolution of 0.05 mrads between pixels. Since surfaces are the most visible feature in the LADAR data, they are used as the geometric feature for generating the aspect graph (other features can be used). Figure 3.4.21 is a list of all the surfaces that are visible from each viewpoint. Each of the tessels which see the same surfaces are grouped together in Figure 3.4.22. Each group represents a node in the aspect graph of the M113. Figure 3.4.23 shows the views corresponding to each node in the aspect graph. The M113 is a "boxy" object and the model is not detailed, so the nodes on the aspect graph happen to represent eight equally spaced views.

As the number of visible surfaces in the model increases, the number of nodes in the aspect graph increases. Figures 3.4.24 - 3.4.28, illustrate this for the M60A1 model. Figure 3.4.24 shows the 32 views of an M60A1; Figure 3.4.25 shows each of the surface numbers on the M60A1 model; Figure 3.4.26 lists the surfaces visible from each tessel; Figure 3.4.27 groups the tessels which view the same surfaces; and finally, Figure 3.4.28 shows the corresponding views of the M60A1 target. Note that the M60A1 is a less boxy target and therefore has more surfaces which results in it having twice as many nodes in its aspect graph than the M113 does. The aspect graphs of both targets will have even more nodes once the BRL-CAD models are converted to TWIN models.

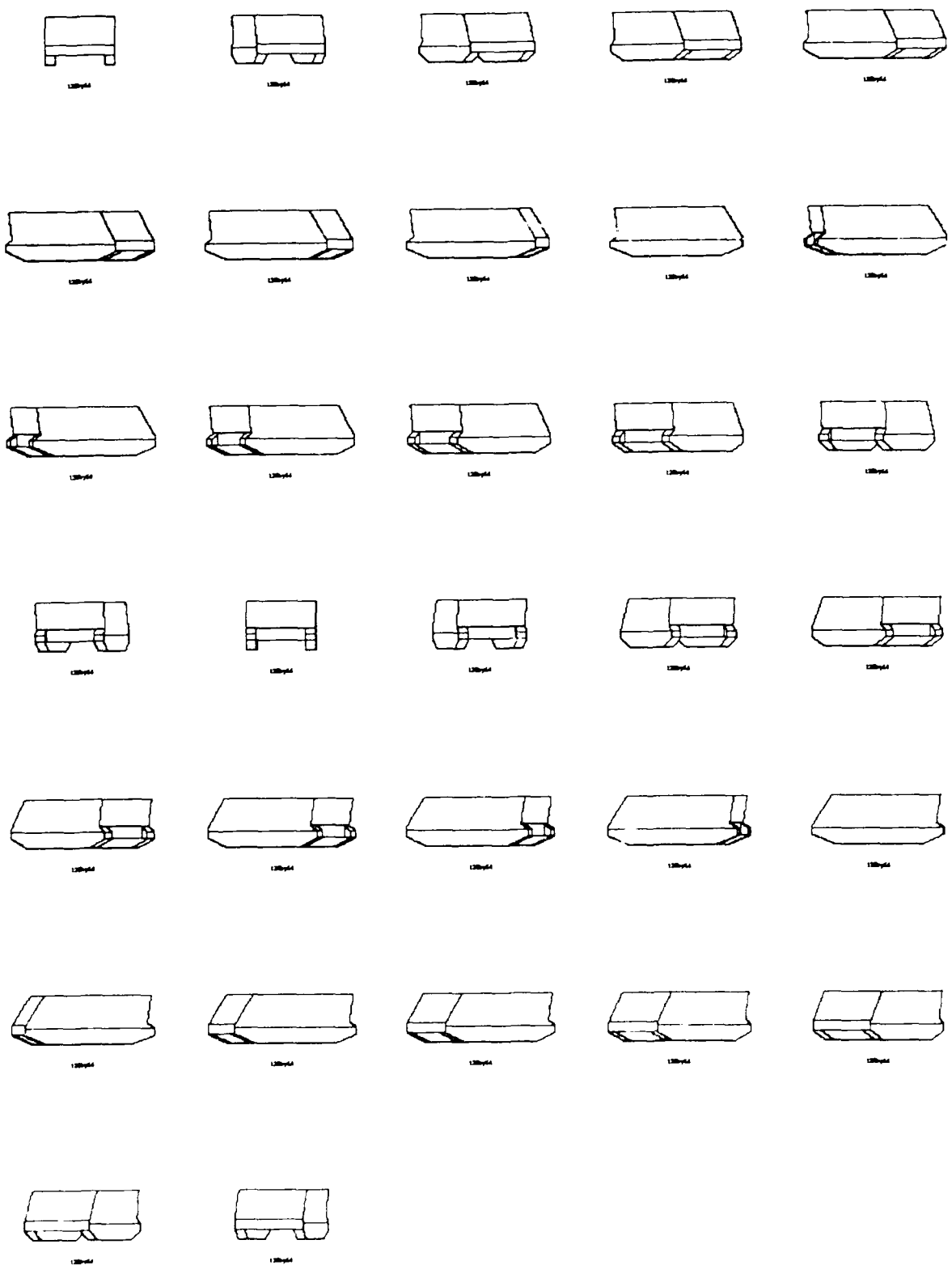


Figure 3.4.20 M113 as viewed from the 32 tessels on the viewing cylinder. Range = 1k.n, resolution = 0.05mrad.

Tessel Number	Visible Surfaces
1	2 3 4 5
2	2 3 4 5 14 21 24
3	2 3 4 5 14 21 24
4	2 3 4 5 14 21 24
5	2 3 4 5 14 21 24
6	2 3 4 5 14 21 24
7	2 3 4 5 14 21 24
8	2 3 4 5 14 21 24
9	2 3 14 21 24
10	6 7 8 9 10 11 12 14 17 19 21 24
11	6 7 8 9 10 11 12 14 17 19 21 24
12	6 7 8 9 10 11 12 14 17 19 21 24
13	6 7 8 9 10 11 12 14 17 19 21 24
14	6 7 8 9 10 11 12 14 17 19 21 24
15	6 7 8 9 10 11 12 14 17 19 21 24
16	6 7 8 9 10 11 12 14 17 19 21 24
17	6 7 8 9 10 11 12 14 17 21 24
18	6 7 8 9 10 11 12 13 17 18 20 22
19	6 7 8 9 10 11 12 13 17 18 20 22
20	6 7 8 9 10 11 12 13 17 18 20 22
21	6 7 8 9 10 11 12 13 17 18 20 22
22	6 7 8 9 10 11 12 13 17 18 20 22
23	6 7 8 9 10 11 12 13 17 18 20 22
24	6 7 8 9 10 11 12 13 17 18 20 22
25	6 13 17 18 20 22
26	2 3 4 5 13 20 22
27	2 3 4 5 13 20 22
28	2 3 4 5 13 20 22
29	2 3 4 5 13 20 22
30	2 3 4 5 13 20 22
31	2 3 4 5 13 20 22
32	2 3 4 5 13 20 22

Figure 3.4.21 Surfaces of M113 which are visible from each viewpoint.

Tessel Numbers	Visible Surfaces
1	2 3 4 5
2 3 4 5 6 7 8	2 3 4 5 14 21 24
9	2 3 14 21 24
10 11 12 13 14 15 16	6 7 8 9 10 11 12 14 17 19 21 24
17	6 7 8 9 10 11 12 14 17 21 24
18 19 20 21 22 23 24	6 7 8 9 10 11 12 13 17 18 20 22
25	6 13 17 18 20 22
26 27 28 29 30 31 32	2 3 4 5 13 20 22

Figure 3.4.22 Figure 3.4.21 with viewpoints grouped together which view the same surfaces.

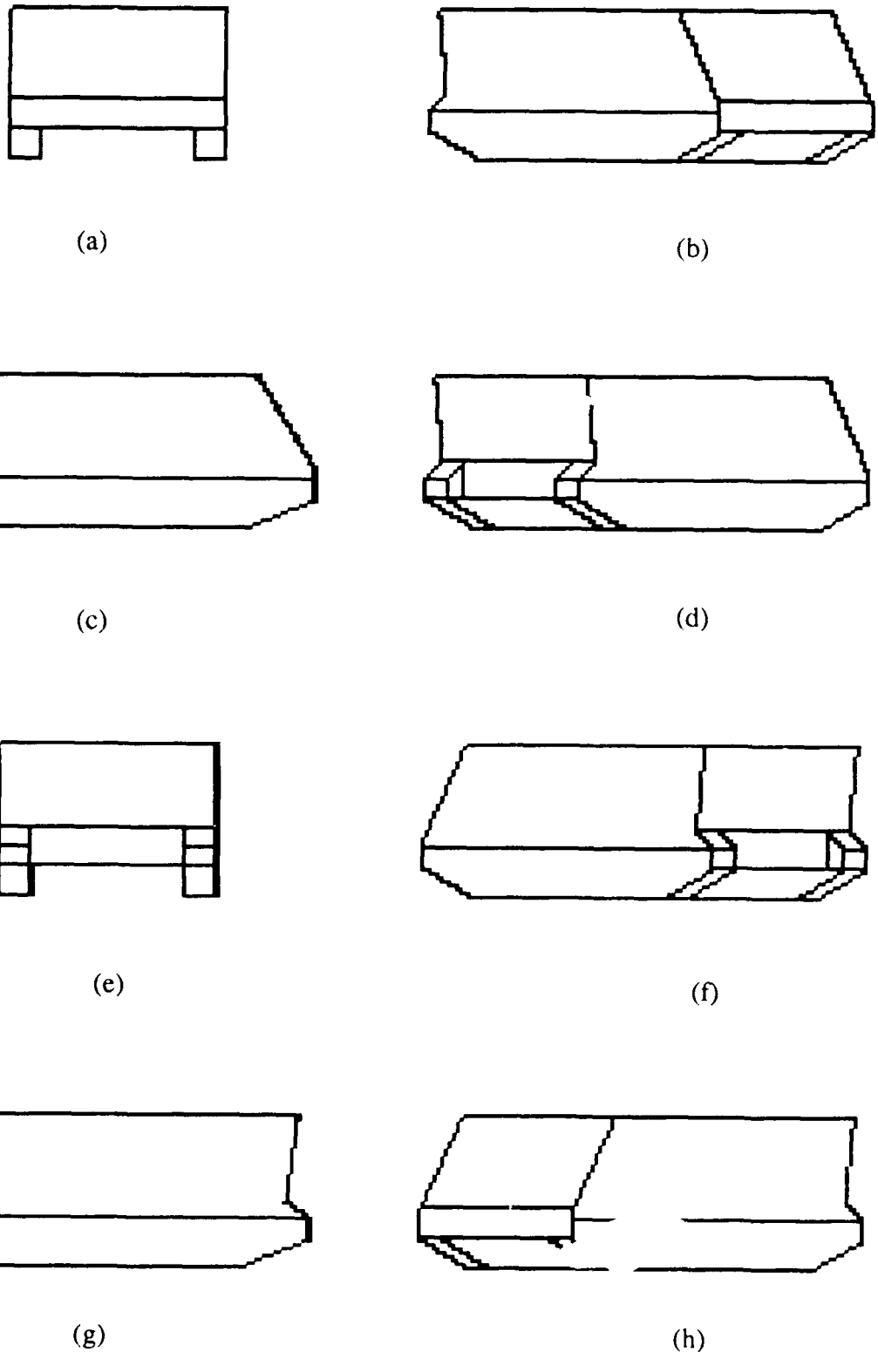


Figure 3.4.23 Views of M113 that correspond to each node in the aspect graph.

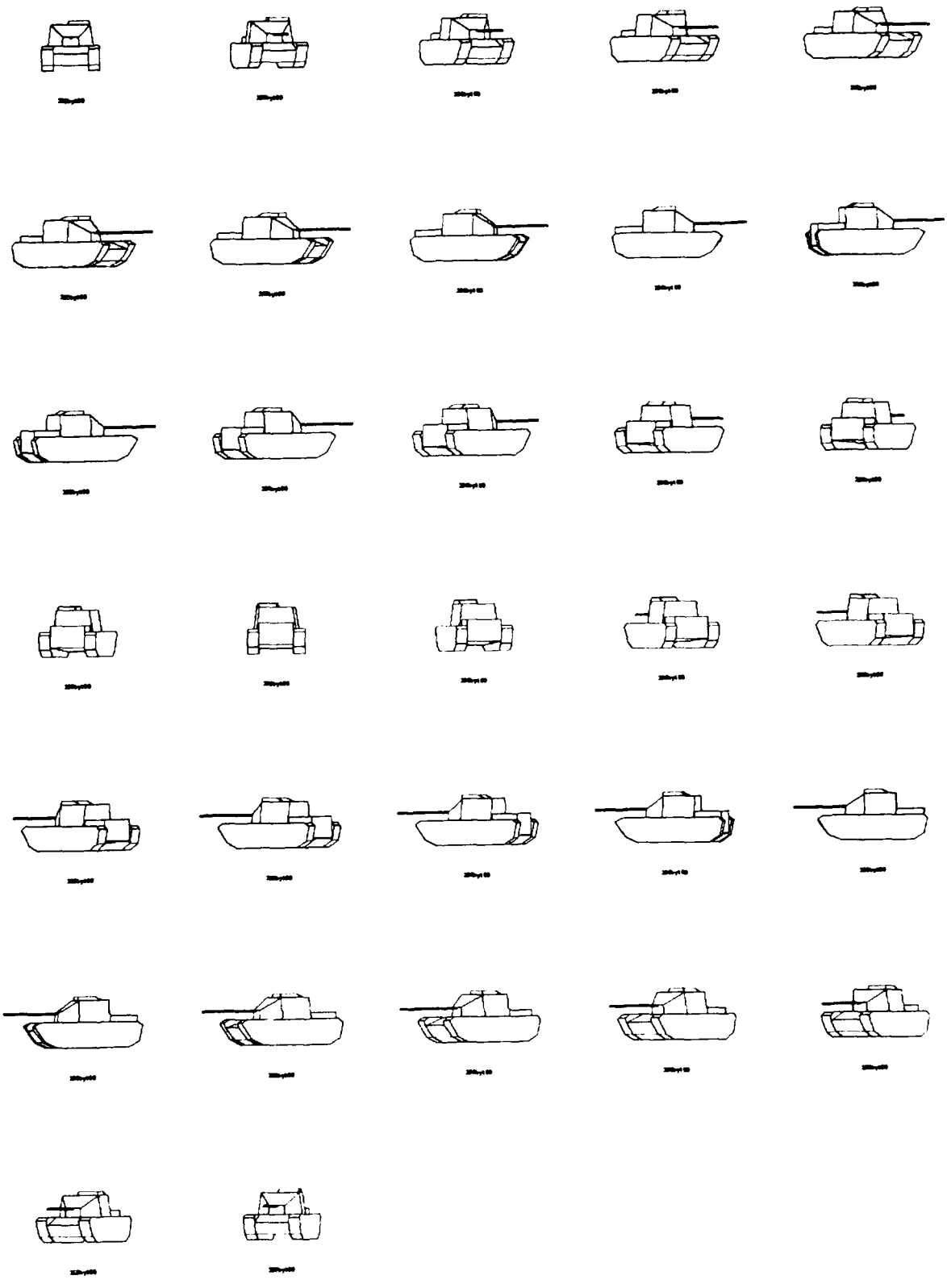


Figure 3.4.24 M06A1 as viewed from the 32 tessels on the viewing cylinder. Range = 1km, resolution = 0.05mrad.

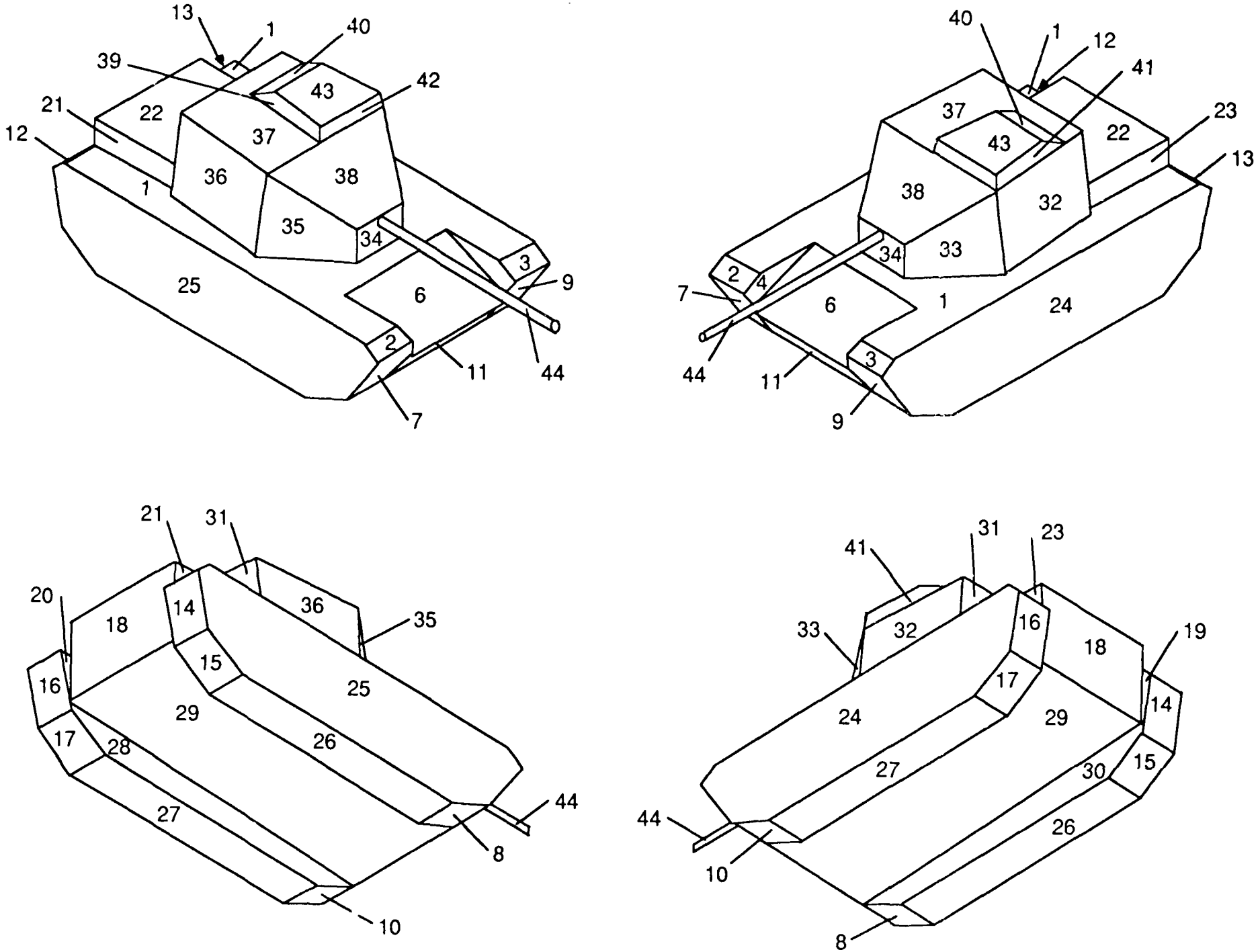


Figure 3.4.25 The surface numbers of the M06A1.

Tessel Number	Visible Surfaces
1	2 3 6 7 8 9 10 11 33 34 35 38 42 45
2	2 3 5 6 7 8 9 10 11 21 25 28 33 34 35 36 38 39 42 44 45
3	2 3 5 6 7 8 9 10 11 21 25 28 33 34 35 36 38 39 42 44 45
4	2 3 5 6 7 8 9 10 11 21 25 28 33 34 35 36 38 39 42 44 45
5	2 3 5 6 7 8 9 10 11 21 25 28 34 35 36 38 39 42 44 45
6	2 3 5 6 7 8 9 10 11 21 25 28 34 35 36 38 39 42 44 45
7	2 3 5 6 7 8 9 10 11 21 25 28 34 35 36 38 39 42 44
8	2 3 5 6 7 8 9 10 11 21 25 28 34 35 36 38 39 42 44
9	7 21 25 35 36 39 44
10	12 13 14 15 16 17 18 20 21 25 28 29 31 35 36 39 40 44
11	12 13 14 15 16 17 18 20 21 25 28 29 31 35 36 39 40 44
12	12 13 14 15 16 17 18 20 21 25 28 29 31 35 36 39 40 44
13	12 13 14 15 16 17 18 20 21 25 28 29 31 35 36 39 40 44
14	12 13 14 15 16 17 18 20 21 25 28 29 31 36 39 40 44
15	12 13 14 15 16 17 18 20 21 25 28 29 31 36 39 40 44
16	12 13 14 15 16 17 18 20 21 25 28 29 31 36 39 40
17	12 13 14 15 16 17 18 29 31 32 36 39 40 41
18	12 13 14 15 16 17 18 19 23 24 29 30 31 32 40 41
19	12 13 14 15 16 17 18 19 23 24 29 30 31 32 40 41 44
20	12 13 14 15 16 17 18 19 23 24 29 30 31 32 40 41 44
21	12 13 14 15 16 17 18 19 23 24 29 30 31 32 33 40 41 44
22	12 13 14 15 16 17 18 19 23 24 29 30 31 32 33 40 41 44
23	12 13 14 15 16 17 18 19 23 24 29 30 31 32 33 40 41 44
24	12 13 14 15 16 17 18 19 23 24 29 30 31 32 33 40 41 44
25	9 23 24 32 33 41 44
26	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 38 41 42 44
27	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 38 41 42 44
28	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 38 41 42 44 45
29	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 38 41 42 44 45
30	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 35 38 41 42 44 45
31	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 35 38 41 42 44 45
32	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 35 38 41 42 44 45

Figure 3.4.26 Surfaces of M06A1 which are visible from each viewpoint.

Tessel Numbers	Visible Surfaces
1	2 3 6 7 8 9 10 11 33 34 35 38 42 45
2 3 4	2 3 5 6 7 8 9 10 11 21 25 28 33 34 35 36 38 39 42 44 45
5 6	2 3 5 6 7 8 9 10 11 21 25 28 34 35 36 38 39 42 44 45
7 8	2 3 5 6 7 8 9 10 11 21 25 28 34 35 36 38 39 42 44
9	7 21 25 35 36 39 44
10 11 12 13	12 13 14 15 16 17 18 20 21 25 28 29 31 35 36 39 40 44
14 15	12 13 14 15 16 17 18 20 21 25 28 29 31 36 39 40 44
16	12 13 14 15 16 17 18 20 21 25 28 29 31 36 39 40
17	12 13 14 15 16 17 18 29 31 32 36 39 40 41
18	12 13 14 15 16 17 18 19 23 24 29 30 31 32 40 41
19 20	12 13 14 15 16 17 18 19 23 24 29 30 31 32 40 41 44
21 22 23 24	12 13 14 15 16 17 18 19 23 24 29 30 31 32 33 40 41 44
25	9 23 24 32 33 41 44
26 27	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 38 41 42 44
28 29	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 38 41 42 44 45
30 31 32	2 3 4 6 7 8 9 10 11 23 24 30 32 33 34 35 38 41 42 44 45

Figure 3.4.27 Figure 3.4.26 with viewpoints grouped together which view the same surfaces.

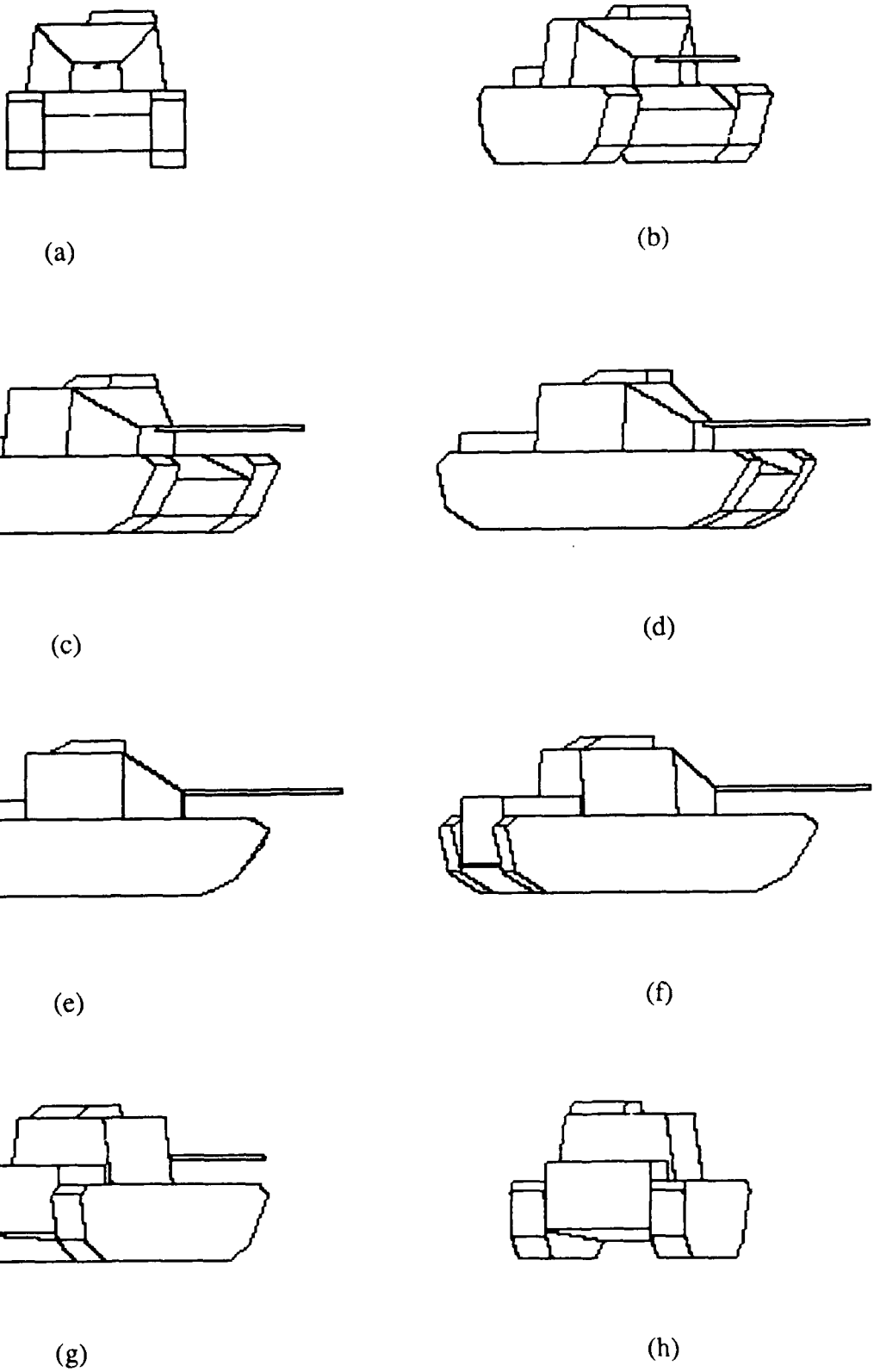
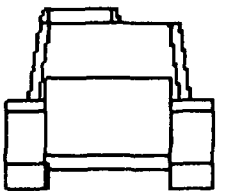
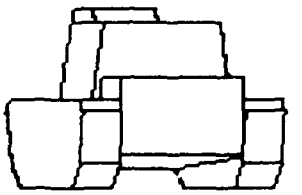


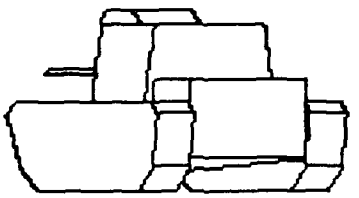
Figure 3.4.28 Views of M60A1 that correspond to each node in the aspect graph.



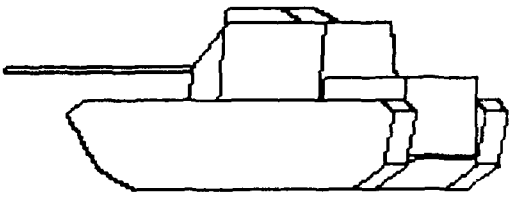
(i)



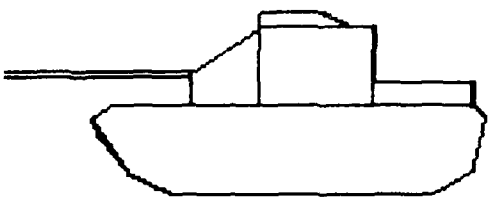
(j)



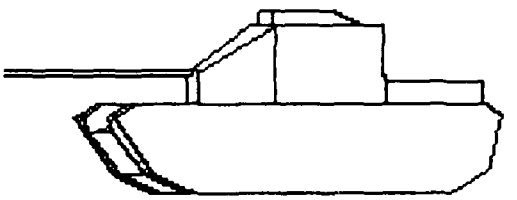
(k)



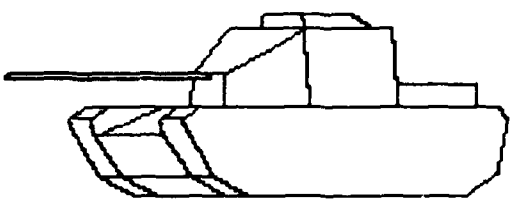
(l)



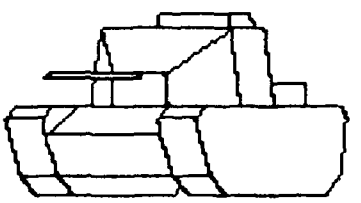
(m)



(n)



(o)



(p)

Figure 3.4.28 (continued)

Aspect graphs themselves can provide a suitable data structure for a model based geometric reasoning system. For example, a classifier could be trained on the silhouettes of the images corresponding to the nodes in the aspect graph. (See Figures 3.4.23 and 3.4.28.) The classifier would match an unknown target to one or more of the silhouettes. Each silhouette corresponds to a given viewpoint of the 3D model and therefore has geometric information about that view of the target. This information can be used to generate hypotheses about the geometric features present in the target. For example, if the unknown target matched Figure 3.4.28 (c), the modeler could hypothesize that if the target is an M60A1, then there must be a 90° edge in the lower half of the target and a jump edge between the bottom half of the target and the top half.

Such an approach is a workable solution if the target models are very simple like the ones from ERIM. If more detailed targets are used, like those from BRL, the number of nodes in the aspect graph would increase greatly. Most of the added details would not be visible in distant LADAR images. The following section presents a method which will systematically reduce the number of nodes in the aspect graph (if needed) to match the detectable features of the target.

3.4.2.3. Multi-Resolution Aspect Graphs

The previous section has shown how aspect graphs can take a 3D model of a target and break it down into a number of candidate views, each seeing a different set of features. If a detailed model is used, the aspect graph will have many nodes (viewpoints). This section presents a new method for reducing the number of nodes in an aspect graph so that detailed models can still be used to classify distant targets that have few pixels on target.

It is possible for a small feature in an object (a feature so small the LADAR sensor can not see it), to cause additional (possibly unneeded) nodes to appear in the aspect graph. The *multi-resolution aspect graph* is an extension of the aspect graph which systematically removes features from consideration when building the aspect graph so that the graph is not influenced by features the sensor will never be able to see. The following two sections discuss how to build a multi-resolution aspect graph based on visible surface area and the angles between two surfaces.

3.4.2.3.1. Surface Area Based Multi-Resolution Aspect Graph

Figure 3.4.29 is a list of all the surfaces in the M60A1 model. The numbers in the *Real Area* column represent the actual area of the surface as measured on the 3D model. The number in the *Viewed Area* column are the number of pixels covered by the surface when it is projected onto a 2D plane assuming a 0.05 mrad resolution and a distance of one kilometer. Since this area changes with viewpoint, the maximum area seen from all of the tessels is the one recorded. Notice that surface 29 has the largest *real area*, but it has one of the smaller *viewed area*s. This is because it is the surface on the bottom of the tank.

Surface Number	Real Area ($meters^2$)	Viewed Area (pixels)
2	0.26793	60.00
3	0.26793	56.00
4	0.34750	129.00
5	0.34750	129.00
6	3.16716	440.00
7	0.88122	315.00
8	0.62581	105.00
9	0.88122	295.00
10	0.62581	98.00
11	2.03758	660.00
12	0.21110	56.00
13	0.21110	60.00
14	0.69235	266.00
15	0.61492	126.00
16	0.69235	285.00
17	0.61492	135.00
18	3.08000	1232.00
19	0.12650	46.00
20	0.12650	46.00
21	0.70000	280.00
23	0.70000	280.00
24	9.92135	3992.00
25	9.92135	3991.00
28	2.61005	373.00
29	13.27565	220.00
30	2.61005	373.00
31	2.34000	935.00
32	2.83907	1120.00
33	1.48808	576.00
34	0.61393	244.00
35	1.48808	576.00
36	2.8396[illegible]	1120.00
38	2.4[illegible]	564.00
39	0.37112	135.00
40	0.63643	110.00
41	0.37112	132.00
42	0.36450	135.00
44	0.16302	142.00
45	0.00707	2.00

Figure 3.4.29 Actual and maximum viewed sizes of the surfaces in the M60A1 model.

Although it is the largest surface, only a small part of it is ever visible at a given time. Surface number 45 is the end of the main gun of the M60A1. Although its *viewed area* is only two pixels, its presence in tessels 5 and 6 (Figure 3.4.27) and absence in tessels 7 and 8 is enough to cause those tessels to be in different nodes on the aspect graph. This surface would most likely never be seen by a LADAR sensor, and should therefore not cause the aspect graph to have two nodes differing by only it.

In the example above, the features are the surfaces of the target. The surfaces can be ordered by their largest viewed area and those surfaces whose area is too small to be detected at the given resolution will not be considered when building the aspect graph. Suppose at the given resolution the smallest reliably detectable surface is 575 pixels (i.e. the number of pixels that can be viewed) , from Figure 3.4.29 it is seen that only surfaces 11, 18, 24, 25, 31, 32, 33, 35, and 36 would be considered in building the aspect graph. Figure 3.4.30 is a list of the nodes in the reduced resolution aspect graph, and Figure 3.4.31 shows the corresponding views. The aspect graph has been simplified to match the resolution of the sensor.

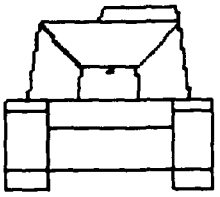
3.4.2.3.2. Edge Angle Based Multi-Resolution Aspect Graph

Edges are another distinctive feature of LADAR imagery that can be used for target classification. There are two types of edges which can appear in a range image, viewpoint *dependent* edges and viewpoint *independent* edges. Viewpoint dependent edges are generally those edges which appear between the target and the background. For example the edges between the turret and the terrain in the background. The model itself gives all the viewpoint independent edges, and these are the edges which will be used here. Figure 3.4.32 lists all the viewpoint independent edges along with their lengths and the angle between the normals of the two surfaces which meet at the edges in the M60A1 model. Figure 3.4.33 (a) plots the count of the number of edges with the same angle and Figure 3.4.33 (b) plots the count of the number of edges with the same length. It is easy to see that most of the edges are 90°. This is expected to change when more detailed models are available since most targets are not so "boxy". Figure 3.4.34 shows a plot of length vs angle for each of the models used. These again show that in these models, most of the edges are 90°.

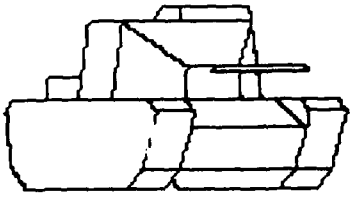
The edge information can be used to adjust the model to the sensor resolution by observing that edges with small angles between surfaces are more difficult to detect than edges with large angles. Also short edges are more difficult to detect than long edges. Our approach is to remove all edges whose length is less than a given threshold and whose angle is less than a given threshold. The thresholds used must be determined by the resolution of the sensor. Since such information is not currently available, Figure 3.4.35 - 3.4.38 were generated to show how the M60A1 model changes as edges are removed.

Tessel Numbers	Visible Surfaces
1	11 33 35
2 3 4	11 25 33 35 36
5 6 7 8	11 25 35 36
9	25 35 36
10 11 12 13	18 25 31 35 36
14 15 16	18 25 31 36
17	18 31 32 36
18 19 20	18 24 31 32
21 22 23 24	18 24 31 32 33
25	24 32 33
26 27 28 29	11 24 32 33
30 31 32	11 24 32 33 35

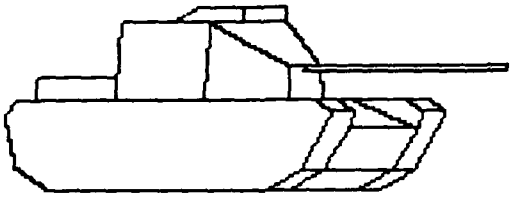
Figure 3.4.30 Nodes of the aspect graph for the M60a1 using only those surfaces larger than 575 $meters^2$.



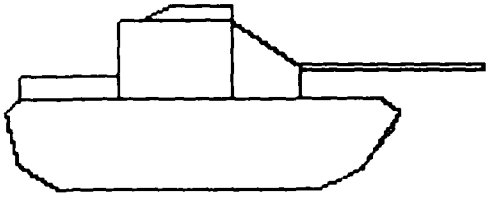
(a)



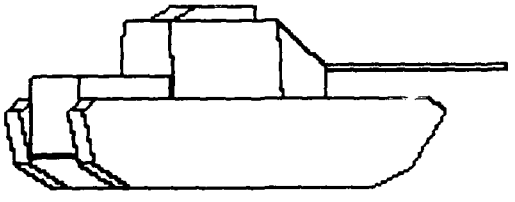
(b)



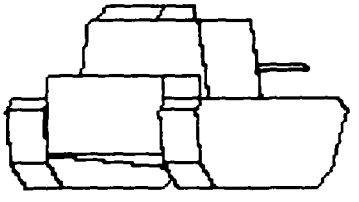
(c)



(d)

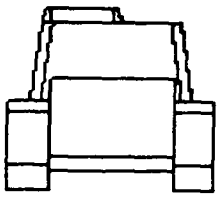


(e)

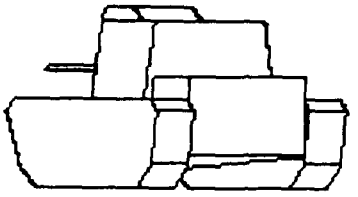


(f)

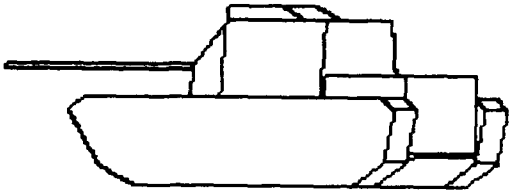
Figure 3.4.31 Views of M60A1 corresponding to the tessels in Figure 4.12.



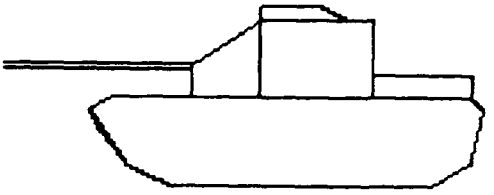
(g)



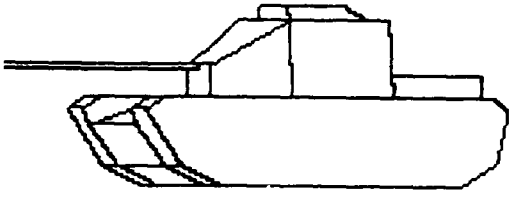
(h)



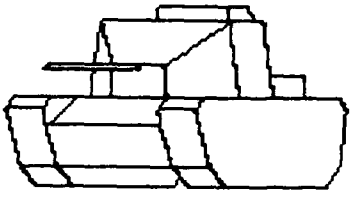
(i)



(j)



(k)



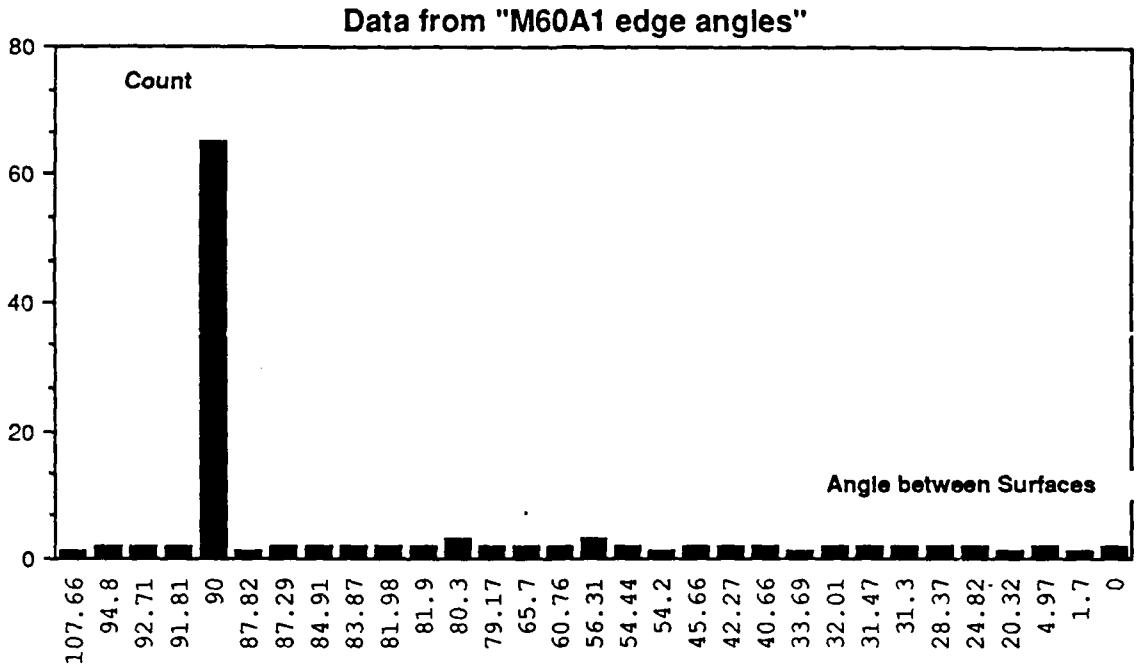
(l)

Figure 3.4.31 (continued)

Face 1	Face 2	Angle (degrees)	length (meters)
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
28	20	0	0.15
30	19	0	0.15
31	1	90	0.15
31	1	90	0.15
42	39	94.8	0.27
42	41	94.8	0.27
19	12	90	0.3
20	13	90	0.3
24	13	90	0.3
25	12	90	0.3
7	4	92.71	0.32
9	5	92.71	0.32
24	3	90	0.38
25	2	90	0.38
4	2	90	0.38
5	3	90	0.38
21	18	90	0.4
23	18	90	0.4
31	21	90	0.4
31	23	90	0.4
37	32	81.98	0.4
34	38	56.31	0.49
38	34	56.31	0.49
40	39	79.17	0.57
41	40	79.17	0.57
34	33	54.44	0.6
35	34	54.44	0.6
10	9	31.47	0.71
12	1	42.27	0.71
13	1	42.27	0.71
14	12	60.76	0.71
15	14	45.66	0.71
16	13	60.76	0.71
17	16	45.66	0.71
2	1	32.01	0.71
26	15	31.3	0.71
26	8	24.82	0.71
27	10	24.82	0.71
27	17	31.3	0.71
3	1	32.01	0.71
7	2	91.81	0.71
8	7	31.47	0.71
9	3	91.81	0.71
24	17	90	0.87

Face 1	Face 2	Angle (degrees)	length (meters)
25	15	90	0.87
28	17	90	0.87
30	15	90	0.87
24	10	90	0.88
25	8	90	0.88
28	10	90	0.88
30	8	90	0.88
11	7	4.97	0.93
11	9	4.97	0.93
19	14	90	0.98
20	16	90	0.98
24	16	90	0.98
25	14	90	0.98
19	18	90	1
20	18	90	1
38	37	33.69	1.06
34	1	90	1.1
43	39	80.3	1.1
43	41	80.3	1.1
40	37	28.37	1.12
43	40	28.37	1.12
4	1	90	1.15
5	1	90	1.15
24	9	87.29	1.24
25	7	87.29	1.24
43	42	90	1.3
42	38	56.31	1.4
32	31	84.91	1.41
33	32	40.66	1.41
36	31	84.91	1.41
36	35	40.66	1.41
6	4	90	1.44
6	5	90	1.44
33	1	83.87	1.49
35	1	83.87	1.49
39	37	80.3	1.61
41	32	1.7	1.61
38	33	65.7	1.62
38	35	65.7	1.62
21	1	90	1.75
22	21	90	1.75
23	1	90	1.75
23	22	90	1.75
32	1	81.98	2.01
36	1	81.9	2.01
37	36	81.9	2.01
37	31	90	2.1
11	6	107.66	2.2
22	18	90	2.2
29	11	54.2	2.2
29	18	87.82	2.2
31	22	90	2.2
6	1	20.32	2.2
26	25	90	4.49
27	24	90	4.49
28	27	90	4.49
30	26	90	4.49
29	28	90	6.03
30	29	90	6.03
24	1	90	6.4
25	1	90	6.4

Figure 3.4.32 Angles and lengths of all the edges in the M60A1 model. Ordered by length.



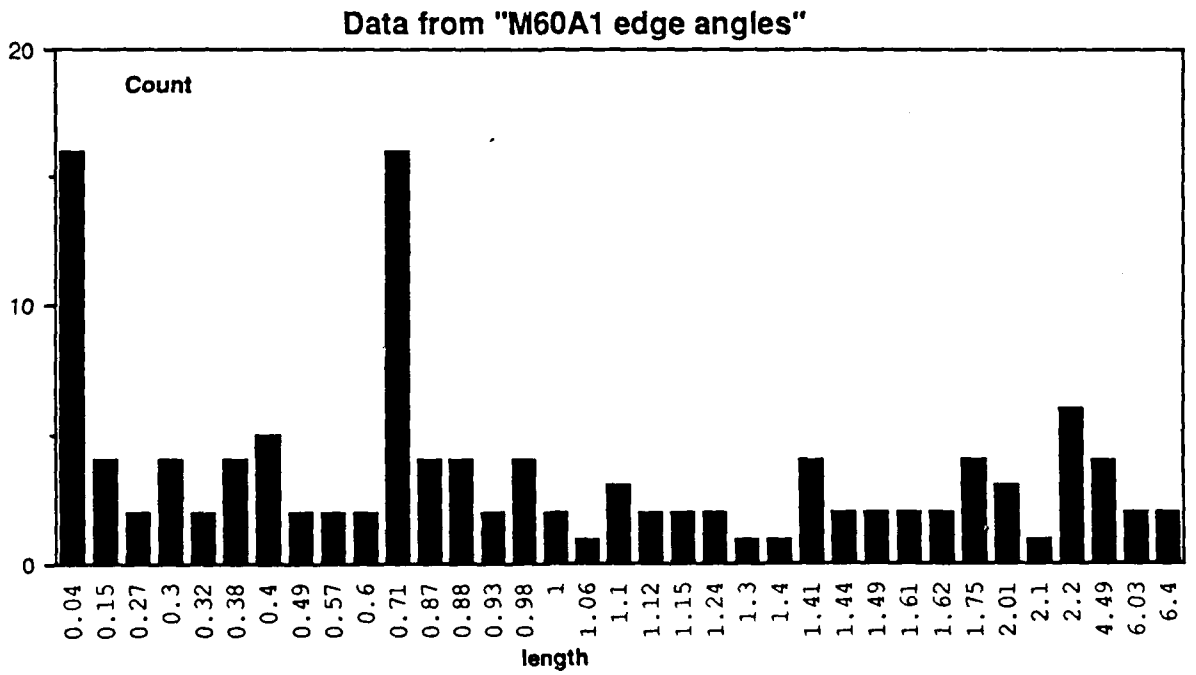


Figure 3.4.33 Count of edges with given angle and given length in M60A1 model.

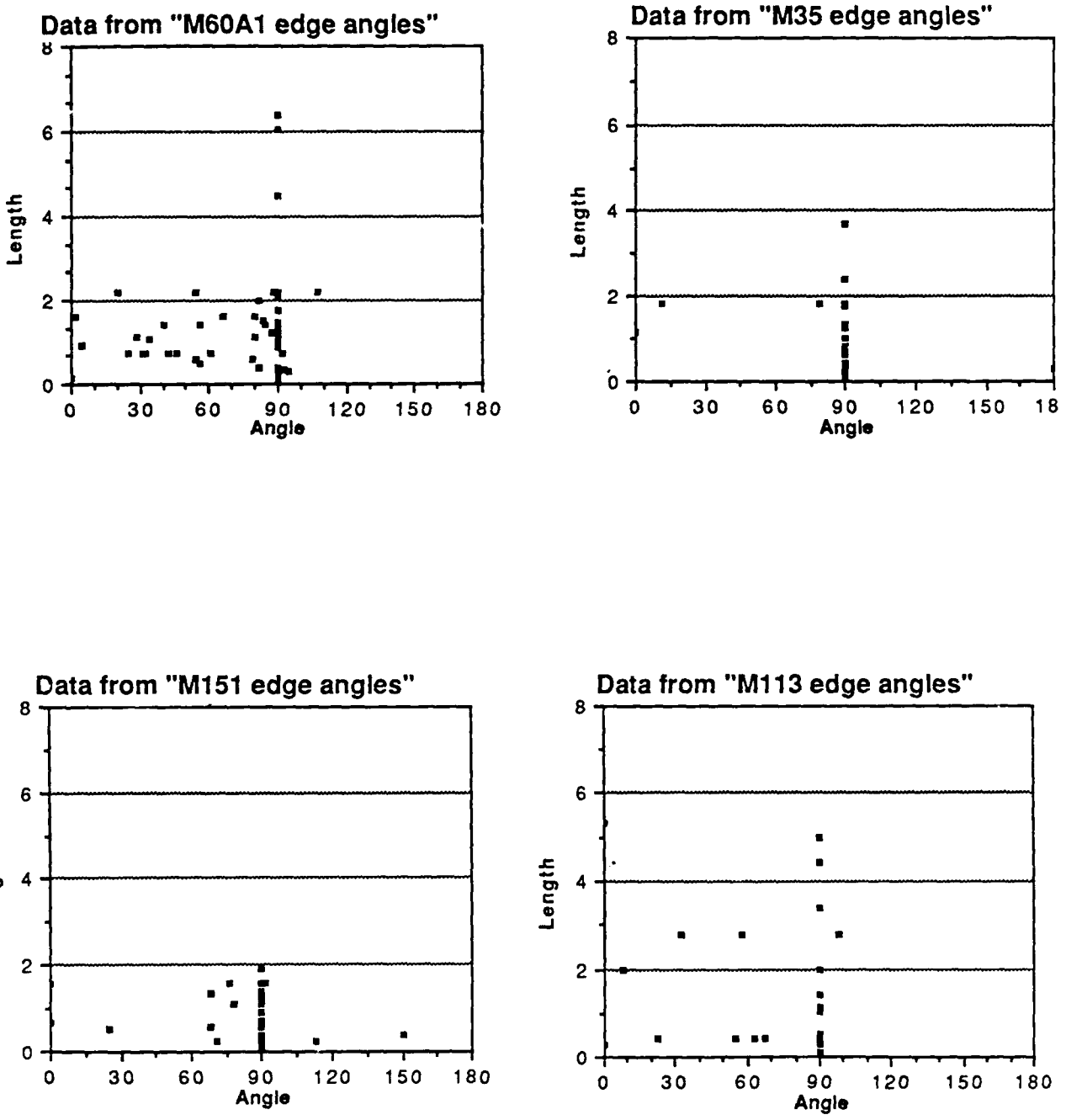


Figure 3.4.34 Plots of edges length vs. edge angle for each of the models used.

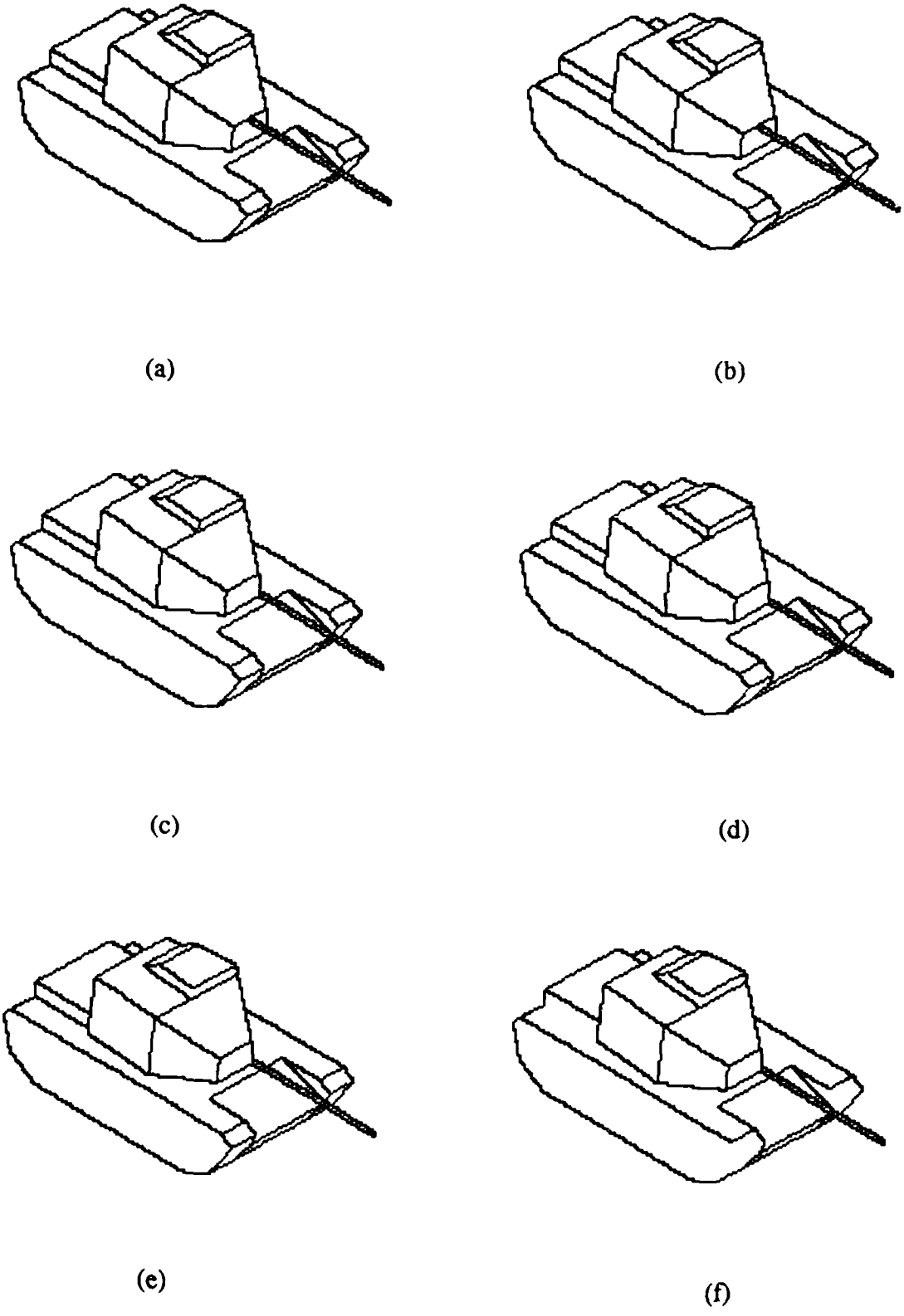
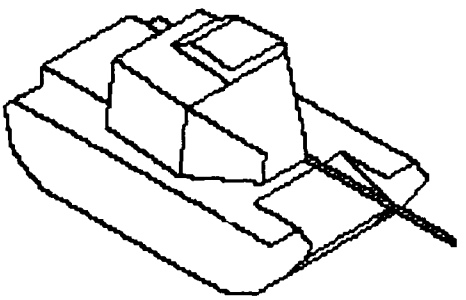
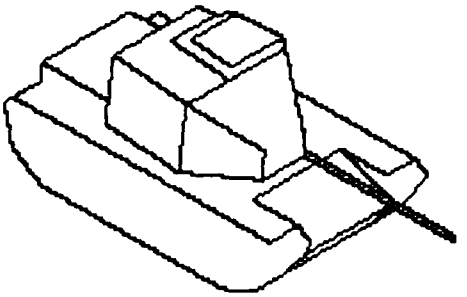


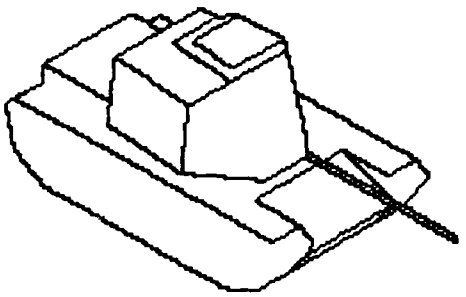
Figure 3.4.35 M60A1 with edges less than a given length removed. a) original M60A1 b) less than 0.0 meters c) less than 0.1 meters d) less than 0.2 meters e) less than 0.3 meters f) less than 0.4 meters g) less than 0.5 meters h) less than 0.6 meters i) less than 0.7 meters j) less than 0.8 meters k) less than 1.0 meters l) less than 1.1 meters m) less than 1.2 meters n) less than 1.4 meters o) less than 1.5 meters p) less than 1.8 meters q) less than 2.2 meters.



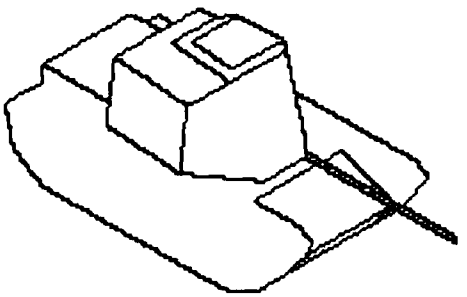
(g)



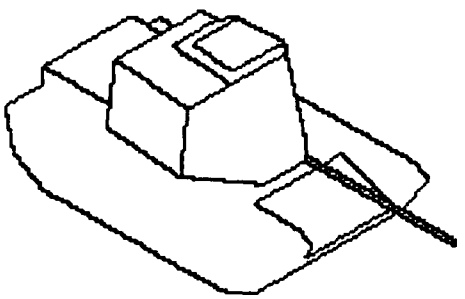
(h)



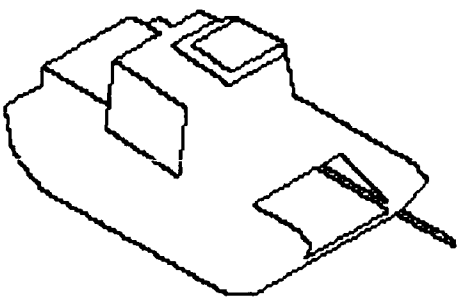
(i)



(j)

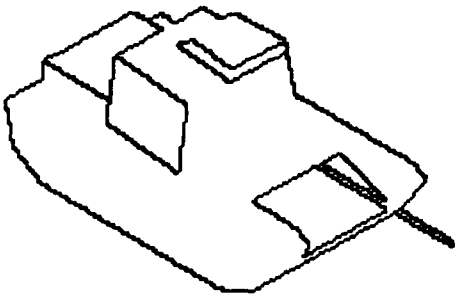


(k)

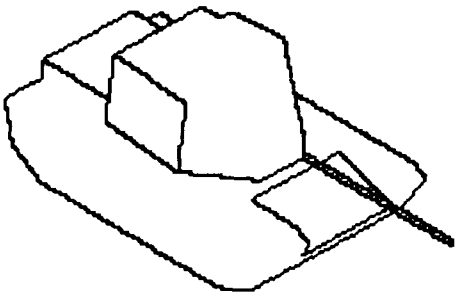


(l)

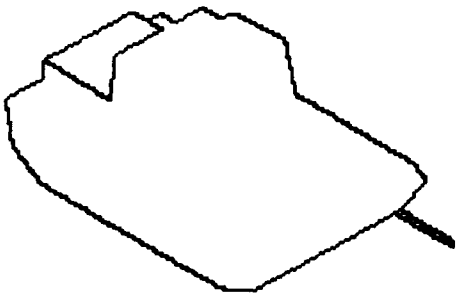
Figure 3.4.35 Continued.



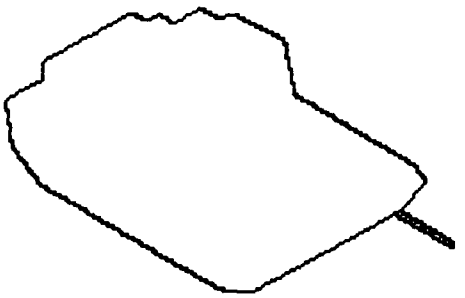
(m)



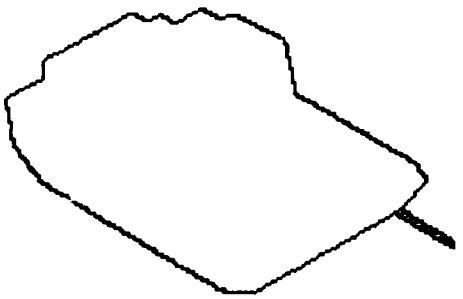
(n)



(o)

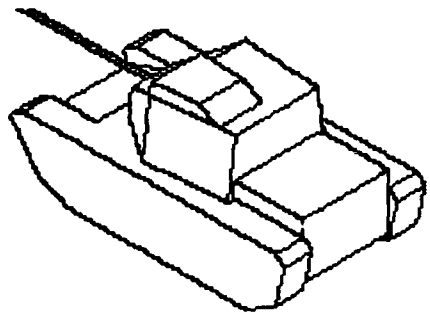


(p)

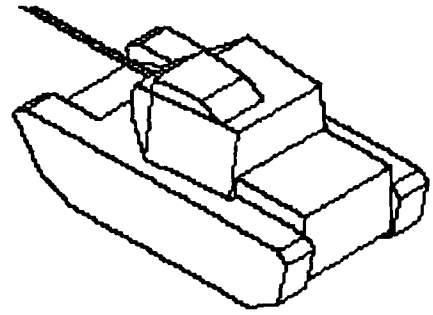


(q)

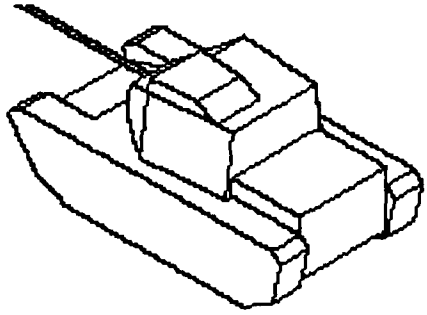
Figure 3.4.35 Continued.



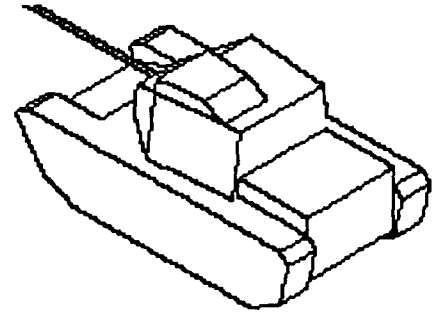
(a)



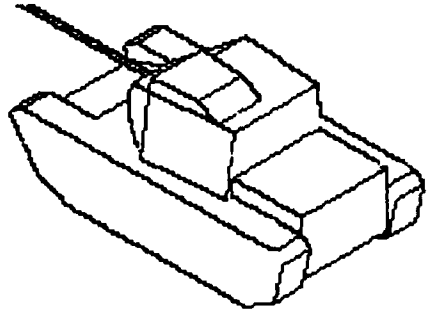
(b)



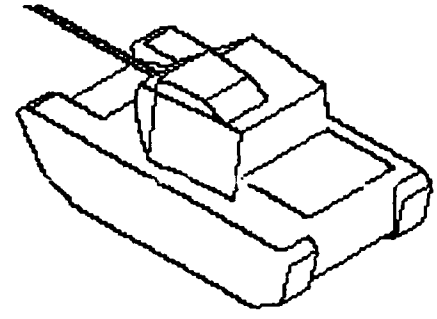
(c)



(d)

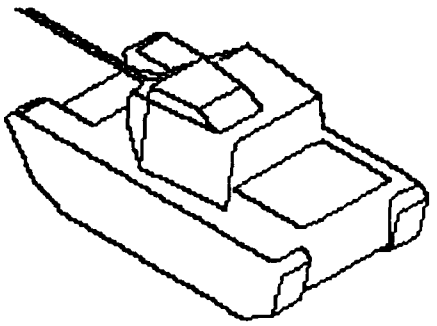


(e)

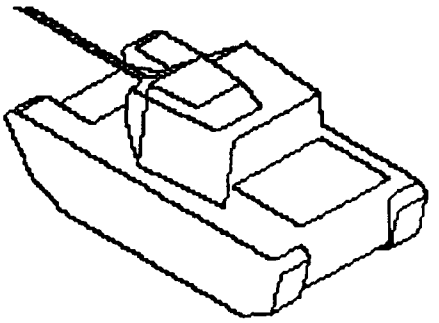


(f)

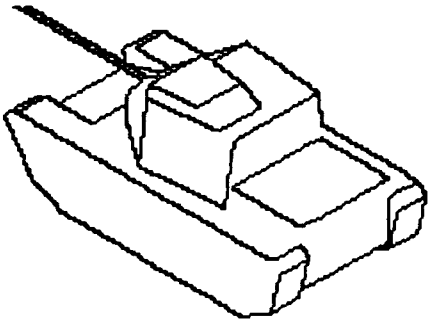
Figure 3.4.36 Back side of Figure 3.4.35.



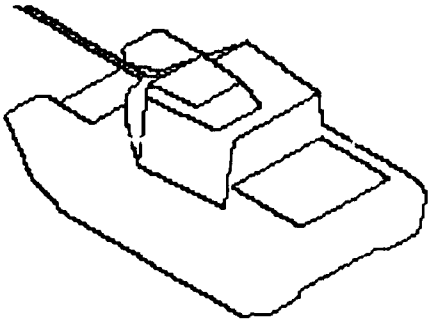
(g)



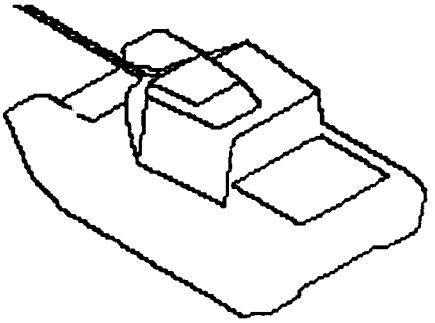
(h)



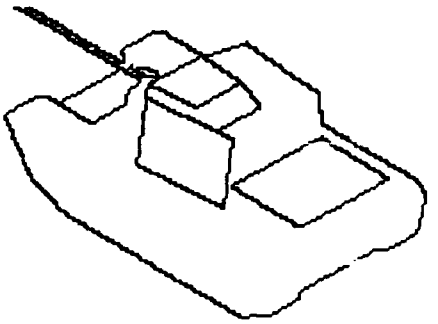
(i)



(j)

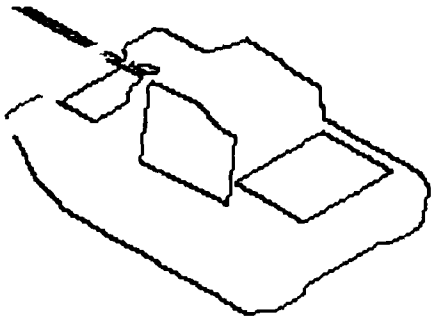


(k)

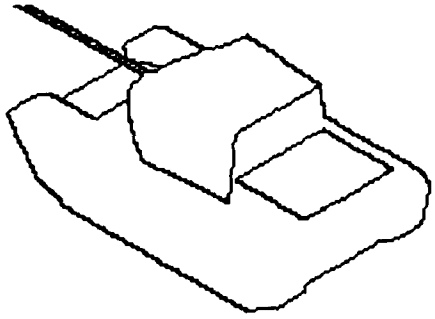


(l)

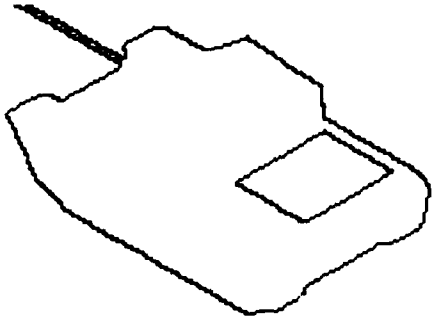
Figure 3.4.36 Continued.



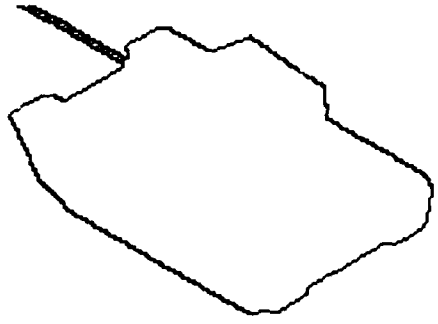
200by150



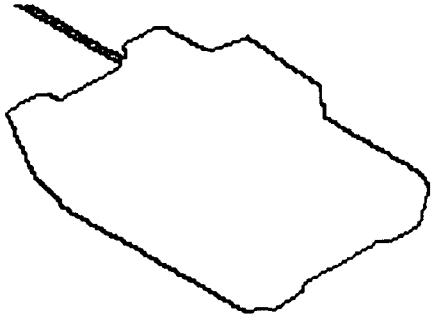
200by150



(o)



(p)



(q)

Figure 3.4.36 Continued.

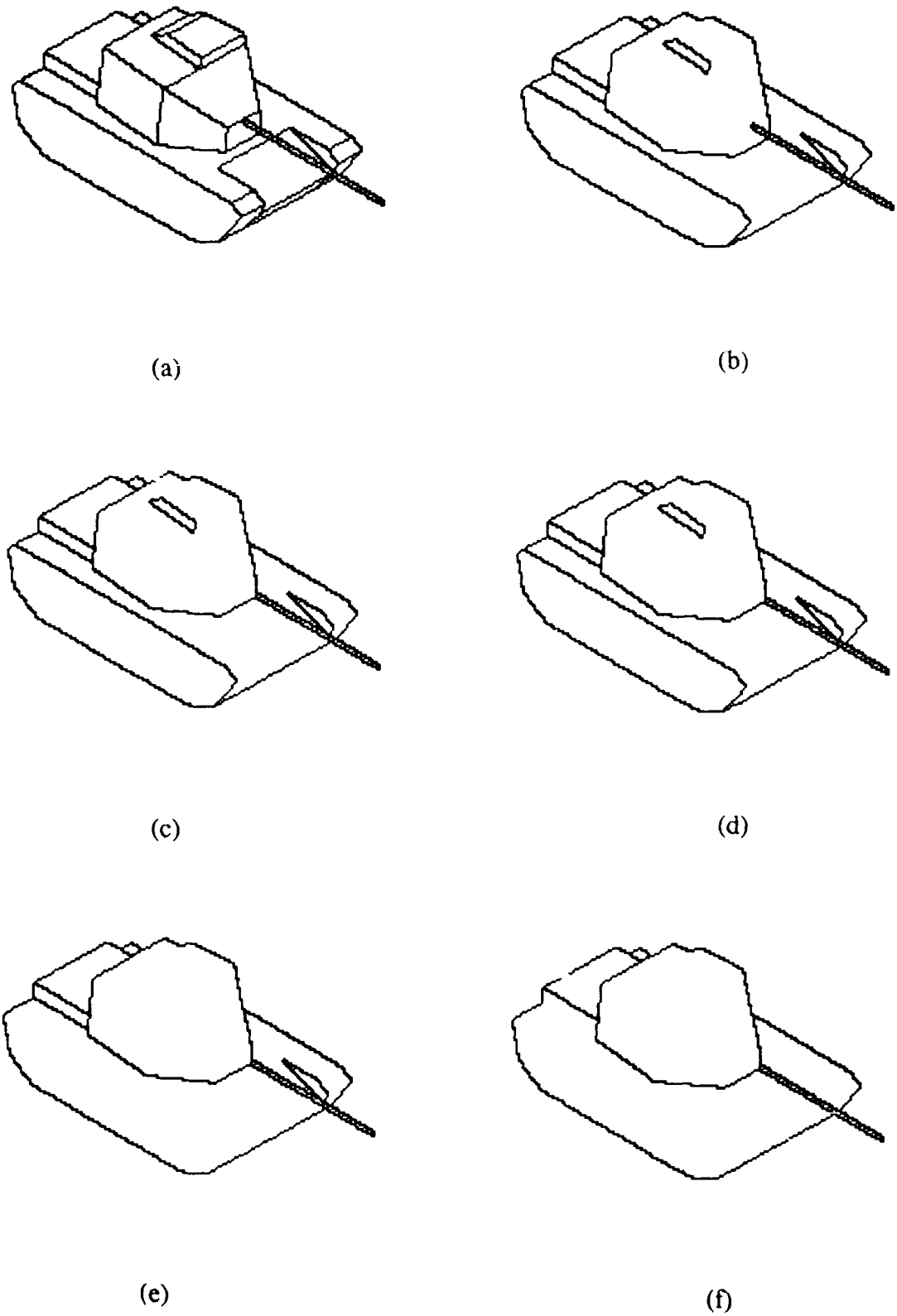
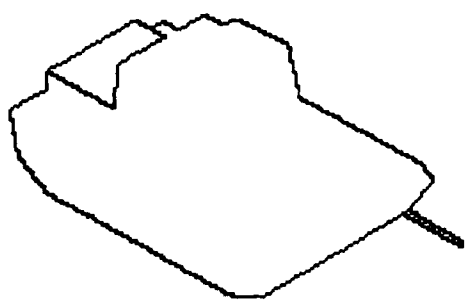
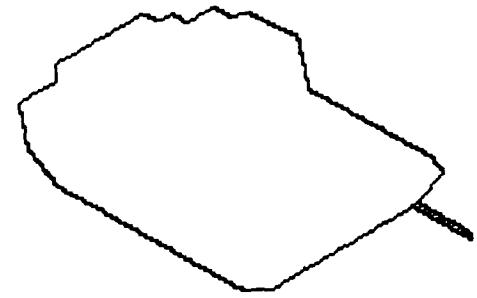


Figure 3.4.37 M60A1 with edges less 70° and less than a given length removed. a) original M60A1. b) less than 0.0 meters c) less than 0.1 meters d) less than 0.2 meters e) less than 0.3 meters f) less than 1.1 meters g) less than 2.2 meters.

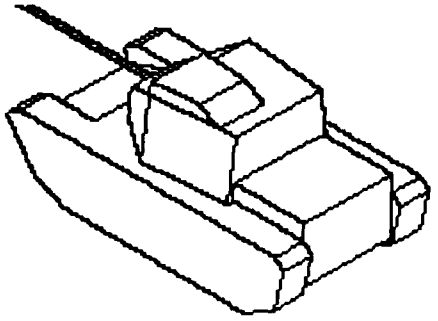


(g)

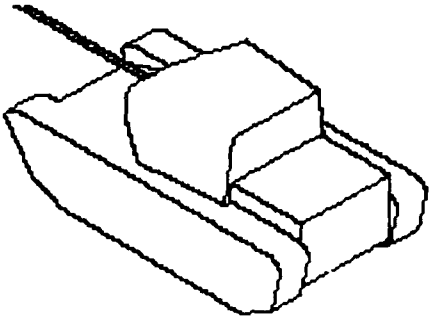


(h)

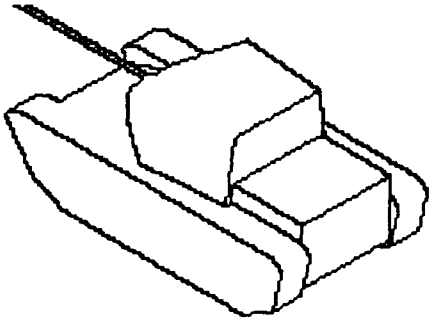
Figure 3.4.37 Continued.



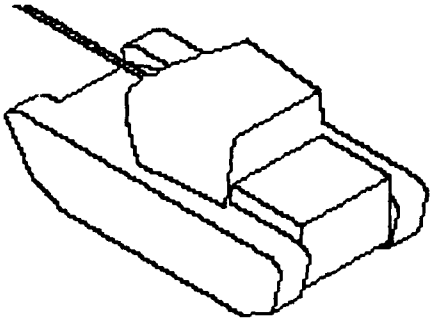
(a)



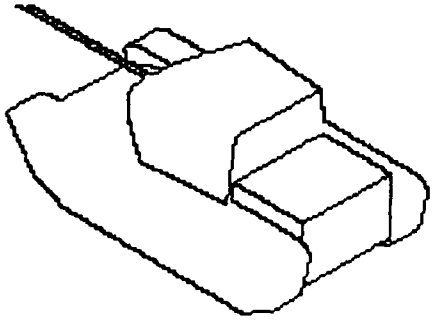
(b)



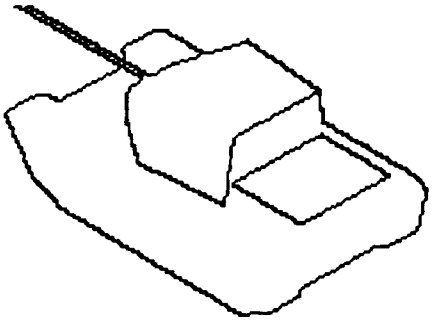
(c)



(d)

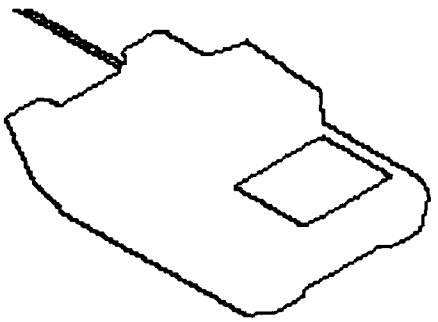


(e)

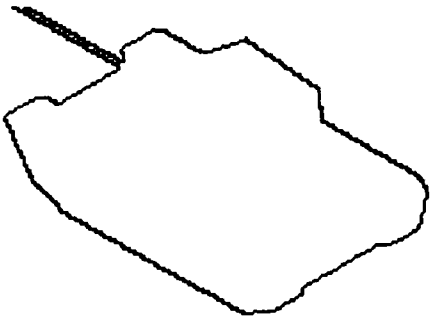


(f)

Figure 3.4.38 The back side of Figure 3.4.37.



(g)



(h)

Figure 3.4.38 Continued.

Edges are removed from the model by assigning the two surfaces which meet at the edge the same surface number. This is done by assigning both surfaces the minimum surface number of the two. Figure 3.4.32 is a list of all the edges in the M60A1 model, ordered from shorted edge to longest. In Figures 3.4.35 and 3.4.36, the edges were removed in the order show in Figure 3.4.32 (i.e. based on length only) starting with the shortest up to the longest. Remember that in an actual system, the sensor would determine the longest detectable edge, therefore it would not have to compute all the views shown here. Theses views are presented to show the appearance of the target at varying resolutions.

Figure 3.4.39 is a list of all edges in the M60A1 model ordered from smallest to largest angle up to 70°. After 70° they are ordered from shortest to longest. Such an ordering might be used if is was know that a sensor could not reliably detect angles less than 70°. In Figures 3.4.37 and 3.4.38, the edges are removed in the order show in Figure 3.4.39. The model simplifies differently than when only length is used (as in Figures 3.4.35 and 3.4.36).

Adjusting both the edge angle threshold and the edge length threshold gives great flexibility in reducing the complexity of a target. Since no information is available concerning the angle and edge lengths that can be detected, and angle of 70° and 0.1 meters is length was chosen (Figure 3.4.37c). The aspect graph is shown in Figure 3.4.40 and the corresponding views are in Figure 3.4.41.

3.4.2.4. Conclusions

Targets appearing in real LADAR data can vary in size from a few pixels to several thousand pixels. Such a wide dynamic range in perceived target size introduces some additional complications in modeling the targets for recognition. The multi-resolution aspect graph is a data structure which can be used to model such targets. It can be used to retrieve data about a given target based on the range to the target.

Adjusting thresholds based on the surface area, edge length, and edge angle gives great control over how a model is simplified. Future work must determine how to select these thresholds based on the sensor being used.

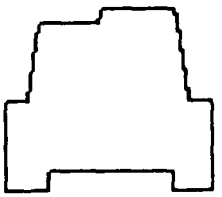
Face 1	Face 2	Angle (degrees)	length (meters)
28	20	0	0.15
30	19	0	0.15
41	32	1.7	1.61
11	7	4.97	0.93
11	9	4.97	0.93
6	1	20.32	2.2
26	8	24.82	0.71
27	10	24.82	0.71
40	37	28.37	1.12
43	40	28.37	1.12
26	15	31.3	0.71
27	17	31.3	0.71
10	9	31.47	0.71
8	7	31.47	0.71
2	1	32.01	0.71
3	1	32.01	0.71
38	37	33.69	1.06
33	32	40.66	1.41
36	35	40.66	1.41
12	1	42.27	0.71
13	1	42.27	0.71
15	14	45.66	0.71
17	16	45.66	0.71
29	11	54.2	2.2
34	33	54.44	0.6
35	34	54.44	0.6
34	38	56.31	0.49
38	34	56.31	0.49
42	38	56.31	1.4
14	12	60.76	0.71
16	13	60.76	0.71
38	33	65.7	1.62
38	35	65.7	1.62
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
44	34	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
45	44	90	0.04
31	1	90	0.15
31	1	90	0.15
42	39	94.8	0.27
42	41	94.8	0.27
19	12	90	0.3
20	13	90	0.3
24	13	90	0.3
25	12	90	0.3
7	4	92.71	0.32
9	5	92.71	0.32

Face 1	Face 2	Angle (degrees)	length (meters)
24	3	90	0.38
25	2	90	0.38
4	2	90	0.38
5	3	90	0.38
21	18	90	0.4
23	18	90	0.4
31	21	90	0.4
31	23	90	0.4
37	32	81.98	0.4
40	39	79.17	0.57
41	40	79.17	0.57
7	2	91.81	0.71
9	3	91.81	0.71
24	17	90	0.87
25	15	90	0.87
28	17	90	0.87
30	15	90	0.87
24	10	90	0.88
25	8	90	0.88
28	10	90	0.88
30	8	90	0.88
19	14	90	0.98
20	16	90	0.98
24	16	90	0.98
25	14	90	0.98
19	18	90	1
20	18	90	1
34	1	90	1.1
43	39	80.3	1.1
43	41	80.3	1.1
4	1	90	1.15
5	1	90	1.15
24	9	87.29	1.24
25	7	87.29	1.24
43	42	90	1.3
32	31	84.91	1.41
36	31	84.91	1.41
6	4	90	1.44
6	5	90	1.44
33	1	83.87	1.49
35	1	83.87	1.49
39	37	80.3	1.61
21	1	90	1.75
22	21	90	1.75
23	1	90	1.75
23	22	90	1.75
32	1	81.98	2.01
36	1	81.9	2.01
37	36	81.9	2.01
37	31	90	2.1
11	6	107.66	2.2
22	18	90	2.2
29	18	87.82	2.2
31	22	90	2.2
26	25	90	4.49
27	24	90	4.49
28	27	90	4.49
30	26	90	4.49
29	28	90	6.03
30	29	90	6.03
24	1	90	6.4
25	1	90	6.4

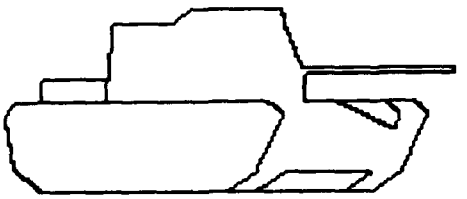
Figure 3.4.39 Angles and lengths of all the edges in the M60A1 model. Sorted first by angle between edges up to 70°, then ordered by length.

Tessel Numbers	Visible Surfaces
1	1
2 3 4 5 6 7 8	1 5 20 21 25
9	1 21 25
10 11 12 13 14 15 16	1 18 20 21 25
17	1 18
18 19 20 21 22 23 24	1 18 19 23 24
25	1 23 24
26 27 28 29 30 31 32	1 4 19 23 24

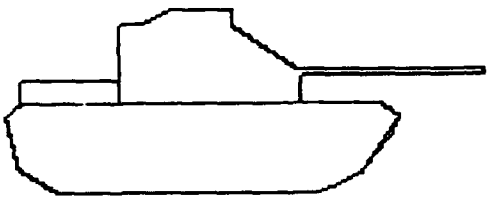
Figure 3.4.40 Figure 3.4.26 with viewpoints grouped together which view the same surfaces of an M60A1 with edges longer than 0.1 meters and angles greater than 70°.



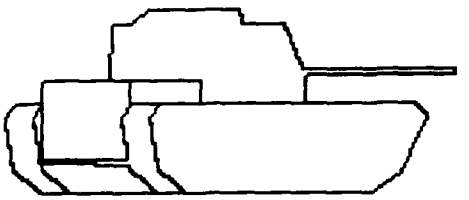
200by150



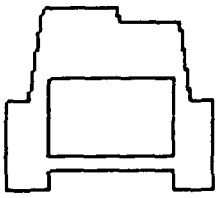
200by150



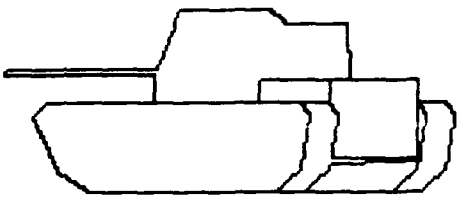
200by150



200by150

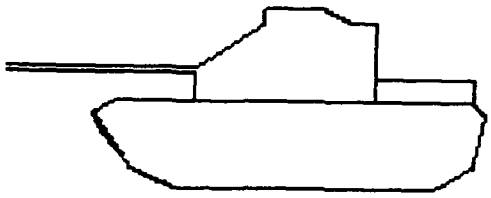


200by150

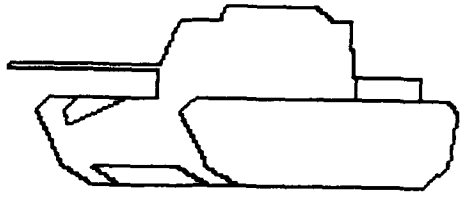


200by150

Figure 3.4.41 Nodes of the aspect graph for the M60a1 using only those edges larger than 0.1 meters with an angle of greater than 70°.



200by150



200by150

Figure 3.4.41 Continued.

4. ELECTRONIC TERRAIN BOARD MODELING

4.1. ETBM via PADL

When developing ATR algorithms, the actual data from the sensor (or sensors) should be used to test the algorithm. Unfortunately this data is not always readily available; field tests are expensive and often postponed, and after a successful field test new scenarios are thought of with an arrangement of targets, background and clutter not collected in the field test. To relieve this problem, the construction of an **Electronic Terrain Board Model** is being pursued. Anyone with such a system will be able to run "electronic" field tests and generate realistic data without the time and cost of a conventional field test. To be successful, an electronic terrain board must contain the following components:

1. geometric models of the targets of interest,
2. models of background and foreground clutter,
3. a model of environmental conditions, and
4. validation routines for testing the integrity of the simulated data.

The following sections address each of the above components.

4.1.1. Geometric Models of Targets

A first-order approximation to modeling targets can be as simple as collecting the wire frame information about each target to be modeled. The paragraphs which follow describe the conversion of the wire frame data from ERIM to geometric models.

4.1.1.1. Converting Wire Frame data to Geometric Models

ERIM has collected wire frame data which describes seventeen different vehicles. Each description contains a list of points in three space, a list of edges which are ordered pairs of points, and a list of surfaces; each surface is a list of edges ordered so that using the right hand rule, if the fingers follow the edges, the thumb is the outward normal of the surface. Table 4.1.1 shows a portion of each list for the M60A1 (All units are in meters). Some of the surface descriptions list edges with negative values; these negative values indicate that the order of the points in the edge is reversed.

PADL [Padl], a solid modeling system originally designed for describing industrial parts, is the geometric modeler being used. Although PADL was not designed for target simulation, it does have facilities for computing range images of objects which is what is needed. PADL does not understand points, edges, or bounded surfaces as input, instead it uses solid objects that are bounded by planes; therefore a program was written to convert points, edges and bounded surfaces to solid objects. This program required human input since the wire frame data did not

Point File				Edge File			Surface File	
Point	x	y	z	Edge	Point1	Point2	Surface	Edges
1.	3.15	1.81	1.6	1.	1	3	1.	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2.	3.15	-1.81	1.6	2.	3	9	2.	19 -18 20 21
3.	-3.25	1.81	1.6	3.	9	15	3.	22 -14 23 24
4.	-3.25	-1.81	1.6	4.	15	17	4.	25 26 -20 -17
5.	-3.47	1.81	1.4	5.	17	21	5.	27 -15 -22 28
6.	-3.47	-1.81	1.4	6.	21	25	6.	29 -25 -16 -27
7.	-3.47	1.10	1.4	7.	25	26	7.	30 -21 -26 31 32
8.	-3.47	-1.10	1.4	8.	26	22	8.	33 -32 34 35
9.	-3.25	1.10	1.6	9.	22	18	9.	-24 36 37 38 -28
10.	-3.25	-1.10	1.6	10.	18	16	10.	-37 39 40 41
11.	-3.25	1.10	2.0	11.	16	10	11.	42 -31 -29 -38
12.	-3.25	-1.10	2.0	12.	10	4	12.	-2 43 44 45
13.	-1.5	1.10	2.0	13.	4	2	13.	-12 46 47 48
14.	-1.5	-1.10	2.0	14.	2	32	14.	-44 49 50 51
15.	-1.5	1.10	1.6	15.	32	30	15.	-50 52 53 54
16.	-1.5	-1.10	1.6	16.	30	29	16.	-47 55 56 57
17.	-1.5	1.25	1.6	17.	29	31	17.	-56 58 59 6
18.	-1.5	-1.25	1.6	18.	31	1	18.	161 62 65 63 66 64
19.	-1.5	1.05	3.0	19.	33	1	19.	-45 -51 -230 -65
20.	-1.5	-1.05	3.0	20.	31	35	20.	-55 -46 -66 -231
21.	0.5	1.43	1.6	21.	35	33	21.	-62 67 68 -3
22.	0.5	-1.43	1.6	22.	36	32	22.	-67 -61 69 78
23.	0.5	1.23	3.0	23.	2	34	23.	-69 -64 -11 71
24.	0.5	-1.23	3.0	24.	34	36	24.	72 73 -4 -68 -70 -71 -10 74
25.	1.7	0.55	1.6	25.	29	37	25.	-74 -9 75 76 77
26.	1.7	-0.55	1.6	26.	37	35	26.	-75 -8 78 79
27.	1.7	0.485	2.2	27.	38	30	27.	-78 -7 80 81
28.	1.7	-0.485	2.2	28.	36	38	28.	-80 -6 82 83
29.	2.0	1.10	1.6	29.	38	37	29.	-82 -5 -73 84
30.	2.0	-1.10	1.6	30.	39	33	30.	-72 -77 -76 85 86 -84
...				...			...	

Table 4.1.1 Partial listing of points, edges, and surfaces for the wireframe model on an M60A1. (All dimensions are in meters.)

group the surfaces into objects bounded by planes as was needed for PADL. Figure 4.1.1 is a sample PADL input for the M60A1. Surface 1 in Table 4.1.1 is the surface between the turret and the main hull. It is a collection of edges 1 through 18. Edge 1 consists of points 1 and 3. Point 1 is at x=3.15, y=1.81, z=1.6. The next section gives a detailed description of the PADL code in Figure 4.1.1.

4.1.1.1.1. PADL Details

Table 4.1.2 gives a list a short description of the PADL commands used in Figure 4.1.1. The text which follows gives a description of how these commands are used.

Table 4.1.2 PADL commands used in Figure 4.1.1

Command	Description
blo	Block primitive object
wed	Wedge primitive object
un	Union operator
diff	Difference operator
at	Location operator
meta	Defines new primitives
plane	Defines a plane

PADL has many primitive objects which can be assembled together to make complex 3-D objects. The code in Figure 4.1.1 uses the primitive objects **blo** (a block) and **wed** (a wedge). (PADL keywords are shown in **boldface**.) These primitives are combined using various operators. The two operators used in this code are **un** which unions two objects together and **dif** which takes the difference between to objects. Objects can be placed at a specified locations by using the **at** command. For these targets, the positive Z-axis is up, the positive Y-axis is the front of the target, and the X-axis is the right of the target as viewed from the target. PADL also allows new primitive objects to be defined by using **meta**. The **meta** command is given a list of planes which bound the object in three space, and a **block** which which contains the new object. Most of the targets were defined by using *meta* objects since surfaces were given in the wire frame data.

The first line in Figure 4.1.1 defines an *m60a1* to be the **union** of the *m60a1_hull*, *m60a1_turret*, *m60a1_sm_turret*, and *m60a1_gun*. (The *m60a1_sm_turret* is the small turret on top of the main turret.) The next line defines *m60a1_box* which is a box that contains the M60A1. The lines that follow define the turret, small turret, gun, and hull. The line

```
m60a1_turret = meta(m60a1_turret0, m60a1_turret1, m60a1_turret2,
      m60a1_turret3, m60a1_turret4, m60a1_turret5, m60a1_turret6,
      m60a1_turret7, m60a1_turret8, box=m60a1_turret_box)
```

```
m60a1 = m60a1_hull un m60a1_turret un m60a1_sm_turret un m60a1_gun
m60a1_box = m60a1_hull_box un m60a1_turret_box un m60a1_sm_turret_box
    un m60a1_gun_box
m60a1_turret = meta(m60a1_turret0, m60a1_turret1, m60a1_turret2,
    m60a1_turret3, m60a1_turret4, m60a1_turret5, m60a1_turret6,
    m60a1_turret7, m60a1_turret8, box=m60a1_turret_box)
m60a1_turret0=plane=(rm=(degy=90.0, degz=180.0, movx=-1.5, movy=-1.05, movz= 3.0));
m60a1_turret1=plane=(rm=(degy=81.9022, degz=-95.1428, movx=-1.5, movy=-1.05, movz= 3.0));
m60a1_turret2=plane=(rm=(degy=83.4285, degz=-53.7462, movx=0.5, movy=-1.23, movz= 3.0));
m60a1_turret3=plane=(rm=(degy=90.0, degz= 0.0, movx=1.7, movy=-0.485, movz=2.2));
m60a1_turret4=plane=(rm=(degy=85.0073, degz=53.7462, movx=1.7, movy=0.485, movz=2.2));
m60a1_turret5=plane=(rm=(degy=81.9022, degz=95.1428, movx=0.5, movy=1.23, movz= 3.0));
m60a1_turret6=plane=(rm=(degy= 0.0, degz= 0.0, movx=-1.5, movy=1.05, movz= 3.0));
m60a1_turret7=plane=(rm=(degy=33.6901, degz= 0.0, movx=0.5, movy=-1.23, movz= 3.0));
m60a1_turret8=plane=(rm=(degy=180.0, degz= 0.0, movx=-1.5, movy=-1.25, movz=1.6));
m60a1_turret_box = blo(x=3.2, y=2.86, z=1.4) at (movx=-1.5, movy=-1.43, movz=1.6)
m60a1_sm_turret = meta(m60a1_sm_turret0, m60a1_sm_turret1,
    m60a1_sm_turret2, m60a1_sm_turret3, m60a1_sm_turret4,
    m60a1_sm_turret5, box=m60a1_sm_turret_box)
m60a1_sm_turret0=plane=(rm=(degy=80.8304, degz=95.0006, movx=0.5, movy=0.17, movz= 3.0));
m60a1_sm_turret1=plane=(rm=(degy=28.369, degz=180.0, movx=-1.1, movy=0.03, movz= 3.0));
m60a1_sm_turret2=plane=(rm=(degy=81.4126, degz=-94.6774, movx=0.5, movy=-1.18, movz=3.27));
m60a1_sm_turret3=plane=(rm=(degy=90.0, degz= 0.0, movx=0.5, movy=-1.23, movz= 3.0));
m60a1_sm_turret4=plane=(rm=(degy= 0.0, degz= 0.0, movx=0.5, movy=-1.18, movz=3.27));
m60a1_sm_turret5=plane=(rm=(degy=180.0, degz= 0.0, movx=-1.1, movy=0.03, movz= 3.0));
m60a1_sm_turret_box = blo(x=1.6, y=1.4, z=0.27) at (movx=-1.1, movy=-1.23, movz= 3.0)
m60a1_gun = meta(m60a1_gun0, m60a1_gun1, m60a1_gun2, m60a1_gun3,
    m60a1_gun4, box=m60a1_gun_box)
m60a1_gun0=plane=(rm=(degy= 0.0, degz= 0.0, movx=5.96, movy=0.05, movz=2.2));
m60a1_gun1=plane=(rm=(degy=90.0, degz=90.0, movx=1.7, movy=0.05, movz=2.2));
m60a1_gun2=plane=(rm=(degy=90.0, degz=-90.0, movx=5.96, movy=-0.05, movz=2.2));
m60a1_gun3=plane=(rm=(degy=90.0, degz= 0.0, movx=5.96, movy=0.05, movz=2.2));
m60a1_gun4=plane=(rm=(degy=180.0, degz= 0.0, movx=1.7, movy=0.05, movz=2.1));
m60a1_gun_box = blo(x=4.26, y=0.1, z=0.1) at (movx=1.7, movy=-0.05, movz=2.1)
```

Figure 4.1.1 PADL input for M60A1.

```
m60a1_hull_con = meta(m60a1_hull_con0, m60a1_hull_con1,
      m60a1_hull_con2, m60a1_hull_con3, m60a1_hull_con4, m60a1_hull_con5,
      m60a1_hull_con6, m60a1_hull_con7, m60a1_hull_con8, m60a1_hull_con9,
      m60a1_hull_con10, m60a1_hull_con11, m60a1_hull_con12,
      m60a1_hull_con13, m60a1_hull_con14, m60a1_hull_con15,
      m60a1_hull_con16, box=m60a1_hull_con_box)
m60a1_hull_con0=plane=(rm=(degy= 0.0, degz= 0.0, movx=3.15, movy=1.81, movz=1.6));
m60a1_hull_con1=plane=(rm=(degy=32.0054, degz= 0.0, movx=3.47, movy=1.81, movz=1.4));
m60a1_hull_con2=plane=(rm=(degy=32.0054, degz= 0.0, movx=3.47, movy=-1.1, movz=1.4));
m60a1_hull_con3=plane=(rm=(degy=123.818, degz= 0.0, movx=2.78, movy=1.81, movz=0.37));
m60a1_hull_con4=plane=(rm=(degy=155.179, degz= 0.0, movx=1.98, movy=1.81, movz= 0.0));
m60a1_hull_con5=plane=(rm=(degy=123.818, degz= 0.0, movx=3.47, movy=-1.1, movz=1.4));
m60a1_hull_con6=plane=(rm=(degy=155.179, degz= 0.0, movx=2.78, movy=-1.1, movz=0.37));
m60a1_hull_con7=plane=(rm=(degy=42.2737, degz=180.0, movx=-3.25, movy=1.1, movz=1.6));
m60a1_hull_con8=plane=(rm=(degy=42.2737, degz=180.0, movx=-3.25, movy=-1.81, movz=1.6));
m60a1_hull_con9=plane=(rm=(degy=103.039, degz=-180.0, movx=-3.47, movy=1.1, movz=1.4));
m60a1_hull_con10=plane=(rm=(degy=148.696, degz=-180.0, movx=-3.25, movy=1.1, movz=0.45));
m60a1_hull_con11=plane=(rm=(degy=103.039, degz=-180.0, movx=-3.47, movy=-1.81, movz=1.4));
m60a1_hull_con12=plane=(rm=(degy=148.696, degz=-180.0, movx=-3.25, movy=-1.81, movz=0.45));
m60a1_hull_con13=plane=(rm=(degy=90.0, degz=-90.0, movx=-2.51, movy=-1.81, movz= 0.0));
m60a1_hull_con14=plane=(rm=(degy=90.0, degz=90.0, movx=1.98, movy=1.81, movz= 0.0));
m60a1_hull_con15=plane=(rm=(degy=180.0, degz= 0.0, movx=-2.51, movy=1.81, movz= 0.0));
m60a1_hull_con16=plane=(rm=(degy=180.0, degz= 0.0, movx=1.98, movy=-1.81, movz= 0.0));
m60a1_hull_con_box = blo(x=6.94, y=3.62, z=1.6) at (movx=-3.47, movy=-1.81, movz= 0.0)
m60a1_hull_bottom = blo(x=6.03, y=2.2, z=0.37) at movx=-3.25, movy=-1.1, movz= 0.0
m60a1_engine = blo(x=1.75, y=2.2, z=1.4) at movx=-3.25, movy=-1.1, movz=0.6
m60a1_hood = wed(x=2.2, y=0.3, z=1.47) at degz=90, degy=-90,
      movx=3.47, movy=-1.1, movz=1.6
m60a1_hull = (m60a1_hull_con dif m60a1_hood dif m60a1_hull_bottom) un m60a1_engine
```

Figure 4.1.1 Continued.

defines the *turret* which is a **meta** object consisting of the nine planes:

```
m60a1_turret0=plane=(rm=(degy=90.0, degz=180.0, movx=-1.5, movy=-1.05, movz= 3.0));
m60a1_turret1=plane=(rm=(degy=81.9022, degz=-95.1428, movx=-1.5, movy=-1.05, movz= 3.0));
m60a1_turret2=plane=(rm=(degy=83.4285, degz=-53.7462, movx=0.5, movy=-1.23, movz= 3.0));
m60a1_turret3=plane=(rm=(degy=90.0, degz= 0.0, movx=1.7, movy=-0.485, movz=2.2));
m60a1_turret4=plane=(rm=(degy=85.0073, degz=53.7462, movx=1.7, movy=0.485, movz=2.2));
m60a1_turret5=plane=(rm=(degy=81.9022, degz=95.1428, movx=0.5, movy=1.23, movz= 3.0));
m60a1_turret6=plane=(rm=(degy= 0.0, degz= 0.0, movx=-1.5, movy=1.05, movz= 3.0));
m60a1_turret7=plane=(rm=(degy=33.6901, degz= 0.0, movx=0.5, movy=-1.23, movz= 3.0));
m60a1_turret8=plane=(rm=(degy=180.0, degz= 0.0, movx=-1.5, movy=-1.25, movz=1.6));
m60a1_turret_box = blo(x=3.2, y=2.86, z=1.4) at (movx=-1.5, movy=-1.43, movz=1.6)
```

which are displayed in Figure 4.1.2. The orientation of the planes came from the surface information in the wire frame data for the M60A1. The definition of the small turret, *sm_turret*, is

```
m60a1_sm_turret = meta(m60a1_sm_turret0, m60a1_sm_turret1,
    m60a1_sm_turret2, m60a1_sm_turret3, m60a1_sm_turret4,
    m60a1_sm_turret5, box=m60a1_sm_turret_box)
m60a1_sm_turret0=plane=(rm=(degy=80.8304, degz=95.0006, movx=0.5, movy=0.17, movz= 3.0));
m60a1_sm_turret1=plane=(rm=(degy=28.369, degz=180.0, movx=-1.1, movy=0.03, movz= 3.0));
m60a1_sm_turret2=plane=(rm=(degy=81.4126, degz=-94.6774, movx=0.5, movy=-1.18, movz=3.27
m60a1_sm_turret3=plane=(rm=(degy=90.0, degz= 0.0, movx=0.5, movy=-1.23, movz= 3.0));
m60a1_sm_turret4=plane=(rm=(degy= 0.0, degz= 0.0, movx=0.5, movy=-1.18, movz=3.27));
m60a1_sm_turret5=plane=(rm=(degy=180.0, degz= 0.0, movx=-1.1, movy=0.03, movz= 3.0));
m60a1_sm_turret_box = blo(x=1.6, y=1.4, z=0.27) at (movx=-1.1, movy=-1.23, movz= 3.0)
```

and the *gun* is

```
m60a1_gun = meta(m60a1_gun0, m60a1_gun1, m60a1_gun2, m60a1_gun3,
    m60a1_gun4, box=m60a1_gun_box)
m60a1_gun0=plane=(rm=(degy= 0.0, degz= 0.0, movx=5.96, movy=0.05, movz=2.2));
m60a1_gun1=plane=(rm=(degy=90.0, degz=90.0, movx=1.7, movy=0.05, movz=2.2));
m60a1_gun2=plane=(rm=(degy=90.0, degz=-90.0, movx=5.96, movy=-0.05, movz=2.2));
m60a1_gun3=plane=(rm=(degy=90.0, degz= 0.0, movx=5.96, movy=0.05, movz=2.2));
m60a1_gun4=plane=(rm=(degy=180.0, degz= 0.0, movx=1.7, movy=0.05, movz=2.1));
m60a1_gun_box = blo(x=4.26, y=0.1, z=0.1) at (movx=1.7, movy=-0.05, movz=2.1)
```

These two are also both **meta** objects which are bounded by **planes** as displayed in Figure 4.1.2. The following line

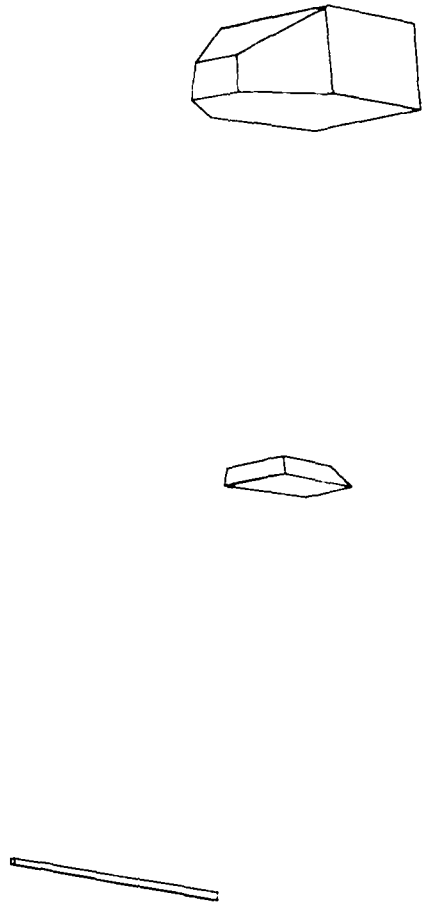


Figure 4.1.2 the PADL objects *m60a1_turret*, *m60a1_sm_turret*, and *m60a1_gun* from Figure 4.1.1.

```
m60a1_hull_con = meta(m60a1_hull_con0, m60a1_hull_con1,
    m60a1_hull_con2, m60a1_hull_con3, m60a1_hull_con4, m60a1_hull_con5,
    m60a1_hull_con6, m60a1_hull_con7, m60a1_hull_con8, m60a1_hull_con9,
    m60a1_hull_con10, m60a1_hull_con11, m60a1_hull_con12,
    m60a1_hull_con13, m60a1_hull_con14, m60a1_hull_con15,
    m60a1_hull_con16, box=m60a1_hull_con_box)
```

which is the first line of the second page of Figure 4.1.1 defines the convex part of the hull of the M60A1. PADL **meta** objects must be convex. The hull of the M60A1 has some concavities, therefore it is defined by the convex part *m60a1_hull_con* and subtracting off (using **dif**) the concave parts. Figure 4.1.3 shows the convex part, *m60a1_hull_con*, and the engine box *m60a1_engine*, the wedge above the hood, *m60a1_hood,* and the space between the treads, *m60a1_hul_bottom*. Figure 4.1.4 shows the M60A1 hull, *m60a1_hull* after subtracting the concave parts off.

The other vehicles are defined in a similar way. After converting, the vehicles were read into PADL and the wire frame, shaded, and range images in Figure 4.1.5-4.1.8 were created. Although the shaded image have the light source to the left of the viewer, they are displayed here as negatives, so the bright surfaces appear dark. Only the range images in the figures are used in our simulations. All the vehicles in Figure 4.1.5 - 4.1.8 were scaled so that they were the size that would be seen by a sensor at 500 meters using a 0.05 mrad instantaneous field of view (IFOV)† in both directions.

4.1.1.2. The Electronic Field Test

With these models in hand an *electronic* field test was performed which generated noiseless LADAR images of each target as viewed from 500 meters. Each test consisted of 60 views of each target, one view every 6 degrees. With this data we were able to generate images from any multiple of 500 meters and any multiple of 0.05 mrad IFOV, by down sampling the targets. Figure 4.1.9 shows an M60A1 as viewed from 500m to 5km.

4.1.1.3. Conclusions

We are now able to model targets "as they come from the factory". For many of our experiments this data will be sufficient. Future work in the modeling area will look into modeling targets "after user modifications". That is, it is a common practice for an operator of a vehicle to attach clutter to the target once a vehicle is in the field. Such clutter should be easily modeled by the fractal trees discussed in the sections that follow.

---

† The IFOV is the angular measurement between adjacent pixels.

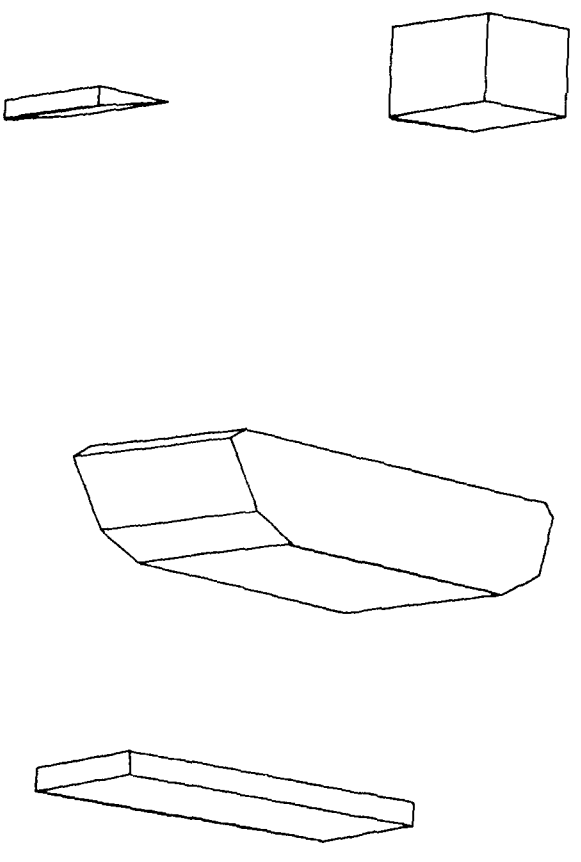


Figure 4.1.3 The PADL objects *m60a1_hull_con*, *m60a1_hull_bottom*, *m60a1_engine*, and *m60a1_hood* from Figure 4.1.1.

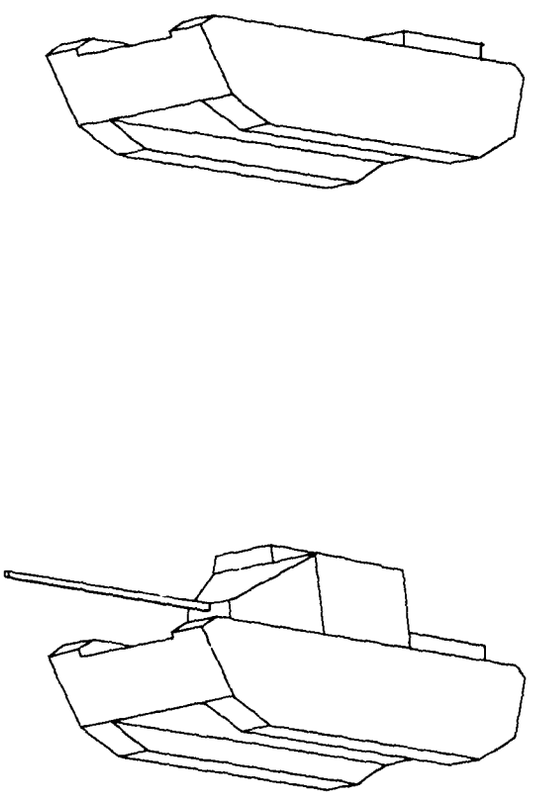
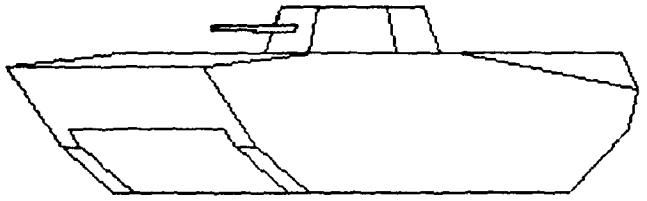
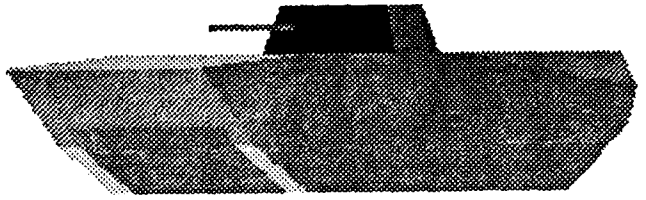


Figure 4.1.4 The PADL object *m60a1_hull* from Figure 4.1.1.

dmp.disp



bmp.shade



bmp.lase

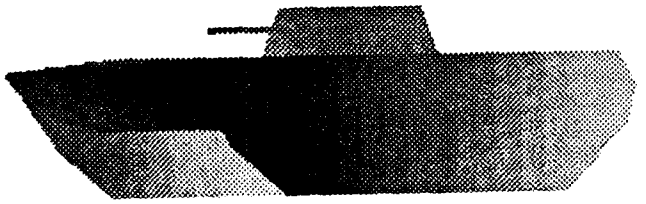
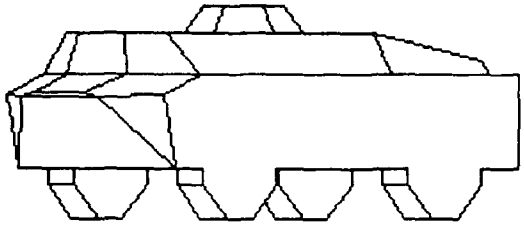
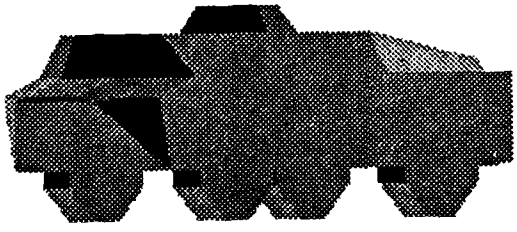


Figure 4.1.5 Wireframe, shaded, and range image of BMP.

brdm2.disp



brdm2.shade



brdm2.lase

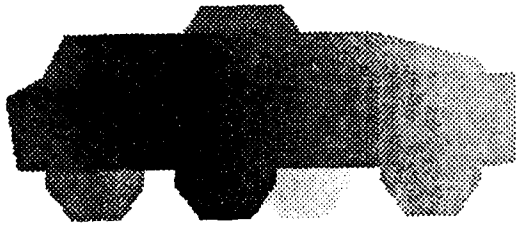
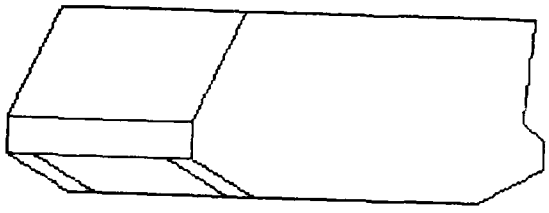
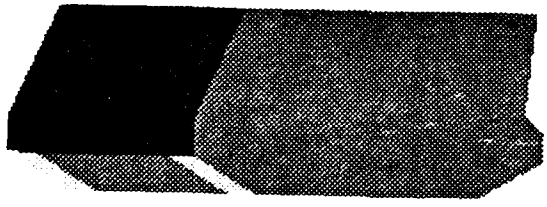


Figure 4.1.6 Wireframe, shaded, and range image of BRDM2.

m113.disp



m113.shade



m113.lase

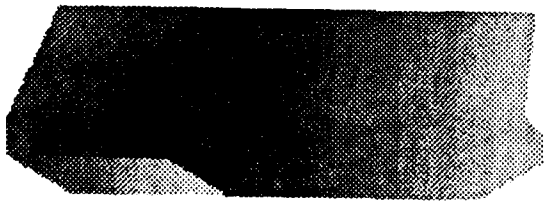


Figure 4.1.7 Wireframe, shaded, and range image of M113.

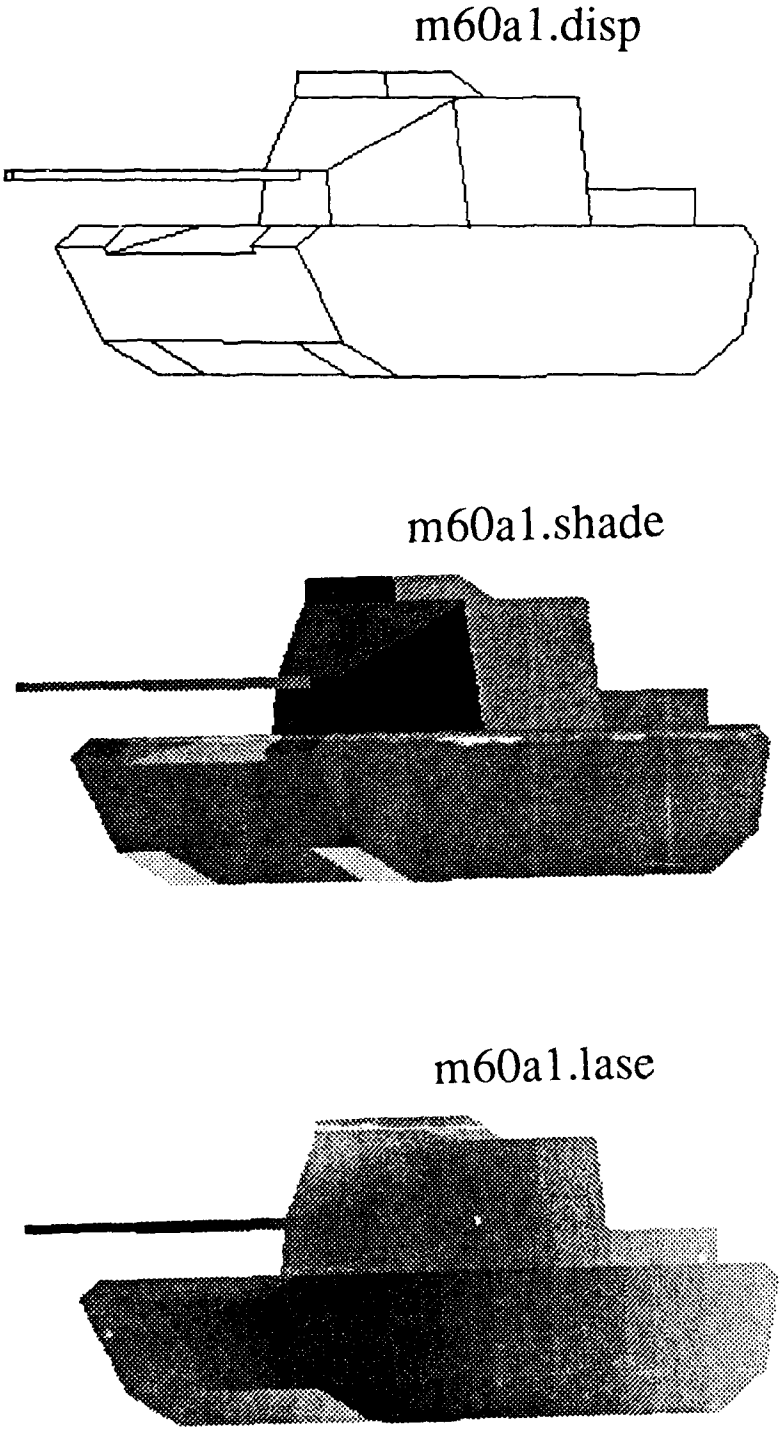


Figure 4.1.8 Wireframe, shaded, and range image of M60A1.

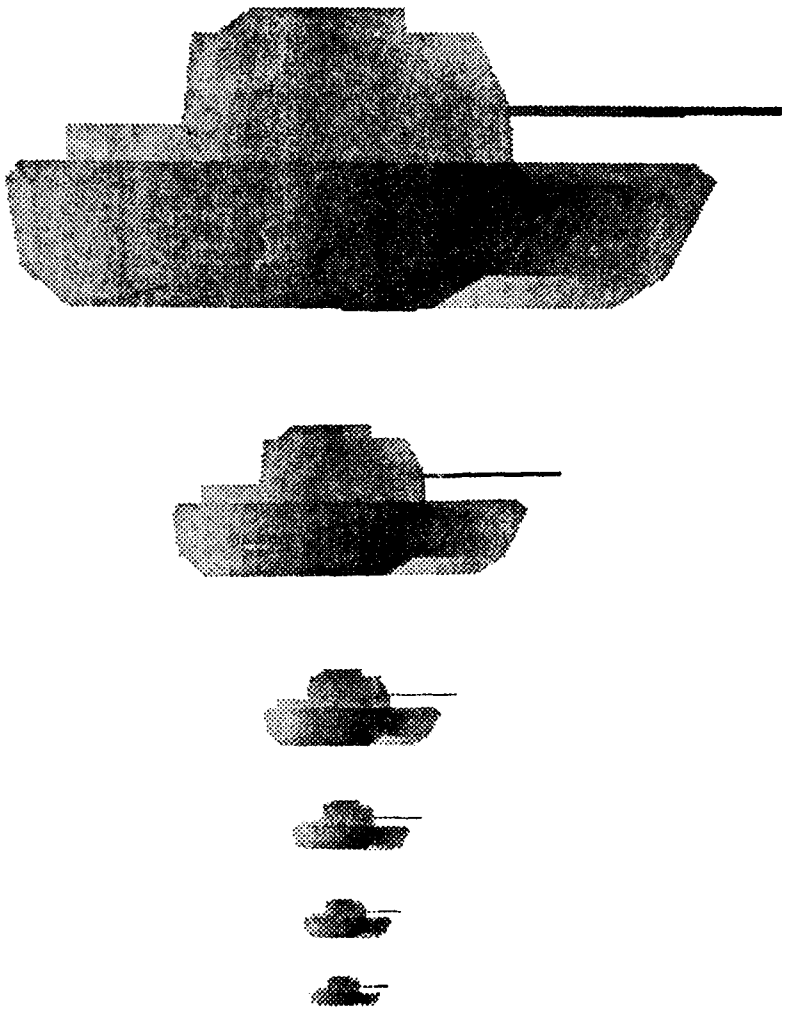


Figure 4.1.9 An M60A1 as viewed from 500m, 1km, 2km, 3km, 4km, and 5km.

4.1.2. Modeling Clutter

Modeling targets alone is not enough to generate realistic scenes. With only models of targets we can simulate a target sitting in the middle of a plane with no clutter around it. Unfortunately real targets are both surrounded by clutter and have clutter attached to them. Fractals have been shown to accurately model natural objects [Mand]. The following sections give a brief introduction to fractals and show how we used them to model the terrain of the earth and trees.

4.1.2.1. Fractals

Fractals, as defined by Mandelbrot [Mand], are a family of shapes which describe many of the fragmented and irregular shapes around us. The most useful fractals involve chance and both their regularities and their irregularities are statistical. The most useful feature of fractals for generating clutter is that the irregularity is similar at all scales. That is, a mountain can be viewed at 10km and many irregularities can be see. Moving to 1km will reveil many finer irregularities. These fragmented patterns described by fractals are needed to model clutter, whether it is at 10km or 10 meters.

4.1.2.2. Fractal-Based Terrain Generation

The generation of mountains is one of the areas fractals have been successful in generating realistic looking objects using only a few parameters. Kornfeld [Korn1] used fractals to generate the crest line structures used in the background of her simulated FLIR images. Figure 4.1.10 shows how she approximates mountain ranges using flat layers at different distances from the viewer. The top edges of the crests were created using fractals. Such an approach is a nice simplification which allows the simulation program to synthesize the FLIR image very quickly, however it cannot be used for simulating range images for the obvious reason that the sensor would pick out the flat layers.

Fournier et al. [FFC] use fractals to generate 3-D mountains by computing the elevation of points above a grid. The grid, as described here, is merely a collection of numbers which can be interpreted as elevations above a plane. The spacing between grid points is not of interest here and is later set according to the dimensions of other objects in the synthetic range image. Each grid point has a certain *neighborhood* associated with it. The elevation of the point is just the average of the neighborhood perturbed by a random amount. Also, the orientation of the neighborhood relative to the point of interest depends on its position. Figure 4.1.11 shows the order in which elevations are computed.

The grid is filled in as follows: After setting the four corner elevations, filling the rest of the grid is merely an exercise in recursion. First the corner points (the ones labeled *0*) are selected. The values chosen for these points depend on the type of terrain the designer is simulating. If a rolling plane is being simulated, all four points will be given about the same value. If the side of a mountain is being simulated, the bottom points labeled *0* may be given one

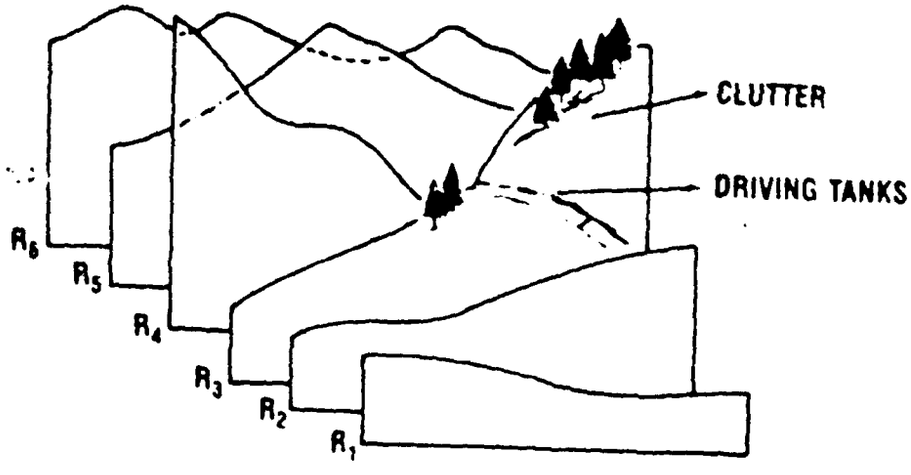


Figure 4.1.10 Schematic of crest lines structure.

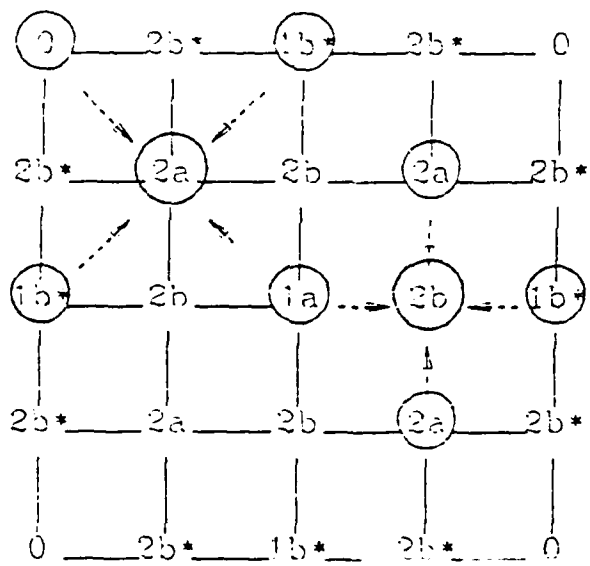


Figure 4.1.11 Order of grid point computation. The four corner points are set before computation begins. Order of computation is: 1a, 1b, 2a, 2b. An (*) indicates a point interpolated from boundary values only.

value, and the top points labeled *0* will be given a greater value. After the corner points are labeled the routine will pick a value for the point labeled *1a* by finding the average for all the points labeled *0* and then adding a random value to it. (The neighbors of *1a* are the *0* points.) Next the values for the *1b* points are found by averaging the *0* points to the left and right (or above and below if the *1b* is on the left or right) with the value of the *1a* point. Next the values of the *2a* points are found by averaging the neighbors that are shown with arrows in Figure 4.1.11. Likewise with point *2b*. Notice that the neighbors of *1a* and *1b* are above, below, left, and right of of them while the neighbors of *2a* and *2b* are oriented differently in that they are on the diagonals from them.

Figure 4.1.12 shows the different phases of a terrain patch at different times in the process of adding more detail. Figure 4.1.13 shows a patch of terrain with some targets on it.

4.1.2.3. Fractal-Based Tree Generation

Background clutter such as mountains is not the only form of clutter in a range image. Another more difficult form of clutter is foreground clutter, which is more difficult because it can obscure the targets of interest. Our first approach to simulating this type of clutter is to simulate trees. The next sections review a couple of techniques used to simulate trees in FLIR simulations and the final section shows the approach being taken here.

4.1.2.3.1. Tree Simulation at CNVEO

Gertrude Kornfeld [Korn2] simulates trees by *cutting* a tree out of actual FLIR imagery and *pasting* it into the simulated imagery. One problem with this approach is that all the trees will look the same. To overcome this problem she distorts the tree image through non-linear scaling before *pasting* it into the synthetic image. Using this approach she can have several trees (46 in one example) in a scene which are distortions of just a few trees (three trees in the same example).

The main problem with this approach is that like the mountains, the trees are flat. The same tree cannot be viewed from different angles. For her work this is fine, but for an Electronic Terrain Board Model, one must be able to describe the terrain board in three space and then view the objects from any angle.

4.1.2.3.2. The GTRI Model of Trees

The Georgia Tech Research Institute has a FLIR simulator called GTVISIT which contains a three dimensional tree model. This model (shown in Figure 4.1.14) contains over 32,000 facets which describe the tree's reflectivity in three dimensions. The simulator allows the tree to be copied, scaled, rotated, and placed anywhere in the scene. This approach overcomes the problem of having a 2-D tree, but due to the large size of the tree, it is the only hardwood tree in the model, and scaling and rotation are used to give the appearance of different trees.

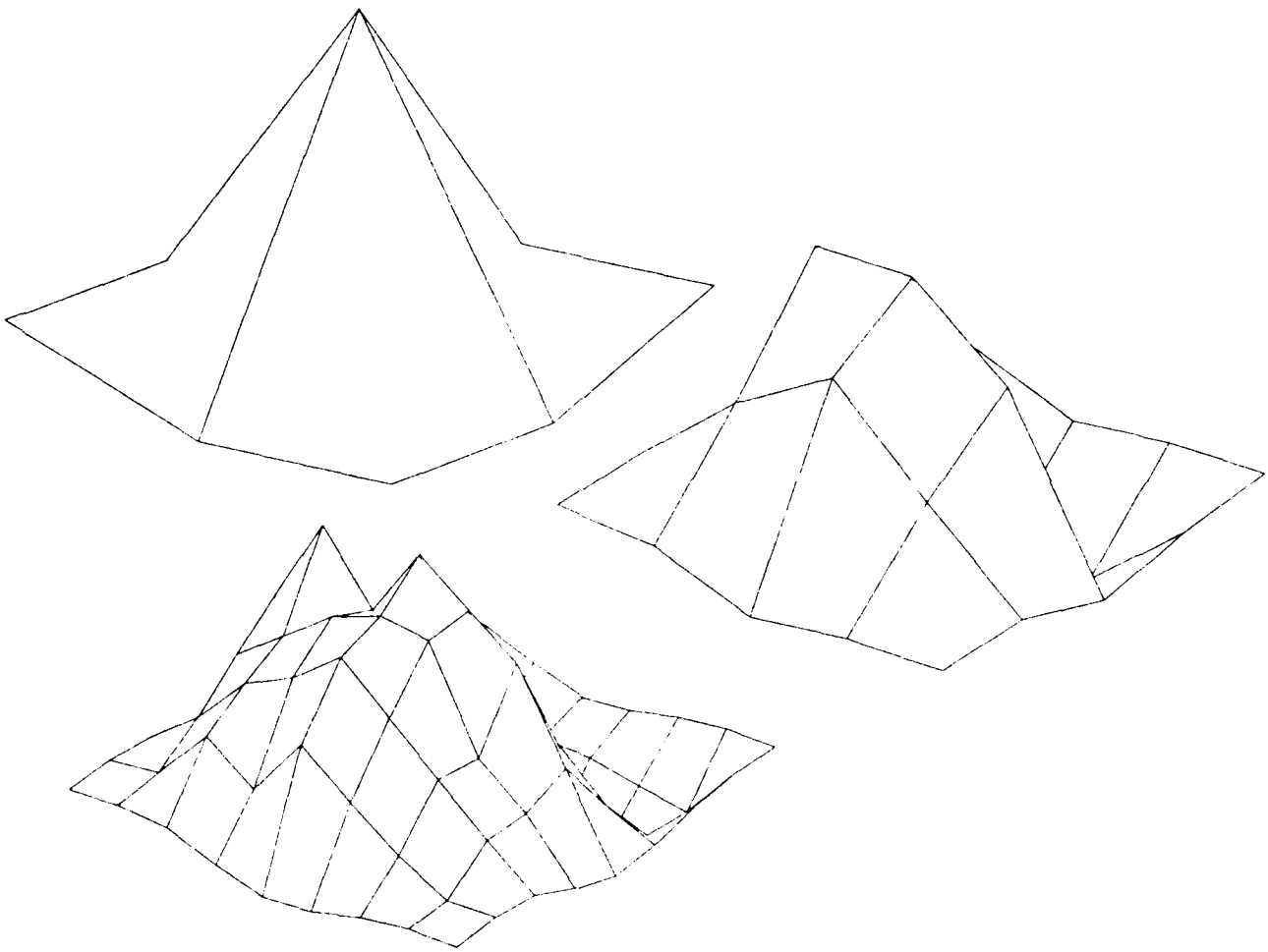


Figure 4.1.12 Different phases of a terrain patch being formed.

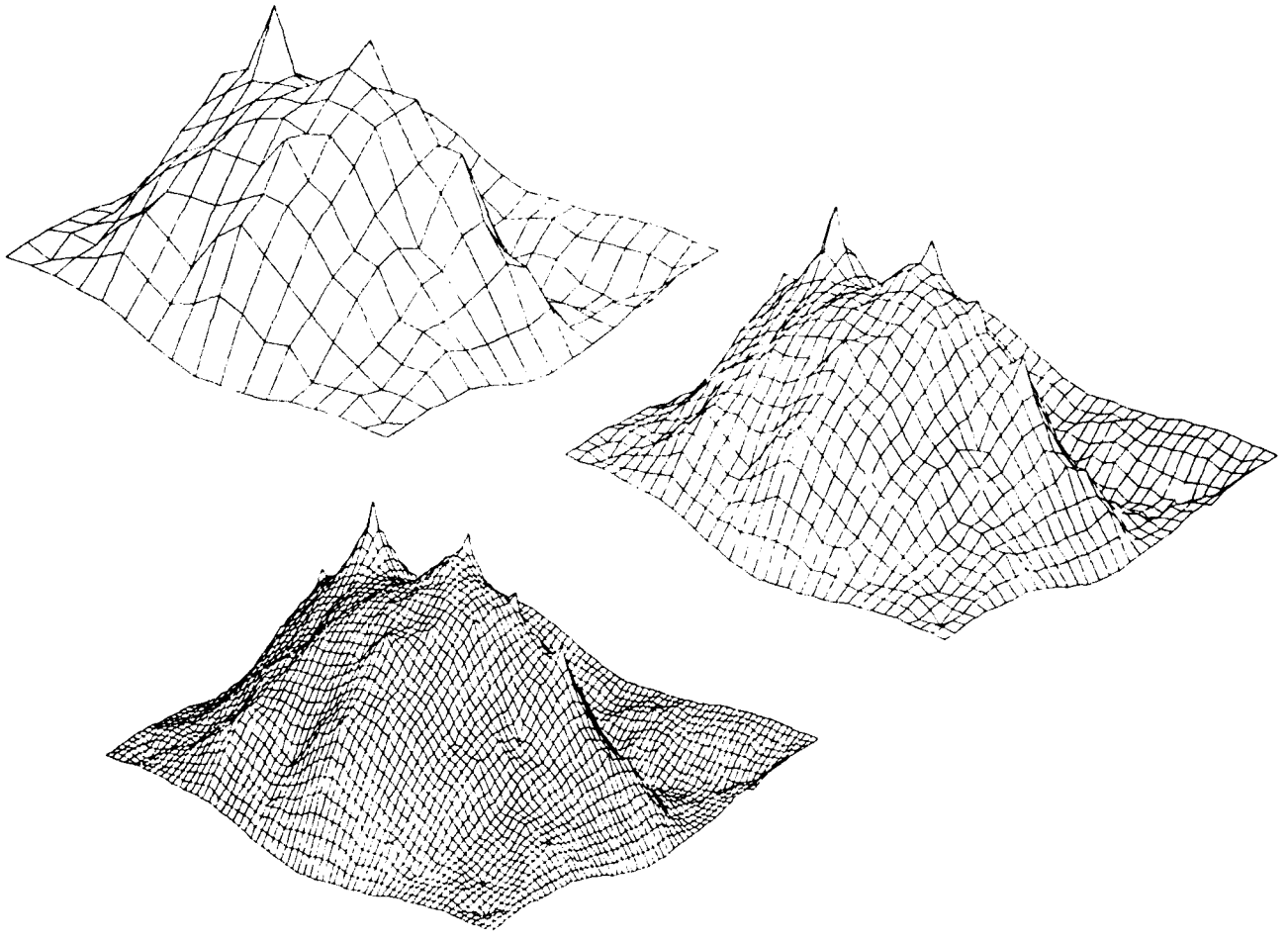


Figure 4.1.12 (Continued)

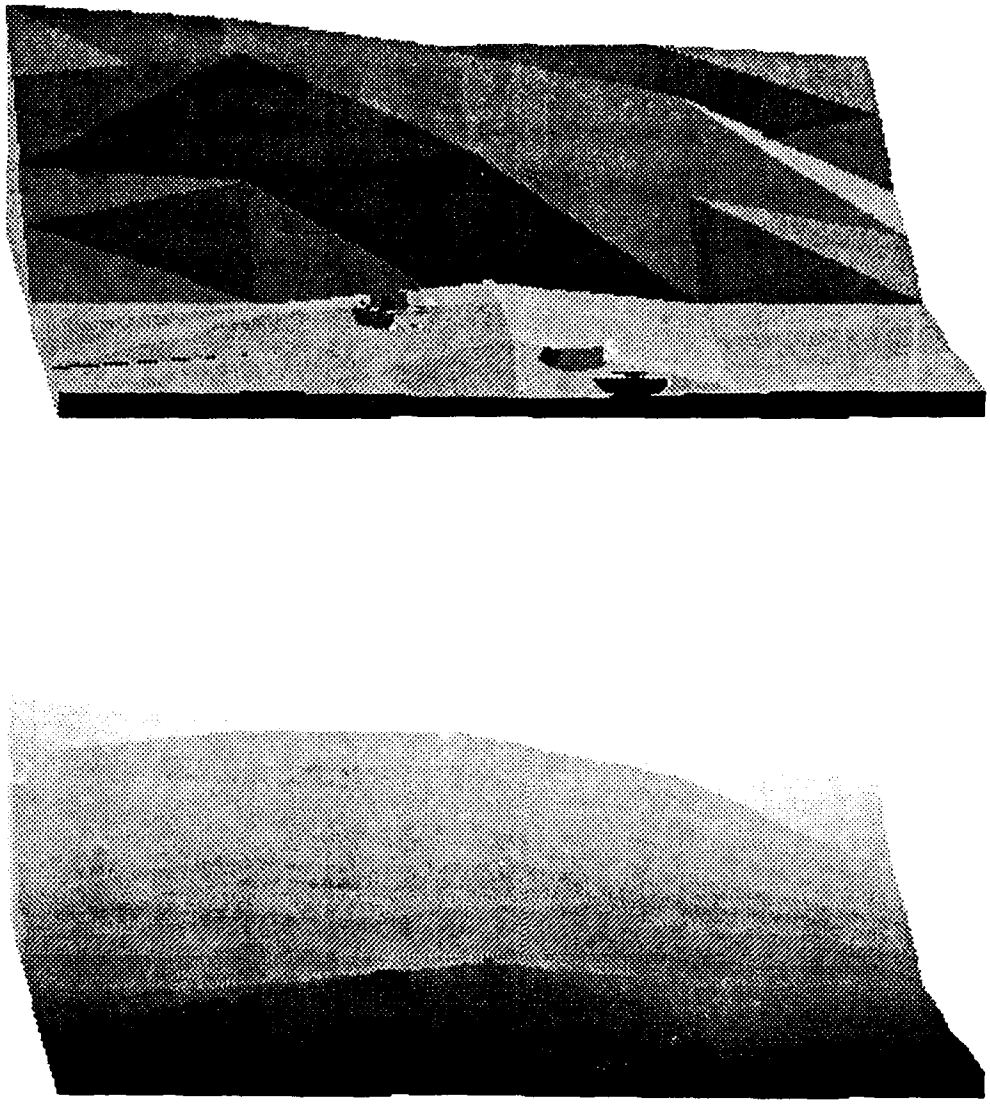


Figure 4.1.13 Shaded and range images of a simulated M60a1 and a M113 on a terrain patch.



Figure 4.1.14 Simulated hardwood tree in GTVISIT model. The same tree appears three times, each time it is scaled and rotated differently.

4.1.2.3.3. The Fractal Approach to Trees

A three dimensional model of a tree must be used for simulating LADAR images. This model should be detailed enough to look like a tree to the sensor being modeled, but be simple enough for several different trees to be used in the same image. One approach is to *grow* the trees as they are needed by using a formal grammar. This approach, presented in [Smit], uses a formal grammar (an example is shown in Figure 4.1.15) which describes how the tree is grown from one generation to the next. Each generation is grown by taking the symbols on the left side of the production rules and replacing them with the symbols on the right of the production rules. If we define the Z-axis as being up, the angle of the branches around the Z-axis and down from the Z-axis can be randomly chosen. The range of the random numbers determines the type of tree being grown.

This approach has the advantage that every tree used can be a different tree. This reduces the possibility of an algorithm becoming accidentally tuned to a given tree. Once a tree is grown, it can be placed on the Electronic Terrain Board Model and viewed from any angle. Figure 4.1.16 is a fourth generation tree using the grammar in Figure 4.1.15. The branches of the tree are cylinders, all of the same diameter, and there are no leaves on the tree. Future work should make the tree look more like a tree as viewed by a LADAR sensor.

4.1.2.4. Real Terrain Board Data

Although fractals can be used to generate real looking terrain, real terrain data looks even more real. Figure 4.1.17 is an image created using the elevation data from the Night Vision Laboratory Terrain Model. This data could have been used in generating the scenes in this report, however it was not available at the time time work was being done.

4.1.3. Noise Degradation of Synthetic Range Imagery

The next two components of the Electronics Terrain Board Model which need to be addressed are the modeling of the sensor and the current environmental conditions. The purpose of these models is to add the correct type and amount of noise to the synthesized image to simulate the sensor and how it interacts with the environment. We lacked specific information on the LADAR sensor, so we chose to examine the noise present in the LADAR image taken during the 1986 A.P. Hill test. The next sections describe our analysis of the noise, and the creation of noisy synthetic images.

Both the general nature of the noise (e.g. Gaussian, uniform, etc.), and how the noise varies with range are studied here.

4.1.3.1. Analysis of LADAR Noise

The analysis which follows is based on experiments performed with the *old* laser range data, i.e. background pixels are merely noise. Depending on the changes made to the sensor,

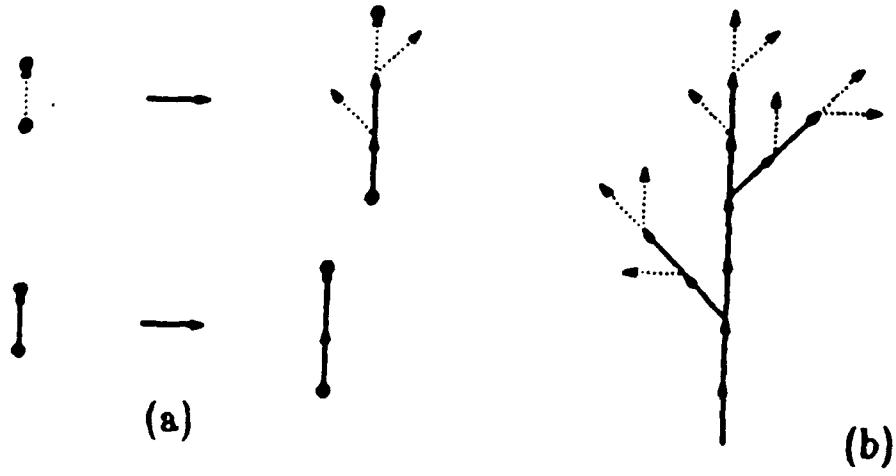


Figure 4.1.15 Grammar for growing a tree. (a) Production rules. (b) Generation $n=2$.

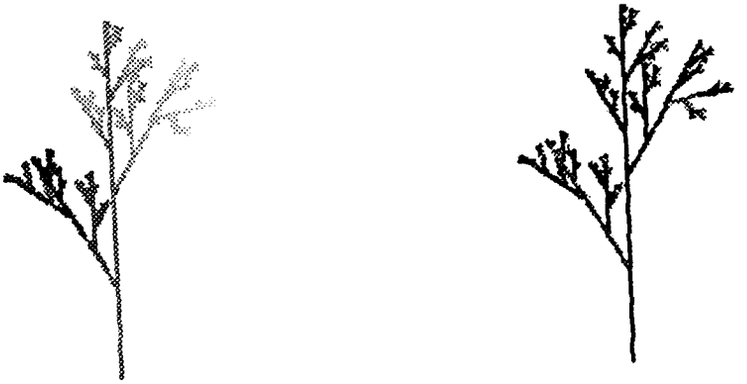


Figure 4.1.16 Fourth generation tree using the grammar in Figure 4.1.15.

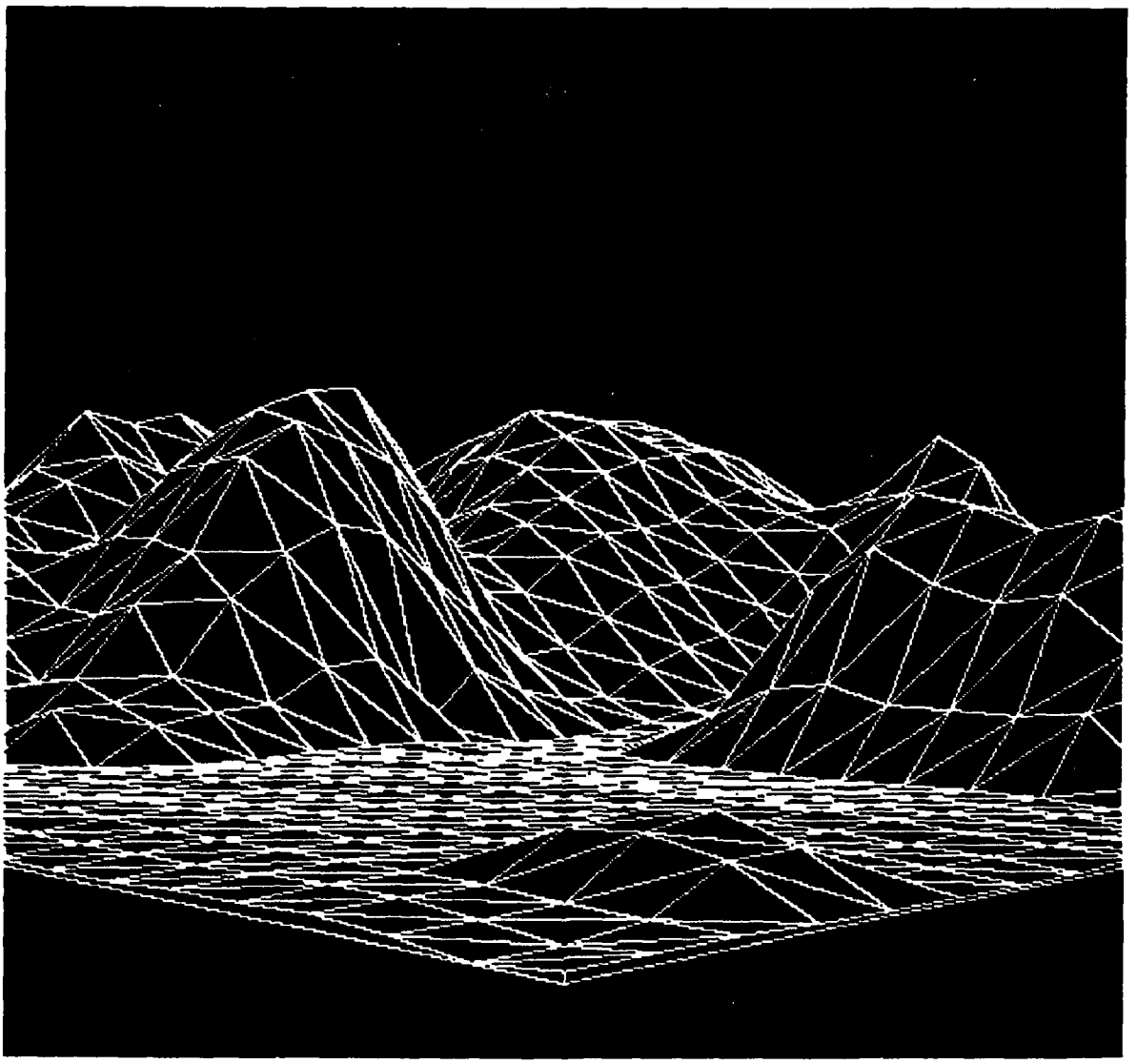


Figure 4.1.17a Mountains generated from NVL terrain board data. Facet image.

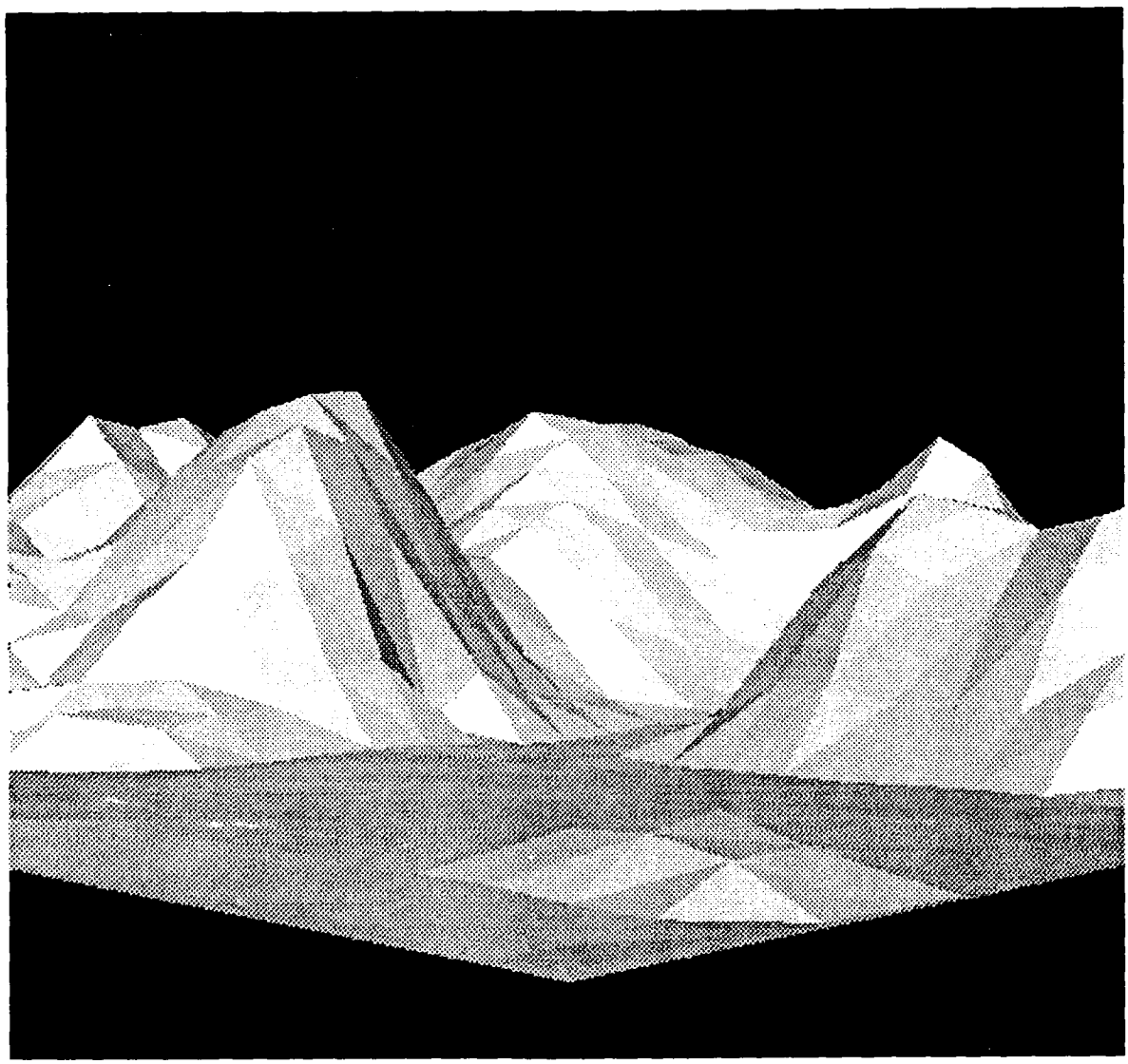


Figure 4.1.17b Mountains generated from NVL terrain board data. Shaded image.

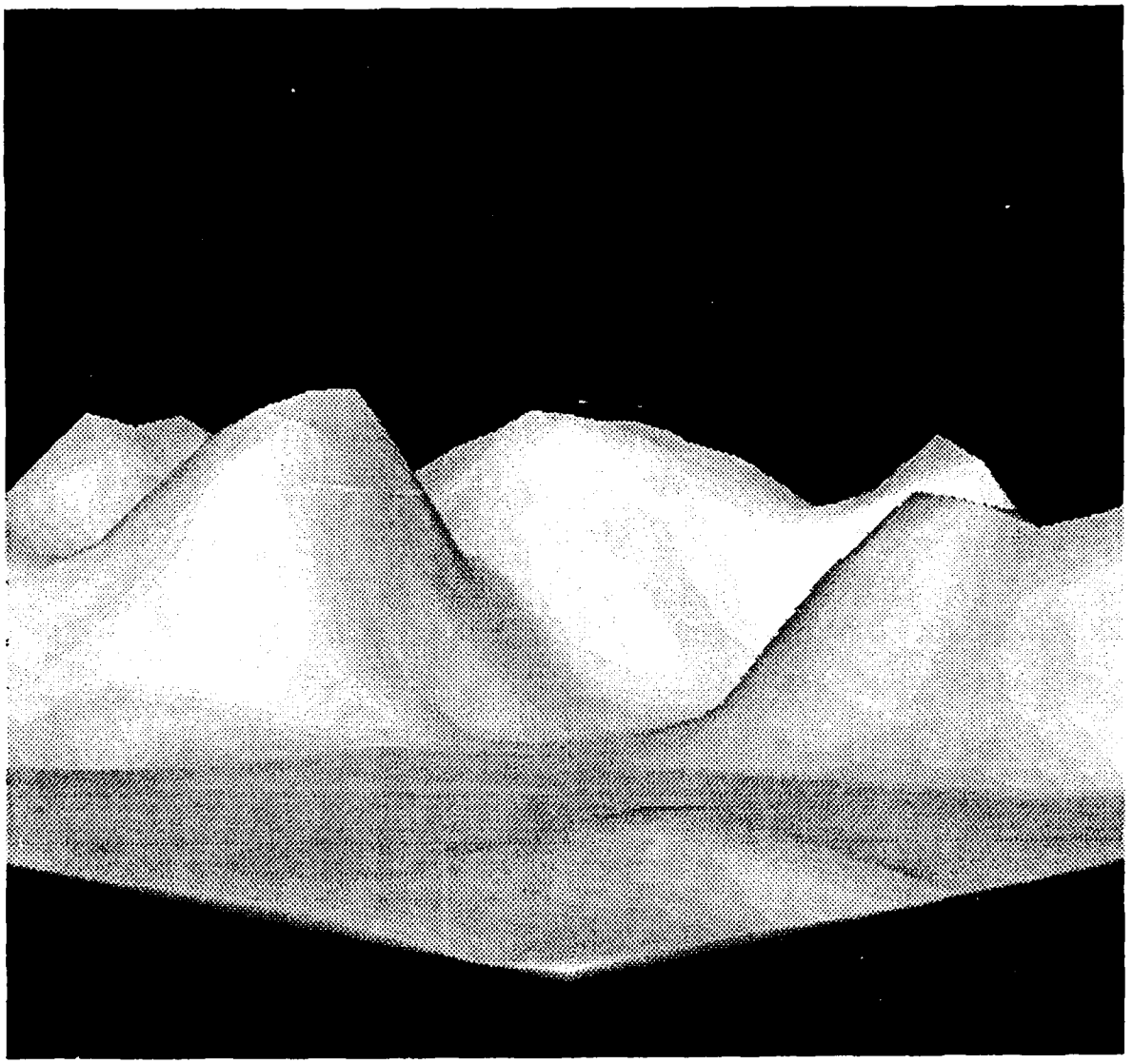


Figure 4.1.17c Mountains generated from NVL terrain board data. Shaded image with smoothing.

the results presented may change drastically. Recall, however, that the main objective here is to determine whether realistic images can be generated, and the actual data used is the only data available for comparison.

4.1.3.1.1. General Characteristics

The five ton truck target is used for the analysis in this section and the next. This target provides a large flat surface from which to extract a window of data at ranges of 1.54, 2.91, and 4.24 km.

Figure 4.1.18 is a sample image from the 1.54 km set showing the hand selected rectangular window used to extract noise data from only the target. In many simulation type problems it may be possible to assume Gaussian noise characteristics. However the histogram of Figure 4.1.19 shows the data of the above window clearly indicates that a more complex model is necessary here. Notice that the extremities of the histogram are flat and exhibit a *uniform* density characteristic while the central region looks somewhat Gaussian.

As is apparent from the background of Figure 4.1.18, if an unreasonable return or no return is received by the sensor, an arbitrary gray level between 0 and 255 is assigned. It seems reasonable to expect that some *on-target* pixels may be assigned similarly. The histogram of Figure 4.1.19 supports this since most pixels contain a gray level corresponding to a range measurement corrupted by noise (the central region of the histogram) while the rest represent a gray level chosen at random.

Once the random pixels are separated from the data the remaining noise is approximated as Gaussian since many factors act together to perturb a given range measurement. Assuming ergodicity (as is done throughout this analysis and the sections to follow), the relative frequency of randomly chosen gray levels within a given window is a good approximation of the probability that a given pixel has a random gray level. The value of this is in determining which pixels of a synthetic range image should be corrupted with *uniform* noise.

As was pointed out earlier, the far ends of the histogram have the *uniform* type of characteristic which would be evident in samples chosen completely at random. Thresholds are chosen, therefore, beyond which the histogram is considered to be *uniform*. Next it is necessary to determine how many of the gray levels of the central region are due to random assignment. This is simple since a basic property of the *uniform* distribution is an even spread throughout the possible range of values. The following is an example of the necessary interpolation:

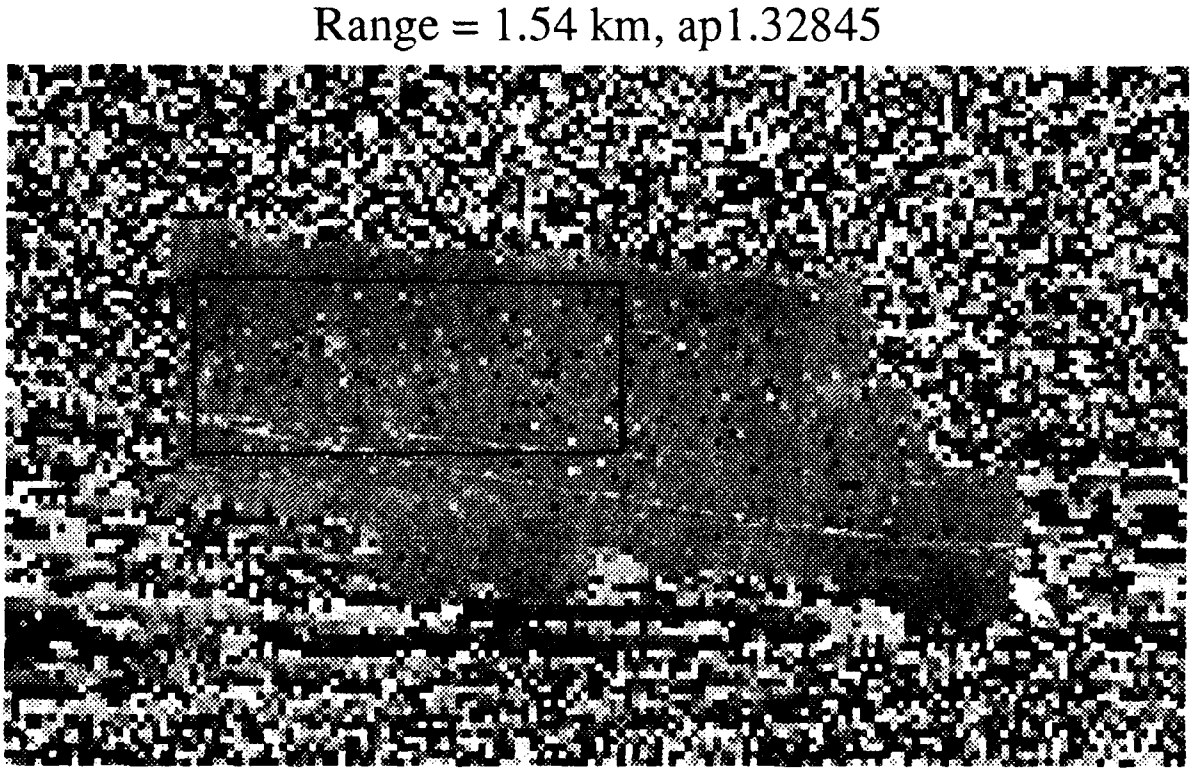


Figure 4.1.18 5 Ton Truck, Image 45, Date 3/28/86, Range 1.54 km. Hand selected window of 1416 pixels from 160 x 96 pixel laser range image.

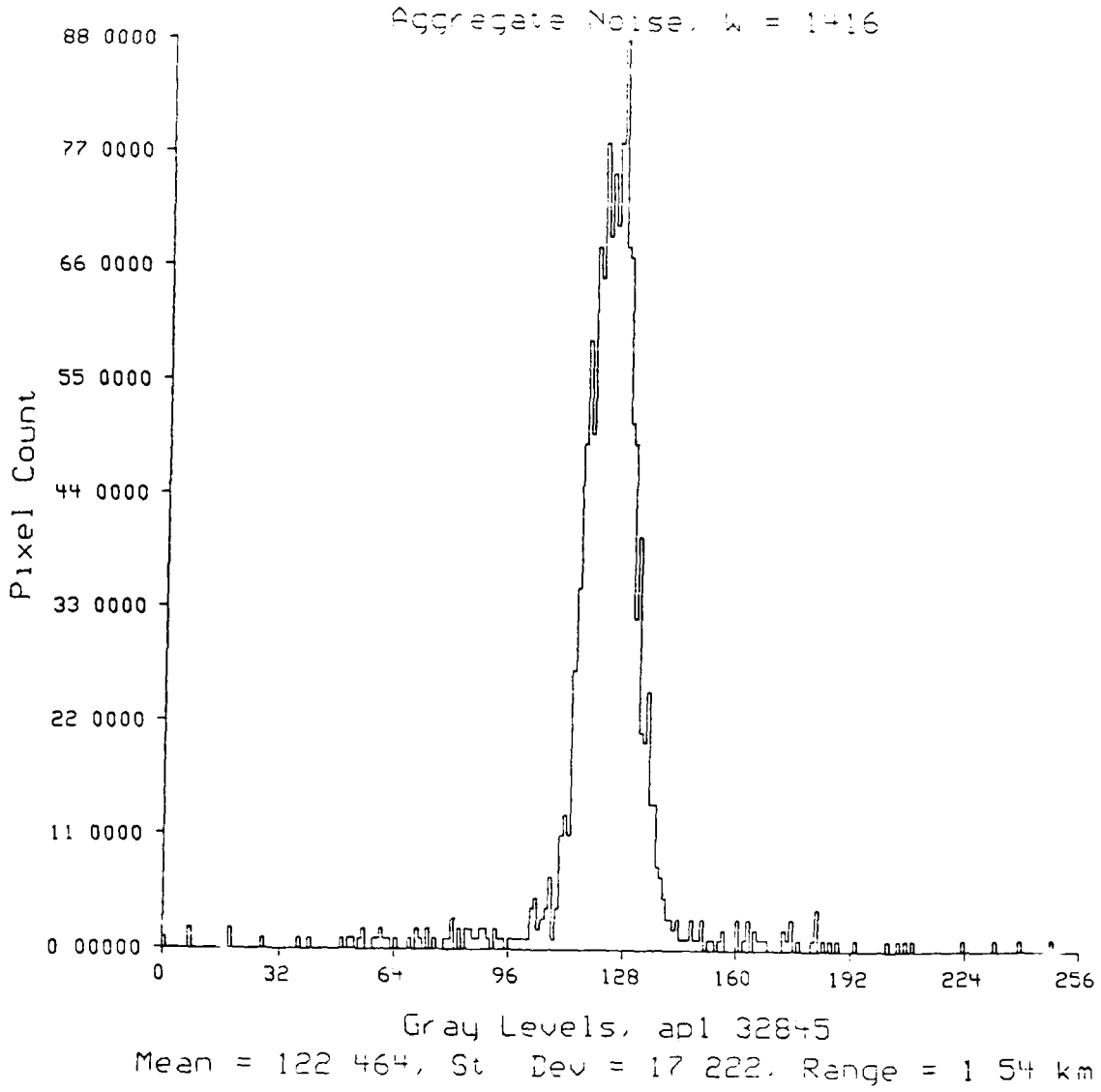


Figure 4.1.19 Aggregate gray level histogram of 1416 pixel window. Mean = 122.464, Standard Deviation = 17.222.

HIGH = gray level greater than mean, above which the histogram is considered uniform.
LOW = gray level less than mean, below which the histogram is considered uniform.
N = (255 - HIGH + 1) + (LOW + 1)
= number of gray levels in the uniform regions.
P = number of pixels in uniform regions.

P/N = average number of pixels per gray level in uniform region
*X = (P/N) * (HIGH - LOW + 1)*
= approximate number of random pixels between HIGH and LOW.

The analysis above completes one important step in the noise synthesis problem. Given a synthetic range image of a target, it is now known what fraction of *on-target* pixels should be corrupted by random noise. The next task is to determine the variance of the Gaussian portion.

In order to estimate the Gaussian characteristic it is merely necessary to subtract the random pixels from the aggregate noise histogram. The problem is determining which pixels between HIGH and LOW are random, and which are true measurements perturbed by Gaussian noise. These random pixels are approximated by randomly choosing X gray levels between HIGH and LOW. If the random pixel value chosen did not appear in the original image, the value will be discarded and a new random value selected. After subtracting these from the aggregate noise, an approximation of the Gaussian noise remains. See Figures 4.1.20 and 4.1.21. Again, the histogram of Figure 4.1.20 does not convey much useful information by itself, but it is used to extract the Gaussian data contained in the aggregate noise histogram.

This concludes the examination of the general noise characteristics. The following section addresses the question of how these characteristics depend on range.

4.1.3.1.2. Noise Variation With Range

Only two parameters of the previous section will be examined here to determine their variation with range. They are:

(1) The drop-out probability.

(2) The variance of the Gaussian data.

Item (1) is essentially the relative frequency of pixels within a given data window that have random gray levels. Item (2) follows from a straightforward calculation following the subtraction of random pixels from the aggregate noise data.

The five ton truck target is again used in the following study, but at ranges of 1.54, 2.91, and 4.24 km. Figures 4.1.18, 4.1.22, and 4.1.23 show sample data at these ranges along with the windows used for data extraction. Figures 4.1.19, 4.1.24, and 4.1.25 are histograms of this data.

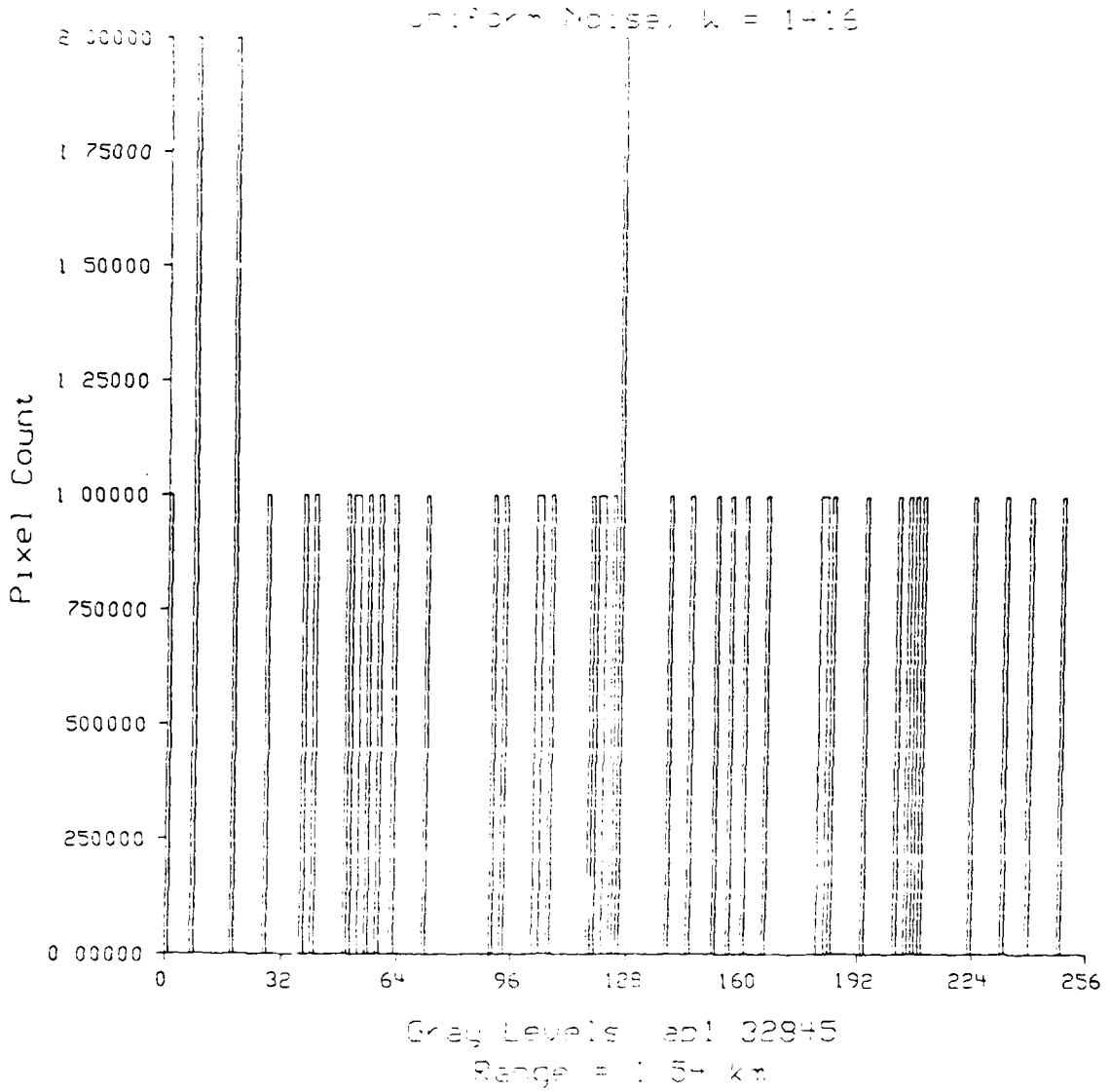


Figure 4.1.20 Histogram of *uniform* gray levels extracted from aggregate histogram of 1416 pixel window.

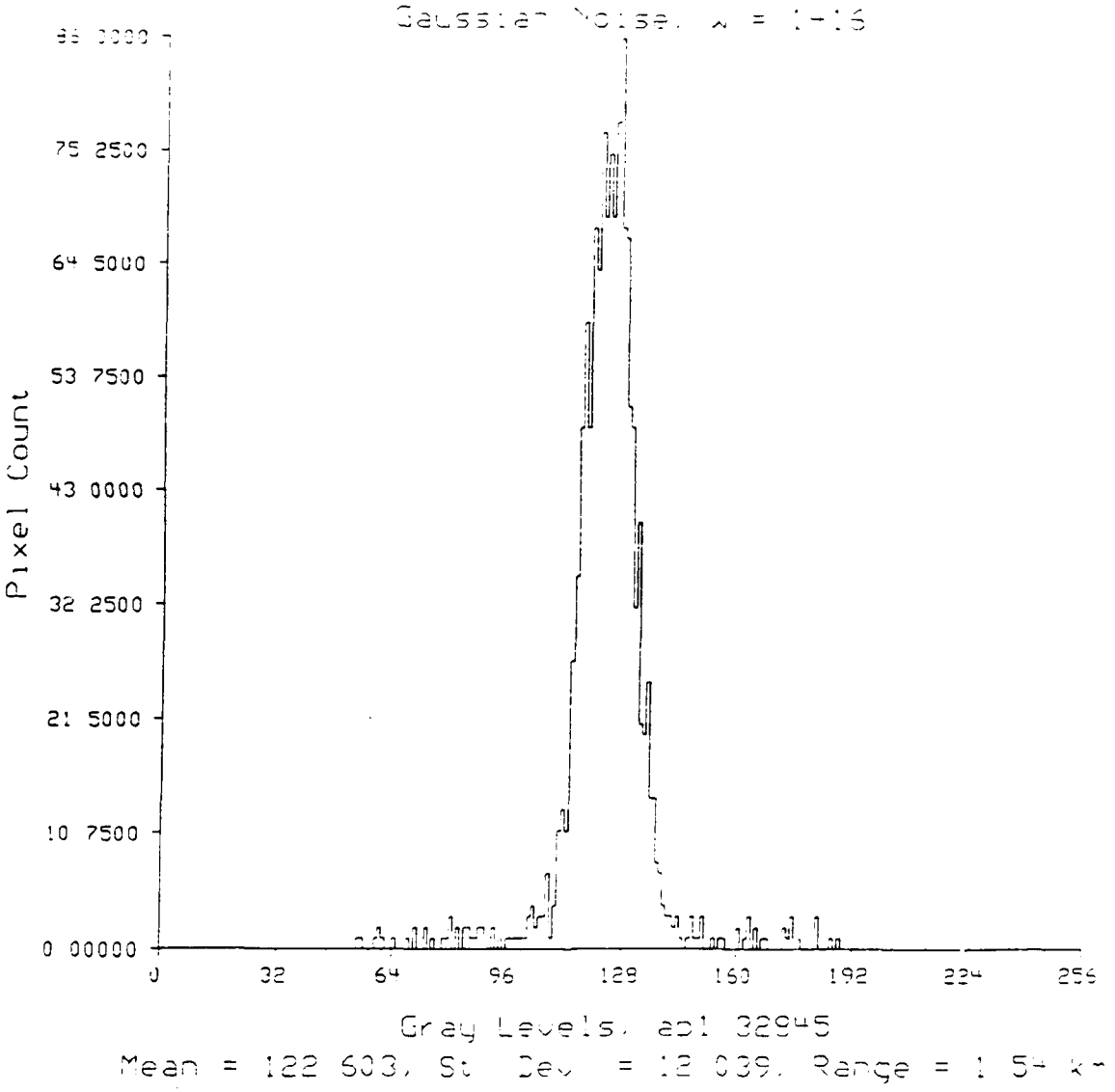


Figure 4.1.21 Histogram of Gaussian gray levels extracted from aggregate histogram of 1416 pixel window. Mean = 122.603, Standard Deviation = 12.039.

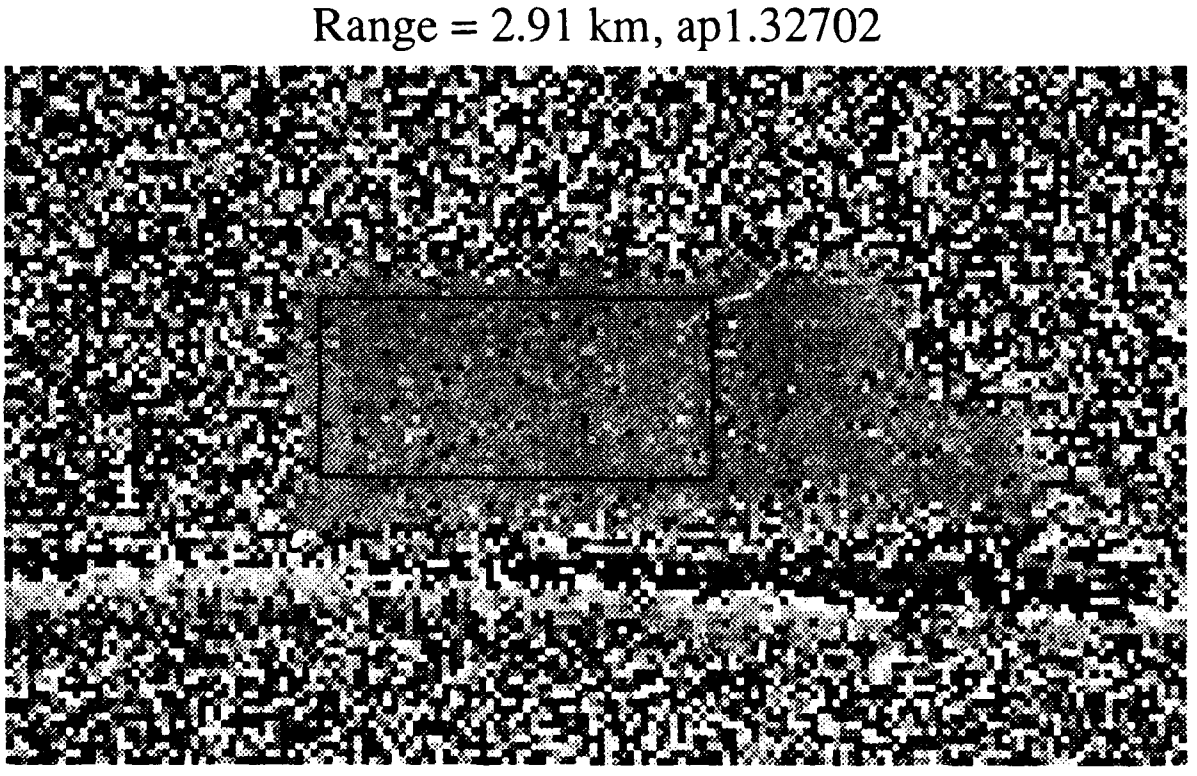


Figure 4.1.22 5 Ton Truck, Image 02, Date 3/27/86, Range 2.91 km. Hand selected window of 1325 pixels from 160 x 96 pixel laser range image.

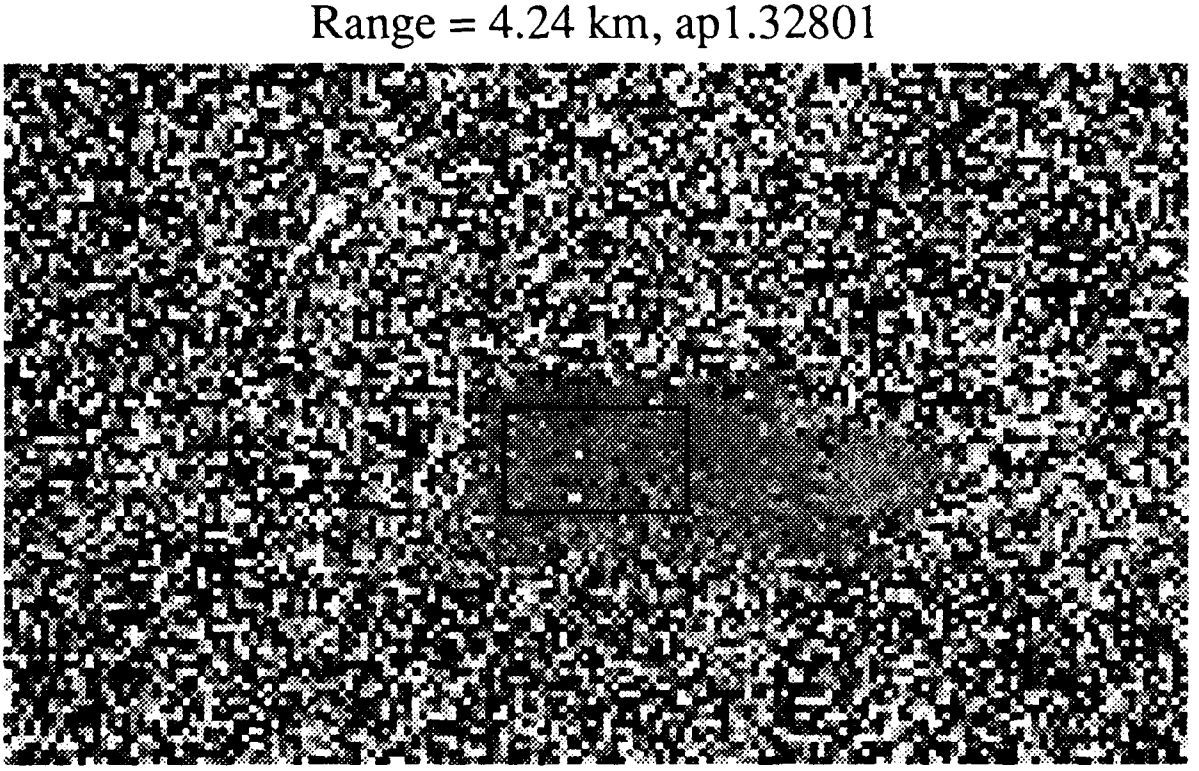


Figure 4.1.23 5 Ton Truck, Image 01, Date 3/28/86, Range 4.24 km. Hand selected window of 375 pixels from 160 x 96 pixel laser range image.

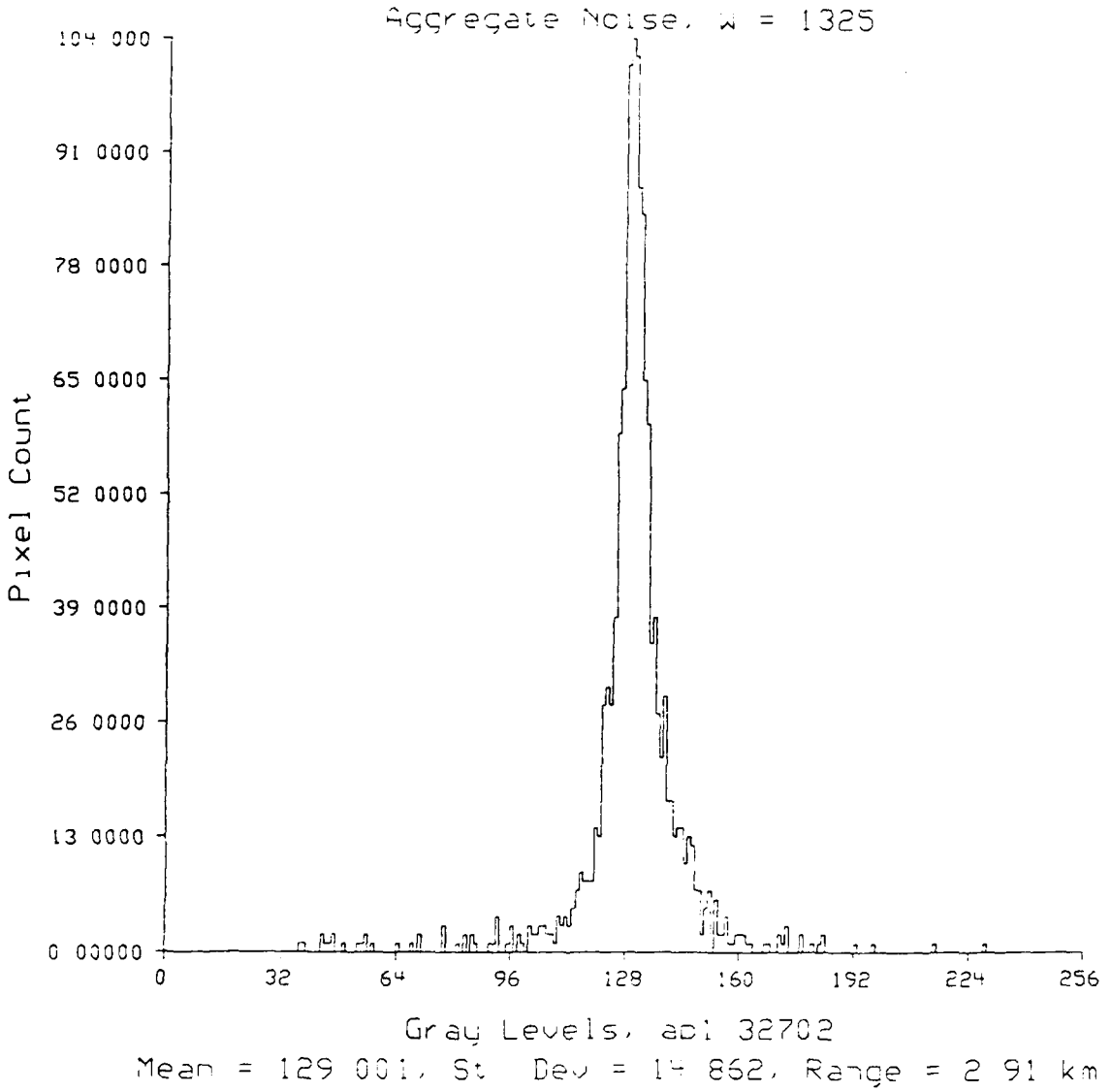


Figure 4.1.24 Aggregate gray level histogram of 1325 pixel window. Mean = 129.001, Standard Deviation = 14.862.

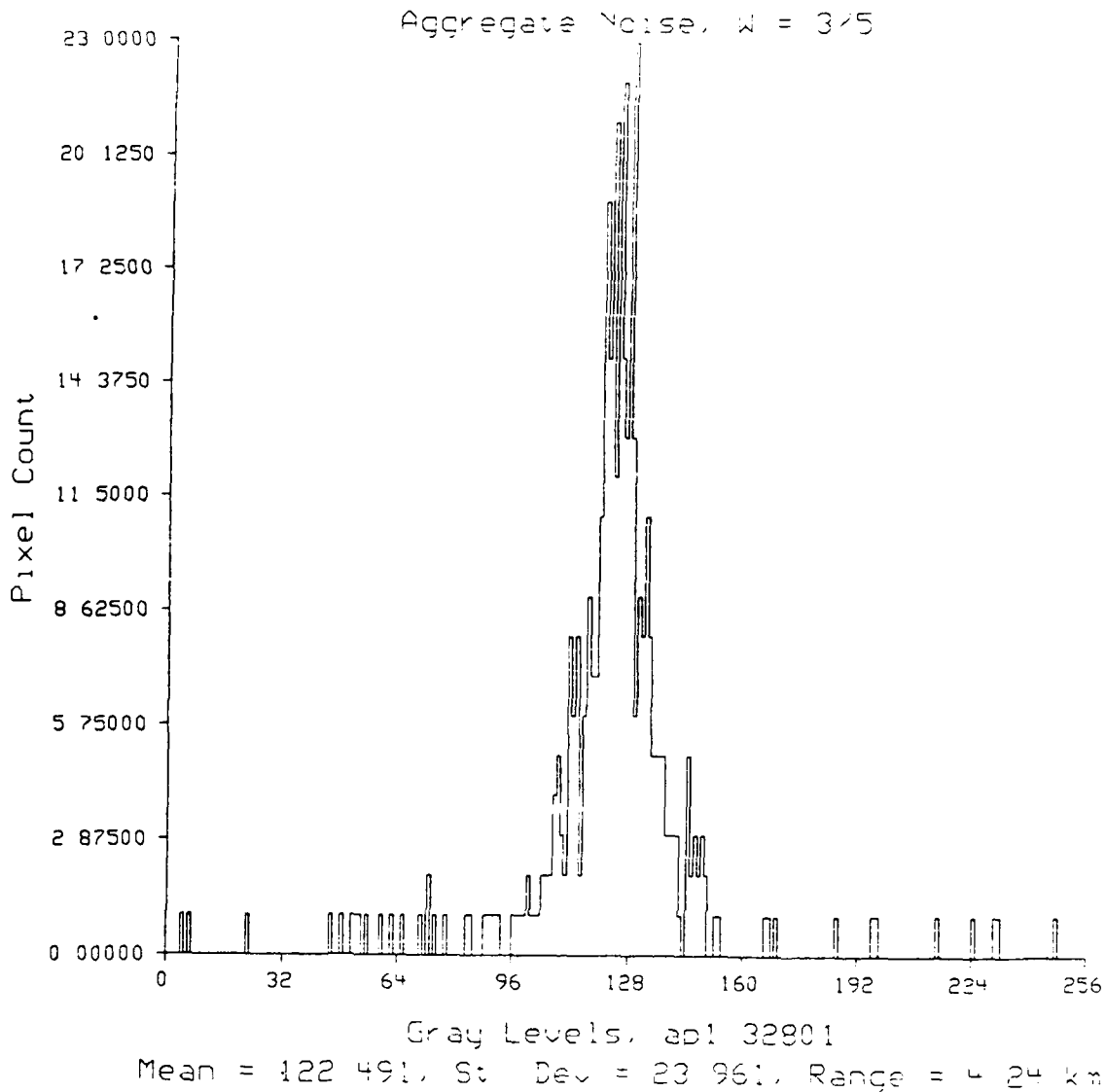


Figure 4.1.25 Aggregate gray level histogram of 375 pixel window. Mean = 122.491, Standard Deviation = 23.961.

First the separation process discussed above is applied to the images of each range measurement set. Next, the parameters of interest are calculated. Plots of this information for all three range sets are superimposed in Figures 4.1.26 and 4.1.27. Table 4.1.3 summarizes the results.

Table 4.1.3 Summary of drop-out probabilities and Gaussian standard deviation statistics for 1.54, 2.91, and 4.24 km. data sets.

Range	Drop-out Probability			Gaussian Standard Deviation		
(km)	Max.	Min.	Avg.	Max.	Min.	Avg.
1.54	0.102	0.031	0.073	12.237	9.388	11.199
2.91	0.032	0.028	0.030	10.816	8.777	9.797
4.21	0.099	0.005	0.027	17.199	7.001	10.353

There are some surprises in these results. For example, the only apparent variation with range is actually counterintuitive. The drop-out probability is consistently *higher* for the 1.54 km data set then for the other two data sets. The reason for this is unclear at present. Another interesting note is the wide variation in the data quality of the 4.24 km data set. The images corresponding to the highest and lowest points of the standard deviation plot for this data set are given in Figures 4.1.28 and 4.1.29.

4.1.3.1.3. Summary

There are many factors that could effect range data that have not been addressed here. The tuning of the sensor for the different range sets, or a difference in prevailing weather conditions might have a dramatic effect on the characteristics of the data. But, from the data presented, it appears that the parameters discussed do not vary with range by any appreciable amount.

Even with the unknown factors mentioned here, the results are still very useful in noise synthesis. The separation of noise into two types of distributions along with the averages and bounds of Table 4.1.3 will make it possible to realistically degrade synthetic range imagery.

4.1.3.2. Noise Degradation of Synthetic Imagery

As with the rest of the noise study presented so far, the primary parameters of interest are: the "Drop-Out Probability" (DOP), and the Gaussian Standard Deviation (GSD). A range of values for these parameters exists from the results presented in the previous section.

Again, by the ergodicity assumption, these parameters which have been determined through spatial averaging will be applied to synthetic imagery on a pixel by pixel basis to provide realistic noise degradation. For example, each *on target* pixel should represent either a

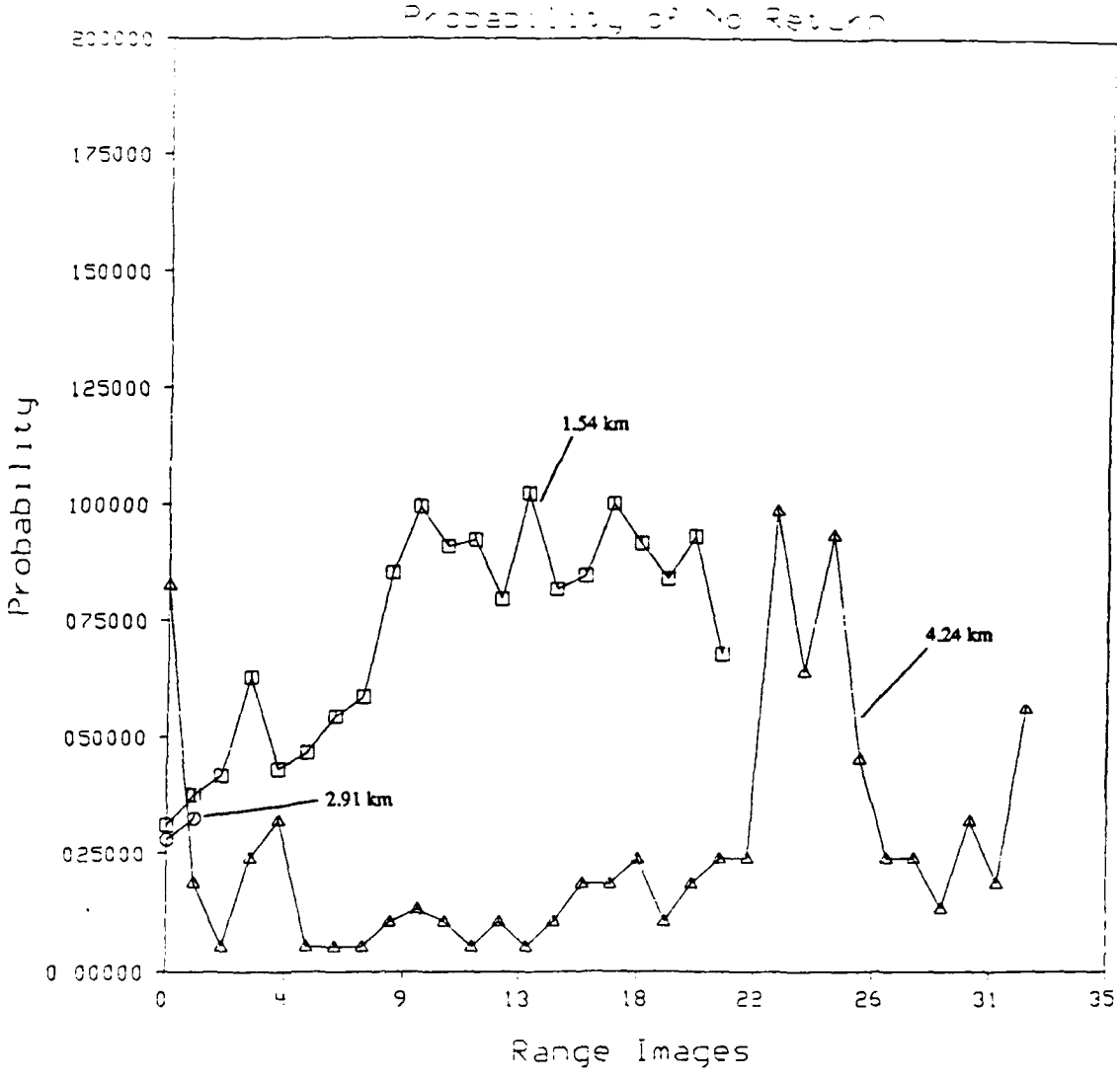


Figure 4.1.26 Composite graph of 1.54, 2.91, and 4.24 km. data. Shows image to image variation of drop-off probability.

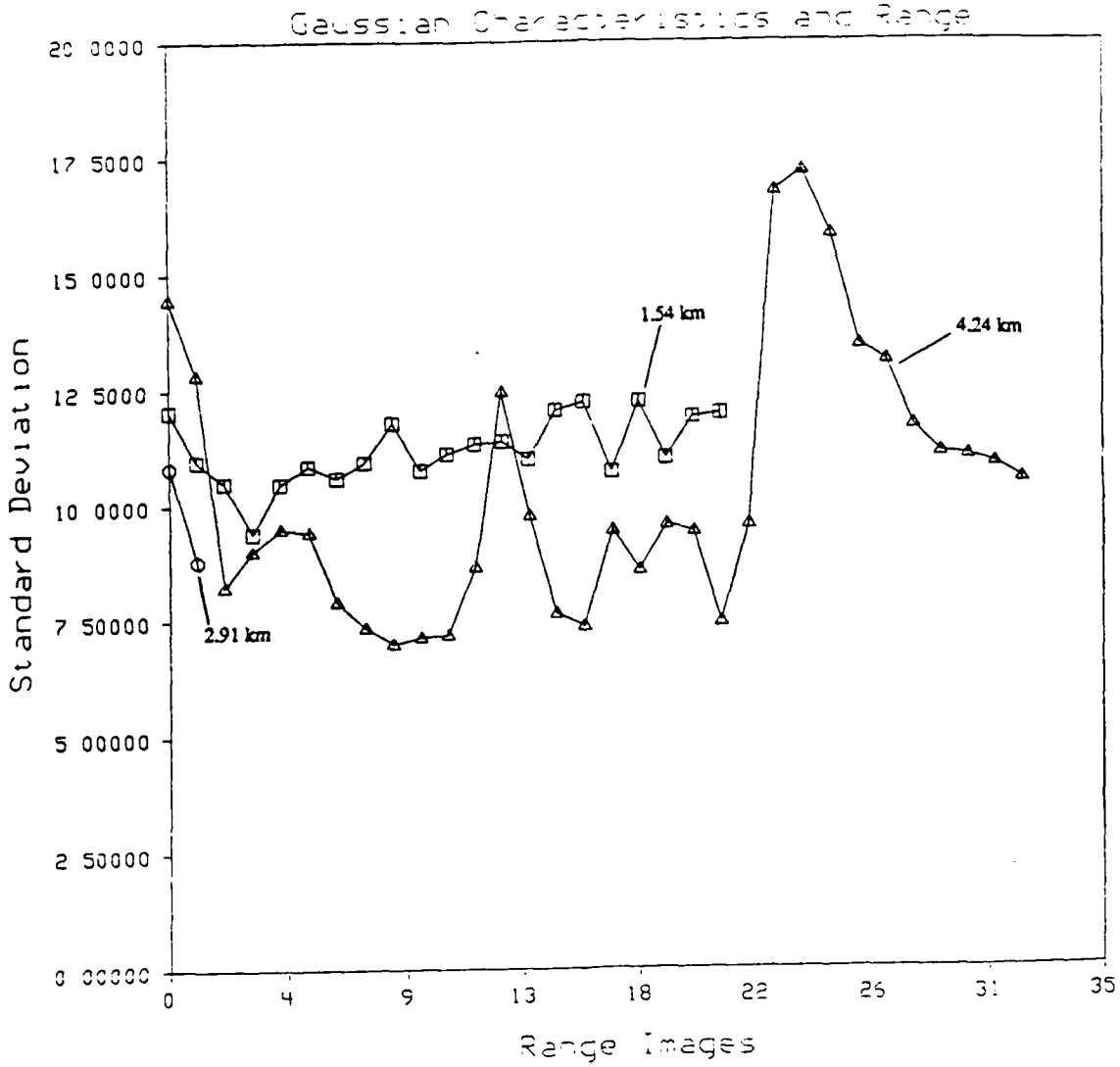


Figure 4.1.27 Composite graph of 1.54, 2.91, and 4.24 km. data. Shows image to image variation of Gaussian characteristics.

Range = 2.91 km, ap1.32828

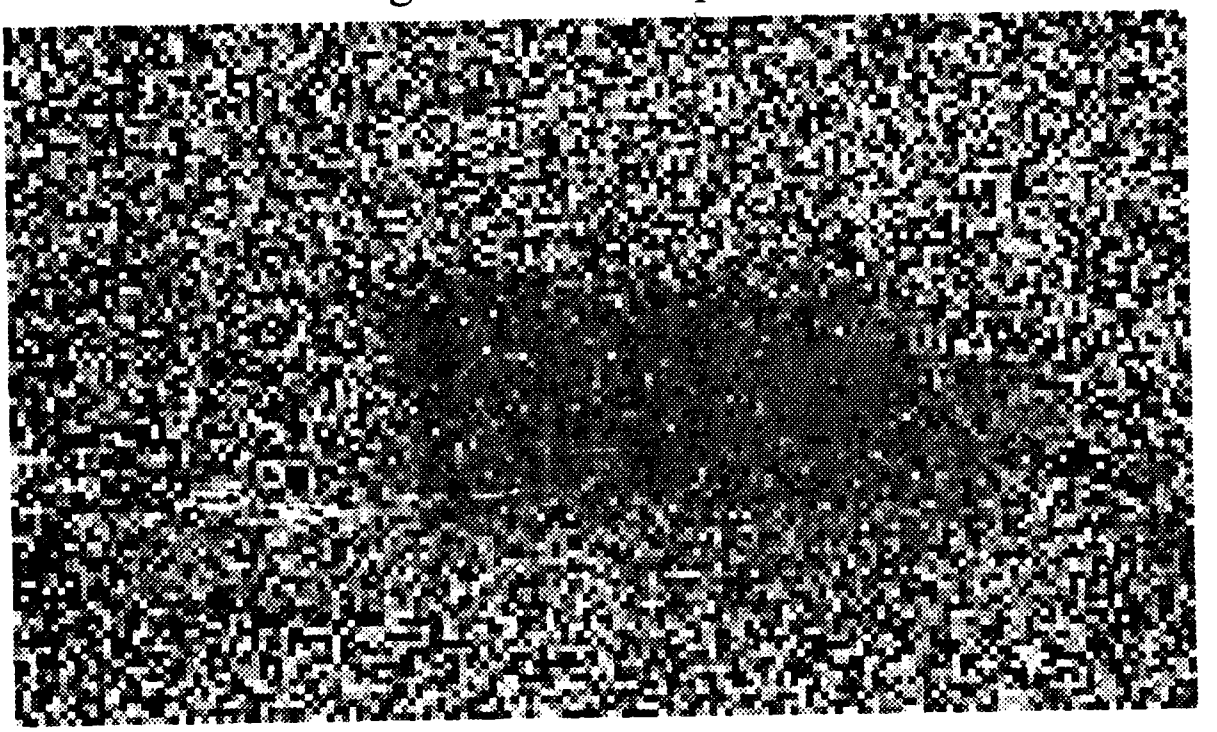


Figure 4.1.28 5 Ton Truck, Image 28, Date 3/28/86, Range 2.91 km. Image with the largest Gaussian standard deviation. Gaussian Standard Deviation = 17.199.

Range = 2.91 km, ap1.32813

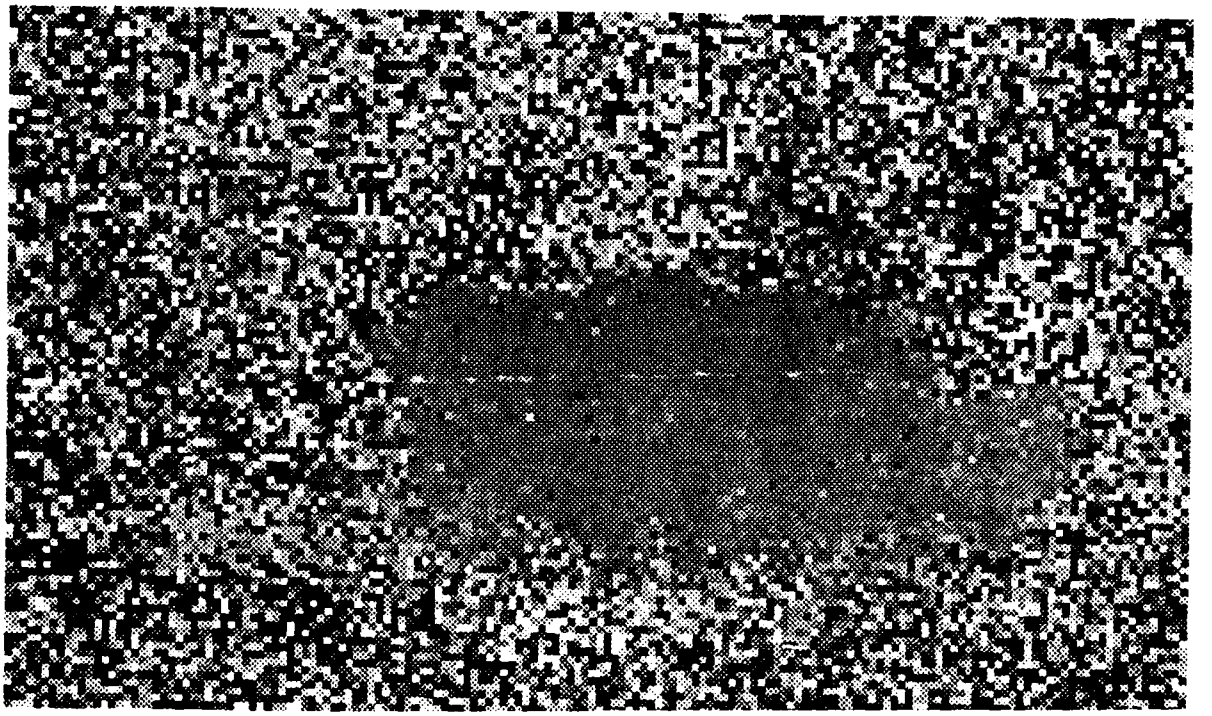


Figure 4.1.29 5 Ton Truck, Image 13, Date 3/28/86, Range 2.91 km. Image with the smallest Gaussian standard deviation. Gaussian Standard Deviation = 7.001.

drop-out measurement or a valid measurement perturbed by Gaussian noise. Thus, after the DOP, and GSD noise parameters are chosen, the synthetic degradation of a synthetic image proceeds as follows.

As each *on target* pixel is considered a random number between 0 and 1 is generated. If this number is less than or equal to the DOP parameter selected, the pixel of interest is replaced with a random gray level between 0 and 255. Otherwise, the pixel is additively corrupted with a Gaussian number of the proper variance. Figures 4.1.30-4.1.32 show examples of original and noise corrupted synthetic images at ranges of 0.5, 3, and 5 km.

Next, it is important to determine how well the synthetic and actual images compare. To perform this comparison the noise analysis developed above is applied to actual tank images, and the noise parameters extracted are used to degrade synthetic PADL imagery. An obvious problem here is the lack of large planar tank surfaces from which to extract a window of data. The windows used cover most of the tank base, and because of this, some of the noise is actually range variation of the tank surface. These variations are not readily discernible, however, and should not affect this visual comparison (see Figures 4.1.33 - 4.1.35).

This last comparison provides some other useful information as well. It provides an excellent test of the noise analysis technique which has been a major portion of this entire section on noise characteristics and their simulation. The noise parameters of the actual imagery are calculated for the indicated window and shown in part (a) of the figure captions. These parameters are then specified as the desired characteristics for the synthetic image. The resultant noisy synthetic image is then analyzed in the same way and its noise parameters appear in part (b) of the figure captions. Note how well the parameters agree.

4.1.3.3. Future Work

The first attempts at synthesizing LADAR images is very encouraging. The synthetic image *look* very much like the *real* images. One visual difference is that the edges of the synthetic target are sharper than the real target. Future work will attempt to measure the blurred edges in the real images and mimic it in the synthetic images. Another problem to be addressed is to verify that the synthetic images *look* the same to the image processing algorithms as the real images.

4.2. ETBM via TWIN

Section 4.1 presented our approach for building an **Electronic Terrain Board Model** (ETBM) using the PADL solid modeler. This section presents the additional capabilities we have gained by moving the modeler to the TWIN Solid modeling package and show some examples of the detailed images that can be produced by it. In addition, the efforts to convert the BRL models to TWIN are discussed.

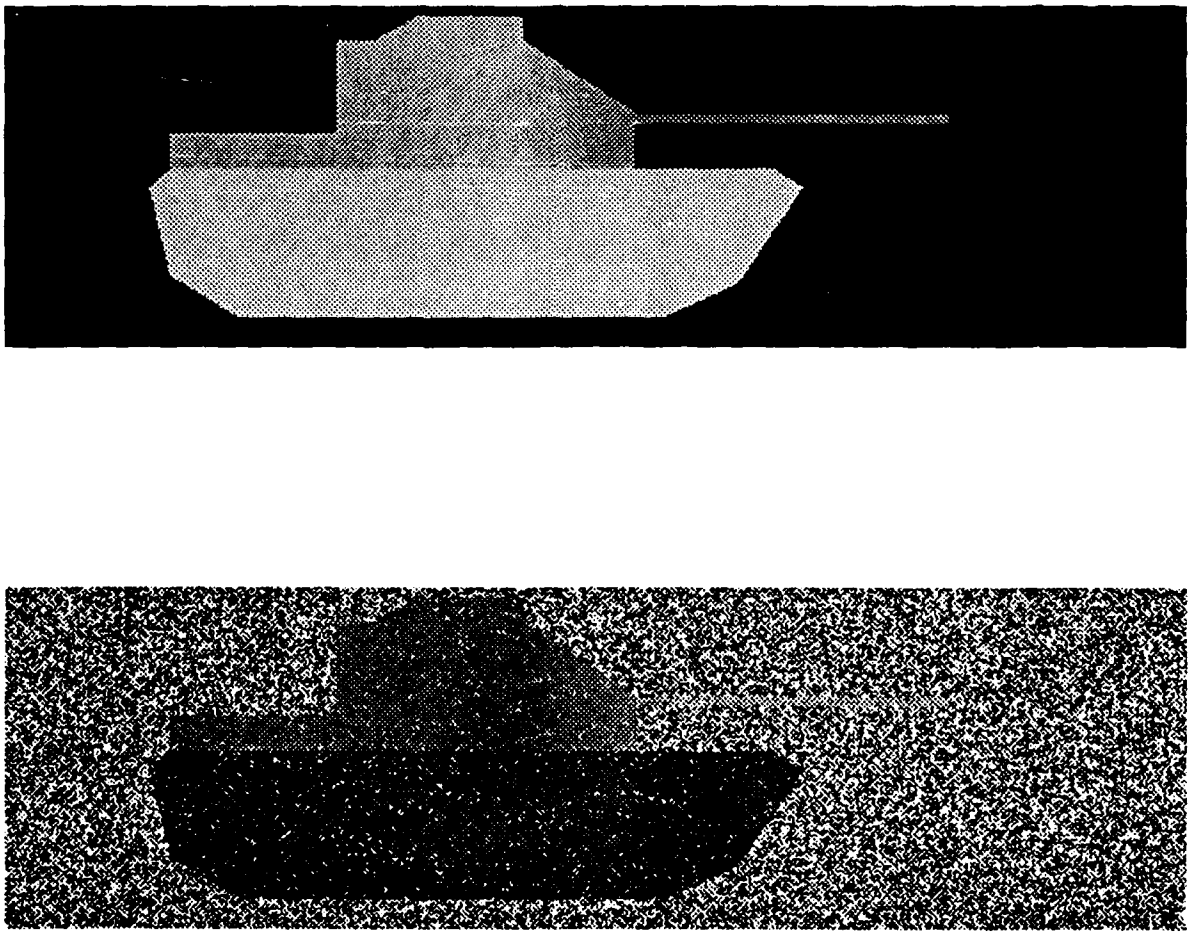


Figure 4.1.30 Clean and noisy images of synthetic PADL m60a1 tank. Range = 0.5 km., Probability of No Return = 0.05, Gaussian Standard Deviation = 12.0.

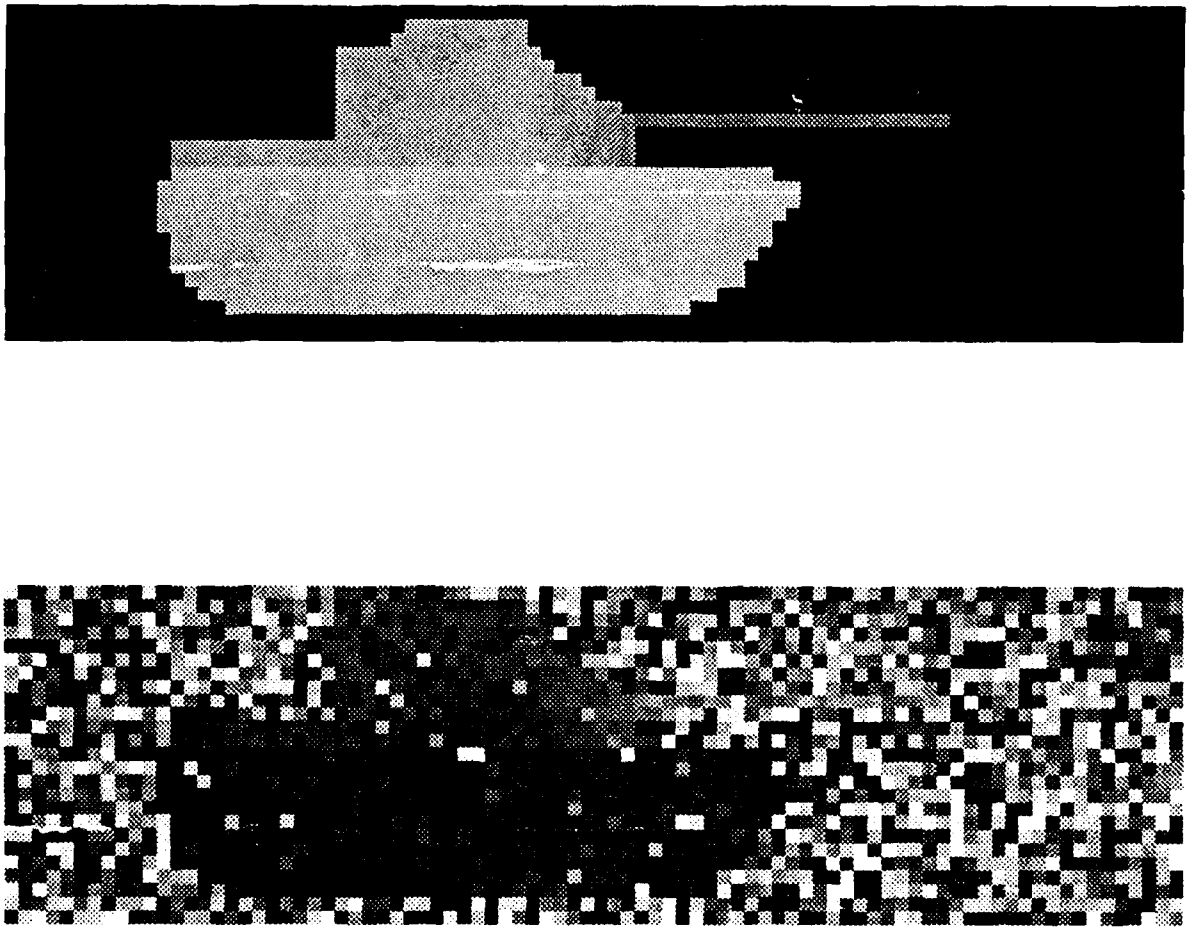


Figure 4.1.31 Clean and noisy images of synthetic PADL m60a1 tank. Range = 3.0 km., Probability of No Return = 0.05, Gaussian Standard Deviation = 12.0.

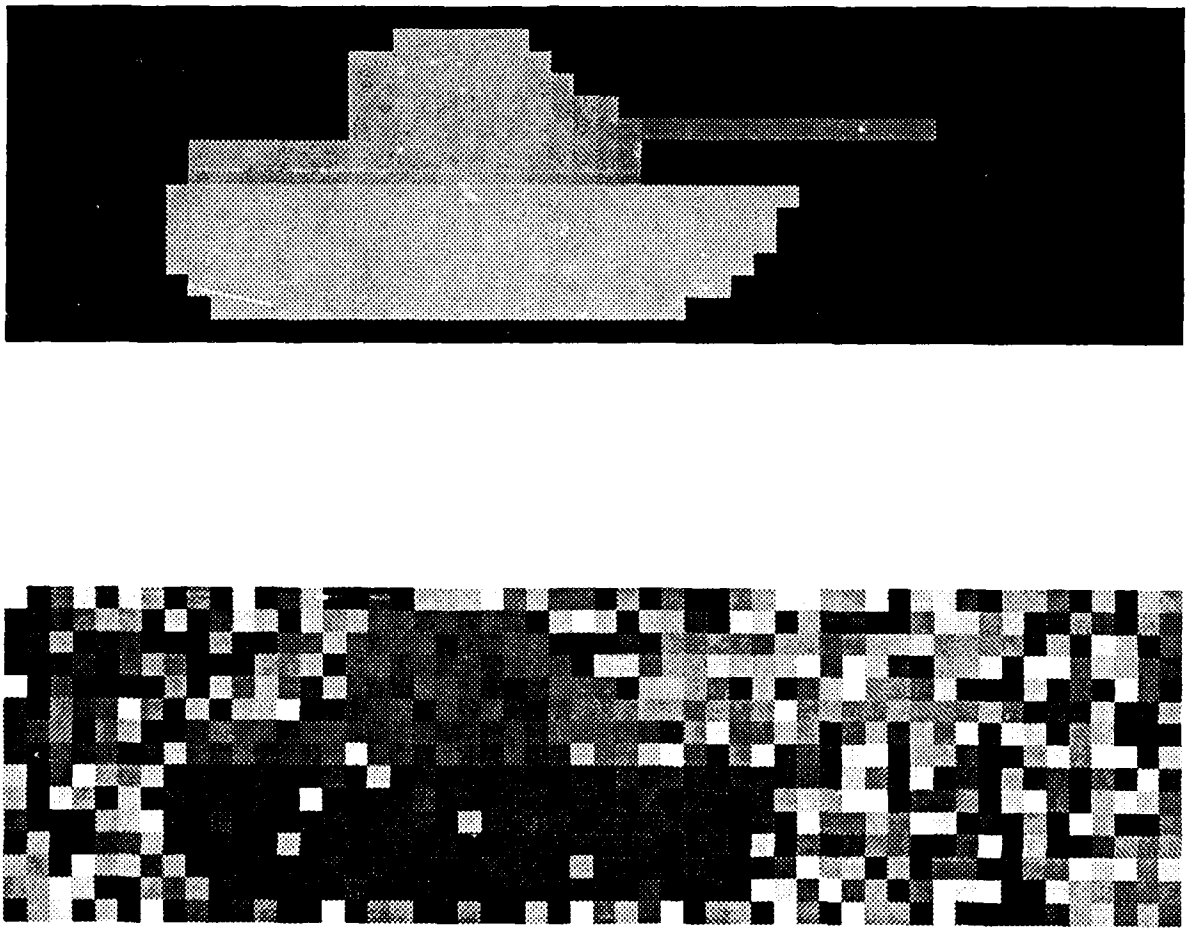
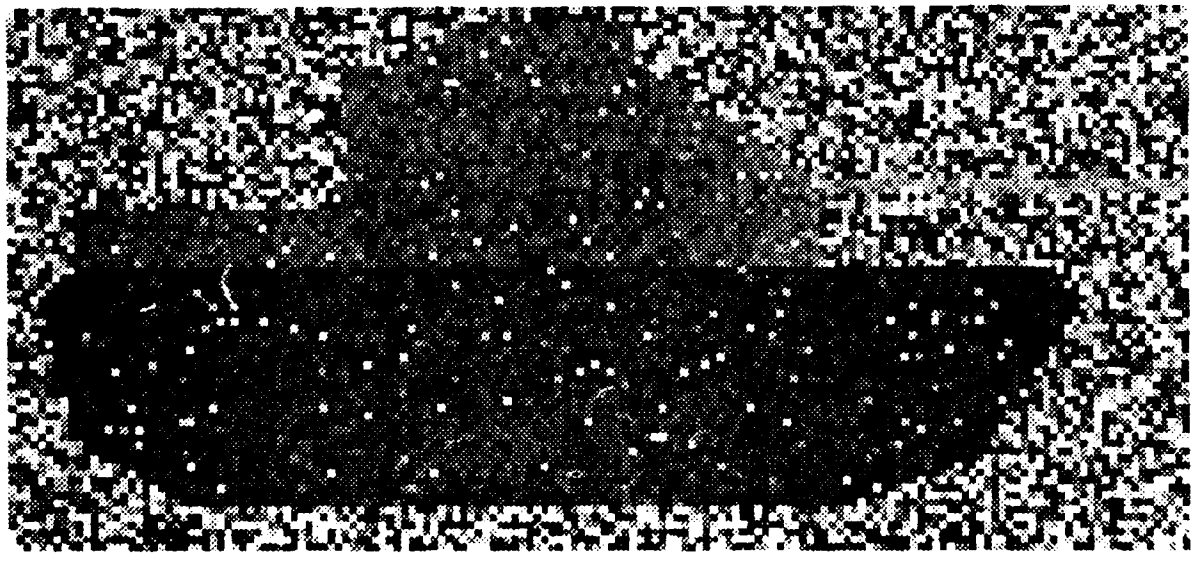


Figure 4.1.32 Clean and noisy images of synthetic PADL m60a1 tank. Range = 5.0 km., Probability of No Return = 0.05, Gaussian Standard Deviation = 12.0.

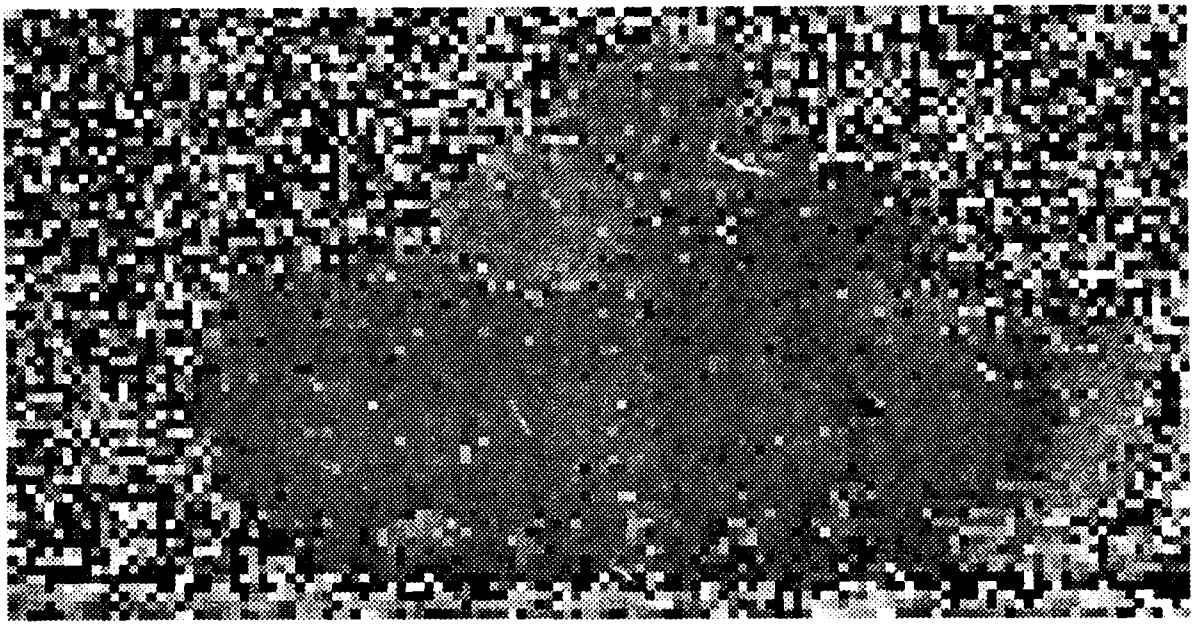


(a)

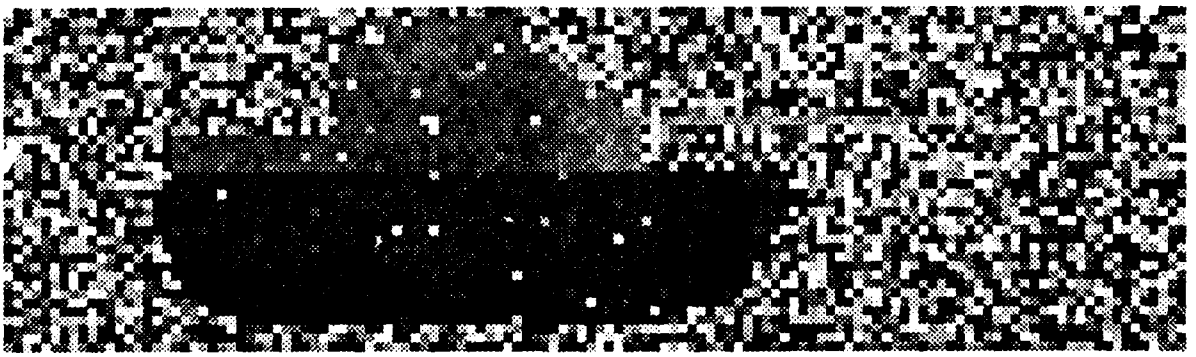


(b)

Figure 4.1.33 Actual vs. synthetic laser range imagery. (a) Actual: range = 1.19 km, Probability of No Return = 0.041, Gaussian Standard Deviation = 14.509. (b) Synthetic: range = 1.0 km Probability of No Return = 0.037, Gaussian Standard Deviation = 14.471.

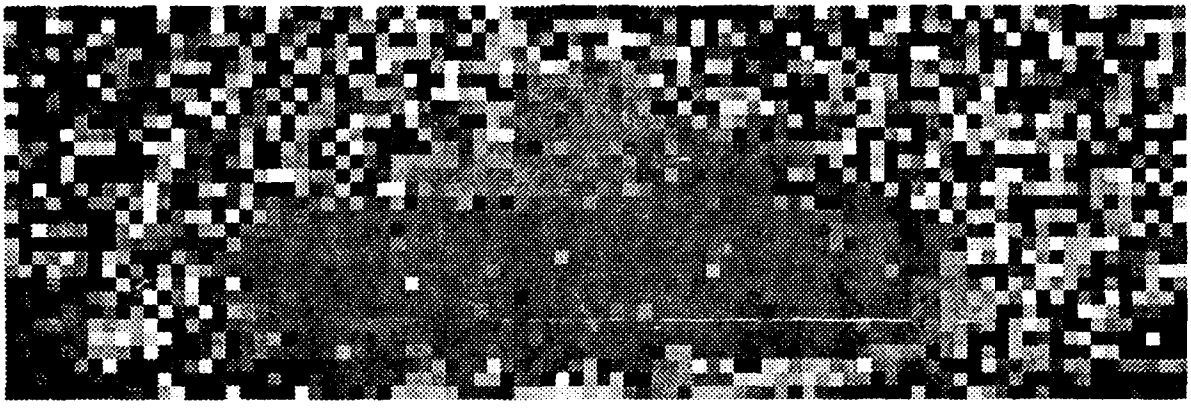


(a)

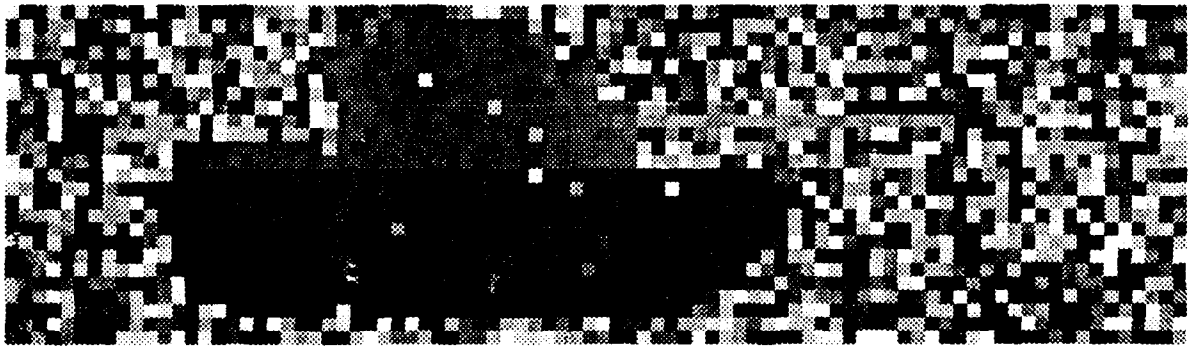


(b)

Figure 4.1.34 Actual vs. synthetic laser range imagery. (a) Actual: range = 1.935 km., Probability of No Return = 0.027, Gaussian Standard Deviation = 8.765. (b) Synthetic: range = 2.0 km, Probability of No Return = 0.019, Gaussian Standard Deviation = 8.719.



(a)



(b)

Figure 4.1.35 Actual vs. synthetic laser range imagery. (a) Actual: range = 2.91 km, Probability of No Return = 0.018, Gaussian Standard Deviation = 6.576. (b) Synthetic: range = 3.0 km, Probability of No Return = 0.018, Gaussian Standard Deviation = 6.426.

4.2.1. The TWIN Solid Modeling Package

The **Electronic Terrain Board Model**, presented in Section 4.1, was built upon the PADL [Padl] solid modeling system. Synthetic range images of targets, terrain, and clutter were all produced using PADL. However, PADL was designed to model industrial parts and not terrain boards, and therefore is unable to handle the large number of objects needed to synthesize a complicated scene. We have switched to the **TWIN Solid Modeling Package** to overcome the size limitation problems and gain more flexibility in the modeler. By doing so we have gained a package that can serve both as a terrain board modeler and a data structure for model based target recognition. TWIN is a boundary representation solid modeler which supports planer surfaces and is available from the Purdue CADLAB [CAD] as a library of C routines. Appendix D is a partial listing of the routines in the package. By interfacing these TWIN routines to our Lisp image processing environment, planer boundary representation (BRep) objects can be created from the same types of primitives as used by PADL, that is, the CSG (constructive solid geometry) primitives. Therefore most models used in PADL can be easily converted to TWIN models. The next paragraph describes how an M60A1 TWIN model is built.

Figure 4.2.1 is a partial list of the points, edges, and surfaces (from ERIM) which describe the M60A1 tank. (Figure 4.2.1 is the same as Table 4.1.1.) The Lisp data for Figure 4.2.1 is shown in Figure 4.2.2. Each symbol, *p#*, is a point which is a list of X, Y, Z coordinates (in meters). Due to large number of points (and edges and surfaces) in the M60A1 model, only the first few points (or edges or surfaces) are show in the figure. Each symbol, *e#*, is an edge which is an ordered list of two points. Each symbol *s#* is a surface which is bounded by the edges in the list. Normal edges are traversed from the first point to the second. Negative edges are traversed from the second point to the first. Finally, *m60a1-all* a list of all the surfaces which bound the M60A1 model. Figure 4.2.3 is program for creating an M60A1 TWIN object given the *m60a1-all* data. The first **setq** calls the function **object-to-twin** which converts the object described by **m60a1-all** to a TWIN object. The second **setq** defines the main gun of the tank as a cylinder which is a primitive TWIN object. The **Cylinder** routine is given the X, Y, Z location of the base of the cylinder, the X, Y, Z direction of the axis of the cylinder, the radius of the cylinder, and finally the number of facets to use to approximate the cylinder. The size and location of the cylinder are described symbolically in terms of the points defined in the ERIM wireframe model. (i.e. *p191*, *p186*, and *p189* are all points from the M60A1 model.) Once the body and the main gun are defined, they are unioned together into one object using the **Combine** function. Therefore the object returned by **make-m60a1** is the union of the **body** and the **gun**. This code is much simpler than the code needed to describe the same tank in PADL. (See Figure 4.1.2). Once the TWIN object is defined, a synthetic image can be defined from Lisp by entering:

```
(render m60a1 :ifov 0.05 :range 1.0 :size '(200 100))
```

which will generate a 200 by 100 pixel range image of an M60A1 at one kilometer with a resolution 0.05 mrad. Figure 4.2.4 is the output of the render program.

Point File				Edge File			Surface File	
Point	x	y	z	Edge	Point1	Point2	Surface	Edges
1.	3.15	1.81	1.6	1.	1	3	1.	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2.	3.15	-1.81	1.6	2.	3	9	2.	19 -18 20 21
3.	-3.25	1.81	1.6	3.	9	15	3.	22 -14 23 24
4.	-3.25	-1.81	1.6	4.	15	17	4.	25 26 -20 -17
5.	-3.47	1.81	1.4	5.	17	21	5.	27 -15 -22 28
6.	-3.47	-1.81	1.4	6.	21	25	6.	29 -25 -16 -27
7.	-3.47	1.10	1.4	7.	25	26	7.	30 -21 -26 31 32
8.	-3.47	-1.10	1.4	8.	26	22	8.	33 -32 34 35
9.	-3.25	1.10	1.6	9.	22	18	9.	-24 36 37 38 -28
10.	-3.25	-1.10	1.6	10.	18	16	10.	-37 39 40 41
11.	-3.25	1.10	2.0	11.	16	10	11.	42 -31 -29 -38
12.	-3.25	-1.10	2.0	12.	10	4	12.	-2 43 44 45
13.	-1.5	1.10	2.0	13.	4	2	13.	-12 46 47 48
14.	-1.5	-1.10	2.0	14.	2	32	14.	-44 49 50 51
15.	-1.5	1.10	1.6	15.	32	30	15.	-50 52 53 54
16.	-1.5	-1.10	1.6	16.	30	29	16.	-47 55 56 57
17.	-1.5	1.25	1.6	17.	29	31	17.	-56 58 59 6
18.	-1.5	-1.25	1.6	18.	31	1	18.	161 62 65 63 66 64
19.	-1.5	1.05	3.0	19.	33	1	19.	-45 -51 -230 -65
20.	-1.5	-1.05	3.0	20.	31	35	20.	-55 -46 -66 -231
21.	0.5	1.43	1.6	21.	35	33	21.	-62 67 68 -3
22.	0.5	-1.43	1.6	22.	36	32	22.	-67 -61 69 78
23.	0.5	1.23	3.0	23.	2	34	23.	-69 -64 -11 71
24.	0.5	-1.23	3.0	24.	34	36	24.	72 73 -4 -68 -70 -71 -10 74
25.	1.7	0.55	1.6	25.	29	37	25.	-74 -9 75 76 77
26.	1.7	-0.55	1.6	26.	37	35	26.	-75 -8 78 79
27.	1.7	0.485	2.2	27.	38	30	27.	-78 -7 80 81
28.	1.7	-0.485	2.2	28.	36	38	28.	-80 -6 82 83
29.	2.0	1.10	1.6	29.	38	37	29.	-82 -5 -73 84
30.	2.0	-1.10	1.6	30.	39	33	30.	-72 -77 -76 85 86 -84
...				...			...	

Figure 4.2.1 Partial listing of points, edges, and surfaces for the wireframe model on an M60A1. (All dimensions are in meters.)

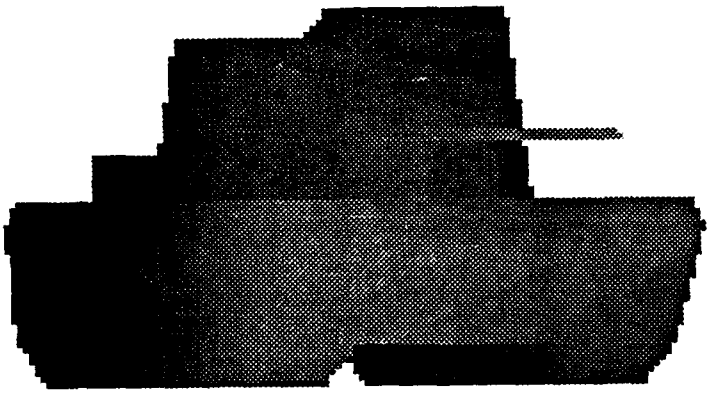
```
;;;       M60 POINT FILE
;;;
;;;                  X      Y      Z      (in meters)
(setq p1        '( 3.15  1.81   1.6))
(setq p2        '( 3.15 -1.81   1.6))
(setq p3        '(-3.25  1.81   1.6))
(setq p4        '(-3.25 -1.81   1.6))
(setq p5        '(-3.47  1.81   1.4))
...

;;;  M60 EDGE FILE
;;;
::;             point1  point2
(setq e1        '(p1    p3))
(setq e2        '(p3    p9))
(setq e3        '(p9    p15))
(setq e4        '(p15   p17))
(setq e5        '(p17   p21))
...
;;; M60 SURFACE FILE
;;;
;;;             Edge numbers
(setq s1        '(e1 e2 e3 e4 e5 e6 e7 e8 e9
                     e10 e11 e12 e13 e14 e15 e16 e17 e18))  ; HULL TOP
(setq s2        '( e19 e-18 e20 e21))           ; LEFT FENDER TOP
(setq s3        '( e22 e-14 e23 e24))           ; RICHT FENDER TOP
(setq s4        '( e25 e26 e-20 e-17))          ; LEFT INSIDE FENDER
(setq s5        '( e27 e-15 e-22 e28))          ; RIGHT INSIDE FENDER
...
;;; The following describes the surfaces that bound an m60a1.
;;; These are the same surfaces described in the m60a1 wireframe data
;;; from ERIM.
(setq m60a1-all '(s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14
                s15 s16 s17 s18 s19 s20 s21 s22 s23 s37 s38 s39 s40
                s65 s66 s67 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33
                s34 s35 s36))
```

Figure 4.2.2 LISP data for the M60A1 model described in Figure 4.2.1.

```
;;; This will create an m60a1 as a TWIN object.
(defun make-m60a1 ()
  (setq body (object-to-twin 'm60a1-all))
  (setq gun  (Cylinder
              (x p191)                      ; x location of base
              (/ (+ (y p191) (y p186)) 2.0) ; y location of base
              (/ (+ (z p191) (z p186)) 2.0) ; z location of base
              (- (x p189) (x p191))         ; x Direction of cylinder
              0.0                           ; y Direction of cylinder
              0.0                           ; z Direction of cylinder
              (/ (- (z p187) (z p191)) 2d0) ; Radius
              8)))                          ; Number of facets to use.
    (Combine body #+ gun)))
```

Figure 4.2.3 LISP program to generate an M60A1 TWIN tank model.



200 by 100

Figure 4.2.4 Rendered M60a1.

The TWIN Solid Modeling Package has proven to be much easier to work with and much faster than the PADL modeler. It is easier to work with because all the routines are written in C, unlike PADL which is written in a non-standard dialect of FORTRAN. This makes the TWIN code easier to read and understand than PADL code. TWIN is faster for a number of reasons. One, we are able to control the size of the images being rendered. i.e. if a 511 by 256 image is needed, that is the size we render. PADL on the other hand was designed to render an image the size of the display device it is using, therefore if a 511 by 256 image is wanted, a 512 by 480 image must be rendered, and then trimmed to the desired size. Reason two is that TWIN approximates non-planar objects such a spheres and ellipses as being planar. This allows a simple scan line rendering algorithm to be used. PADL does not approximate non-planar object and uses a more time consuming ray tracing algorithm to render. Although there may be cases where approximations cannot be used, our application can use them. Finally, TWIN can "compile" CSG primitives into a complex object once and save that object. Subsequent uses of the object do not require recompiling. PADL, one the other hand, has no method for saving the resulting compilation of CSG primitives. Instead, it must rebuild the object each time PADL is restarted. Some of the BRL objects have as many as 5,000 primitives. Substantial times savings will be realized if such an object can be compiled only once and the resulting boundary representation used over and over again.

Section 3.4.2 of this report has shown how geometric features are readily extracted from the TWIN modeler. The following section describes how a LADAR sensor is simulated.

4.2.2. The Electronic Field Test

The four basic elements needed to run an electronic field test are the terrain, the targets, clutter, and sensor noise. This section describes how, using our TWIN based modeler, these four elements are combined. The actual steps are:

Start with a 2D elevation array
Create a terrain patch
Select the targets for the image
Place the targets on the patch
Render the image
Generate ground truth data
Convert to 32 bit integer
Create range ambiguities
Blur the image
Add noise

Each of the following sections discusses each step.

4.2.2.1. Create a 2D Elevation Array

The first step in an electronic field test is to design the terrain for the test. This is done by creating a two dimensional array of elevations over an evenly spaced grid. If "real" terrain data is available, it can be used, otherwise fractal based methods can be used to generate the data. Here is the LISP code to make such an array:

```
(setq elevation (make-array '(17 17))
```

This creates the storage needed for the elevation array. The following data was generated using the fractal techniques presented in Section 4.1,2.2.

```
((48 10 48 50 55 53 54 48 46 10 48 57 54 50 54 52 46)
 (46 48 10 52 53 53 56 52 52 45 57 54 48 46 10 55 56)
 (48 46 49 10 55 48 48 49 49 48 48 51 53 52 49 51 51)
 (48 54 46 51 10 53 57 57 50 46 10 56 57 57 49 54 50)
 (48 56 54 46 53 10 [illegible] 49 57 51 10 50 50 46 10 54)
 (48 50 55 49 46 54 10 53 54 51 46 56 10 50 51 49 48)
 (48 49 51 51 49 46 55 10 51 51 57 46 49 54 54 51 48)
 (48 53 48 54 51 48 46 56 10 53 49 49 48 54 48 46 10)
 (10 51 48 55 48 51 53 46 49 57 56 50 46 10 49 52 49)
 (48 10 54 56 49 55 53 53 48 57 54 55 52 49 54 49 52)
 (46 49 10 49 51 54 48 57 46 10 48 52 49 49 48 54 46)
 (51 46 50 10 56 57 54 53 48 45 49 53 56 46 10 57 51)
 (52 50 46 52 10 56 57 54 48 48 10 51 56 54 49 53 56)
 (57 56 52 46 53 10 49 52 48 46 52 10 48 49 49 56 48)
 (50 51 53 48 46 54 10 54 48 56 46 57 57 53 46 10 50)
 (52 52 50 54 52 46 56 10 48 53 55 46 10 56 51 49 49)
 (53 53 55 56 56 55 46 57 48 54 53 53 49 53 51 52 53))
```

4.2.2.2. Create Terrain Patch

The next step is to create the terrain patch from the elevation data. This is done with the following LISP command:

```
(setq patch (create-patch elevation
                          :extent '((-25 -25 -1)
                                    ( 25  25  0))))
```

The `:extent` keyword is used to give the size of terrain patch. The above command sets the minimum x and y values to −25 meters and the maximum x and y values to 25 meters, thus creating a 50 by 50 meter patch. The minimum z value is −1 meter and the maximum is 0 meters. Changing the relative extent of the z value affects the "ruggedness" of the patch. Figure 4.2.5 is a rendering of the above patch.

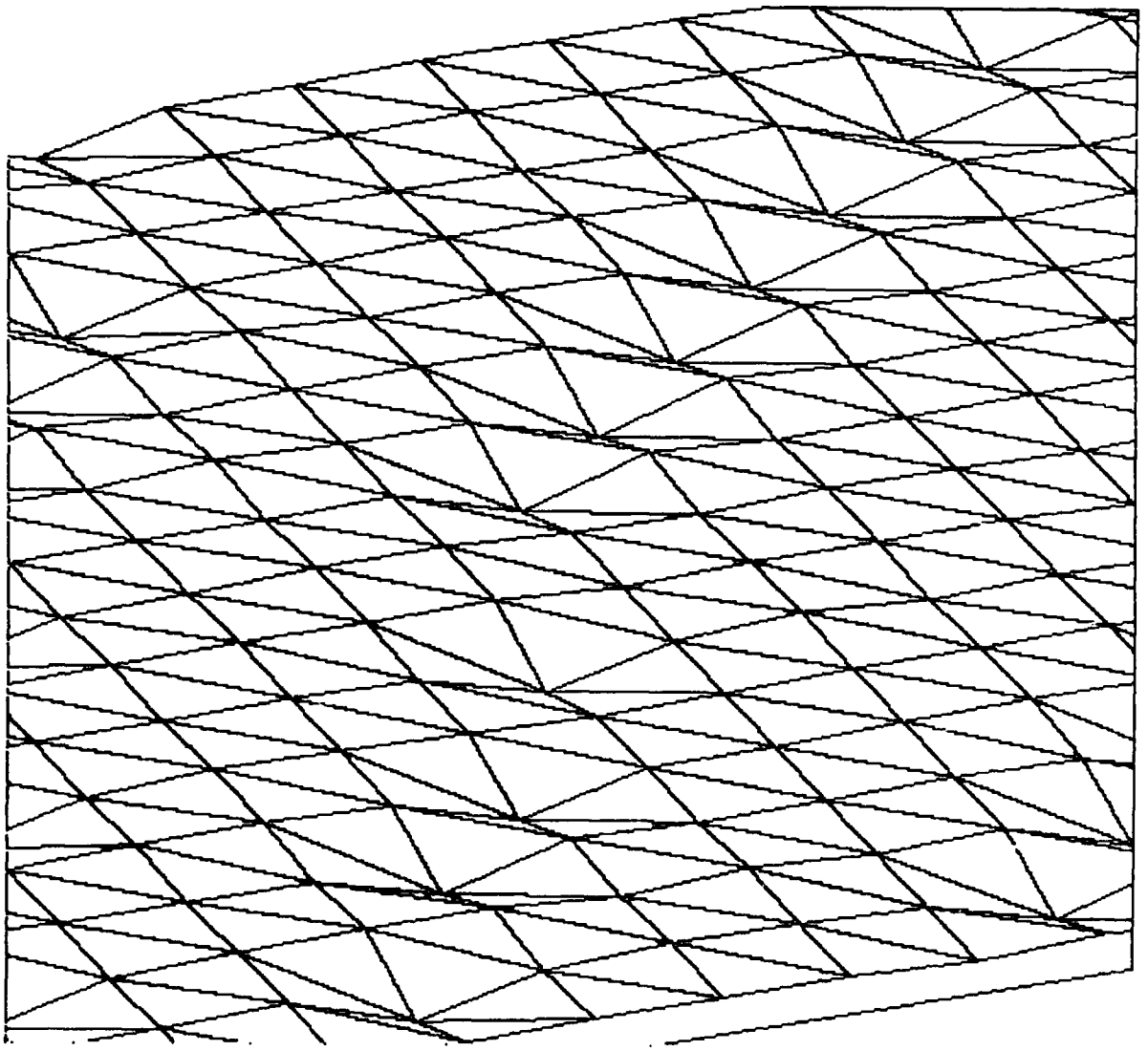


Figure 4.2.5 Sample terrain patch created with fractals.

4.2.2.3. Select Targets

The next step is to select the targets to be used in the field test. Figure 4.2.6 shows the four targets currently available. A target can appear any number of times in a scene and any orientation. Section 4.2.3 discusses our efforts to convert the BRL targets to TWIN which will increase the selection.

4.2.2.4. Place targets on patch

The next step is to describe the location and orientation of each of the targets on the patch. The following code places five targets on the patch:

```
(setq tanks (place-objects-on-patch patch
                        '((m60a1 0.0 (0 0))
                          (m60a1 90.0 (10 10))
                          (m35   180  (-6 6))
                          (m35   270  (-10 -10))
                          (m113  35   (6 -6)))))
```

The targets are places in the following locations:

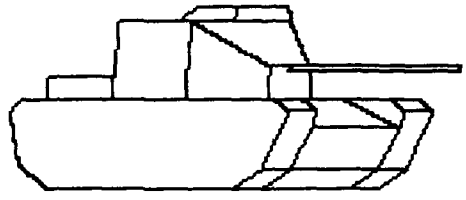
Target	Rotation about Z (in degrees)	Location (in meters)	
M60A1	0.0	0	0
M60A1	90.0	10	10
M35	180.0	-6	-6
M35	270.0	-10	-10
M113	35.0	6	-6

Knowledge about the elevations of the given locations on the patch are used to automatically select the correct Z elevation.

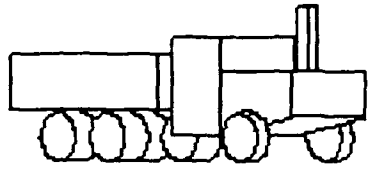
4.2.2.5. Render

Rendering the above scene with the following command will create four images: range, shading, edges, and faces.

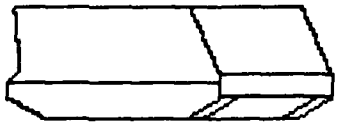
```
(setq images (render tanks
                        :shading "all"
                        :camfrom '( 10   4 12)
                        :camto   '(-10  -4  8)
                        :range 1.0
                        :size '(512 480)))
```



200by150



200by150



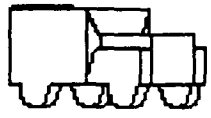


Figure 4.2.6 Target models available in TWIN modeling system.

The rendering routine can render the scene from any viewpoint (`:camfrom`) to and viewpoint (`:camto`) for any given range and sensor resolution (`:range :resolution`). Figures 4.2.7 - 4.2.10 show the resulting images.

4.2.2.6. Display Ground Truth

The next step is the display the ground truth table. The ground truth table lists every surface in the scene and its location and orientation relative the the viewing plane. The following command prints the table:

```
(print-ground-truth tanks)
```

Figure 4.2.11 is a partial listing of the ground truth of the sample scene.

4.2.2.7. Convert to 32 bit integer

Up to this point, the pixels in the range images are floating point values which are the distance from the Z axis in the model space. The following routine converts these values to 32 bit integers at the requested distance. Since the units in the model space are meters and a scale of 100 is used, the result units for the range image is centimeters.

```
(setq range (range-to-int (first images)
                          :offset 1000
                          :scale   100))
```

4.2.2.8. Create range ambiguities

As presented in previous reports, taking the mod of each pixel 1875 will create range ambiguities in the image which simulate the ambiguities of the fine range channel of the sensor.

```
(setq range (mod-image range 1875))
```

This is shown in Figure 4.2.12.

4.2.2.9. Blur the image

The following code will blur the image by replacing each pixel with the average of the pixels around it. Although this does blur the image, it most likely does not simulate the blurring of a real LADAR sensor. More information is needed about the sensor before its blurring can be mimiced.

```
(setq range (blur-image range))
```

This is shown in Figure 4.2.13.

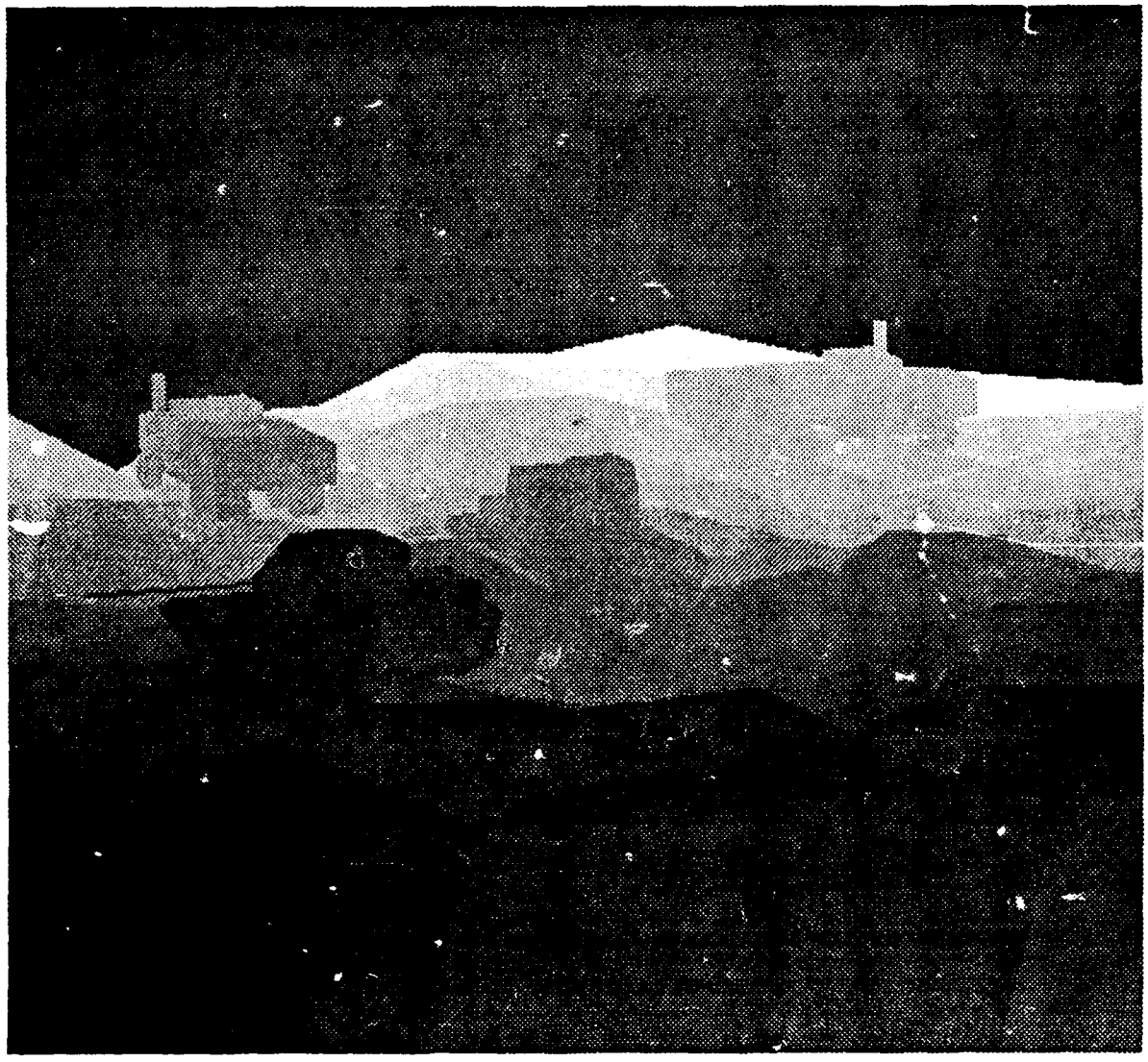


Figure 4.2.7 Range image of scene created in Section 4.2.2.5.

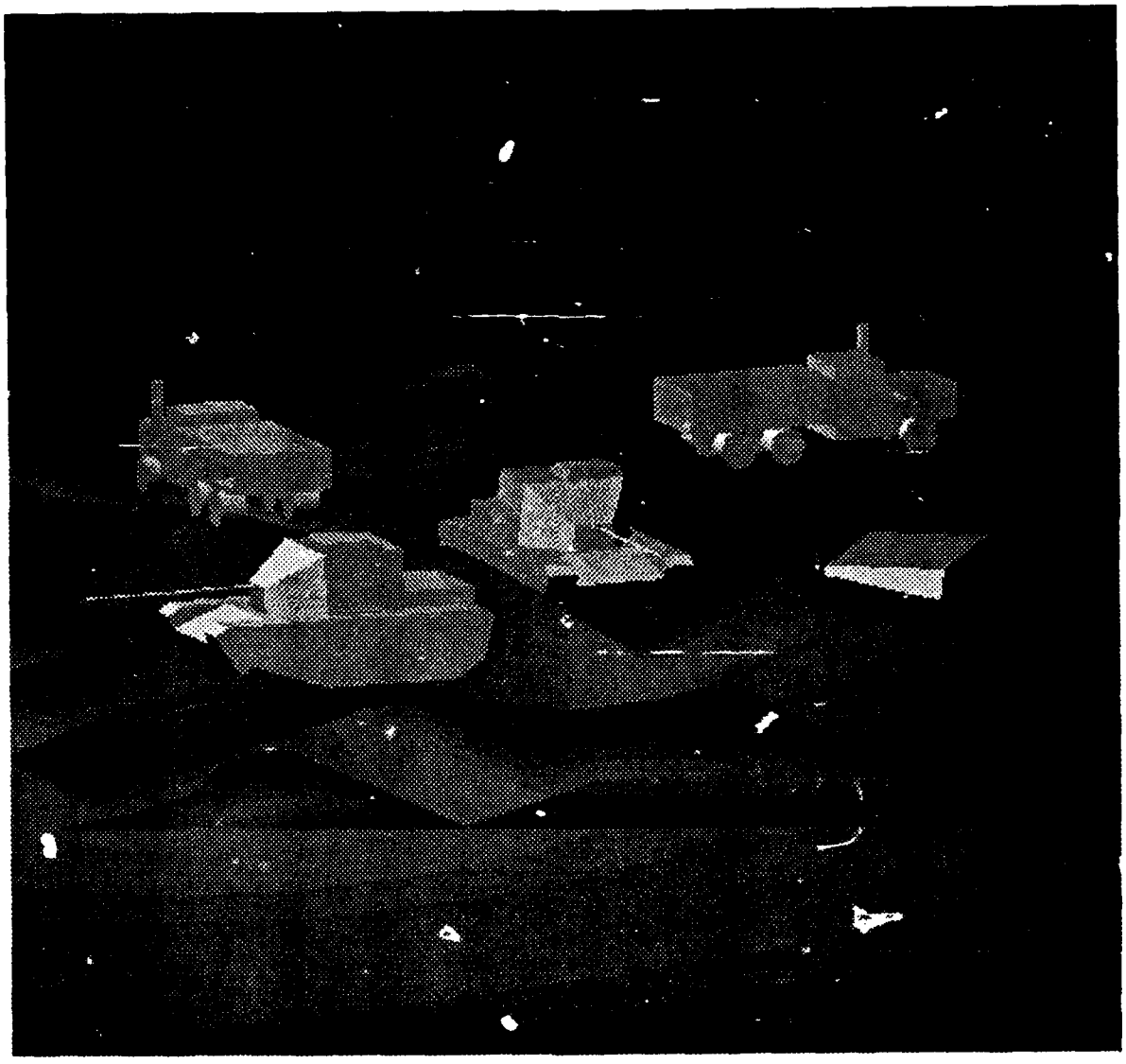


Figure 4.2.8 Shaded image of scene created in Section 4.2.2.5.

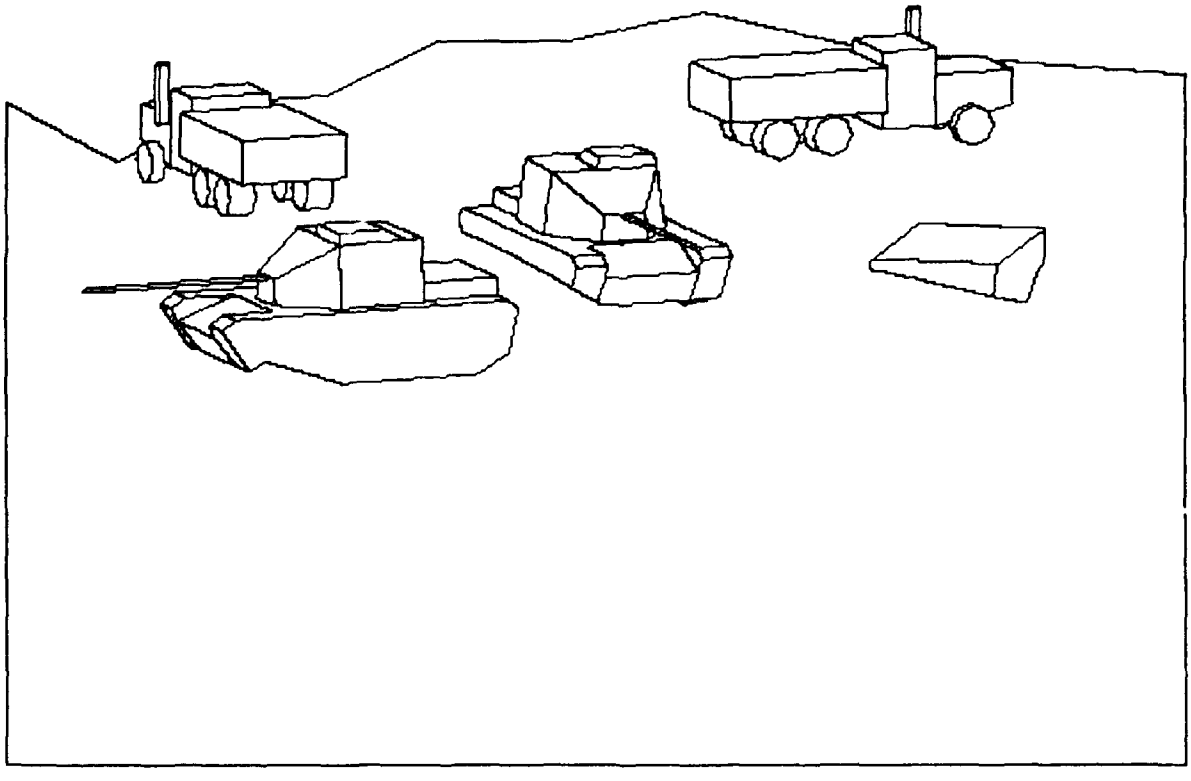


Figure 4.2.9 Edge image of scene created in Section 4.2.2.5.

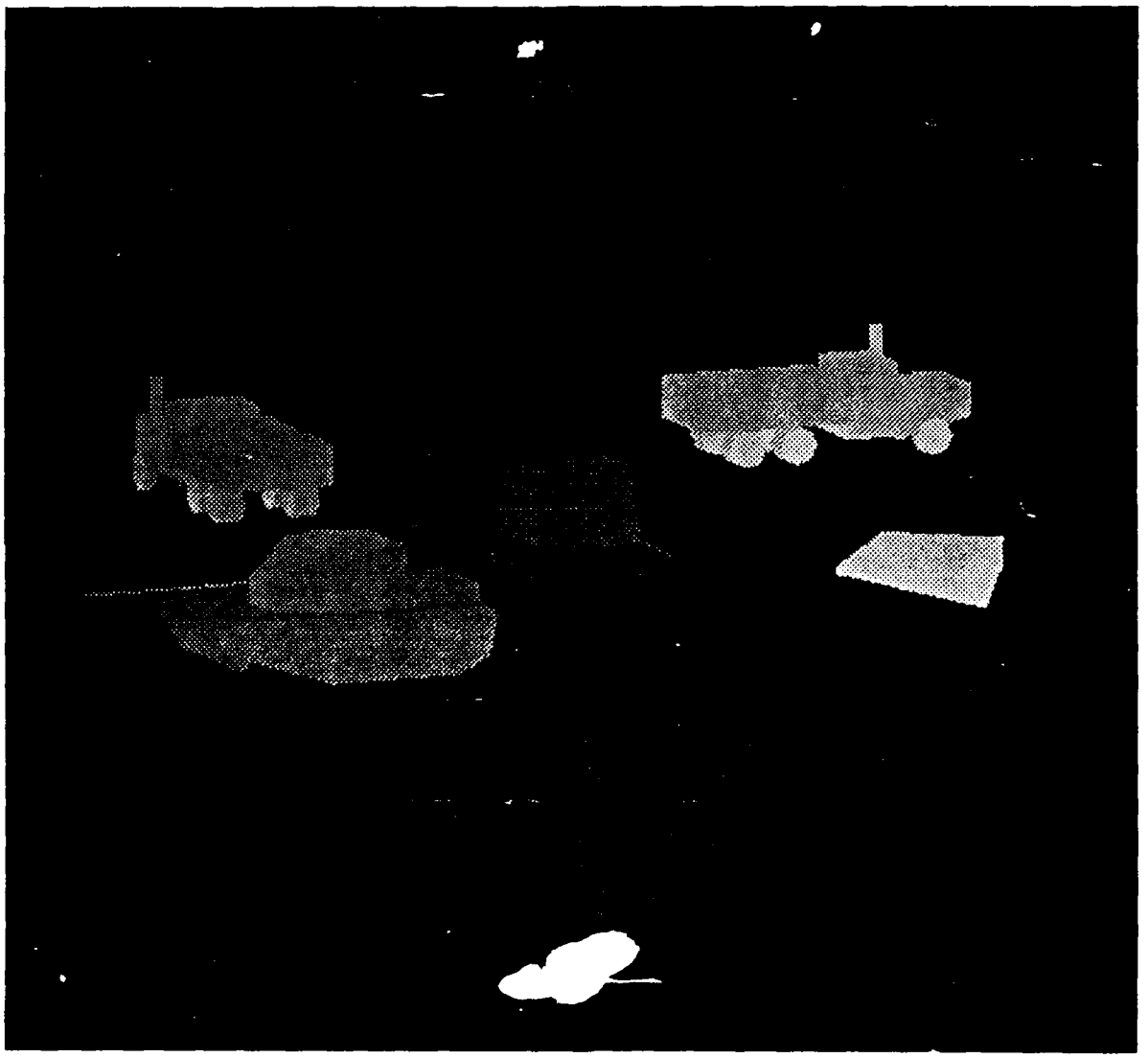


Figure 4.2.10 Faces image of scene created in Section 4.2.2.5. The grey values in the faces image index into the ground truth table.

group	Area	min-ext	max-ext	centroid	normal
2	295.70	(198.25,228.54,-14.60)	(313.01,255.14,-7.44)	(256.8253 242.3849 -10.871665)	(-0.00,-0.98, 0.18)
3	46.88	(245.79,254.18,-7.70)	(261.35,260.16,-7.18)	(253.5674 257.1697 -7.43846)	(0.20,-0.74, 0.64)
4	46.88	(299.82,250.23,-8.76)	(315.39,256.21,-8.24)	(307.60474 253.22339 -8.501042)	(0.20,-0.74, 0.64)
5	24.56	(250.43,250.28,-8.75)	(261.35,264.69,-7.44)	(257.80124 257.08667 -7.871617)	(0.93,-0.07,-0.37)
6	24.56	(291.28,247.30,-9.55)	(302.20,261.71,-8.24)	(298.6542 254.10316 -8.674944)	(-0.93, 0.07, 0.37)
7	350.26	(250.43,247.30,-9.55)	(301.31,264.69,-7.60)	(275.86942 255.99329 -8.57681)	(0.13,-0.86, 0.49)
8	169.45	(243.04,259.20,-8.26)	(261.35,278.07,-7.18)	(253.84595 267.84653 -7.69662)	(0.26, 0.69, 0.67)
9	23.77	(237.10,277.11,-9.06)	(256.22,282.64,-8.00)	(246.65952 279.87466 -8.527515)	(0.16, 0.96, 0.22)
10	154.27	(297.08,255.25,-9.32)	(315.39,274.13,-8.24)	(305.24637 264.09274 -8.707351)	(0.35, 0.68, 0.64)
11	23.77	(291.13,273.17,-10.12)	(310.26,278.69,-9.06)	(300.69684 275.92838 -9.590096)	(0.16, 0.96, 0.22)
12	326.18	(256.22,261.71,-9.06)	(301.31,277.11,-7.60)	(278.76627 269.40875 -8.333074)	(0.29, 0.74, 0.61)
13	22.62	(196.61,232.48,-13.78)	(211.43,236.63,-13.28)	(204.02386 234.5563 -13.527309)	(-0.25,-0.84,-0.48)
14	22.62	(250.65,228.54,-14.84)	(265.47,232.68,-14.34)	(258.06122 230.60999 -14.589891)	(-0.25,-0.84,-0.48)
15	241.65	(196.61,235.67,-13.78)	(211.43,256.06,-13.49)	(204.02386 245.86302 -13.632289)	(-0.36, 0.06,-0.93)
16	105.13	(198.25,255.09,-13.75)	(216.93,267.42,-12.90)	(207.58922 261.25507 -13.321914)	(-0.19, 0.75,-0.63)
17	241.65	(250.65,231.72,-14.84)	(265.47,252.11,-14.55)	(258.06122 241.9167 -14.694872)	(-0.36, 0.06,-0.93)
18	105.13	(252.29,251.15,-14.81)	(270.97,263.47,-13.96)	(261.6266 257.30872 -14.384494)	(-0.19, 0.75,-0.63)
19	1036.93	(211.43,221.63,-14.52)	(252.29,252.15,-13.47)	(231.85962 234.92247 -13.97644)	(-0.37,-0.17,-0.91)
20	8.94	(209.80,232.48,-13.78)	(211.43,255.09,-13.54)	(211.02458 243.84747 -13.695734)	(0.93,-0.07,-0.37)
21	8.94	(250.65,229.50,-14.58)	(252.29,252.11,-14.34)	(251.87758 240.86398 -14.499061)	(-0.93, 0.07, 0.37)
22	49.47	(211.43,224.62,-13.54)	(224.43,238.41,-11.87)	(217.93243 231.51517 -12.702987)	(-0.93, 0.07, 0.37)
23	125.77	(211.43,221.63,-14.27)	(265.28,230.55,-11.87)	(238.35895 226.09067 -13.068134)	(0.00,-0.98, 0.18)
24	49.47	(252.29,221.63,-14.34)	(265.28,235.43,-12.67)	(258.78543 228.53171 -13.506313)	(0.93,-0.07,-0.37)
25	701.11	(263.84,228.54,-14.84)	(315.39,277.73,-8.50)	(288.58954 253.78618 -11.896009)	(0.93,-0.07,-0.37)
26	701.11	(196.61,233.44,-13.52)	(248.16,282.64,-7.18)	(221.36783 258.69534 -10.574172)	(-0.93, 0.07, 0.37)
27	104.14	(203.75,266.45,-13.15)	(250.28,282.64,-8.80)	(227.01295 274.5451 -10.975836)	(-0.00, 0.98,-0.18)
28	104.14	(257.78,262.51,-14.22)	(304.32,278.69,-9.86)	(281.0503 270.5988 -12.038416)	(-0.00, 0.98,-0.18)
29	184.44	(252.29,249.16,-14.55)	(297.08,278.69,-9.06)	(270.11285 263.51236 -12.390967)	(-0.93, 0.07, 0.37)
30	503.58	(211.43,249.16,-14.52)	(297.08,277.11,-8.26)	(254.25447 263.1367 -11.3911915)	(-0.01, 0.98,-0.22)
31	184.44	(211.43,252.15,-13.75)	(256.22,281.67,-8.26)	(229.25986 266.4958 -11.587641)	(0.93,-0.07,-0.37)
32	787.80	(221.65,207.97,-12.80)	(268.07,238.62,-11.70)	(244.85826 228.07399 -12.260244)	(-0.37,-0.17,-0.91)
33	243.20	(264.36,207.97,-12.80)	(286.27,241.76,-10.71)	(273.8639 221.74771 -11.827543)	(0.88,-0.22,-0.42)
34	66.82	(277.63,214.51,-11.04)	(286.27,247.03,-9.49)	(281.32422 234.65791 -10.214957)	(0.96,-0.06, 0.26)
35	206.69	(258.41,235.32,-9.62)	(278.84,248.52,-9.13)	(268.6273 238.39363 -9.334141)	(0.37, 0.17, 0.91)
36	433.44	(233.16,217.84,-9.99)	(259.62,248.52,-9.13)	(247.017 237.15837 -9.540345)	(-0.53, 0.05, 0.84)
37	157.01	(221.65,210.82,-11.89)	(236.87,245.64,-9.81)	(229.25987 228.22997 -10.848946)	(-0.95,-0.09, 0.30)
38	83.23	(225.36,207.97,-12.47)	(268.07,217.84,-9.81)	(249.74825 212.18248 -11.335811)	(0.00,-0.98, 0.18)
39	456.76	(236.87,214.51,-10.71)	(282.55,236.63,-9.13)	(263.64447 226.11214 -9.776019)	(0.21,-0.72, 0.66)
40	21.24	(247.27,207.18,-11.71)	(257.49,216.40,-10.17)	(253.07529 211.34956 -10.820647)	(0.94,-0.11, 0.31)

Figure 4.2.11 Partial list of surfaces in Figure 4.2.10.

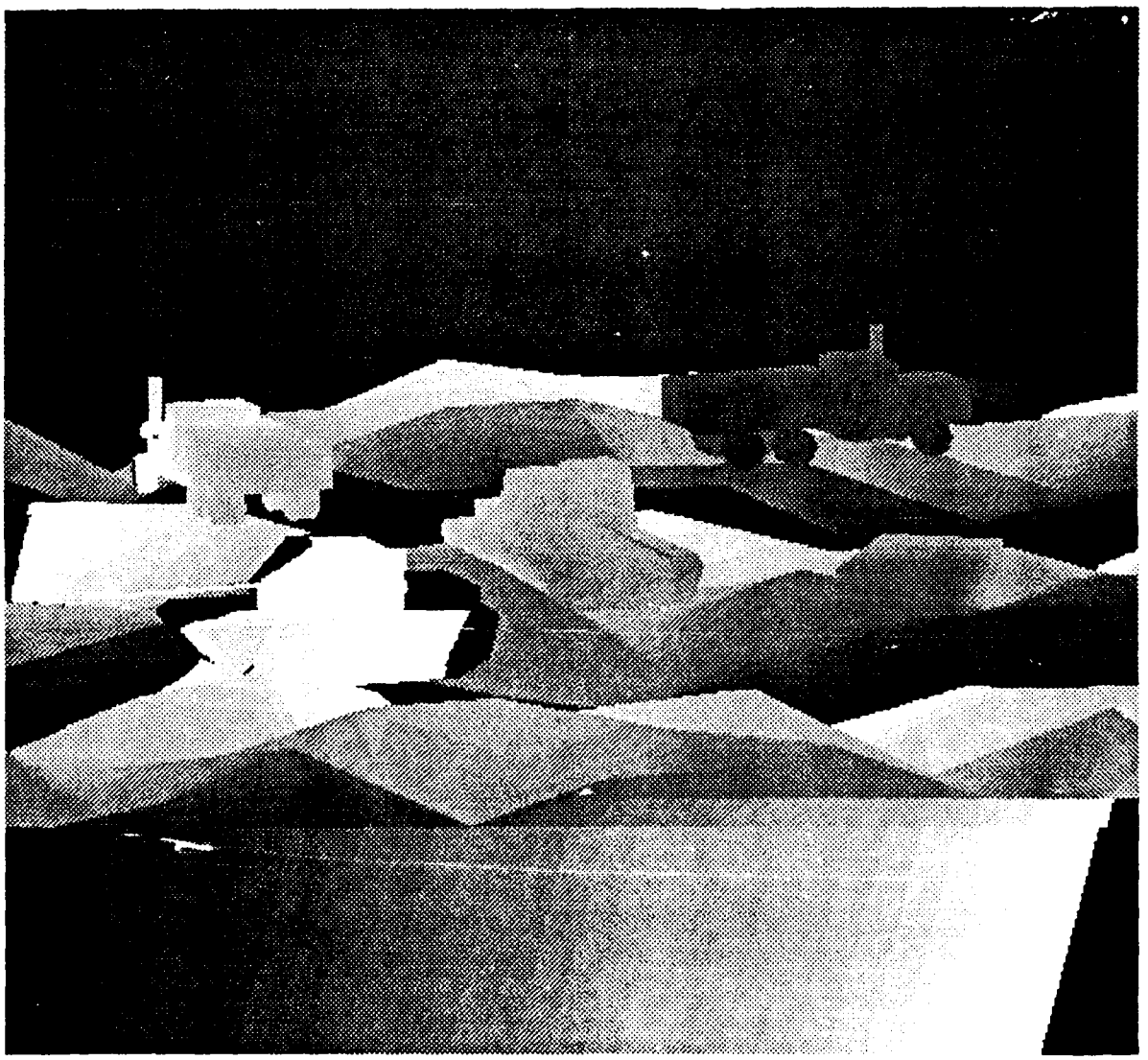


Figure 4.2.12 Synthetic range image with range ambiguities added.

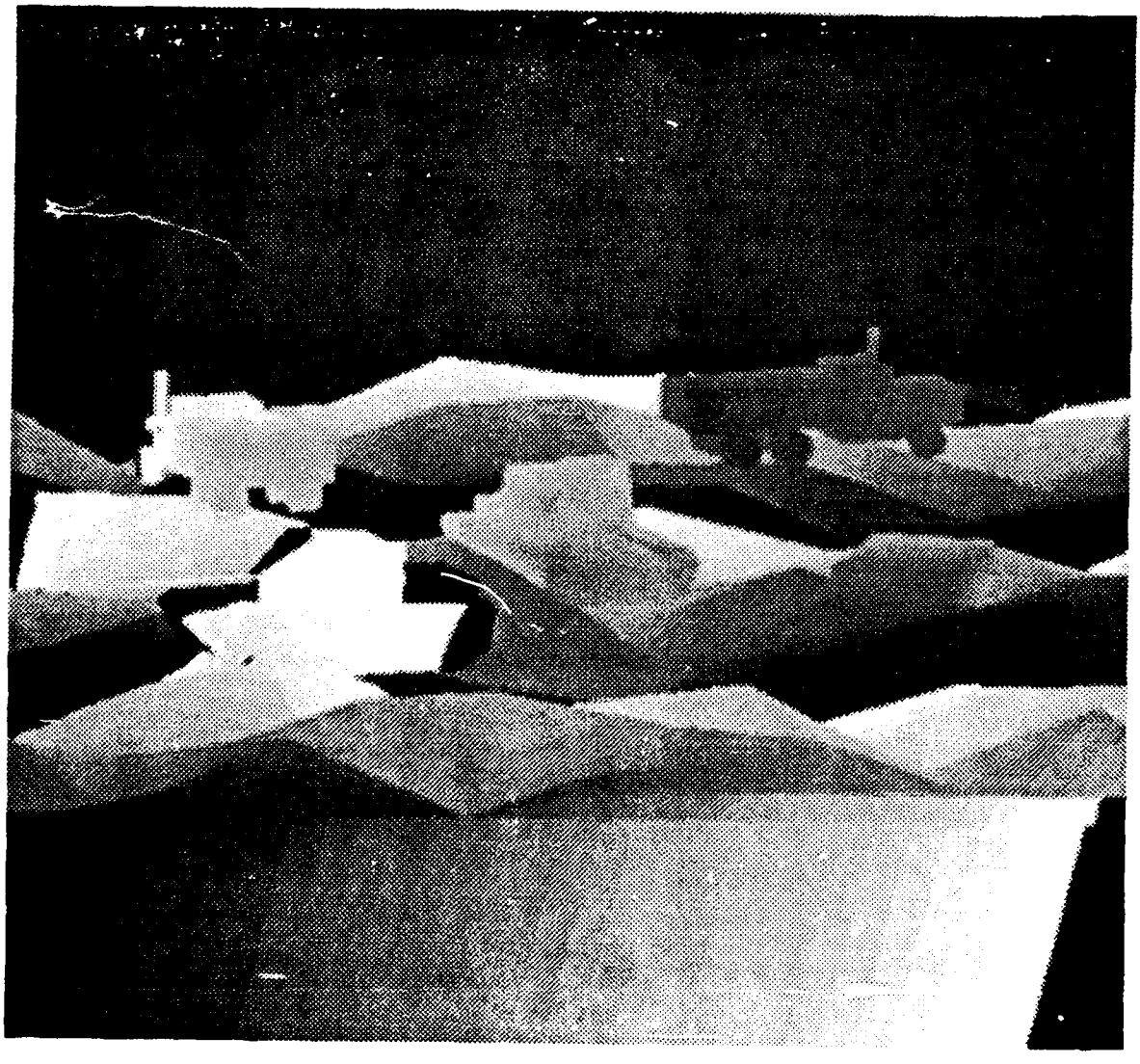


Figure 4.2.13 Blurred synthetic range image.

4.2.2.10. Add noise

Finally noise is added to the image which matches the noise measured in real LADAR images. The noise added here is the same as that described in Section 4.3.2 of [KaYo88].

```
(setq range (add-noise range  :sky 275))
```

Since the noise characteristics of a no-return signal differ from the characteristics of background clutter, the value of the sky is passed to the add-noise routine so it knows when to add the no-return type noise. This is shown is Figure 4.2.14.

4.2.2.11. Future Work

Use better blurring, based on real data. Study the new LADAR data to know what kind of noise to add. Add clutter.

4.2.3. Conversion of BRL Objects to TWIN Objects

In order to convert solid objects from the representation used by the BRL modeler to one compatible with the TWIN library, one must first understand how complex objects are represented in BRL modeler. The BRL modeler represents solid objects via CSG trees, much as PADL does. Figure 4.2.15 shows a simple CSG tree that could be used to define a solid object. In this figure, the oval nodes in the tree represent primitive objects; these objects, such as spheres, cubes, etc., are the elemental objects used by the BRL modeler. The rectangular nodes in the tree represent solids created by the boolean combination of lower-level objects. As in most CSG systems, BRL objects can be combined using three regularized boolean operations: union (u), intersection (i), and difference (-). A rigid body transformation (rotation, translation) is applied to each of the low-level objects before they are combined to form the parent object. These transformations are represented as homogeneous transformation matrices (indicated in the figure with the symbols T_j and I).

There are two sub-tasks that must be performed to convert solid objects from the representation used in the BRL modeler to that used by the TWIN library. First, TWIN objects that represent BRL primitive objects must be generated. These objects are generated while the BRL model is being read into the system. When a BRL primitive is encountered in the definition for a solid, the appropriate library subroutine is called immediately and the TWIN structure representing the primitive is generated. After TWIN structures for all the BRL primitive objects have been generated, they are combined into increasingly complex objects. The following is a list of the primitive solids generated by the BRL modeler.

ARB8 An ARB8 is a solid with 8 arbitrarily placed vertices. This primitive solid is used to represent such objects as cubes, parallelpipeds and wedges. The BRL modeler also uses the ARB8 structure to represent primitive objects with less than 8 vertices by setting the coordinates of some of the 8 vertices to the same point in 3-space. For example in a 7 vertex solid, the coordinates of 2 vertices

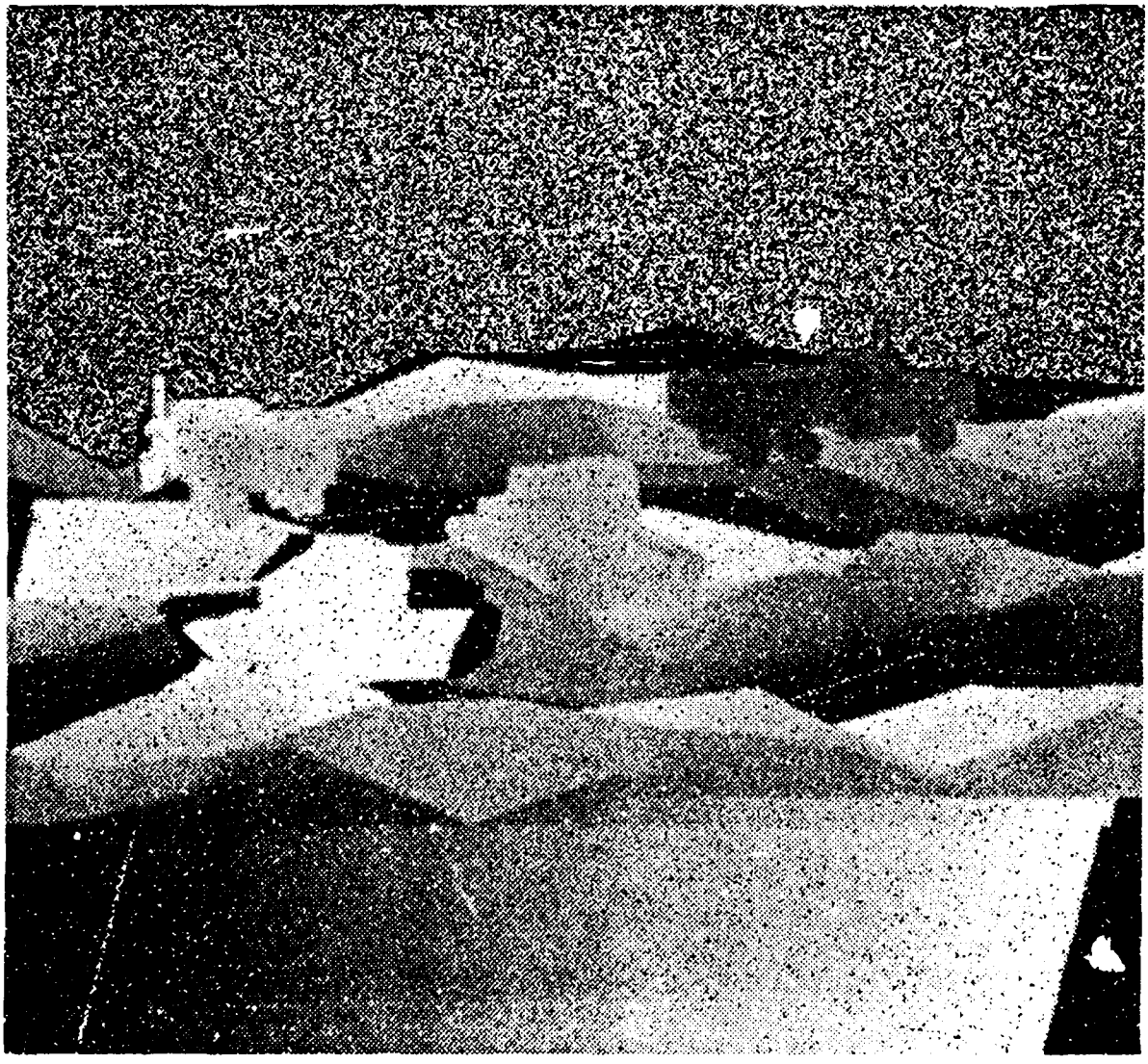


Figure 4.2.14 Synthetic range image with noise added.

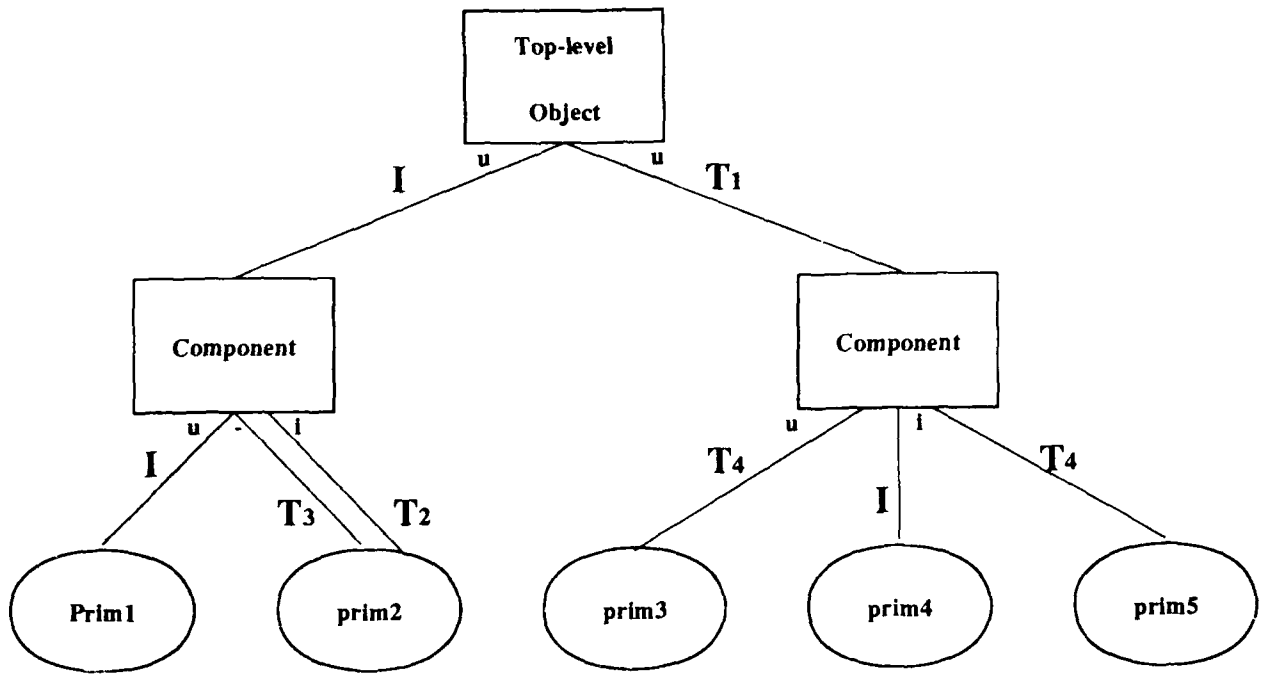


Figure 4.2.15 BRL CSG tree.

would be set to the same point.

TGC The TGC (truncated generalized cylinder) primitive is used to represent cylinders, cones, and elliptical cones.

ELLG The ELLG (generalized ellipsoid) primitive is used to represent spheres, ellipsoids and ellipsoids of revolution.

TOR The TOR primitive is used to represent toroidal solids.

Most of the primitive solids listed here have equivalent TWIN counterparts and thus can be generated directly by TWIN subroutine calls; however, a number of difficulties were encountered when implementing some of the BRL primitives in TWIN. For example, there was no subroutine in the TWIN library to generate solids with 8 arbitrary vertices. We have been able to overcome this difficulty by writing our own routines to create TWIN objects with 8 arbitrarily specified vertices. Also, the definition for a truncated generalized cylinder used in the BRL package is slightly more general than that used in the TWIN library. We have implemented the TGC using the slightly less general TWIN definition; if a TGC requiring the more general BRL definition is encountered, a warning message is printed. So far, we have never encountered a TGC object that required the more general definition. At this time, we have not implemented the code to generate TWIN solids from ARB8's with less then 8 distinct vertices, but this should not present any major difficulties.

After the primitive TWIN solids are generated, they must be combined into more complex solids as defined by the structure of the object's CSG tree. During the conversion process, a post-order traversal of the CSG tree is performed, and a TWIN solid is created for each node as it is visited. Because each node is visited after all of its children are visited in a post-order tree traversal, the TWIN solids corresponding with a node's children are created before they are needed to be combined into the parent node.

We have encountered problems when combining TWIN objects into more complex entities; specifically, the TWIN combination routines sometimes produce invalid objects when two valid objects are combined. We have determined that the *tolerance problem* is the cause of this behavior. Tolerancing is a fundamental problem of solid modeling that must be overcome, to some extent, in all solid modeling systems. Two of the most obvious manifestations of the tolerance problem are the following: When is a point inside the object and when is it outside? When should two points be considered to be equivalent? The tolerance problem also arises in many more subtle ways when combining solid models. Obviously, if the modeler cannot solve these problems, then it cannot produce correct results. At the current time, the TWIN library is not able to overcome many of these tolerance problems; however, the TWIN library is evolving rapidly and thus may encorporate more sophisticate tolerancing schemes in the future.

4.2.4. Conclusions

The conversion to the TWIN solid modeler has been successful. We are now able to model more complex scenes than possible with PADL. Many problems have been overcome in converting the BRL targets into TWIN targets. Unfortunately the tolerance problem is one that hasn't been solved at this time. Improvements in TWIN could overcome this problem.

5. REFERENCES

[And73] Anderberg, M. R. (1973). *Cluster Analysis for applications,* Academic Press, New York and London, pp. 190.

[AndKak87] K.M. Andress and A.C. Kak, "PSEIKI: A Production System Environment for Integrating Knowledge with Images," Technical Report, School of Electrical Engineering, Purdue University.

[BaHa65] Ball, G. H., and Hall, D. J. (1965). ISODATA, *A Novel Method of Data Analysis and Pattern Classification,* AD 699616. Stanford Res. Inst., Menlo Park, California.

[BeJa85] P.J. Besl and R.C. Jain, "Three-Dimensional Object Recognition," *Computing Surveys*, Volume 17, Number 1, March 1985.

[BeJa86] P.J. Besl and R.C. Jain, "Invariant Surface Characteristics for 3D Object Recognition in Range Images," *Computer Vision, Graphics, and Image Processing,* 33, 1986, pp. 33-80.

[Besl88] Paul J. Besl, "Geometric Modeling and Computer Vision," To be published in the Proceedings of the IEEE (Special Issue on Computer Vision), April 21, 1988.

[BFKM86] Lee Brownston, Robert Farrell, Elaine Kant, and Nancy Martin, "Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming," Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1986.

[BonMur76] J.A. Bondy and U.S.R. Murty, "Graph Theory with Applications," North-Holland, New York, 1976.

[Bratko] I. Bratko, *PROLOG Programming for Artificial Intelligence,* Addison-Wesley Publishing Company, Inc., Reading, MA, 1986, pp. 314-358.

[BroFar] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5,* Addison-Wesley Publishing Company, Inc., Reading, MA, 1985.

[CAD] "TWIN Solid Modeling Package User's Manual," School of Mechanical Engineering, Engineering Research Center, Purdue University, April 13, 1987.

[Ch62] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Ann. Math. Stat. 23*, pp 493-507 (Chapters 3, 9).

[ChKa88] C-H Chen and A.C. Kak, "Classification of Primitive Surfaces in Range Data Using Surface Normals", in preparation.

[CroKa87] R. L. Cromwell and A. C. Kak, "Low and Intermediate Level Processing of Range Maps," Technical Report TR-EE-87-41, School of Electrical Engineering, Purdue University.

[Duda] R. Duda, J. Gaschnig, and P. Hart, "Model design in the Prospector consultant system for mineral exploration," in *Expert Systems in the Microelectronic Age*, D. Michie ed., Edinburgh University Press, 1979.

[Date86] C.J. Date, "An Introduction to Database Systems, Volume 1," Addison-Wesley, Reading, Massachusetts, 1986.

[Eber76] Robert B. Eberlein, "An iterative gradient edge detection Algorithm," *Computer Graphics and Image Processing*, Vol. 5, 1976, pp. 245-253.

[FaPr79] I.D. Faux and M.J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood Limited, Chichester, England, 1979.

[FFC] A. Fournier, D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models," *Communications of the ACM*, Vol. 25, No. 6, June 1982.

[Fo65] Forgy, E. W. (1965). *Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications*. Biometric Soc. Meetings, Riverside, California (Abstract in Biometrics 21, No. 3, 768).

[Fu72] K. Fukunaga, "Introduction to Statistical Pattern Recognition," Academic Press, Inc., Orlando, Florida, 1972.

[Fu86] K. Fukunaga, "Statistical Pattern Classification," from *Handbook of Pattern Recognition and Image Processing* edited by T. Young, K.S. Fu., Academic Press, Inc., New York, New York., 1986.

[FuHa] Raymond R. Hayes, "Chap. 4 - The Reduced Parzen Classifier," *Statistical Classifier Design and Evaluation*, pp. 68-82, PhD. Thesis, Purdue University, May 1988.

[FuHu87] Keinosuke Fukunaga and Donald M. Hummels, "Bayes Error Estimation Using Parzen and k-NN Procedures," submitted for publication in *IEEE Trans. Pattern Anal. Machine Intell.*.

[Ha84] R. M. Haralick, "Digital Step Edges from Zero Crossing of Second Directional Derivatives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, No. 1, January 1984, pp 58-68.

[HaHe87] C. Hansen and T. Henderson, "CAGD-Based Computer Vision." *Proceedings of the Workshop on Computer Vision*, Nov. 30 - Dec. 2, 1987, Miami, FL.

[HaSh85] Robert M. Haralick and Linda G. Shaprio, "SURVEY: Image Segmentation Techniques" *Compter Vision, Graphics, and Image Processing*, Vol. 29, pp 100-132.

[HoPa76] S. L. Horowitz and T. Pavlidis, "Picture Segmentation by a Tree Traversal Algorithm," Journal of the ACM, Vol.23, No.2, April 1976.

[Hu62] Ming-Kuei Hu, "Visual Pattern Recognition by Moment Invariants," *IRE Transactions on Information Theory*, February 1962, pp 179-187.

[Hughes84] Hughes Aircraft Company, "Bandwidth Reduction and Intelligent Target Tracking (BRITT) Final Report," Contract No. DAAK70-82-C-0210,CDRLA002, June 1984, pp 2-50 - 2-62.

[HuKa88] S. A. Hutchinson, R. L. Cromwell, and A. C. Kak, "Planning Sensing Strategies in a Robot Work Cell with Multi-Sensor Capabilites," *Proceedings of the 1988 IEEE International Conference on Robotics* and Automation", April 24-29, 1988, Franklin Plaza Hotel, Philadelphia, Pennsylvania.

[KaCh87] A.C. Kak, A. Vayda, R. Cromwell, Y. Kim, and C-H Chen, "Knowledge-based Robotics," *Proceedings of the 1987 IEEE Conference on Robotics and Automation,* 1987, pp. 637-646.

[KaLa83] H.M. Kalayeh and D.A. Landgrebe, "Predicting the Required Number of Training Samples," *IEEE Trans. Pattern Anal. Machine Intell.,* Vol. PAMI-5, No. 6, Nov. 1983, pp. 664-667.

[KaYo87] A.C. Kak, M.A. Yoder, K.M. Andress, S.G. Blask, and B. Shamee, "First Annual Progress Report on Research in Computer Vision for Autonomous Systems," Contract Number DAAK20-85-C-0293, Submitted to Tim Williams, Night Vision and Electro-Optics Laboratory, Fort Belvoir, VA 22060-5677.

[KaYo88] A.C. Kak, M.A. Yoder, K.M. Andress, S.G. Blask, and T.A. Underwood, "Second Annual Progress Report on Research in Computer Vision for Autonomous Systems," Contract Number DAAK20-85-C-0293, Submitted to Tim Williams, Night Vision and Electro-Optics Laboratory, Fort Belvoir, VA 22060-5677.

[KoDo76] J. J. Koenderink, A. J. Van Doorn, "The Singularities of the Visual Mapping," *Biological Cybernetics*, 24:51-59, 1976.

[Korn1] G.H. Kornfeld, "Tactical Vehicles and Background Clutter," *Proceedings of the Seventh Annual KRC Symposium on Ground Vehicle Infrared Signature*, Keweenaw Research Center, August 27,28 1985.

[Korn2] G.H. Kornfeld, "Computer Generation of Infrared Imagery," *Applied Optics*, Vol. 24, No. 24, 15 December 1985.

[Ma67] MacQueen, J. B. (1967). *Some Methods for Classification and Analysis of Multivariate Observations*. Proc. Symp. Math. Statist. and Probability, 5th, Berkeley, 1, 281-297, AD 669871, Univ. of California Press, Berkeley.

[Ma79] S. Maitra, "Moment Invariants," *Proceedings of the IEEE*, Vol. 67, No. 4, April 1979, pp 697-698.

[Mand] B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freemand and Company, San Francisco, 1983.

[MM84] Martin Marietta Corporation, "Context Cueing Techniques," Technical Report DAAK70-82-C-0215, June 1984, pp 90-93.

[NeBa80] R. Nevatia, K. R. Babu, "Linear feature extraction and description," *Computer Graphics and Image Processing*, Vol. 13, 1980, pp 257-269.

[NeSm87] J.E. Nettleton and A.W. Smiley, "Multi-Sensor Field Test", *Center for Night Vision & Electro-Optics Laser Division, Ladar Radar Team*, Fort A. P. Hill, June 1987.

[Padl] E.E. Hartquist, H.A. Marisa, "PADL-2 User's Manual," The Production Automation Project, The University of Rochester, Rochester, New York 14627.

[Phil87] Personal correspondence with Jonathan Phillips, 10-Dec-1987.

[Rayt] G. R. Osche, D. S. Young, and W. J. Wilson, "IR Startle Technology Demonstrator System: Final Report," Raytheon Company, Electro-Optics Systems Laboratory, 528 Boston Post Road, Sudbury, MA 01776, page 19.

[RosKak84] A. Rosenfeld and A. C. Kak, "Digital Picture Processing," Vol. 1, Academic Press, 1982, pp 97, 167, 317.

[ScSy71] Scott, A. J., and Symons, M. J. (1971). *Clustering Methods Based on Likelihood Ratio Criteria*. Biometrics 27, No. 2, 387-398.

[Shaf76] G. Shafer, "A Mathematical Theory of Evidence," Princeton University Press, 1976.

[Smit] A.R. Smith, "Plants, Fractals, and Formal Languages," *Computer Graphics*, Vol. 18, No. 3, July 1984.

[SQL] SQL is a trademark of International Business Machines Corporation.

[Te80] M. R. Teague, "Image Analysis via The General Theory of Moments," *Journal of the Optical Society of America*, Vol. 70, No. 8, August 1980, pp 920-930.

[TeCh86] Teh and R. Chin, "On Digital Approximation of Moment Invariants," *Computer Vision, Graphics, and Image Processing*, Vol. 33, 1986, pp 318-326.

[TI85] TMS32030 User's Guide, SPRU004A, Texas Instruments Incorporated, 1985.

[Trees68] H. L. Van Trees, "Detection, Estimation and Modulation Theory, Part 1: Detection, Estimation and Linear Modulation Theory," John Wiley, New York, 1968.

[Trees71] H. L. Van Trees, "Detection, Estimation and Modulation Theory, Part 3: Radar/Sonar Signal Processing and Gaussian Signals in Noise," John Wiley, New York, 1971.

[VeWi87] J. G. Verly, B. D. Williams, and D. E. Dudgeon, "Automatic Object Recognition from Range Imagery Using Appearance Models," *Proceeding of the IEEE Computer Society Workshop on Computer Vision,* Fontainebleau Hilton, Miami Beach, Florida, November 30 - December 2, 1987.

[UNIFY] UNIFY is a registered trademark of Unify Corporation in Lake Oswego, Oregon.

[Wa63] Ward, Jr., J. H. (1963). *Hierarchical Grouping to Optimise an Objective Function.* J. Amer. Statist. Assoc. 58, No. 301, 236-244.

[Wo70] Wolfe, J. H. (1970), *Pattern Clustering by Multivariate Mixture Analysis.* Multivariate Behavioral Res. 5, No. 3, 329-350.

[YaKa86a] H.S. Yang and A.C. Kak, "Determination of the Identity, Position, and Orientation of the Topmost Object in a Pile," *Computer Vision, Graphics, and Image Processing,* 36, 1986, pp. 229-255.

[YaKa86b] H.S. Yang and A.C. Kak, "Determination of the Identity, Position, and Orientation of the Topmost Object in a Pile: Some Further Experiments," *Proceedings of the 1986 IEEE Conference on Robotics and Automation,* 1986.

APPENDIX A: UTILITIES

A.1. A DATABASE FOR MANAGING TARGET IMAGES

After acquiring the Terrain Board, Eglin Turntable, and BRITT target images from NVL, we found that we had a rich variety of images. These images contained a wide selection of targets viewed from many ranges and angles. Most images had a header which contained information on the targets in the image (information such as ranges, orientation, etc), and the prevailing environmental conditions. However, selecting a set of targets for a given experiment was difficult because the header formats were not the same for all the images. In addition, most headers were hundreds of bytes long and contained extra information which was not needed. The experiments (classification in particular) required the selection of several targets with similar characteristics For example, an M35 truck as viewed from 2.5 km and 100 feet altitude. Finding such a set of targets is difficult and time consuming since hundreds of headers, each consisting of hundreds of bytes of information, must be scanned.

To resolve this problem we extracted the pertinent information from each of the headers and then used a *relational database manager* to help select the different sets of targets needed for the experiments.

A generic relational database is described in the following section. Section 1.2 gives details on what information was extracted from the headers and stored in the database. Finally, Section 1.3 gives examples of how to use the UNIFY Relational Database Management System [UNIFY], to locate target of a given description in the database.

A.1.1. RELATIONAL DATABASES

Briefly, a relational system is one in which:

1. the data is perceived by the user as tables (and nothing but tables); and
2. the operators at the user's disposal (e.g., for data retrieval) are operators that generate new tables from old. [Date86]

For example, there will be one operator to extract a subset of the rows of a given table, and another to extract a subset of the columns - and of course a row subset and a column subset of a table may both in turn be regarded as tables themselves. New tables may be permanently saved as part of the database or merely displayed on a terminal.

Each permanent table (or relation, as it is called) is given a unique name. The BRITT target data has been stored in a table named **britt**; Terrain Board and Eglin Turntable data will similarly be stored in their own tables. Each of the n columns of a table has a unique name called an *attribute*. Each row of a table is called a *tuple*, short for n-tuple, which is made up of

n fields containing the attribute values. A tuple is not allowed to have a null or blank value for one of its attributes. Each table must have a key attribute or set of attributes whose value or values uniquely identify a tuple in the relation. For example, if there is a single key attribute, then no two tuples in the table may have the same value for that attribute.

A.1.2. DATABASE DESIGN

Table A.1 gives a description of the attributes stored in each tuple of the **britt** database.

Each tuple in the britt relation represents a target and its associated information. Figure A.1 is a listing of a portion of the **britt** database which shows the values of the various attributes. Similar table definitions will be made for the rest of our target data.

A.1.3. QUERYING THE DATABASE

After a database is built, most any database manager can be used. We choose to use the UNIFY Relational Database Management System [UNIFY] since it is readily available and supported on the location computer system. UNIFY uses the SQL [SQL] (pronounced SEQUEL) query language developed by IBM in conjunction with DML, the Data Manipulation Language.

A SQL query consists of clauses, each of which begins with a keyword. The following is a list of SQL/DML keywords:

and	help	separator
asc	in	set
avg	insert	start
between	into	sum
by	is	unique
count	lines	unlock
delete	max	update
desc	min	where
edit	not	write
end	or	
fields	order	
from	records	
group	restart	
having	select	

Since these commands are described in the UNIFY manuals we will only briefly discuss those commands needed to extract the targets for the BRITT classification experiments.

There are required and optional SQL clauses. The required clauses are as follows:

Table A.1 Attributes of **britt** database.

id:	The key attribute of the relation. Every target is assigned a unique three digit i.d. number.
type:	The type of vehicle. In this case, one of **APC, JEEP, TANK,** or **TRUCK**.
model:	The specific kind of vehicle. For example, there were two kinds of **TANKs, M48** and **M551.**
range:	The distance to the center of field of view of the parent image from which the target image was extracted.
angle:	Aspect angle from which the target is seen.
size:	The extracted target image is *size* by *size* square.
target:	Name of file containing the target image with the target of interest in its center. This image is extracted from the provided image with multiple targets in it. The britt target files have been named **britt###**, where ### is the target id.
parent:	Name of the file from which the target was extracted.
x:	x-coordinate in the parent file of the center of the target file.
y:	y-coordinate in the parent file of the center of the target file.

id	type	model	range	angle	size	target	parent	x	y
1	TANK	M551	5.000000	180	128	britt001	file001	415	264
2	TANK	M551	5.000000	180	128	britt002	file001	410	290
3	APC	M113	5.000000	180	128	britt003	file001	180	236
4	APC	M114	5.000000	180	128	britt004	file001	303	225
5	TRUCK	M35	5.000000	180	128	britt005	file001	289	308
6	JEEP	M151	5.000000	0	128	britt006	file001	339	154
7	TANK	M551	5.000000	180	128	britt007	file002	468	299
8	TANK	M551	5.000000	180	128	britt008	file002	461	328
9	APC	M113	5.000000	180	128	britt009	file002	239	262
10	APC	M114	5.000000	180	128	britt010	file002	357	251
11	TRUCK	M35	5.000000	180	128	britt011	file002	338	355
12	JEEP	M151	5.000000	0	128	britt012	file002	396	172
13	TANK	M551	3.500000	180	128	britt013	file003	564	261
14	TANK	M551	3.500000	180	128	britt014	file003	555	303
15	APC	M113	3.500000	180	128	britt015	file003	255	209
16	APC	M114	3.500000	180	128	britt016	file003	416	193
17	TRUCK	M35	3.500000	180	128	britt017	file003	373	358
18	JEEP	M151	3.500000	0	128	britt018	file003	476	87
19	TANK	M551	3.500000	180	128	britt019	file004	556	274
20	TANK	M551	3.500000	180	128	britt020	file004	544	316
21	APC	M113	3.500000	180	128	britt021	file004	249	222
22	APC	M114	3.500000	180	128	britt022	file004	410	209
23	TRUCK	M35	3.500000	180	128	britt023	file004	355	364
24	JEEP	M151	3.500000	0	128	britt024	file004	476	109
25	TANK	M551	2.500000	180	128	britt025	file005	56	265
26	TANK	M551	2.500000	180	128	britt026	file005	553	320
27	APC	M113	2.500000	180	128	britt027	file005	121	211
28	APC	M114	2.500000	180	128	britt028	file005	345	194
29	TRUCK	M35	2.500000	180	128	britt029	file005	297	363
30	JEEP	M151	2.500000	0	128	britt030	file005	408	95
31	TANK	M551	2.500000	180	128	britt031	file006	536	257
32	TANK	M551	2.500000	180	128	britt032	file006	522	309
33	APC	M113	2.500000	180	128	britt033	file006	97	193
34	APC	M114	2.500000	180	128	britt034	file006	320	182
35	TRUCK	M35	2.500000	180	128	britt035	file006	262	348
36	JEEP	M151	2.500000	0	128	britt036	file006	387	86
37	TANK	M551	5.000000	45	128	britt037	file007	469	249
38	TANK	M551	5.000000	45	128	britt038	file007	453	288
39	APC	M113	5.000000	45	128	britt039	file007	235	222
40	APC	M114	5.000000	45	128	britt040	file007	339	214
41	TRUCK	M35	5.000000	45	128	britt041	file007	368	343
42	JEEP	M151	5.000000	0	128	britt042	file007	398	115
43	TANK	M551	5.000000	45	128	britt043	file008	470	269
44	TANK	M551	5.000000	45	128	britt044	file008	455	308
45	APC	M113	5.000000	45	128	britt045	file008	240	240
46	APC	M114	5.000000	45	128	britt046	file008	342	234
47	TRUCK	M35	5.000000	45	128	britt047	file008	368	353
48	JEEP	M151	5.000000	0	128	britt048	file008	404	146
49	TANK	M551	3.500000	45	128	britt049	file009	504	234
50	TANK	M551	3.500000	45	128	britt050	file009	483	282

Figure A.1 A sample section of the **britt** database.

select (attribute list)
from (table names)

Some of the optional clauses are:

where (expression is true or false)
into (an ASCII file)

The required **select** and **from** clauses go hand in hand. The **select** clause specifies which attributes (or columns) to print out for the relations (tables) specified by the **from** clause. For example:

```
sql> select id, type
sql> from britt /
```

causes the **id** and **type** columns of the **britt** table to be displayed. The '/' character tells the SQL parser that we are done entering a query and no additional optional clauses follow. One may print out the entire britt table with the following command:

```
sql> select *
sql> from britt /
```

Here the '*' will match all attribute names, and so all columns of the britt table will be printed.

We have seen how to select specific columns of our tables, but not specific rows. For this we need to make use of the **where** clause. The **where** clause will select a tuple only if its expression is true for that tuple. For example:

```
sql> select *
sql> from britt
sql> where range = 5.0 /
```

will select all the tuples with values of 5.0 for their range field. We can now begin to make more complex queries. The query:

```
sql> select target
sql> from britt
sql> where angle = 0.0 and type = 'TANK*' /
```

will print out all of the target filenames containing front views (corresponding to aspect angle=0 degrees) of TANKs. The type attribute is stored as a string, and so must be quoted. The '*' character matches all trailing blanks in the field. We can also make nested queries:

```
sql> select target
sql> from britt
sql> where  angle = 0 and type = select type
sql>                             from britt
sql>                             where model = 'M48*' or
sql>                                   model = 'M551*' /
```

Starting from the innermost query, the tuples with model values of M48 or M551 (the two kinds of tanks) are selected, then the type column of this temporary table is selected, then all tuples in britt with types in this set and angle values of 0 are selected, and finally the target column of this temporary table is selected and displayed. The final result is the same as the previous query, a list of target filenames containing front views of tanks.

The operators >, <, and ^= (not equal) may all appear in the where expression. Arithmetic expressions using the operators +,-,/, and * and attribute names used like variables are allowed anywhere a simple attribute name is allowed in **select**, and **where** clauses. The following is a way to request an attribute within a range of values:

```
sql> select id
sql> from britt
sql> where type = 'JEEP*' and angle between 0.0 and 180.0 /
```

which will give the obvious result.

The order of the query output can be specified by using the order by clause. The data can be sorted by multiple fields in both ascending and descending order:

```
sql> select id, angle, target
sql> from britt
sql> where range between 2.5 and 5.0
sql> order by angle asc, id desc /
```

which will get the id number, angle of view, and filename of all targets between 2.5 and 5.0 kilometers away and output them in ascending order by angle and descending order by id number for targets with the same angle.

The last important part of the query language is the ability to send results of queries to ASCII files for use as input to other programs. The "lines 0" command for suppressing the table header comes in handy here. An example of such a query:

```
sql> lines 0
sql> select target
sql> from britt
sql> where type = 'TRUCK*' and
sql>       angle = 180 and
sql>       range = 3.0
sql> into savefile /
```

Names of the files containing rear views of trucks at a range of 3.0 kilometers are saved in file **savefile**. The file **savefile** may then be used as input to a program performing classification experiments. As another example, the following query could be used as input to a program to extract targets from their parent files for a newly generated database:

```
sql> select parent, x, y, size, target
sql> from britt
sql> into extract.infile /
```

A.1.4. SELECTING THE BRITT CLASSES

With the above tutorial in mind, extracting the targets needed for the three classes used in the BRITT classification experiments is easy. The *tank* s class consisted of 50 M551 tanks at a range of 2.5km. The following request would extract all M551 tanks at 2.5 km:

```
sql> select target
sql> from britt
sql> where range = 2.5 and model = 'M551*' /
```

The *apcs* class consisted of 25 M113 apcs and 25 M114 apcs at a range of 2.5km. They were selected by using the following query:

```
sql> select target
sql> from britt
sql> where range = 2.5 and [model = 'M113*' or model = 'M114*'] /
```

The final class consisted of M35 *trucks*. There were not enough trucks at the range of 2.5km, so trucks at all ranges were used:

```
sql> select target
sql> from britt
sql> where model = 'M35*' /
```

All of the above queries returned more targets than needed, so then each of the targets was viewed and the most suitable targets were used.

A.1.5. CONCLUSIONS

The use of a database manager has greatly simplified the selection of target images. All the targets from the Eglin turntable, the NVL terrain board, and the TI data set that we have received will be placed in the database so that they can be easily located when needed. Additional attributes such as background clutter and image quality may be added to the database if they will help in the target selection.

A.2. RANGE DATA FROM STRUCTURED LIGHT

This section reports on the acquisition of structured light range data.

A.2.1. LIGHT STRIPE IMAGES: WHAT ARE THEY?

In our Robot Vision Lab we use a single-slit projection system for the acquisition of 3-D range data. Our sensor consists of a projector and a camera. The projector illuminates the scene with a single stripe of light, and the camera records the interaction of the stripe with scene objects. To collect range data from the scene, a robot arm moves the sensor in a straight line and records the illuminated stripes at equal intervals along the direction of motion (see Figure A.2). The robot arm moves in the direction of the arrow in the figure, stopping at equal intervals along the way to collect the image of a single stripe.

The stripe images as recorded on the camera can be translated easily into what is called the pixel offset data. In Figure A.3 we have shown what is meant by pixel offset data. In that figure, with pixel P in the source-viewpoint frame, we associate the offset $d(i,j)$ as obtained from the location of the corresponding illuminated pixel in the camera image. This pixel offset data can be translated into a range map of the scene. The offset values are multiplied by a calibration matrix to obtain the (x,y,z) coordinates of points on the detected stripe.

A program has been written to convert the raw (x,y,z) data of the range map into an image in which pixel brightness corresponds to distance from the sensor path. The resulting range image is basically an orthogonal projection of the scene, and so is very much like the laser range images we receive from NVL. There are slight differences, however. Due to occlusion problems, the range images we generate have some areas with no valid range information. This is due to the geometry of the sensor. Because the camera is next to the projector, there will be cases in which the projected stripe will lie on some part of the object that is hidden from the camera by some other part of the object. The result is a shadow-like void of missing information in the range image. Also, points more than a few feet from the scanner are not detected well due to the spreading of the stripe. We therefore have an abrupt falloff in valid data as we move far enough from the scanner. The distance at which this phenomenon occurs is obvious when one observes the composite stripe or range images.

A.2.2. OUR SIMULATED TARGET RANGE IMAGES

We have constructed scale models of several tanks and generated range maps from light stripe images taken of them. Figure A.4 shows composite light stripe images of an M48 tank model taken from eight different aspect angles. Figure A.5 shows the corresponding range images, where lighter pixels correspond to nearby pixels, darker pixels are farther away, and white pixels are areas of no information due to occlusion and spreading of the stripe, as discussed above. Note the artificial contours in the range images, which is especially noticeable in the ground plane on which the tank sits. The false edges between bands are due to the gray

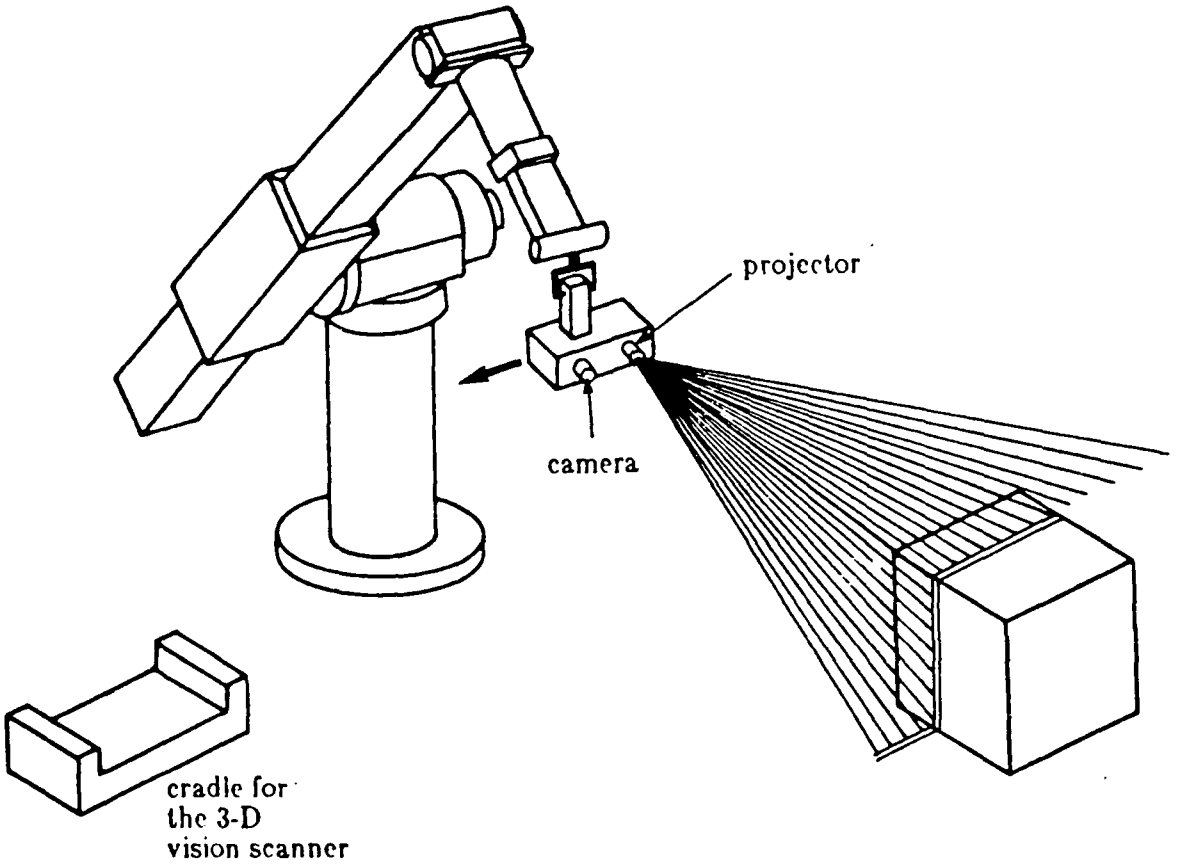


Figure A.2 Light stripe image collection using a linear scan with the sensor by a robot.

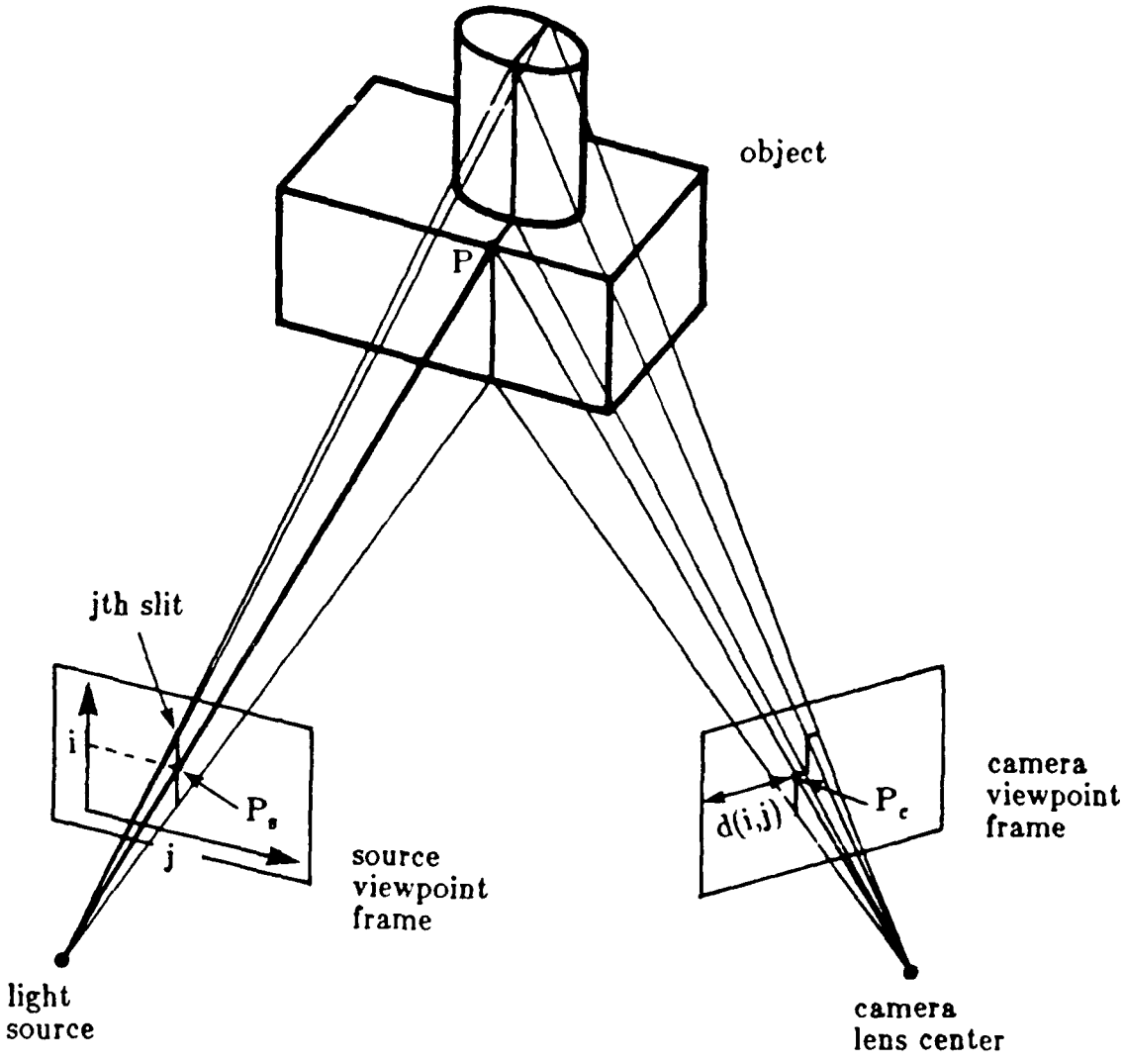


Figure A.3 Shown are the source viewpoint frame (light source plane) and the camera viewpoint frame (camera image plane). Pixel offset d(i,j) is the horizontal distance in the camera image corresponding to point P on the j-th stripe projected by the source. The quantity d(i,j) is measured from the left hand edge of the i-th scan line of the camera image.

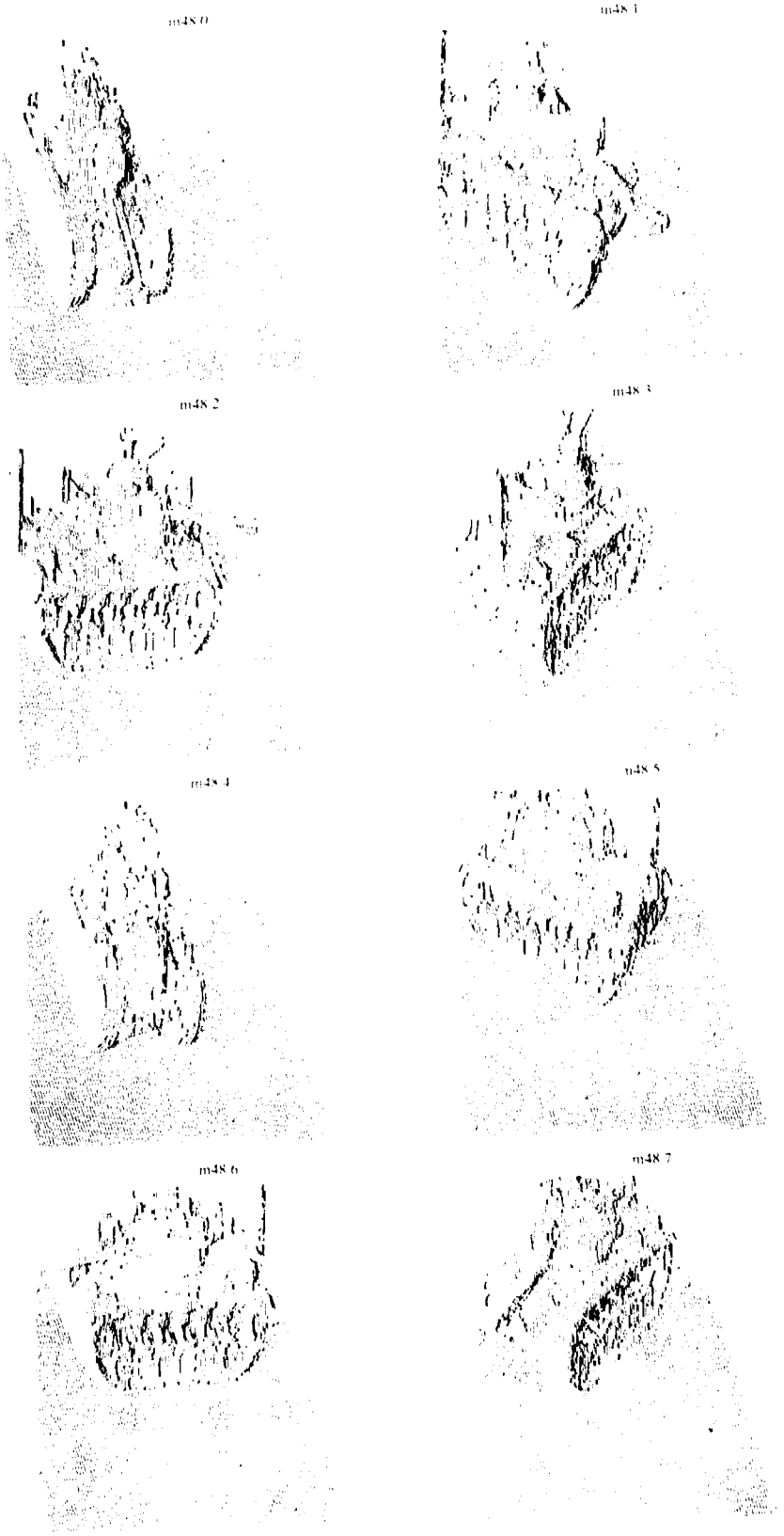


Figure A.4 Composite light stripe images of a model of an M48 tank.

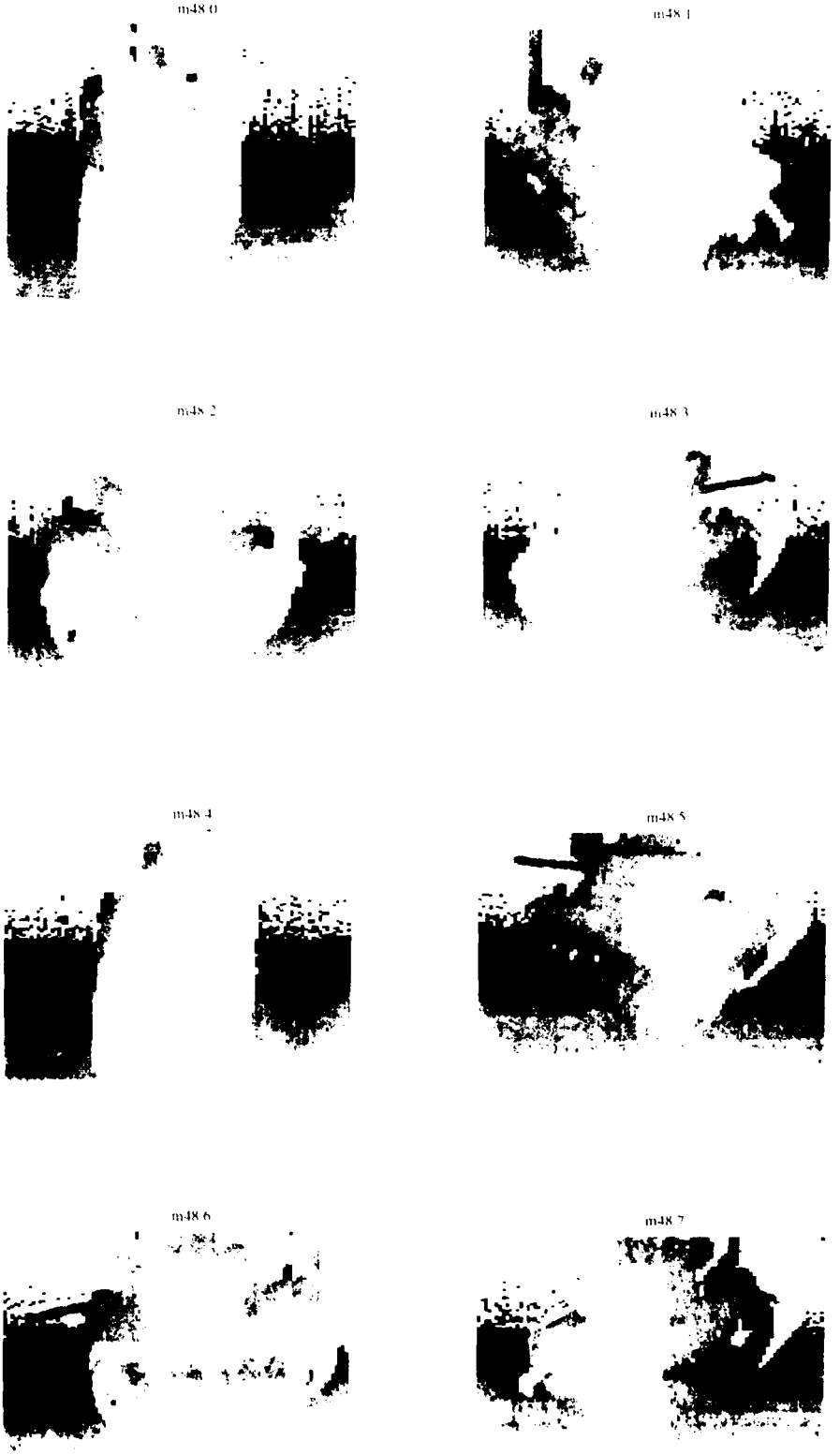


Figure A.5 Range images of a model of an M48 tank computed from offset data.

value quantification of the image hardcopy device, and are not really present in the image itself.

A.2.3. CONCLUSIONS

The figures show that we are able to generate range images of targets in our lab. Future work in this area will include down sampling the images so that the number of pixels on target will be approximately the same as real LADAR images.

APPENDIX B: DETAILED DESCRIPTION OF LADAR DETECTION PROGRAMS

The purpose of this appendix is to describe the LADAR target detector developed by the Robot Vision Lab at Purdue, and how to use the programs that it is composed of. The detector that this document describes is the first attempt at detecting tactical targets using a single line of LADAR data. The theory of our detector was describe in Section 3.2.4. In that section, we proposed a pixel based detector that uses the range value of a pixel and the ranges values of the neighboring pixels to classify the pixel as belonging to the background or the object. We also discussed some aspects of detection theory and explained the need for estimating the density functions of the target and the background for robust detection.

The detection process is non-trivial in that it requires the use of a number of programs, all of which interact with each other, its use . Because of this, there are a number of sections to this appendix. These sections include a description of what the detection programs do and how they are used. A description of the data files that the programs expect to see and examples demonstrating program usage are also included. It should also be noted that the programs as supplied can work with up to 50 dimensional data vectors and this can be increased by merely changing a single parameter and recompiling the programs.

B.1. PROGRAM DESCRIPTIONS

The target detector consists of four programs. These programs are: **I2V** and **V2I** convert data from an image to a data vector and back again. **MAKECLASS** estimates the density function of the target and background and stores relevant parameters for the detector, and **QCLASS** is the actual detector. All these programs will be described in greater detail in the following sections.

B.1.1. Operation of I2V

The purpose of I2V (image to vector) is to extract data vectors from an image so the fact that the data vectors are coming from an image and their composition is hidden from the detector. I2V actually works in two modes depending on whether it is being used for training the detector or if it is generating vectors for detection. These two modes of operation will be called training and detection modes respectively. Since I2V is easiest to understand in detection mode, we will discuss that first.

In detection mode, I2V was designed to generate a data vector for every pixel in the input image and pass these vectors to the detector. To be precise, the data vector will be built as follows: If the target is at the farthest range, choose N range values from the scan line centered around the pixel being considered; N is the minimum number of pixels needed to guarantee that an entire target (plus some background) is covered. These range values are then used to build an N dimensional vector. If the target is closer then the maximum range, down-sample the scan line around the pixel so that N range values still cover the entire target (plus some background).

In this mode, I2V reads the image from standard input and writes a vector file to standard output.

In training mode, operation of I2V is similar to operation in detection mode. In this mode, the data vector is computed identically as in the detection mode; the only difference in the two modes is how the program reads the image files and where it writes the data vector files. In training mode, I2V reads two image files, one containing the actual image and a second one containing the segmented version of the image (a pixel in the segmented image is defined to be zero if the pixel is in the background nonzero otherwise). It also writes to two data vector files, one for target pixels and another for background pixels. After the data vector has been computed, the pixel in the segmented image is checked; if the pixel is zero, the data vector is written to the background vector file otherwise it is written to the object vector file (The format of the types of data files will be described later).

B.1.2. Operation of MAKECLASS

MAKECLASS is the program used to train the detector. To do this, it creates an estimate of the density function and writes the density function's statistics (mean and covariance matrix) to a file for use by the detector. Because the density function was assumed to be Gaussian (otherwise the process would be intractable) it is completely described by these statistics. The input to MAKECLASS is a data vector file created by I2V in training mode and the output is a parameter file containing the necessary statistics.

B.1.3. Operation of QCLASS

QCLASS is the actual detector. In fact, it is a very general purpose detector and can work with any type of data. It reads the data vectors generated by I2V in detection mode, does the classification as object/background and outputs its result to the standard output. To accomplish the classification, it reads in a data vector for a pixel and then computes the value of the target and background density estimates at that point. If these values are called *f(target)* and *f(noise)* respectively, then a decision that the pixel is part of a target is made if $f(target) > C * f(noise)$ otherwise the pixel is said to be noise. QCLASS then outputs the decision to the standard output (it outputs a "1" if it thinks the pixel is from an object, "0" otherwise). Note: C corresponds to a threshold and depends on the a-priori class probabilities and the costs of false detections and detection misses. At the current time, C is set equal to 1.

B.1.4. Operation of V2I

V2I (vector to image) is the program that reassembles the classified pixels back into an image. It merely reads in the "0's" and "1's" produced by qclass and outputs unsigned chars (bytes) in the form of an image.

B.2. PROGRAM OPERATION

B.2.1. I2V Operation

The following are the command line parameters that I2V uses.

```
size=<image size>          -- size of image (for square images)
                           -- or
rows=<# rows>              -- # of rows in image
cols=<# cols>              -- # of cols in image
x_in=<x target size>       -- x size of target (in pixels)
x_out=<x dim>              -- x size of target (after down-sampling)
y_in=<y target size>       -- should be 1 for ladar detection
y_out=<y target size>      -- should be 1 for ladar detection
-mean                      -- include if want to normalize by mean
                           -- of row being scanned
skip=<sampling density>    -- used when training detector
                           -- skip to every nth pixel
                           -- used to reduce amount of data processed

-- Command line parameters to use only when training classifier
    grey=<file name>       -- file name of grey scale file (input)
    seg=<file name>        -- file name of segmentation file (input)
    back=<file name>       -- file name of background vector file (output)
    obj=<file name>        -- file name of object vector file (output)
                           -- Note: the output files append
                           -- the new data vectors to the
                           -- end of the corresponding
                           -- files so samples can be
                           -- gathered from multiple files.

-- Command line parameters to use only when detecting targets

    STDIN                  -- input image file
    STDOUT                 -- output data vector file

-- Note that if x_in = x_out and y_in = y_out then no resampling is done.
```

B.2.2. MAKECLASS Operation

The following are the command line parameters that MAKECLASS uses.

```
dim=<dimension>          -- data vector dimension
                         -- should be equal to x_out * y_out
                         -- as sent to I2V

STDIN                    -- input data vector file
STDOUT                   -- output parameter file
-- Note: the name of the output file should have the form:
        <core name>.0    -- for background data vector file
        <core name>.1    -- for object data vector file
```

B.2.3. QCLASS Operation

The following are the command line parameters that QCLASS uses.

```
dim=<dimension>          -- data vector dimension
                         -- should be equal to x_out * y_out
                         -- as sent to I2V
file=<file name core>    -- parameter file name core. Parameter
                         -- file names should be <file name core>.#
                         -- see the example commands for further
                         -- clarification.
<file name core>.0       -- background parameter file name
<file name core>.1       -- object parameter file name

STDIN                    -- input data vector file
                         -- should have one vector per pixel
STDOUT                   -- output data vector file
                         -- 1 => object
                         -- 0 => background
                         -- should have one vector per pixel
```

B.2.4. V2I Operation

The following are the command line parameters that V2I uses.

size=<image size>	-- size of image (for square images)
	-- or
rows=<# rows>	-- # of rows in image
cols=<# cols>	-- # of cols in image
STDIN	-- input data vector file
	-- 1 => object
	-- 0 => background
	-- should have one vector per pixel
STDOUT	-- output image

B.3. SAMPLE COMMANDS

The following are samples provided to make the explanations a little more concrete.

```
-- This example shows how to train the detector
-- The first two (i2v) commands pull training samples from two images

i2v x_in=49 x_out=25 y_in=1 y_out=1 rows=96 cols=160 -mean skip=11 #
        grey=images.324.im03                                       #
        seg=segment.324.im03                                       #
        back=background obj=object

i2v x_in=49 x_out=25 y_in=1 y_out=1 rows=96 cols=160 -mean skip=11 #
        grey=images.324.im03                                       #
        seg=segment.324.im03                                       #
        back=background obj=object

-- Note that the character "#" means continue the command on the next line.

-- The next two commands compute the density function statistics

makeclass dim=25 file=background > class.0
makeclass dim=25 file=background > class.1

-- This example shows how to run an actual detection experiment

i2v x_in=49 x_out=25 y_in=1 y_cut=1 rows=96 cols=160 -mean          #
        < image.324.im03                                           #
        | qclass dim=25 -v file=class class.1 class.0              #
        | v2i rows=96 cols=160 > detect.32403
```

B.4. DATA FILE TYPES

There are three types of files used in the detection process, they are image files, sample vector files and parameter files.

B.4.1. Image Files

The image files consist of unsigned character (byte) data stored in the usual raster scan fashion. Therefore, a 160 x 96 range image would consist of 15360 bytes of range information.

B.4.2. Data Vector files

The data vector files are created to interface with the detection programs QCLASS and MAKECLASS. These are ascii files with one data vector per line. The lines consist of N floating point numbers separated by spaces or tabs. So the files end up looking like:

```
sample1[0]  sample1[1]  ...    sample1[N]

sample2[0]  sample2[1]  ...    sample2[N]

.
.
.

sampleM[0] sampleM[1]...    sampleM[N]
```

Where N is the dimensionality of the data. The following is an example of a (very short) 5 dimensional data vector file:

```
2.124582  0.466372  1.442297  0.129548  0.475008  ; 1st pixel
0.449099  1.355932  1.053654  1.675483  2.020944  ; 2nd pixel
0.120965  2.307648  1.265484  0.027915  1.823786
1.262410  0.635385  1.329293  1.103564  1.981398
0.344403  0.041000  1.279213  1.254613  1.303813
1.256044  0.249545  1.380816  1.106317  0.133091
2.087861  1.588770  1.264362  1.006499  0.673772
0.507408  1.330907  1.222771  1.505589  0.091500
1.247726  1.214453  1.430725  1.347544  1.763452
```

Notice that anything after a comma in a line is defined as a comment and is ignored.

B.4.3. Parameter Files

The parameter files contain the covariance matrix and mean vector of the object and background density functions. Because the densities are assumed to be Gaussian, these are the only values needed to determine density function. Thus, the detector trainer needs only to store these two parameters for the detector to use. The format of these files is similar to the format of the data vector files; they too are ascii files. The first N valid vector lines are assumed to be the covariance matrix of the class, and the next valid vector line is assumed to be the density function's mean vector. The format of the files looks like this:

```
covar[0,0]  covar[0,1]  ...  covar[0,N]

covar[1,0]  covar[1,1]  ...  covar[1,N]

.
.
.

covar[N,0]  covar[N,1]  ...  covar[N,N]

mean[0]          mean[1]          ...  mean[N]
```

Where once again N is the dimensionality of the data. A sample 5 dimensional parameter file is shown next:

```
 0.373155    0.0659588   -0.0151386   -0.0715956   -0.0860195   ; covariance
 0.0659588   0.300289     0.0780137   -0.0323843   -0.0698119
-0.0151386   0.0780137    0.248101     0.0794427   -0.0335361
-0.0715956  -0.0323843    0.0794427    0.268688     0.0508662
-0.0860195  -0.0698119   -0.0335361    0.0508662    0.304258

 0.989099    1.01034      1.03651      0.957537     0.922343  ; mean vector
```

Once again note the ability to use comments in the data lines.

APPENDIX C: DERIVATION OF MAITRA'S INVARIANT MOMENTS

This appendix is not intended for a reader well conversant with the theory of image moments. In fact, even a reader who is not familiar with this theory might wonder about why we have taken the trouble of rederiving the results that are amply documented in the archival literature.

Our motivation for rederiving the expressions snown here was the discovery of an error in the moment-invariants used in the Martin Marietta report. This apparently was caused by a typographical error in the original 1962 paper by Hu [Hu62]; this error being subsequently reported upon by Maitra in 1979 [Ma79]. This and the other errors that Maitra found in the literature that preceded him prompted us to rederive for ourselves all the major results.

In what follows, we will rederive expressions for the region level features used in [MM84]. The main features are *Moment Invariants* based on Hu's paper [Hu62]. Moment invariants, while invariant under rotation, translation and scale change, are not invariant under illumination change. Maitra's invariants [Ma79] are invariant under illumination change, rotation, translation and scale change; and thus are sufficient to characterize an image under these transformations. However, note that not all the features listed in Table 9 of [MM84] are necessary for target characterization. **Specifically, the moment invariants and Maitra's invariants are not independent.**

C.1. Moments

Since Hu's paper [Hu62], moments have been applied to pattern recognition problems. Features that are invariant under translation, rotation, and scaling can be derived from the two dimensional moments. The two dimensional moments represent a countable collection of weighted averages of an image as can be seen from the following definition:

Definition 1: The *two dimensional moment of* $(p+q)th$ order is defined by the following Riemman integral,

$$m_{pq} = \iint x^p y^q \rho(x,y)\,dx\,dy$$

for $p,q = 0,1,2,\ldots$, where ρ is a piecewise continuous function defined over a *finite region* in the plane.

Moments are a faithful representation of an image. If we consider the image in the xy plane to be a piecewise continuous function whose values are the intensity over the pixels of an image, then the following theorem [Hu62] asserts that the moment sequence can reconstruct the image, and furthermore the moments are unique.

Theorem 2: The double moment sequence $\{m_{pq}\}$ as defined above, is uniquely determined by $\rho(x,y)$ provided that the integration is carried over finite region. Conversely, $\rho(x,y)$ is uniquely determined by the set $\{m_{pq}\}$. See [Hu62] for the proof.

C.1.1. Central Moments

It is more convenient to work with central moments which correspond to moments computed about the centroid. In this section we define and relate the central moments to the moments.

Definition 3: The *Central moment* of $(p+q)$th order is defined by

$$\mu_{pq} = \iint (x-\bar{x})^p (y-\bar{y})^q \rho(x,y)dxdy \qquad p,q = 0,1,2,... \tag{C.1}$$

where $\bar{x} = \dfrac{m_{10}}{m_{00}}$ and $\bar{y} = \dfrac{m_{01}}{m_{00}}$.

It is clear from (C.1) that central moments μ_{pq} do not change under the following transformation,

$$x' \leftarrow x+\alpha$$

$$y' \leftarrow y+\beta$$

that is, $\mu'_{pq} = \mu_{pq}$ under translation of coordinates by α, β, where μ'_{pq} are the moments in the translated coordinate system.

Central moments can be obtained from the moments $\{m_{pq}\}$. To do so, consider (C.1) and use the binomial expansion, i.e.,

$$(x-\bar{x})^p = \sum_{k=0}^{p} \binom{p}{k} (-1)^k \bar{x}^k x^{p-k}$$

where

$$\binom{p}{k} = \frac{p!}{(p-k)!k!}.$$

One can obtain a similar expression for $(y-\bar{y})$. Substituting the binomial expansions in (C.1) we obtain

$$\mu_{pq} = \iint \sum_{k=0}^{p} \sum_{l=0}^{q} \binom{p}{k} \binom{q}{l} (-1)^{k+l} \bar{x}^k \bar{y}^l x^{p-k} y^{q-l} \rho(x,y)dxdy$$

by arranging the terms and interchanging the order of summation and integration we get

$$\mu_{pq} = \sum_{k=0}^{p} \sum_{l=0}^{q} (-1)^{k+l} \binom{p}{k} \binom{q}{l} \bar{x}^k \bar{y}^l m_{p-k,q-l} \tag{C.2}$$

Equation (C.2) yields the following relationships between central moments and moments of order less than or equal to three:

$$\mu_{00} = m_{00}$$

$$\mu_{01} = \mu_{10} \equiv 0$$

$$\mu_{20} = m_{20} - \bar{x} m_{10}$$

$$\mu_{11} = m_{11} - \bar{y}\bar{x} m_{00}$$

$$\mu_{02} = m_{02} - \bar{y} m_{01}$$

$$\mu_{30} = m_{30} - 3\bar{x} m_{20} + 2\bar{x}^2 m_{10}$$

$$\mu_{21} = m_{21} - 2\bar{x} m_{11} - \bar{y} m_{20} + 2\bar{x}^2 m_{01}$$

$$\mu_{12} = m_{12} - 2\bar{y} m_{11} - \bar{x} m_{02} + 2\bar{y}^2 m_{10}$$

$$\mu_{03} = m_{03} - 3\bar{y} m_{02} + 2\bar{y}^2 m_{01}$$

Since central moments are translation invariant, and are uniquely related to moments by (C.2), they can also represent images. Thus, from now on, we consider only central moments.

C.2. Moment Invariants

We now define the notion of invariance. The main idea is to obtain "invariants" under various transformations. Specifically, if an image undergoes a change in size, rotation or translation, the "invariants" do not change. Thus the image may be characterized independently of these changes.

Given an image over a region in the xy plane, moment invariants characterize the image independent of linear transformations. If we let μ be a particular set of moments and I be an invariance function over the set μ, that is, $I: \gamma \subseteq \mu \rightarrow R$,then the necessary condition for invariance can be stated as

$$I(\mu') = I(\mu)$$

where μ' is the resulting moment set under transformations.

In order to maintain theoretical consistency with [MM84], the approach followed in this report is that of Hu [Hu62], rather than Teague's [Te80], although the latter possess the advantage of simplicity.

Before we proceed with moment invariant derivation, the following definitions are necessary.

Definition 4: A homogeneous polynomial of the form

$$f \equiv a_{p,0}u^p + \begin{pmatrix} p \\ 1 \end{pmatrix} a_{p-1,1}u^{p-1}v + \cdots + \begin{pmatrix} p \\ p-1 \end{pmatrix} a_{1,p-1}uv^{p-1} + a_{0,p}v^p$$

is called a *binary algebraic form* and is denoted by

$$f \equiv (a_{p,0};a_{p-1,1};...;a_{0,p}) (u,v)$$

Definition 5: Let $(a_{p,0};a_{p-1,1};...;a_{0,p}) (u,v)$ be a binary algebraic form. A homogeneous polynomial $I(a_{p,0}, a_{p-1,1}, \ldots, a_{0,p})$ is called an *algebraic invariant of weight w* if

$$I(a'_{p,0}, a'_{p-1,0}, \ldots, a'_{0,p}) = \Delta^w I(a_{p,0}, a_{p-1,1}, \ldots, a_{0,p})$$

where $a'_{p,0}, a'_{p-1,1}, \ldots, a'_{0,p}$ are the resulting coefficients of the binary algebraic form when (u,v) is transformed into (u',v') by

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix} \begin{bmatrix} u' \\ v' \end{bmatrix}$$

and

$$\Delta = \det \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix} \neq 0.$$

Remark: If $w = 0$ then I is called an *absolute invariant,* otherwise I is called a *relative invariant.*

The key element in relating moments to the algebraic theory of invariants is the moment generating function. As shown below, the moment generating function can be expressed as an infinite sum of binary algebraic forms. Once this relation is expressed, the invariants are deduced from the theory of algebraic invariants.

With definition (4) we can express the moment generating function as an infinite sum of binary algebraic forms. Recall the moment generating function is given by,

$$M(u,v) = \iint e^{(ux+vy)} \rho(x,y) dx dy$$

where ρ is a piecewise continuous function having finite support. Expanding $e^{(ux+vy)}$ in power series, we obtain

$$M(u,v) = \iint \sum_{p=0}^{\infty} \frac{(ux+vy)^p}{p!} \rho(x,y) dx dy$$

and interchanging the order of summation and integration, we get

$$M(u,v) = \sum_{p=0}^{\infty} \frac{1}{p!} \iint (ux+vy)^p \rho(x,y) dx dy$$

$$M(u,v) = \sum_{p=0}^{\infty} \frac{1}{p!} \iint \sum_{k=0}^{\infty} \begin{bmatrix} p \\ k \end{bmatrix} u^{p-k} v^k x^{p-k} y^k \rho(x,y) dx dy$$

$$M(u,v) = \sum_{p=0}^{\infty} \frac{1}{p!} \sum_{k=0}^{p} \begin{bmatrix} p \\ k \end{bmatrix} u^{p-k} v^{k} \mu_{p-k,k}$$

and noting that the inner summation term is $(\mu_{p,0}; \mu_{p-1,1}; \ldots; \mu_{0,p})\,(u,v)$ to obtain

$$M(u,v) = \sum_{p=0}^{\infty} \frac{1}{p!} (\mu_{p,0}; \mu_{p-1,1}; \ldots; \mu_{0,p})\,(u,v) \tag{C.3}$$

When applied to moments [Hu62], the theory of invariants tells us how to extract features from an image such that the features remain unchanged if the image undergoes linear transformations. The following is a derivation of the invariants under the following transformations:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{C.4}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix} \begin{bmatrix} u' \\ v' \end{bmatrix} \quad ; \qquad ux + vy = u'x' + v'y'. \tag{C.5}$$

Let's consider the change in the moment generating function under (C.4) and (C.5). Recall equation (C.3)

$$M(u,v) = \sum_{p=0}^{\infty} \frac{1}{p!} (\mu_{p,0}; \ldots; \mu_{0,p})\,(u,v) \tag{C.6}$$

and the definition of the moment generating function

$$M(u,v) = \iint e^{(ux+vy)} \rho(x,y)\,dx\,dy \tag{C.7}$$

by applying the transformations (C.4) and (C.5) to (C.7), we obtain

$$M'(u',v') = \iint \sum_{p=0}^{\infty} \frac{1}{p!} (u'x' + v'y')^{p} \rho'(x',y') \frac{1}{|J|}\,dx'\,dy'$$

with

$$|J| = \left| \det \begin{bmatrix} \delta & -\beta \\ -\gamma & \alpha \end{bmatrix} \right| \qquad \rho' = \rho$$

Similarly we can obtain for equation (C.6)

$$M'(u',v') = \frac{1}{|J|} \sum_{p=0}^{\infty} \frac{1}{p!} (\mu'_{p,0}; \ldots; \mu'_{0,p})\,(u',v') \tag{C.8}$$

provided that the transformed moments are defined by

$$\mu'_{pq} = \iint x'^{p} y'^{q} \rho'(x',y')\,dx'\,dy' \qquad p,q = 0,1,2,\ldots$$

From the theory of algebraic invariants, the transformation law for the binary algebraic form

$$(\mu_{p,0};\ \mu_{p-1,1};\ \ldots;\ \mu_{0,p})\ (u,v)$$

is the same as that for

$$(ux + vy)^p = (x^p;\ x^{p-1}\ y;\ \ldots;y^p)\ (u,v).$$

Thus if we let $a_{p,0}, \ldots, a_{0,p}$ be the moments in (C.3), and combining with (C.6) and (C.8) we have the following theorem [Hu62].

Theorem 6: If the algebraic form of order p has an algebraic invariant of weight w, then the moments of order p have the same algebraic invariant with the additional factor $|J|$, that is,

$$I(\mu'_{p,0},\ldots,\mu'_{0,p}) = |J|\ \Delta^w I(\mu_{p,0},\ldots,\mu_{0,p})$$

C.2.1. Invariants under scale change

Consider the following *similitude* transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \alpha \in R$$

Each coefficient of any algebraic form is an algebraic invariant of weight $p+q$, that is

$$a'_{pq} = \alpha^{p+q}\ a_{pq}.$$

By applying Theorem 3 we obtain,

$$\mu'_{pq} = |J|\alpha^{p+q}\ \mu_{pq}$$

Under Similitude transformation we have $|J| = \alpha^2$, which yields

$$\mu'_{pq} = \alpha^2\ \alpha^{p+q}\ \mu_{pq}$$

combined with the zero-th order moment relation

$$\alpha = \sqrt{\frac{\mu'_{00}}{\mu_{00}}}$$

to yield,

$$\mu'_{pq} = \frac{\mu'_{00}}{\mu_{00}} \left(\frac{\mu'_{00}}{\mu_{00}}\right)^{\frac{p+q}{2}} \mu_{pq}$$

$$\frac{\mu'_{pq}}{\mu'^{\frac{p+q}{2}+1}_{00}} = \frac{\mu_{pq}}{\mu^{\frac{p+q}{2}+1}_{00}}. \qquad \text{(C.9)}$$

Equation (C.9) defines absolute moment invariants under similitude transformation.

Remark : Since $\mu_{10} = \mu_{01} \equiv 0$ equation (C.9) is nontrivial for $p+q = 2,3,....$

C.2.2. Moment Invariants under Orthogonal Transformations

Now we consider invariants under orthogonal transformations. Rotation of coordinate systems is an element of such transformations. Define the *proper* orthogonal transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad J = 1$$

then the moment invariants are [Hu62]:

$$I_{p,0} = \mu_{p0} - i\binom{p}{1}\mu_{p-1,1} - \binom{p}{2}\mu_{p-2,2} + i\binom{p}{3}\mu_{p-3,3} + \cdots + i^p\mu_{0p}$$

$$I_{p-1,1} = (\mu_{p0} + \mu_{p-2,2}) - i(p-2)(\mu_{p-1,1} + \mu_{p-3,3}) + \cdots + (-i)^{p+2}(\mu_{2,p-2} + \mu_{0p})$$

$$I_{p-2,2} = (\mu_{p0} + 2\mu_{p-2,2} + \mu_{p-4,4}) - i(p-4)(\mu_{p-1,1} + 2\mu_{p-3,3} + \mu_{p-5,5}) +$$

$$\cdots + (-i)^{p-4}(\mu_{4,p-4} + 2\mu_{2,p-2} + \mu_{0p})$$

$$I_{p-r,r} = [(\mu_{p0};\mu_{p-2,2};...;\mu_{p-2r,2r})(1,1)^r;\ (\mu_{p-1,1};\mu_{p-3,3};...;\mu_{p-2r-1,2r+1})(1,1)^r$$
$$;...;(\mu_{2r,p-2r};\mu_{2r+2,p-2r-2};...;\mu_{0p})(1,1)^r](1,-i)^{p-2r} \qquad p-2r>0$$

$$I_{\frac{p}{2},\frac{p}{2}} = \mu_{p0} + \binom{p/2}{1}\mu_{p-2,2} + \binom{p/2}{2}\mu_{p-4,4} + \cdots + \mu_{0p} \qquad \textit{for even } p$$

$$I_{r,p-r} = I^*_{p-r,r}$$

where * is complex conjugation.

The invariants obtained from the second-order moments are given by I_{11} and $I_{02}I_{20}$. From the third order moments, we get $I_{30}I_{03}$, $I_{21}I_{12}$, $(I_{30}I_{12}^3 + I_{03}I_{21}^3)$, and $\frac{1}{i}(I_{30}I_{12}^3 - I_{03}I_{21}^3)$, a skew invariant. Another invariant may be derived from a combination of second and third order invariants: $(I_{20}I^2_{21} + I_{02}I^2_{21})$.

In general for $p \geq 4$ there are $\left\lceil \frac{p}{4} \right\rceil$ invariants given by $I_{p0}\, I_{0p}$, $I_{p-1,1}\, I_{1,p-1}, \ldots I_{p-r,r}\, I_{r,p-r}$, and also, when p is even, $I_{\frac{p}{2},\frac{p}{2}}$, where $\lceil x \rceil$ is the smallest integer greater than or equal to x. Also combined with $(p-2)$ moments we have $[\frac{p}{2} - 1\,]$ invariants,

$$(I_{p-1,1}\, I_{0,p-2} + I_{1,p-1}\, I_{p-2,0}),$$
$$(I_{p-2,2}\, I_{1,p-3} + I_{2,p-2}\, I_{p-3,1})$$
$$\ldots\ldots$$
$$(I_{p-r,r}\, I_{r-1,p-r+1} + I_{r,p-r}\, I_{p-r+1,r-1}) \qquad p - 2r > 0$$

combined with second order moments,

$I^2_{[\frac{p}{2}],[\frac{p}{2}]+1}\, I_{20} + I^2_{[\frac{p}{2}]+1,[\frac{p}{2}]}\, I_{02}$ if p is odd

$I_{\frac{p}{2}-1,\frac{p}{2}+1}\, I_{20} + I_{\frac{p}{2}+1,\frac{p}{2}-1}\, I_{02}$ if p is even

which give us a total of $(p+1)$ independent invariants.

C.2.3. Summary of moment invariants

The central moments $\{\mu_{pq}\}$ are invariant to translation of coordinate system. The invariants under similitude transformation (scale) are defined by equation (C.9). In Section C.2.2 we listed the invariants under orthogonal transformation. These can be combined to produce scale, rotation and translation invariants as follows:

Let $\phi_1, \ldots, \phi_7$ be the invariance functions for second and third order moments, (obtained by evaluating the second and third order invariants from section C.2.2) then

$$\phi_1 = (\mu_{20} + \mu_{02}) \tag{C.10}$$
$$\phi_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \tag{C.11}$$
$$\phi_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \tag{C.12}$$
$$\phi_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \tag{C.13}$$
$$\phi_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] \tag{C.14}$$
$$+ (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$$
$$\phi_6 = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] \tag{C.15}$$

$$+ 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$$

$$\phi_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] \qquad \text{(C.16)}$$

$$- (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$$

Equations (C.10) through (C.16) are the invariants under translation and rotation. To make them invariant to scale change, normalize according to (C.9), that is, define η_{pq} by,

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{p+q}{2}+1}} \qquad \text{(C.17)}$$

and replace μ_{pq} by η_{pq} in (C.10) through (C.16).

Note: The normalization given by (C.17) disagrees with the normalization given in Table 9 of [MM84]. We also note typographical error in the normalization equation (30) of [Hu62].

C.3. Maitra invariants

Moment that are invariants under scale, rotation and translation may be sufficient to characterize the image. However, if it is desired to have invariants under illumination changes, the new invariants must incorporate the illumination conditions.

Maitra [Ma79], has considered this problem and obtained illumination invariants denoted by *Maitra invariants*. These invariants incorporate scale, rotation, translation and illumination changes and are based on moment invariants.

Let $g_1(x,y)$ and $g_2(x,y)$ be two grey value images related by the following transformations,

$$g_1(x,y) = k\, g_2(x',y') \qquad k \neq 0 \qquad \text{(C.18)}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \alpha \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \qquad \text{(C.19)}$$

where θ is an angle of rotation, (a,b) is translation, α is a scale factor and k is the change in illumination.

To obtain Maitra invariants, compute the moment invariants for $g_1(x,y)$ and $g_2(x',y')$ to obtain $\phi_1,\ldots,\phi_7$ and $\phi'_1,\ldots,\phi'_7$ respectively. The relations among the invariants are,

$$\phi_1 = \frac{k}{\alpha^4}\,\phi'_1 \qquad \text{(C.20)}$$

$$\phi_2 = \frac{k^2}{\alpha^8}\,\phi'_2 \qquad \text{(C.21)}$$

$$\phi_3 = \frac{k^2}{\alpha^{10}} \phi'_3 \tag{C.22}$$

$$\phi_4 = \frac{k^2}{\alpha^{10}} \phi'_4 \tag{C.23}$$

$$\phi_5 = \frac{k^4}{\alpha^{20}} \phi'_5 \tag{C.24}$$

$$\phi_6 = \frac{k^3}{\alpha^{14}} \phi'_6 \tag{C.25}$$

$$\phi_7 = \frac{k^4}{\alpha^{20}} \phi'_7 \tag{C.26}$$

with the zero-th order moments μ_{00} , μ'_{00}

$$\mu_{00} = \frac{k}{\alpha^2} \mu'_{00}.$$

By eliminating the constants from (C.20) through (C.26) the following become invariants under (C.18) and (C.19) called Maitra invariants.

$$\beta_1 = \frac{\sqrt{\phi_2}}{\phi_1} \tag{C.27}$$

$$\beta_2 = \frac{\phi_3 \mu_{00}}{\phi_2 \phi_1} \tag{C.28}$$

$$\beta_3 = \frac{\phi_4}{\phi_3} \tag{C.29}$$

$$\beta_4 = \frac{\sqrt{\phi_5}}{\phi_4} \tag{C.30}$$

$$\beta_5 = \frac{\phi_6}{\phi_4 \phi_1} \tag{C.31}$$

$$\beta_6 = \frac{\phi_7}{\phi_5} \tag{C.32}$$

Equation (C.27) through (C.32) are invariant under (C.18) and (C.19). However β_4 is undefined when ϕ_5 is negative. To eliminate the ambiguity caused by the square-root, we will use

$$\beta_4 = \frac{\phi_5}{\phi_4^2}$$

in addition to β_1, β_2, β_3, β_5, β_6, as defined in (C.27)-(C.29), (C.31), (C.32).

We note another disagreement in β_2 of Table 10 in [MM84].

C.4. Conclusions

Maitra's invariants are invariant under the transformations given in equation (C.18) and (C.19), and thus the moment invariants are redundant as features but are necessary to obtain Maitra invariants.

The moment invariants are truly invariant only in the continuous domain. For the case of digital imagery, some of the invariance is lost due to undersampling and quantization errors. Both these errors decrease as the image size increases [TeCh86]. Maitra's invariants tend to be less sensitive to the size of an image. It is of course possible to investigate the various effects that influence the invariance property of different types of moments.

APPENDIX D: A PARTIAL LIST OF TWIN ROUTINES

A partial list of TWIN routines for creating and manipulating TWIN objects:

Function Name	Description
Cone	Create a boundary representation of a cone approximation.
Cylinder	Create a boundary representation of a cylinder approximation.
EllCone	Create a boundary representation of an elliptical cone approximation.
Ellipse	Create a boundary representation of an ellipsoid approximation.
Fillet	Create a boundary representation of a fillet shape.
Ppiped	Create a boundary representation of parallelpiped (box).
SolidTorus	Create a boundary representation of a solid torus approximation (torus with no hole).
Sphere	Create a boundary representation of a sphere approximation.
Torus	Create a boundary representation of a torus approximation.
Wedge	Create a boundary representation of a wedge.
Combine	Perform a boolean operation on two TWIN objects. The operation can either be union, subtraction, or intersection.
ReadObj	Read a TWIN object from a file and return a pointer to it.
WriteObj	Write a TWIN object to a file.
render	Create a rendering of a TWIN object.

Appendix.E.1


```
%%%
```

```
%%
%% Present program based on Figure 14.12 of Bratko
%%

% display the conclusion of a consultation and 'how' explanation
present(Answer) :-
        nl, showconclusion(Answer),
        nl, write('Would you like to see how? '),
        getreply(Reply),
        ( Reply = yes, !, show_how(Answer);      % Show solution tree
          true ).

% Get a meaningful answer from the user
getreply(Reply) :-
        read(Answer),
        means(Answer, Meaning), !,      % Answer means something?
                % Yes: valid reply
                Reply = Meaning;
                % No: try again
                nl, write('Answer  unknown, please try again.'), nl,
                getreply(Reply).

means(yes, yes).
means(y, yes).
means(no, no).
means(n, no).
means(why, why).
means(w, why).

% Show conclusion and confidence value
showconclusion((Conclusion : Prob) was Found) :-
        write(Conclusion), write(' : '), write(Prob).

% Show how solution was derived
show_how(Solution) :-
        nl, show_how(Solution, 0), !.           % Indent by 0

show_how(Answer1 and Answer2, H) :- !,          % Indent by H
        show_how(Answer1, H),
        tab(H), write(and), nl,
        show_how(Answer2, H).

show_how(Answer1 or Answer2, H) :- !,           % Indent by H
        tab(H), write('('), write(P), nl,
        show_how(Answer1, H),
        tab(H), write(or), nl,
        show_how(Answer2, H).

show_how((not Answer : P) was Found, H) :-      % Indent by H
        tab(H), write('not '), write(Answer), write(' : '), write(P),
        write(' was '),
        show_evidence(Found, H).                % Show evidence

show_how((Answer : P) was Found, H) :-          % Indent by H
        tab(H), write(Answer), write(' : '), write(P),
        write(' was '),
        show_evidence(Found, H).                % Show evidence

show_evidence(Derived from Answer, H) :- !,
        write(Derived), write(' from'),         % Show rule name
        nl, H1 is H + 5,
        show_how(Answer, H1).                   % Show antecedant

show_evidence('derived from' Answer, H) :- !,
        write('derived from'),                  % Show rule name
        nl, H1 is H + 5,
        show_how(Answer, H1).                   % Show antecedant

show_evidence(Found, _) :-              % Found = 'told' or 'found as fact'
        write(Found), nl.


%%%
%%%  Utilities : various useful prolog subroutines
%%%


% member(Item, List): succeeds if Item is a member of List.
%       Item: element whose membership is to be determined
%       List: list to be checked

member(Item, [Item | Tail]).    % it is if it is the first item

member(Item, [Head | Tail]) :-  % otherwise check the rest of list
        member(Item, Tail).


% del(X, L1, L2): delete element X from list L1 to produce result list L2
%       X: element to be deleted
%       L1: list to delete it from
%       L2: resulting output list

del(X, [X | L], L).

del(X, [Y | L], [Y | L1]) :-
        del(X, L, L1).


% conc(L1, L2, L3): concatenate L2 onto L1 to obtain L3
%       L1: first input list
%       L2: second input list to be appended to L1
%       L3: output list that is concatenation of L1 & L2

conc([],L,L).

conc([X | L1], L2, [X | L3]) :-
        conc(L1,L2,L3).


% min(X, Y, Z): the minimum value of (X, Y) is Z
%       X, Y: the input elements whose minimum is to be returned
%       Z: output element equal to the minimum value between X & Y

min(X, Y, X) :-
        X =< Y, !.

min(X, Y, Y).


% max(X, Y, Z): the maximum value of (X, Y) is Z
%       X, Y: the input elements whose maximum is to be returned
%       Z: output element equal to the maximum value between X & Y

max(X, Y, X) :-
        X >= Y, !.

max(X, Y, Y).
```

Appendix.E.2


```
%%
%% A knowledge base for identifying tanks and apc's from range images
%%

% operator definitions

:- op(100, xfx, isa).

% production rules

rule1 : if
                medium_turret(S1, S2, S3, S4)
                and
                medium_deck(S5, S6)
                and
                medium_track(S7)
                and
                bmp_adjacencies(S1, S2, S3, S4, S5, S6, S7)
        then
                target isa bmp
        with
                strength(0.001, 3000).

rule2 : if
                small_turret(S1, S2, S3)
                and
                large_deck(S4, S5, S6)
                and
                car_front(S7, S8)
                and
                car_side(S9, S10, S11)
                and
                brdm2_adjacencies(S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11)
        then
                target isa brdm2
        with
                strength(0.01, 4000).

rule3 : if
                large_side(S1)
                and
                small_track(S2)
                and
                m113_adjacencies(S1, S2)
        then
                target isa m113
        with
                strength(0.0001, 2000).

rule4 : if
                large_turret(S1, S2, S3, S6)
                and
                small_deck(S4)
                and
                large_track(S5)
                and
                m60a1_adjacencies(S1, S2, S3, S4, S5, S6)
        then
                target isa m60a1
        with
                strength(0.001, 3000).

rule5 : if
                small_dome(S1, S2, S3)
        then
                small_turret(S1, S2, S3)
        with
                strength(0.001, 1000).

rule6 : if
                medium_dome(S1, S2, S3)
                and
                small_gunbarrel(S4)
                and
                adjacent(S3, S4, jmp)
        then
                medium_turret(S1, S2, S3, S4)
        with
                strength(0.5, 2000).

rule7 : if
                hatch(S1)
                and
                adjacent(S1, S2, jmp)
                and
                turret_body(S2)
                and
                adjacent(S2, S3, snd)
                and
                turret_front(S3)
                and
                adjacent(S3, S4, jmp)
                and
                large_gunbarrel(S4)
        then
                large_turret(S1, S2, S3, S4)
        with
                strength(0.2, 4000).

rule8 : if
                elevation(S, 1.8) and height(S, 0.4) and width(S, 1.75)
                and
                area(S, 0.7) and planar(S)
        then
                small_deck(S)
        with
                strength(0.2, 10000).

rule9 : if
                med_rear_deck(S1)
                and
                adjacent(S1, S2, snd)
                and
                med_fore_deck(S2)
        then
                medium_deck(S1, S2)
        with
                strength(0.01, 2000).

rule10 : if
                lg_rear_deck(S1)
                and
                adjacent(S1, S2, snd)
                and
                lg_mid_deck(S2)
                and
                adjacent(S2, S3, snd)
```

Appendix.E.1


```
% numbervars(T, N, M): number the variables in term T from N to M.
%       T: term to number the variables in
%       N: number of first variable (input)
%       M: number of last variable (output)

numbervars(Term, N, Nplus1) :-
        var(Term), !,                   % Variable?
        Term = var/N,
        Nplus1 is N + 1.

numbervars(Term, N, M) :-
        Term =.. [Functor | Args],      % Structure or atomic
        numberargs(Args, N, M).         % Number variables in arguments

numberargs([], N, N) :- !.

numberargs([X | L], N, M) :-
        numbervars(X, N, N1),
        numberargs(L, N1, M).
```

Appendix.E.2


```
                and
                lg_fore_deck(S3)
        then
                large_deck(S1, S2, S3)
        with
                strength(0.1, 5000).

rule11 : if
                elevation(S, 0.35) and height(S, 0.7) and width(S, 5.3)
                and
                area(S, 3.08) and planar(S)
        then
                small_track(S)
        with
                strength(0.05, 10000).

rule12 : if
                elevation(S, 0.5) and height(S, 1.0) and width(S, 6.0)
                and
                area(S, 5.5) and planar(S)
        then
                medium_track(S)
        with
                strength(0.05, 10000).

rule13 : if
                elevation(S, 0.8) and height(S, 1.6) and width(S, 6.94)
                and
                area(S, 10.3) and planar(S)
        then
                large_track(S)
        with
                strength(0.05, 10000).

rule14 : if
                top_front(S1)
                and
                adjacent(S1, S2, snd)
                and
                bottom_front(S2)
        then
                car_front(S1, S2)
        with
                strength(0.01, 2000).

rule15 : if
                med_side(S1)
                and
                adjacent(S1, S2, crv)
                and
                wheel(S2)
                and
                adjacent(S1, S3, crv)
                and
                wheel(S3)
                and
                xdiff(S2, S3, 3.10)
        then
                car_side(S1, S2, S3)
        with
                strength(0.05, 3000).

rule16 : if
                elevation(S, 1.35) and height(S, 1.30) and width(S, 5.3)
                and
                area(S, 6.2) and planar(S)
        then
                large_side(S)
        with
                strength(0.001, 5000).

rule17 : if
                adjacent(S1, S5, jmp)
                and
                adjacent(S3, S6, jmp)
                and
                adjacent(S5, S7, snd)
                and
                adjacent(S6, S7, snd)
        then
                bmp_adjacencies(S1, S2, S3, S4, S5, S6, S7)
        with
                strength(0.001, 10000).

rule18 : if
                adjacent(S1, S5, jmp)
                and
                adjacent(S2, S5, jmp)
                and
                adjacent(S3, S5, jmp)
                and
                adjacent(S4, S9, snd)
                and
                adjacent(S5, S9, snd)
                and
                adjacent(S6, S7, snd)
                and
                adjacent(S7, S9, snd)
                and
                adjacent(S8, S9, snd)
        then
                brdm2_adjacencies(S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11)
        with
                strength(0.001, 10000).

rule19 : if
                adjacent(S1, S2, jmp)
        then
                m113_adjacencies(S1, S2)
        with
                strength(0.001, 10000).

rule20 : if
                adjacent(S2, S4, jmp)
                and
                adjacent(S2, S5, jmp)
                and
                adjacent(S3, S5, jmp)
                and
                adjacent(S4, S5, jmp)
        then
                m60a1_adjacencies(S1, S2, S3, S4, S5, S6)
        with
                strength(0.001, 10000).

rule21 : if
                elevation(S1, 2.15) and area(S1, 0.14) and planar(S1)
                and
```

Appendix.E.2


```
                adjacent(S1, S2, snd)
                and
                elevation(S2, 2.15) and area(S2, 0.14) and planar(S2)
                and
                adjacent(S2, S3, snd)
                and
                elevation(S3, 2.15) and area(S3, 0.14) and planar(S3)
        then
                small_dome(S1, S2, S3)
        with
                strength(0.1, 5000).

rule22 : if
                elevation(S1, 1.75) and area(S1, 0.2) and planar(S1)
                and
                adjacent(S1, S2, snd)
                and
                elevation(S2, 1.75) and area(S2, 0.2) and planar(S2)
                and
                adjacent(S2, S3, snd)
                and
                elevation(S3, 1.75) and area(S3, 0.2) and planar(S3)
        then
                medium_dome(S1, S2, S3)
        with
                strength(0.1, 5000).

rule23 : if
                elevation(S, 1.75)
                and
                height_to_width(S, 0.05'2)
                and
                area(S, 0.0857)
        then
                small_gunbarrel(S)
        with
                strength(0.5, 5000).

rule24 : if
                elevation(S, 3.135) and height(S, 0.27) and width(S, 1.35)
                and
                area(S, 0.3645) and planar(S)
        then
                hatch(S)
        with
                strength(0.2, 2000).

rule25 : if
                elevation(S, 2.3) and height(S, 1.4) and width(S, 2.0)
                and
                area(S, 2.8) and planar(S)
        then
                turret_body(S)
        with
                strength(0.001, 1000).

rule26 : if
                elevation(S, 2.3) and height(S, 1.4) and width(S, 1.2)
                and
                area(S, 1.26) and planar(S)
        then
                turret_front(S)
        with
                strength(0.01, 2000).

rule27 : if
                elevation(S, 2.15)
                and
                height_to_width(S, 0.0235)
                and
                area(S, 0.426)
        then
                large_gunbarrel(S)
        with
                strength(0.5, 5000).

rule28 : if
                elevation(S, 1.35) and height(S, 0.4) and width(S, 2.9)
                and
                area(S, 0.58) and planar(S)
        then
                med_rear_deck(S)
        with
                strength(0.1, 2000).

rule29 : if
                elevation(S, 1.25) and height(S, 0.5) and width(S, 6.5)
                and
                area(S, 2.67) and planar(S)
        then
                med_fore_deck(S)
        with
                strength(0.001, 1000).

rule30 : if
                elevation(S, 1.7) and height(S, 0.3) and width(S, 1.75)
                and
                area(S, 0.525) and planar(S)
        then
                lg_rear_deck(S)
        with
                strength(0.1, 2000).

rule31 : if
                elevation(S, 1.775) and height(S, 0.45) and width(S, 1.9)
                and
                area(S, 0.855) and planar(S)
        then
                lg_mid_deck(S)
        with
                strength(0.001, 1000).

rule32 : if
                elevation(S, 1.775) and height(S, 0.45) and width(S, 0.38)
                and
                area(S, 0.17) and planar(S)
        then
                lg_fore_deck(S)
        with
                strength(0.5, 3000).

rule33 : if
                elevation(S, 1.425) and height(S, 0.25) and width(S, 0.95)
                and
                area(S, 0.68) and planar(S)
        then
                top_front(S)
        with
```

```
                strength(0.5, 3000).

rule34 : if
                elevation(S, 0.95) and height(S, 0.80) and width(S, 0.40)
                and
                area(S, 0.32) and planar(S)
        then
                bottom_front(S)
        with
                strength(0.5, 3000).

rule35 : if
                elevation(S, 1.05) and height(S, 1.0) and width(S, 5.2)
                and
                area(S, 4.0) and planar(S)
        then
                med_side(S)
        with
                strength(0.001, 2000).

rule36 : if
                elevation(S, 0.55) and height(S, 1.1) and width(S, 1.1)
                and
                area(S, 1.0) and planar(S)
        then
                wheel(S)
        with
                strength(0.01, 1000).

% facts

fact : elevation(S, E) :-
        attr_location(S, X, Y),
        Y < 1.02 * E,
        Y > 0.98 * E.

fact : xdiff(S1, S2, D) :-
        attr_location(S1, X1, Y1),
        attr_location(S2, X2, Y2),
        Xdiff is X2 - X1,
        Xdiff < 1.02 * D,
        Xdiff > 0.98 * D.

fact : ydiff(S1, S2, D) :-
        attr_location(S1, X1, Y1),
        attr_location(S2, X2, Y2),
        Ydiff is Y2 - Y1,
        Ydiff < 1.02 * D,
        Ydiff > 0.98 * D.

fact : height(S, H) :-
        attr_hwa(S, Height, Width, Area),
        Height < 1.02 * H,
        Height > 0.98 * H.

fact : width(S, W) :-
        attr_hwa(S, Height, Width, Area),
        Width < 1.02 * W,
        Width > 0.98 * W.

fact : area(S, A) :-
        attr_hwa(S, Height, Width, Area),
        Area < 1.02 * A,
        Area > 0.98 * A.

fact : height_to_width(S, HW) :-
        attr_hwa(S, Height, Width, Area),
        Ratio is Height / Width,
        Ratio < 1.02 * HW,
        Ratio > 0.98 * HW.

fact : planar(S) :-
        attr_fit(S, Error),
        Error < 10000,
        Error > 0.

fact : adjacent(S1, S2, Etype) :-
        rel_adjacent(S1, S2, Etype);
        rel_adjacent(S2, S1, Etype).

% a priori probabilities

prior(target isa bmp, 0.25).
prior(target isa brdm2, 0.25).
prior(target isa m113, 0.25).
prior(target isa m60a1, 0.25).
prior(small_turret(_,_,_), 0.0625).
prior(medium_turret(_,_,_,_), 0.08333).
prior(large_turret(_,_,_,_), 0.08333).
prior(small_deck(_), 0.08333).
prior(medium_deck(_,_), 0.08333).
prior(large_deck(_,_,_), 0.0625).
prior(small_track(_), 0.125).
prior(medium_track(_), 0.08333).
prior(large_track(_), 0.08333).
prior(car_front(_,_), 0.0625).
prior(car_side(_,_,_), 0.0625).
prior(large_side(_), 0.125).
prior(bmp_adjacencies(_,_,_,_,_,_,_,_), 0.08333).
prior(brdm2_adjacencies(_,_,_,_,_,_,_,_,_,_,_,_), 0.0625).
prior(m113_adjacencies(_,_), 0.125).
prior(m60a1_adjacencies(_,_,_,_,_,_,_), 0.08333).
prior(small_dome(_,_,_), 0.0625).
prior(medium_dome(_,_,_), 0.04167).
prior(small_gunbarrel(_), 0.04167).
prior(hatch(_), 0.02083).
prior(turret_body(_), 0.02083).
prior(turret_front(_), 0.02083).
prior(large_gunbarrel(_), 0.02083).
prior(med_rear_deck(_), 0.04167).
prior(med_fore_deck(_), 0.04167).
prior(lg_rear_deck(_), 0.02083).
prior(lg_mid_deck(_), 0.02083).
prior(lg_fore_deck(_), 0.02083).
prior(top_front(_), 0.03125).
prior(bottom_front(_), 0.03125).
prior(med_side(_), 0.02083).
prior(wheel(_), 0.02083).
prior(elevation(_,_), 0.00001).
prior(xdiff(_,_,_), 0.00001).
prior(ydiff(_,_,_), 0.00001).
prior(height(_,_), 0.00001).
prior(width(_,_), 0.00001).
prior(area(_,_), 0.00001).
prior(height_to_width(_,_), 0.00001).
prior(planar(_), 0.93).
prior(adjacent(_,_,_), 0.00001).
```

```
;;;
;;; A set of rules for identifying tanks and apc's from range images
;;;


;; Flow of Control Production Rules

(p Phase-Finished                                                    ; 001
   { (Phase ^status active) <phase> }
-->
   (modify <phase> ^status finished)
                                                                   ) ; 001

(p Expand-to-Hypothesize                                             ; 002
   { (Phase ^description expand ^status finished) <phase> }
-->
   (modify <phase> ^description hypothesize ^status active)
                                                                   ) ; 002

(p Hypothesize-to-Build                                              ; 003
   { (Phase ^description hypothesize ^status finished) <phase> }
-->
   (modify <phase> ^description build ^status active)
                                                                   ) ; 003

(p Build-to-Clean_Up                                                 ; 004
   { (Phase ^description build ^status finished) <phase> }
-->
   (modify <phase> ^description clean ^status active)
                                                                   ) ; 004

(p Output_Results                                                    ; 005
   { (Phase ^description clean ^status finished) <phase> }
-->
   (remove <phase>)
                                                                   ) ; 005


;; Phase 1 Production Rules : Fill in missing surface attributes & relations

(p Expand-H_to_W                                                     ; 101
   (Phase ^description expand ^status active)
   { (Surface ^h_to_w nil ^height <h> ^width <w>) <s> }
-->
   (modify <s> ^h_to_w (compute <h> // <w>) )
                                                                   ) ; 101

(p Expand-Adjacent_1                                                 ; 102
   (Phase ^description expand ^status active)
 - (Adjacent ^first 0 ^second 0)
-->
   (make Adjacent ^edge_type jmp ^first 0 ^second 0)
   (make Adjacent ^edge_type snd ^first 0 ^second 0)
   (make Adjacent ^edge_type crv ^first 0 ^second 0)
                                                                   ) ; 102

(p Expand-Adjacent_2                                                 ; 103
   (Phase ^description expand ^status active)
   (Surface ^id { <s>  <> nil  <> 0 })
 - (Adjacent ^first <s> ^second 0)
-->
   (make Adjacent ^edge_type jmp ^first <s> ^second 0)
   (make Adjacent ^edge_type snd ^first <s> ^second 0)
   (make Adjacent ^edge_type crv ^first <s> ^second 0)
                                                                   ) ; 103

(p Expand-Adjacent_3                                                 ; 104
   (Phase ^description expand ^status active)
   (Adjacent ^edge_type <type> ^first <s1> ^second <s2>)
 - (Adjacent ^edge_type <type> ^first <s2> ^second <s1>)
-->
   (make Adjacent ^edge_type <type> ^first <s2> ^second <s1>)
                                                                   ) ; 104


;; Phase 2 Production Rules : Hypothesize Surface Identities

(p H-medium_dome_panel                                               ; 201
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type medium_dome_panel ^confidence <conf> ^surfaces 0)
   (Surface     ^id           { <s>  <> nil  <> 0 }
                ^y_loc        { > 1.715  < 1.785 }    ; elevation 1.75 +- 2%
                ^area         { > 0.196  < 0.204 }    ; area 0.2 +- 2%
                ^fit_error    { < 1000 }              ; planar
                )
-->
   (make Construct      ^type medium_dome_panel
                        ^confidence (compute <number>  * <conf>)
                        ^surfaces <s>                )
                                                                   ) ; 201

(p H-small_gunbarrel                                                 ; 202
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type small_gunbarrel ^confidence <conf> ^surfaces 0)
   (Surface     ^id           { <s>  <> nil  <> 0 }
                ^y_loc        { > 1.715  < 1.785 }    ; elevation 1.75 +- 2%
                ^area         { > 0.0840  < 0.0874 }  ; area 0.0857 +- 2%
                ^h_to_w       { > 0.05606  < 0.05834 }; h_to_w 0.0572 +- 2%
                )
-->
   (make Construct      ^type small_gunbarrel
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                )
                                                                   ) ; 202

(p H-med_rear_deck                                                   ; 203
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type med_rear_deck ^confidence <conf> ^surfaces 0)
   (Surface     ^id           { <s>  <> nil  <> 0 }
                ^y_loc        { > 1.323  < 1.377 }    ; elevation 1.35 +- 2%
                ^height       { > 0.392  < 0.408 }    ; height 0.4 +- 2%
                ^width        { > 2.842  < 2.958 }    ; width 2.9 +- 2%
                ^area         { > 0.5684  < 0.5916 }  ; area 0.58 +- 2%
                ^fit_error    { < 1000 }              ; planar
                )
-->
   (make Construct      ^type med_rear_deck
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                )
                                                                   ) ; 203

(p H-med_fore_deck                                                   ; 204
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
```

```
   (Construct ^type med_fore_deck ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 1.225  < 1.275 }    ; elevation 1.25 +- 2%
                ^height          { > 0.49  < 0.51 }      ; height 0.5 +- 2%
                ^width           { > 6.37  < 6.63 }      ; width 6.5 +- 2%
                ^area            { > 2.6166  < 2.7234 }  ; area 2.67 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type med_fore_deck
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 204

(p H-medium_track                                                   ; 205
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type medium_track ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 0.49  < 0.51 }      ; elevation 0.5 +- 2%
                ^height          { > 0.98  < 1.02 }      ; height 1.0 +- 2%
                ^width           { > 5.88  < 6.12 }      ; width 6.0 +- 2%
                ^area            { > 5.39  < 5.61 }      ; area 5.5 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type medium_track
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 205

(p H-small_dome_panel                                               ; 206
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type small_dome_panel ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 2.107  < 2.193 }    ; elevation 2.15 +- 2%
                ^area            { > 0.1372  < 0.1428 }  ; area 0.14 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type small_dome_panel
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 206

(p H-lg_rear_deck                                                   ; 207
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type lg_rear_deck ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 1.666  < 1.734 }    ; elevation 1.7 +- 2%
                ^height          { > 0.294  < 0.306 }    ; height 0.3 +- 2%
                ^width           { > 1.715  < 1.785 }    ; width 1.75 +- 2%
                ^area            { > 0.5145  < 0.5355 }  ; area 0.525 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type lg_rear_deck
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 207

(p H-lg_mid_deck                                                    ; 208
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type lg_mid_deck ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 1.7395  < 1.8105 }  ; elevation 1.775 +- 2%
                ^height          { > 0.441  < 0.459 }    ; height 0.45 +- 2%
                ^width           { > 1.862  < 1.938 }    ; width 1.9 +- 2%
                ^area            { > 0.8379  < 0.8721 }  ; area 0.855 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type lg_mid_deck
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 208

(p H-lg_fore_deck                                                   ; 209
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type lg_fore_deck ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 1.7395  < 1.8105 }  ; elevation 1.775 +- 2%
                ^height          { > 0.441  < 0.459 }    ; height 0.45 +- 2%
                ^width           { > 0.3724  < 0.3876 }  ; width 0.38 +- 2%
                ^area            { > 0.1666  < 0.1734 }  ; area 0.17 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type lg_fore_deck
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 209

(p H-top_front                                                      ; 210
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type top_front ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 1.3965  < 1.4535 }  ; elevation 1.425 +- 2%
                ^height          { > 0.245  < 0.255 }    ; height 0.25 +- 2%
                ^width           { > 0.931  < 0.969 }    ; width 0.95 +- 2%
                ^area            { > 0.6664  < 0.6936 }  ; area 0.68 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type top_front
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                    )
                                                                    ) ; 210

(p H-bottom_front                                                   ; 211
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type bottom_front ^confidence <conf> ^surfaces 0)
   (Surface     ^id              { <s>  <> nil  <> 0 }
                ^y_loc           { > 0.931  < 0.969 }    ; elevation 0.95 +- 2%
                ^height          { > 0.784  < 0.816 }    ; height 0.8 +- 2%
                ^width           { > 0.392  < 0.408 }    ; width 0.4 +- 2%
                ^area            { > 0.3136  < 0.3264 }  ; area 0.32 +- 2%
                ^fit_error       { < 1000 }              ; planar
                                                         )
-->
   (make Construct      ^type bottom_front
                        ^confidence (compute <number> * <conf>)
```

```
                    ^surfaces <s>                )
                                                                    ) ; 211

(p H-med_side                                                       ; 212
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type med_side ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 1.029  < 1.071 }    ; elevation 1.05 +- 2%
                ^height         { > 0.98  < 1.02 }      ; height 1.0 +- 2%
                ^width          { > 5.096  < 5.304 }    ; width 5.2 +- 2%
                ^area           { > 3.92  < 4.08 }      ; area 4.0 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type med_side
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 212

(p H-wheel                                                          ; 213
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type wheel ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 0.539  < 0.561 }    ; elevation 0.55 +- 2%
                ^height         { > 1.078  < 1.122 }    ; height 1.1 +- 2%
                ^width          { > 1.078  < 1.122 }    ; width 1.1 +- 2%
                ^area           { > 0.98  < 1.02 }      ; area 1.0 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type wheel
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 213

(p H-large_side                                                     ; 214
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type large_side ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 1.323  < 1.377 }    ; elevation 1.35 +- 2%
                ^height         { > 1.274  < 1.326 }    ; height 1.3 +- 2%
                ^width          { > 5.194  < 5.406 }    ; width 5.3 +- 2%
                ^area           { > 6.076  < 6.324 }    ; area 6.2 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type large_side
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 214

(p H-small_track                                                    ; 215
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type small_track ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 0.343  < 0.357 }    ; elevation 0.35 +- 2%
                ^height         { > 0.686  < 0.714 }    ; height 0.7 +- 2%
                ^width          { > 5.194  < 5.406 }    ; width 5.3 +- 2%
                ^area           { > 3.0184  < 3.1416 }  ; area 3.08 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type small_track
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 215

(p H-hatch                                                          ; 216
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type hatch ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 3.0723  < 3.1977 }  ; elevation 3.135 +- 2%
                ^height         { > 0.2646  < 0.2754 }  ; height 0.27 +- 2%
                ^width          { > 1.323  < 1.377 }    ; width 1.35 +- 2%
                ^area           { > 0.3572  < 0.3718 }  ; area 0.3645 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type hatch
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 216

(p H-turret_body                                                    ; 217
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type turret_body ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 2.254  < 2.346 }    ; elevation 2.3 +- 2%
                ^height         { > 1.372  < 1.428 }    ; height 1.4 +- 2%
                ^width          { > 1.96  < 2.04 }      ; width 2.0 +- 2%
                ^area           { > 2.744  < 2.856 }    ; area 2.8 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type turret_body
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 217

(p H-turret_front                                                   ; 218
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type turret_front ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 2.254  < 2.346 }    ; elevation 2.3 +- 2%
                ^height         { > 1.372  < 1.428 }    ; height 1.4 +- 2%
                ^width          { > 1.176  < 1.224 }    ; width 1.2 +- 2%
                ^area           { > 1.2348  < 1.2852 }  ; area 1.26 +- 2%
                ^fit_error      { < 1000 }              ; planar
                                                        )
-->
   (make Construct      ^type turret_front
                        ^confidence (compute <number> * <conf>)
                        ^surfaces <s>                   )
                                                                    ) ; 218

(p H-large_gunbarrel                                                ; 219
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type large_gunbarrel ^confidence <conf> ^surfaces 0)
   (Surface     ^id             { <s>  <> nil  <> 0 }
                ^y_loc          { > 2.107  < 2.193 }    ; elevation 2.15 +- 2%
```

Appendix.F

```
                ^area           ( > 0.4175  < 0.4345 )  ; area 0.426 +- 2%
                ^h_to_w         ( > 0.02303  < 0.02397 ); h_to_w 0.0235 +- 2%
                                )

-->
   (make Construct      ^type large_gunbarrel
                        ^confidence (compute <number> * <conf>)
                        )
                        ^surfaces <s>                                    ) ; 219

                                                                           ; 220

(p H-small_deck
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type small_deck ^confidence <conf> ^surfaces 0)
   (Surface     ^id             ( <s>  <> nil  <> 0 )
                ^y_loc          ( > 1.764  < 1.836 )    ; elevation 1.8 +- 2%
                ^height         ( > 0.392  < 0.408 )    ; height 0.4 +- 2%
                ^width          ( > 1.715  < 1.785 )    ; width 1.75 +- 2%
                ^area           ( > 0.686  < 0.714 )    ; area 0.7 +- 2%
                ^fit_error      ( < 1000 )              ; planar
                                )

-->
   (make Construct      ^type small_deck
                        ^confidence (compute <number> * <conf>)
                        )
                        ^surfaces <s>                                    ) ; 220

                                                                           ; 221

(p H-large_track
   (Phase ^description hypothesize ^status active)
   (World ^classes <number>)
   (Construct ^type large_track ^confidence <conf> ^surfaces 0)
   (Surface     ^id             ( <s>  <> nil  <> 0 )
                ^y_loc          ( > 0.784  < 0.861 )    ; elevation 0.8 +- 2%
                ^height         ( > 1.568  < 1.632 )    ; height 1.6 +- 2%
                ^width          ( > 6.8012  < 7.0788 )  ; width 6.94 +- 2%
                ^area           ( > 10.094  < 10.506 )  ; area 10.3 +- 2%
                ^fit_error      ( < 1000 )              ; planar
                                )

-->
   (make Construct      ^type large_track
                        ^confidence (compute <number> * <conf>)
                        )
                        ^surfaces <s>                                    ) ; 221

                                                                           ; 222

(p H-Clean_Up
   (Phase ^description hypothesize ^status finished)
   { <prior> (Construct ^type <t> ^surfaces 0) }
   (Construct ^type <t> ^surfaces { <> 0 } )
-->
   (remove <prior>)                                                        ) ; 222


;; Phase 3 Production Rules : Build Higher Level Constructs               ; 301

(p Build-bmp
   (Phase ^description build ^status active)
   (Construct   ^type medium_turret     ^confidence <c1>
                ^surfaces {<s1> <> nil} <s2> <s3> <s4>)
   (Construct   ^type medium_deck       ^confidence <c2>
                ^surfaces {<s5> <> nil} <s6>)
   (Construct   ^type medium_track      ^confidence <c3>
                ^surfaces <s7>)
   (Adjacent ^first <s1> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s3> ^second <s6> ^edge_type jmp)
   (Adjacent ^first <s5> ^second <s7> ^edge_type snd)
   (Adjacent ^first <s6> ^second <s7> ^edge_type snd)
-->
   (make Construct      ^type bmp
                        ^confidence (compute <c1> + <c2> + <c3>)
                        ^surfaces <s1> <s2> <s3> <s4> <s5> <s6> <s7>)   ) ; 301

                                                                           ; 302

(p B-medium_turret
   (Phase ^description build ^status active)
   (Construct   ^type medium_dome       ^confidence <c1>
                ^surfaces {<s1> <> nil} <s2> <s3>)
   (Construct   ^type small_gunbarrel   ^confidence <c2>
                ^surfaces <s4>)
   (Adjacent ^first <s3> ^second <s4> ^edge_type jmp)
-->
   (make Construct      ^type medium_turret
                        ^confidence (compute <c1> + <c2>)
                        ^surfaces <s1> <s2> <s3> <s4>)                  ) ; 302

                                                                           ; 303

(p B-medium_dome
   (Phase ^description build ^status active)
   (Construct   ^type medium_dome_panel         ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type medium_dome_panel         ^confidence <c2>
                ^surfaces {<s2>  <> <s1>})
   (Construct   ^type medium_dome_panel         ^confidence <c3>
                ^surfaces {<s3>  <> <s1>  <> <s2>})
   (Adjacent ^first <s1> ^second <s2> ^edge_type snd)
   (Adjacent ^first <s2> ^second <s3> ^edge_type snd)
-->
   (make Construct      ^type medium_dome
                        ^confidence (compute <c1> + <c2> + <c3>)
                        ^surfaces <s1> <s2> <s3>)                       ) ; 303

                                                                           ; 304

(p B-medium_deck
   (Phase ^description build ^status active)
   (Construct   ^type med_rear_deck     ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type med_fore_deck     ^confidence <c2>
                ^surfaces <s2>)
   (Adjacent ^first <s1> ^second <s2> ^edge_type snd)
-->
   (make Construct      ^type medium_deck
                        ^confidence (compute <c1> + <c2>)
                        ^surfaces <s1> <s2>)                            ) ; 304

                                                                           ; 305

(p Build-brdm2
   (Phase ^description build ^status active)
   (Construct   ^type small_turret      ^confidence <c1>
                ^surfaces {<s1> <> nil} <s2> <s3>)
   (Construct   ^type large_deck        ^confidence <c2>
                ^surfaces {<s4> <> nil} <s5> <s6>)
   (Construct   ^type car_front         ^confidence <c3>
                ^surfaces {<s7> <> nil} <s8>)
   (Construct   ^type car_side          ^confidence <c4>
                ^surfaces {<s9> <> nil} <s10> <s11>)
   (Adjacent ^first <s1> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s2> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s3> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s4> ^second <s9> ^edge_type snd)
```

```
   (Adjacent ^first <s5> ^second <s9> ^edge_type snd)
   (Adjacent ^first <s6> ^second <s7> ^edge_type snd)
   (Adjacent ^first <s7> ^second <s9> ^edge_type snd)
   (Adjacent ^first <s8> ^second <s9> ^edge_type snd)
-->
   (make Construct      ^type brdm2
                        ^confidence (compute <c1> + <c2> + <c3> + <c4>)
                        ^surfaces <s1> <s2> <s3> <s4> <s5> <s6>
                                  <s7> <s8> <s9> <s10> <s11>)
                                                                    ) ; 305

(p B-small_turret                                                   ; 306
   (Phase ^description build ^status active)
   (Construct   ^type small_dome        ^confidence <c>
                ^surfaces (<s1> <> nil) <s2> <s3>)
-->
   (make Construct      ^type small_turret
                        ^confidence <c>
                        ^surfaces <s1> <s2> <s3>)
                                                                    ) ; 306

(p B-small_dome                                                     ; 307
   (Phase ^description build ^status active)
   (Construct   ^type small_dome_panel  ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type small_dome_panel  ^confidence <c2>
                ^surfaces (<s2>  <> <s1>))
   (Construct   ^type small_dome_panel  ^confidence <c3>
                ^surfaces (<s3>  <> <s1>  <> <s2>))
   (Adjacent ^first <s1> ^second <s2> ^edge_type snd)
   (Adjacent ^first <s2> ^second <s3> ^edge_type snd)
 - (Construct ^type small_dome ^surfaces <s3> <s2> <s1>) ; prevent permutations
-->
   (make Construct      ^type small_dome
                        ^confidence (compute <c1> + <c2> + <c3>)
                        ^surfaces <s1> <s2> <s3>)
                                                                    ) ; 307

(p B-large_deck                                                     ; 308
   (Phase ^description build ^status active)
   (Construct   ^type lg_rear_deck      ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type lg_mid_deck       ^confidence <c2>
                ^surfaces <s2>)
   (Construct   ^type lg_fore_deck      ^confidence <c3>
                ^surfaces <s3>)
   (Adjacent ^first <s1> ^second <s2> ^edge_type snd)
   (Adjacent ^first <s2> ^second <s3> ^edge_type snd)
-->
   (make Construct      ^type large_deck
                        ^confidence (compute <c1> + <c2> + <c3>)
                        ^surfaces <s1> <s2> <s3>)
                                                                    ) ; 308

(p B-car_front                                                      ; 309
   (Phase ^description build ^status active)
   (Construct   ^type top_front         ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type bottom_front      ^confidence <c2>
                ^surfaces <s2>)
   (Adjacent ^first <s1> ^second <s2> ^edge_type snd)
-->
   (make Construct      ^type car_front
                        ^confidence (compute <c1> + <c2>)
                        ^surfaces <s1> <s2>)
                                                                    ) ; 309

(p B-car_side                                                       ; 310
   (Phase ^description build ^status active)
   (Construct   ^type med_side  ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type wheel     ^confidence <c2>
                ^surfaces <s2>)
   (Construct   ^type wheel     ^confidence <c3>
                ^surfaces (<s3>  <> <s2>))
   (Adjacent ^first <s1> ^second <s2> ^edge_type crv)
   (Adjacent ^first <s1> ^second <s3> ^edge_type crv)
 - (Construct ^type car_side ^surfaces <s1> <s3> <s2>)   ; prevent permutations
;;;             xdiff(S2, S3, 3.10)
-->
   (make Construct      ^type car_side
                        ^confidence (compute <c1> + <c2> + <c3>)
                        ^surfaces <s1> <s2> <s3>)
                                                                    ) ; 310

(p Build-m113                                                       ; 311
   (Phase ^description build ^status active)
   (Construct   ^type large_side        ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type small_track       ^confidence <c2>
                ^surfaces <s2>)
   (Adjacent ^first <s1> ^second <s2> ^edge_type jmp)
-->
   (make Construct      ^type m113
                        ^confidence (compute <c1> + <c2>)
                        ^surfaces <s1> <s2>)
                                                                    ) ; 311

(p Build-m60a1                                                      ; 312
   (Phase ^description build ^status active)
   (Construct   ^type large_turret      ^confidence <c1>
                ^surfaces (<s1> <> nil) <s2> <s3> <s6>)
   (Construct   ^type small_deck        ^confidence <c2>
                ^surfaces <s4>)
   (Construct   ^type large_track       ^confidence <c3>
                ^surfaces <s5>)
   (Adjacent ^first <s2> ^second <s4> ^edge_type jmp)
   (Adjacent ^first <s2> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s3> ^second <s5> ^edge_type jmp)
   (Adjacent ^first <s4> ^second <s5> ^edge_type jmp)
-->
   (make Construct      ^type m60a1
                        ^confidence (compute <c1> + <c2> + <c3>)
                        ^surfaces <s1> <s2> <s3> <s4> <s5> <s6>)
                                                                    ) ; 312

(p B-large_turret                                                   ; 313
   (Phase ^description build ^status active)
   (Construct   ^type hatch             ^confidence <c1>
                ^surfaces <s1>)
   (Construct   ^type turret_body       ^confidence <c2>
                ^surfaces <s2>)
   (Construct   ^type turret_front      ^confidence <c3>
                ^surfaces <s3>)
   (Construct   ^type large_gunbarrel   ^confidence <c4>
                ^surfaces <s4>)
   (Adjacent ^first <s1> ^second <s2> ^edge_type jmp)
   (Adjacent ^first <s2> ^second <s3> ^edge_type snd)
```

```
   (Adjacent ^first <s3> ^second <s4> ^edge_type jmp)
-->
   (make Construct      ^type large_turret
                        ^confidence (compute <c1> + <c2> + <c3> + <c4>)
                        ^surfaces <s1> <s2> <s3> <s4>)
                                                                    ) ; 313


;; Phase 4 Production Rules : Perform Garbage Collection

(p Clean_Up-Adjacencies                                             ; 401
   (Phase ^description clean ^status active)
   { <adj> (Adjacent) }
-->
   (remove <adj>)
                                                                    ) ; 401

(p Clean_Up-Surfaces                                                ; 402
   (Phase ^description clean ^status active)
   { <surf> (Surface) }
-->
   (remove <surf>)
                                                                    ) ; 402

(p Clean_Up-Prior                                                   ; 403
   (Phase ^description clean ^status active)
   { <prior> (Construct ^surfaces nil) }
-->
   (remove <prior>)
                                                                    ) ; 403

(p Clean_Up-Unlikely                                                ; 404
   (Phase ^description clean ^status active)
   (World ^threshold <thresh>)
   { <unlikely> (Construct ^confidence {<= <thresh>}) }
-->
   (remove <unlikely>)
                                                                    ) ; 404


;; Output Results

(p Output-Type_and_Confidence                                       ; 501
   { <object> (Construct ^type { <t> <> output } ^confidence <c>) }
 - (Phase)
-->
   (write (crlf) | Object class : | <t> (crlf))
   (write |Confidence : | <c> (crlf))
   (write |Surfaces : |)
   (modify <object> ^type output ^confidence nil)
                                                                    ) ; 501

(p Output-Surfaces                                                  ; 502
   { <object> (Construct ^type output ^surfaces { <s> <> nil }) }
 - (Phase)
-->
   (write (rjust 4) <s>)
   (bind <first-surface> (litval surfaces))
   (bind <second-surface> (compute <first-surface> + 1))
   (modify <object> ^surfaces (substr <object> <second-surface> inf) nil)
                                                                    ) ; 502

(p Output-Complete                                                  ; 503
   { <object> (Construct ^type output ^surfaces nil) }
 - (Phase)
-->
   (write (crlf))
   (remove <object>)
                                                                    ) ; 503


;; Rule to Initialize Working Memory
;;      When an element of class Start enters working memory,
;;      this rule initializes the database of a priori probabilities.

(p Initialize                                                       ; 999
   { <initialize> (Start) }
-->
   (remove <initialize>)
   (make Phase ^description expand ^status active)
   (make World ^classes 4 ^threshold 0.5)
   (make Construct ^type bmp               ^confidence 0.25    )
   (make Construct ^type medium_track      ^confidence 0.08333 ^surfaces 0)
   (make Construct ^type medium_deck       ^confidence 0.08333 )
   (make Construct ^type med_rear_deck     ^confidence 0.04166 ^surfaces 0)
   (make Construct ^type med_fore_deck     ^confidence 0.04166 ^surfaces 0)
   (make Construct ^type medium_turret     ^confidence 0.08333 )
   (make Construct ^type small_gunbarrel   ^confidence 0.04166 ^surfaces 0)
   (make Construct ^type medium_dome       ^confidence 0.04166 )
   (make Construct ^type medium_dome_panel ^confidence 0.01389 ^surfaces 0)
   (make Construct ^type brdm2             ^confidence 0.25    )
   (make Construct ^type small_turret      ^confidence 0.0625  )
   (make Construct ^type small_dome        ^confidence 0.0625  )
   (make Construct ^type small_dome_panel  ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type large_deck        ^confidence 0.0625  )
   (make Construct ^type lg_rear_deck      ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type lg_mid_deck       ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type lg_fore_deck      ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type car_side          ^confidence 0.0625  )
   (make Construct ^type med_side          ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type wheel             ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type car_front         ^confidence 0.0625  )
   (make Construct ^type top_front         ^confidence 0.03125 ^surfaces 0)
   (make Construct ^type bottom_front      ^confidence 0.03125 ^surfaces 0)
   (make Construct ^type m113              ^confidence 0.25    )
   (make Construct ^type large_side        ^confidence 0.125   ^surfaces 0)
   (make Construct ^type small_track       ^confidence 0.125   ^surfaces 0)
   (make Construct ^type m60a1             ^confidence 0.25    )
   (make Construct ^type large_turret      ^confidence 0.08333 )
   (make Construct ^type hatch             ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type turret_body       ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type turret_front      ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type large_gunbarrel   ^confidence 0.02083 ^surfaces 0)
   (make Construct ^type small_deck        ^confidence 0.08333 ^surfaces 0)
   (make Construct ^type large_track       ^confidence 0.08333 ^surfaces 0)
                                                                    ) ; 999

;; Make initialization class element
(make Start)
```