Idaho National Engineering Laboratory

INEL/EXT-97-00003

February 1997

Automatic TLI Recognition System, General Description

G. D. Lassahn

LOCKHEED MARTIN

INEL/EXT-97-00003

Automatic TLI Recognition System, General Description

G. D. Lassahn

HQ PROJECT MANAGER - Michael O'Connell
PROJECT NUMBER - ST474E

Published February 1997

Idaho National Engineering Laboratory
EG&G Idaho, Inc.
Idaho Falls, Idaho 83415

Prepared for the
U.S. Department of Energy
Office of Arms Control
under DOE Idaho Field Office
Contract DE-AC07-76ID01570

ABSTRACT

This report is a general description of an automatic target recognition system developed at the Idaho National Engineering Laboratory for the Department of Energy. A user's manual is a separate volume, *Automatic TLI Recognition System, User's Guide*, and a programmer's manual is *Automatic TLI Recognition System, Programmer's Guide*.

This system was designed as an automatic target recognition system for fast screening of large amounts of multi-sensor image data, based on low-cost parallel processors. This system naturally incorporates image data fusion, and it gives uncertainty estimates. It is relatively low cost, compact, and transportable. The software is easily enhanced to expand the system's capabilities, and the hardware is easily expandable to increase the system's speed. In addition to its primary function as a trainable target recognition system, this is also a versatile, general-purpose tool for image manipulation and analysis, which can be either keyboard-driven or script-driven. This report includes descriptions of three variants of the computer hardware, a description of the mathematical basis if the training process, and a description with examples of the system capabilities.

CONTENTS

ABSTRACT ii

INTRODUCTION 1

TRAINABLE MODE PROCESS DESCRIPTION 4

TRAINABLE MODE MATHEMATICAL APPROACH 6
- Surveillance Process 6
- Training Process 6

HARDWARE DESCRIPTION 14
- ATR1 Hardware 16
- ATR2 Hardware 19
- ATR3 Hardware 22

SOFTWARE DESCRIPTION 24
- Mask Creation Program G 24
- Training Program F 24
- Surveillance Program E 26
- ATR1 SCSI Software 26

SYSTEM TESTS 27

CONCLUSIONS 34

REFERENCES 35

APPENDIX A: EXAMPLE 1: Median Filter Testing A-1

APPENDIX B: EXAMPLE 2: Roads and Riverbanks B-1

APPENDIX C: EXAMPLE 3: Buried Waste Location C-1

APPENDIX D: EXAMPLE 4: Finding Airplanes D-1

APPENDIX E: EXAMPLE 5: Stack Shadows E-1

Automatic TLI Recognition System, General Description

INTRODUCTION

This report comprises three parts, printed in separate volumes. This part is a general description of the automatic target recognition (ATR) system and gives some indication of its capabilities. The second part[1] is a user's manual for people who do the hands-on image data analysis, giving instructions on how to use this ATR system. The third part[2] is a programmer's manual for people who want to modify or expand the software. The software described here is version 14 of programs E, F, and G, and version 7 of TSCSI.

The purpose of this task is to develop an Automatic Treaty-Limited Item Recognition System or, in the jargon of the literature, an automatic target recognition system. This task was started as part of the development of the Airborne Multisensor Pod System (AMPS), a reconnaissance system that can be attached to and carried aloft by any of several aircraft. This AMPS system includes several imaging sensors, such as a visible light camera, an infrared imaging system, and a radar imaging system. Each scene of interest can be photographed by each sensor, so we can have several different images of each scene. The several images can be analyzed by a computer to determine the presence of previously specified objects of interest in the scene. In this report, we will refer to these objects of interest as "targets", not to imply that we intend to shoot at them, but because of the established language in the technical community. The development of this computer system, an *automatic target recognition* (ATR) system, is the present task.

This task includes choosing and assembling the computer hardware, developing the ATR algorithms, and developing software to implement the algorithms on the selected hardware. Special requirements for this ATR system include *image fusion* and *uncertainty estimation*. "Image fusion" means that the ATR system must analyze the several images jointly to produce a single, high-confidence statement of the presence of a target, as opposed to analyzing the several images separately and producing several separate, lower-confidence, possibly contradictory, indications of the presence of targets. "Uncertainty estimation" means that the ATR system must tell the user what confidence the user should have in the ATR system's report of the presence or absence of targets. In addition, there are the common requirements for speed, portability, and low cost of the ATR system.

This ATR system was designed specifically for the AMPS project, but it should be equally applicable to other projects. This ATR system can work with any number of images per scene; or, it can work with the most common and simple case of one single image per scene. This ATR system was designed to scan a scene for the presence of traditional objects such as trucks, golf balls, or airport runways. However, it can equally well find "objects" or targets such as a particular type of vegetation, a chemical spill, or some specific type of terrain. This ATR system can be useful in automatically scanning any set of images for any type of target, assuming of course that the target is in some way discernible in the images. The images do not need to be images in the traditional sense of light intensity versus position in two dimensions; for this ATR system; an image is any two-dimensional array of numeric values. This ATR system informs the user of the importance of keeping each calculated feature used to discriminate between target and background, and thus gives the user the option of improving efficiency by eliminating unnecessary calculations and unnecessary sensors.

The following discussions often assume that the ATR system is used in a way that we will call the trainable mode, which includes using the training process to be described shortly. It is not necessary to use this training process or this trainable mode of operation; the system is quite versatile and can be used with other approaches to target recognition and for general purpose image manipulation and analysis.

The software described in this report uses the general appoach of splitting each image among several parallel processors. We expect that a future report will describe a different software system, in which no image is distributed among several nodes, but rather any one scene is analyzed entirely by one node. This difference in philosophy has major implications for system performance, as will be discussed later in this report.

Three hardware implementations of the system -- called ATR1, ATR2, and ATR3 -- have been assembled and tested. Table 1 lists some properties of the three systems. Various parts of Table 1 will be explained in different parts of this report. The Table 1 values for calculation speed, memory, cost, etc. are for one daisy node, not for the whole array of parallel processor nodes. The cost values include a pro-rated share of the cost of the mother board needed to support the node.

The appendices give examples of tests and simple applications of this ATR system. Appendix A gives a very simple example that illustrates the use of the system and demonstrates basic concepts such as marking a scene for the training process. Appendix B is a very simple example of image fusion. Appendix C illustrates the use of data that would not normally be considered image data, in an unconventional application of image analysis, and it also shows that reasonable results can be obtained with no expert knowledge of the application. Appendix D is a more realistic but still rather small example of a target recognition application, and Appendix E is a more complicated example.

Table 1: Properties of Daisy Nodes in the Three Hardware Systems.

system name	ATR1	ATR2	ATR3
processor type	T805	T9000	Alpha 21066
clock speed [MHz]	30	20	166(233)
nominal link speed [Mbits/second]	20	100	250
nominal link speed [Mbytes/second]	1.8	10	17
nominal calculation speed [Mflops]	4.3 peak	10 peak 6 sustained	126 LINPAC
memory (RAM) [Mbytes]	4	16	16
approximate cost	$1080	$2000	$10100
relative calculation speed			
expected	1.00	2.33	29.3
measured	1.00	2.18	21.7
relative calculation speed per cost			
expected	1.00	1.26	3.13
measured	1.00	1.18	2.32
relative communication speed			
expected	1.00	5.55	9.44
measured	1.00	1.99	0.76
setup time per message pair [microseconds]	225	180	1760

TRAINABLE MODE PROCESS DESCRIPTION

The trainable mode of using this ATR system is based on the assumption that targets can be distinguished from background by a quantitative measurement of the presence of *local features.* The term "local" normally implies that the features are small enough to be defined within a region that is small compared to the target, although the formalism does not impose any limit on the size of the features (except of course that the features must be smaller than the whole image). Examples of simple features are edges; lines; intensity averaged over a small region; and, speckle intensity. More complicated features include run length; other descriptors of size or shape of some region; and, local co-occurrence matrices. One could define as a local feature something as complicated as a detailed image of a particular target object, but this would not be a normal or easy use of the present formalism, and this use would not be covered by the usual interpretation of the phrase "local feature".

The *training* process is required by this trainable mode requires a set of images, one image from each imaging sensor, for each of several training scenes. The system will work with as few as one training scene, but better results might be expected from the use of many, perhaps hundreds, of training scenes. The set of training scenes must include some (at least one) targets and some background. A knowledgable person must examine each training scene and, for each scene, create a *mask.* The mask is a set of data with the format of an image (one byte per pixel, for example), but with each pixel value being 2, 1, or 0 indicating whether that pixel is part of a target, background, or unspecified region in the training scene. Thus, the entire set of training data comprises several real images and one mask "image" for each of several – or possibly only one – real world training scenes. Each training scene should contain either some target region or some background region or both; a training scene which contains neither target nor background is useless and should be discarded from the set of training data.

In addition to designating target and background regions in the training scenes, a skilled operator must select an initial set of local features that might distinguish between the target and background regions. If this operator-selected initial set of features does not include features that can distinguish target from background, the ATR computer will still function, but the uncertainty in target identification will be large, perhaps so large that the ATR system results will be useless. If, on the other hand, the operator-selected initial set of features includes more features than are necessary to distinguish target from background, no real harm is done. The ATR system will tell the operator the importance of each feature so that the operator can, if he wishes, discard unimportant features and make the final process more efficient. The only disadvantages of an operator selecting too large a feature set initially are some computational inefficiency in the training process (which is probably tolerable) and, in the extreme case, the inability of the computer programs to handle so many features. (The formalism can in principle handle any number of featues, but of course any particular computer program will be limited.) Thus, although great skill is not essential in selecting the initial set of features, a clever choice can make the ATR process much more efficient and effective.

One additional task of the operator is to select a value for the relative weighting of two types of error. A *type 1 error* is the incorrect designation of a background region as a target by the ATR system; in a *type 2 error*, a target is wrongly called background. The ATR system could make either of these two error rates as small as desired, at the expense of making the other error rate larger. We would like to make both error rates small simultaneously, but that may not always be possible. The procedure used here is to form a weighted sum of the two error rates, with weights W_1 and 1.0-W_1, where $0 < W_1 < 1$, and to minimize this weighted sum. The W_1

value is the relative weight of type 1 errors; type 1 errors are considered more important if W_1 is larger (closer to 1 than to 0).

The ATR training process takes as input (1) the training images and masks; (2) the definitions of the operator-selected feature set; and, (3) the value of the type 1 error weight W_1. The training process then calculates values for coefficients, optimized to minimize the total (weighted sum) error rate. These coefficients are used in the surveillance process, to be discussed shortly. The training process also gives indications of the importance of each feature, and an estimate of the total error rate. The training program does this training process for the complete feature set specified by the operator, and also for the subset in which the least important of the original features is removed, and also for the best subset of this subset, and so on for all successive best subsets of features until the smallest subset is too small to allow a calculation (usually, one feature). This allows the operator to select an efficient subset of features with an acceptably low error rate, if such a subset exists. Or, in the worst case, the operator must recognize that this ATR system, with the feature sets that he has tried, cannot distinguish his targets from his background.

One product of the training process is the magnitude of the minimized total error rate. This is used as the estimated *uncertainty* in the ATR system for this particular set of features and coefficients, for this type of image data. This is a prediction of the fraction of pixels that will be misclassified as "target" or "background" in the surveillance process.

After an acceptable feature set is found, the coefficients for that feature set are used in setting up the surveillance program part of the ATR system. Then, the set of images (no mask) from any scene is input to the surveillance program, and the program tells whether there is a target in the scene. This surveillance process does not require any special skill on the part of the operator. Since this surveillance process takes multiple images as input and yields one single result image as output, it accomplishes *image data fusion*, the simultaneous analysis of several input images to produce a single result.

TRAINABLE MODE MATHEMATICAL APPROACH

Surveillance Process

The surveillance process is used to scan large amounts of data with minimal expert interaction. This process accepts as input a set of T images of one scene; the mathematical approach allows T to be any value greater than 0. From the T scene images, we calculate F feature images $\mathbf{F}_{s,f}$, where s is the scene number and f is the feature number, f=1,2,...F. The user decides the number of feature images F and the nature of each feature image. Each feature image is simply an "image", a two-dimensional array of values in image format, that is calculated from the scene images by any methods that the analyst chooses. Thus, any single feature image might be derived from a single scene image, from another feature image, or from any combination of scene images and feature images. A raw scene image can be used as a feature image. The ATR algorithm always adds one special feature image to the set, the feature image numbered f=0, in which every pixel is 1 regardless of the content of the scene images. The user should not include this, or any image in which all the pixels have the same value, as one of his F feature images. Each feature image is multiplied by a coefficient C_f whose value has been determined in a training process, and these scaled feature images are then added to obtain a result image $\mathbf{R}_s$ for this scene s:

$$\mathbf{R}_s = \sum_{f=0}^{F} C_f \, \mathbf{F}_{s,f} \quad \text{for each scene s.} \qquad [1]$$

This operation requires that all the feature images must be of the same size. If a pixel in the result image $\mathbf{R}_s$ has a value greater than the value Q determined from the training process, that pixel is called a target pixel; otherwise, the pixel is called a background pixel. Thus, the result image classifies every pixel in the scene as either target or background. This surveillance process is very simple in principle, although the calculation of some of the feature images from the scene images may be complicated in practice.

Training Process

The values of the coefficients C_f and Q, which are used in the surveillance process, are determined by a training process. This training process uses S training scenes, indexed s=1,2,...S. For each scene, there are T training images, indexed t=1,2,...T, and one three-level mask image. The mask pixel values are 2 for target regions, 1 for background regions, and 0 for unspecified regions. These regions have been selected manually by some knowledgeable person. The T training images are from T different sensors, so that all T training images contain different information. The T training images are assumed to be in registration. Let $\mathbf{T}_{s,t}$ be the training scene images, and let $\mathbf{M}_s$ be the mask, for each scene s.

For each scene s, we extract F feature images $\mathbf{F}_{s,f}$, f=1,2,...F, from the T scene images. We define a result image $\mathbf{R}_s$ for each training scene s:

$$\mathbf{R}_s = \sum_{f=0}^{F} C_f \mathbf{F}_{s,f} \quad \text{for each } s=1,2,...S. \qquad [2]$$

The feature image with index f equal to 0 does not exist explicitly, but is implicitly defined to be an image in which each pixel has the value 1. It is mathematically advantageous to include this image in the set of feature images. The coefficients C_f are to be determined in this training process. The C_f are the same for all s. We would like to be able to choose values for the C_f to make $\mathbf{R}_s$ equal to $\mathbf{M}_s$ in regions 1 and 2 (we don't care about region 0), for every s. We do not expect to be able to accomplish this exactly.

We use a least squares fitting procedure to make a first estimate of the coefficients C_f. We find the C_f values that minimize the sum of the squares of the weighted differences between the result images and the mask images for regions 1 and 2, with the weight for each pixel being inversely proportional to the number of pixels in that pixel's region. That is, we minimize

$$\begin{aligned} SSQ &= \sum_{s=1}^{S} \underbrace{\sum_{i,j}}_{\text{sum over all region 1 + region 2 pixels}} \text{weight}(s,i,j)\,[\mathbf{R}_s(i,j) - \mathbf{M}_s(i,j)]^2 \\ &= \sum_{s=1}^{S} \underbrace{\sum_{i,j}}_{\text{sum over all region 1 pixels}} [\mathbf{R}_s(i,j) - \mathbf{M}_s(i,j)]^2 / M_{1,0} \\ &+ \sum_{s=1}^{S} \underbrace{\sum_{i,j}}_{\text{sum over all region 2 pixels}} [\mathbf{R}_s(i,j) - \mathbf{M}_s(i,j)]^2 / M_{2,0} \qquad [3] \end{aligned}$$

where $M_{1,0}$ and $M_{2,0}$ are simply the number of pixels in region 1 and region 2 respectively. The indexes i,j denote a particular pixel in the image. The minimization condition is

$$\sum_{f=0}^{F} C_f \left\{ \sum_{s} \sum_{\substack{i,j \\ \text{region 1 + region 2}}} \text{weight}(s,i,j)\, \mathbf{F}_{s,f}(i,j)\, \mathbf{F}_{s,g}(i,j) \right\}$$

$$= \left\{ \sum_{s} \sum_{\substack{i,j \\ \text{region 1 + region 2}}} \text{weight}(s,i,j)\, \mathbf{M}_s(i,j)\, \mathbf{F}_{s,g}(i,j) \right\}$$

or

$$\sum_{f=0}^{F} C_f \left\{ \sum_{s} \sum_{\substack{i,j \\ \text{region 1}}} \mathbf{F}_{s,f}(i,j)\, \mathbf{F}_{s,g}(i,j) / M_{1,0} \; + \; \sum_{s} \sum_{\substack{i,j \\ \text{region 2}}} \mathbf{F}_{s,f}(i,j)\, \mathbf{F}_{s,g}(i,j) / M_{2,0} \right\}$$

$$= \left\{ \sum_{s} \sum_{\substack{i,j \\ \text{region 1}}} \mathbf{M}_s(i,j)\, \mathbf{F}_{s,g}(i,j) / M_{1,0} \; + \; \sum_{s} \sum_{\substack{i,j \\ \text{region 2}}} \mathbf{M}_s(i,j)\, \mathbf{F}_{s,g}(i,j) / M_{2,0} \right\}$$

$$\text{for each } g=0,1,2,\ldots F. \qquad [4]$$

This represents F+1 inhomogeneous linear equations in F+1 unknowns, the C_f for f=0,1,2,...F. These can be easily solved unless two (or more) of the feature images contain the same information, in which case one (or more) of the redundant feature images should be deleted from the set.

After these first estimate C_f values are determined, we can calculate the $\mathbf{R}_s$ and generate two histograms, one for values of **R** pixels in region 1 and another for values of **R** pixels in region 2, both summed over all s=1,2,...S. The region 1 histogram should have a peak near the pixel value 1, and the region 2 histogram should have a peak near the pixel value 2. We would like these two histograms to appear as well separated peaks, but in practice we expect that the high end tail of the region 1 histogram will overlap the low end tail of the region 2 histogram. We define a parameter Q such that, for any scene, we call a pixel a target pixel if its value in the result image **R** is greater than Q, and we call the pixel a background pixel if its value in the result image **R** is less than Q. The value of Q will of course be between 1 and 2, between the positions of the two histogram peaks. We would like to be able to choose a Q value such that, in the training scenes, every region 2 pixel is called a target pixel and every region 1 pixel is called a background pixel. This is clearly impossible if the two histograms overlap, which condition we expect in practice. In this case, for whatever value of Q we choose, there will be some region 1 pixels with **R** values greater than Q, or there will be some region 2 pixels with **R** values less than Q, or both. These pixels represent errors in identifying targets. A region 1 pixel with an **R** value greater than Q is incorrectly called a target pixel; this will be referred to as a type 1 error. A region 2 pixel with an **R** value less than Q will be called a type 2 error. Part of the purpose of the training process is to find the value of Q that minimizes the total error probability.

We approximate the two histograms, after normalization, with two continuous distribution functions (probability density functions) D_1 and D_2. We define two error probabilities E_1 and E_2, representing type 1 and type 2 errors:

$$E_1 = \int_{Q}^{\infty} D_1(x)\, dx \qquad [5]$$

and

$$E_2 = \int_{-\infty}^{Q} D_2(x)\, dx. \qquad [6]$$

We define a total error rate as

$$E = W_1\, E_1 + W_2\, E_2 \qquad [7]$$

with $W_1+W_2=1.0$, where W_1 and W_2 are positive coefficients that have been selected previously to indicate the relative importance of the two types of error. We now want to find the values of Q and all the C_f to minimize the total error E. We have first estimates of the C_f values from the least squares fitting procedure; our first estimate of Q can be 1.5 (although there may be better first estimates of Q). We adjust all these parameter values by an iterative minimization procedure. It is necessary to impose two constraints while minimizing E: the mean values of the two distributions must be constrained to be 1 and 2. We use the method of Lagrange multipliers and minimize the quantity $E + \lambda_1\, \mu_1 + \lambda_2\, \mu_2$ while $\mu_1=1$ and $\mu_2=2$, by adjusting the values of C_f and Q. λ_1 and λ_2 are the Lagrange multipliers, and μ_1 and μ_2 are the mean values of D_1 and D_2.

Let M be the number of moments needed to define D_1 or D_2. If D_1 is the Gaussian distribution, for example, M=2; D_1 is defined by the mean (moment 1) and the variance (moment 2 - [moment $1]^2$), and thus by the first two moments. (For normalized distributions, the zeroth moment is always 1. We do not count this as one of the defining moments.) Let $M_{1,m}$ and $M_{2,m}$ be the m-th moments (about zero) of the distributions D_1 and D_2, for m=1,2,...M. Note that the above μ_1 and μ_2 are the same as $M_{1,1}$ and $M_{2,1}$. These moments are useful intermediate variables in the minimization of E.

Let the C_f and Q represent the current estimates of the coefficients as defined above, and let B_f+C_f and P+Q represent the optimum values of those coefficients. It is helpful to linearize E by expanding in a Taylor's series about the current values:

$$E(P+Q,B+C) = E(Q,C) + P\frac{\partial E(Q,C)}{\partial Q} + \sum_{f=0}^{F} B_f \frac{\partial E(Q,C)}{\partial C_f}$$

$$+ P^2 \frac{\partial^2 E(Q,C)}{\partial Q^2}/2 + P \sum_f B_f \frac{\partial^2 E(Q,C)}{\partial Q \partial C_f} + \sum_f \sum_g B_f B_g \frac{\partial^2 E(Q,C)}{\partial C_f \partial C_g}/2$$

$$+ \text{higher order terms to be neglected} \qquad [8]$$

or, using p and q to represent the sets of P with B_f and Q with C_f respectively and using the gradient operator ∇_q for simpler notation,

$$E(p+q) = E(q) + p \cdot \nabla_q E(q) + (p \cdot \nabla_q)^2 E(q)/2 + \text{h.o.t.} \qquad [9]$$

Then the minimization condition is

$$0 = \nabla_p[E(p+q) + \lambda_1 M_{1,1}(p+q) + \lambda_2 M_{2,1}(p+q)]$$

$$= \nabla_q E(q) + p \cdot \nabla_q \nabla_q E(q) + \lambda_1 \nabla_p M_{1,1}(p+q) + \lambda_2 \nabla_p M_{2,1}(p+q) \qquad [10]$$

with the constraints

$$1 = M_{1,1}(p+q) \qquad [11]$$

and

$$2 = M_{2,1}(p+q). \qquad [12]$$

More explicitly, equations 10-12 can be written as

$$B_0 \frac{\partial^2 E}{\partial C_0 \partial C_0} + B_1 \frac{\partial^2 E}{\partial C_1 \partial C_0} + \dots + B_F \frac{\partial^2 E}{\partial C_F \partial C_0} + P \frac{\partial^2 E}{\partial Q \partial C_0} + \lambda_1 \frac{\partial M_{1,1}}{\partial C_0} + \lambda_2 \frac{\partial M_{2,1}}{\partial C_0} = - \frac{\partial E}{\partial C_0} \qquad [13.0]$$

$$B_0 \frac{\partial^2 E}{\partial C_0 \partial C_1} + B_1 \frac{\partial^2 E}{\partial C_1 \partial C_1} + \dots + B_F \frac{\partial^2 E}{\partial C_F \partial C_1} + P \frac{\partial^2 E}{\partial Q \partial C_1} + \lambda_1 \frac{\partial M_{1,1}}{\partial C_1} + \lambda_2 \frac{\partial M_{2,1}}{\partial C_1} = - \frac{\partial E}{\partial C_1} \qquad [13.1]$$

$$\vdots$$

$$B_0 \frac{\partial^2 E}{\partial C_0 \partial C_F} + B_1 \frac{\partial^2 E}{\partial C_1 \partial C_F} + \dots + B_F \frac{\partial^2 E}{\partial C_F \partial C_F} + P \frac{\partial^2 E}{\partial Q \partial C_F} + \lambda_1 \frac{\partial M_{1,1}}{\partial C_F} + \lambda_2 \frac{\partial M_{2,1}}{\partial C_F} = - \frac{\partial E}{\partial C_F} \qquad [13.F]$$

$$B_0 \frac{\partial^2 E}{\partial C_0 \partial Q} + B_1 \frac{\partial^2 E}{\partial C_1 \partial Q} + \dots + B_F \frac{\partial^2 E}{\partial C_F \partial Q} + P \frac{\partial^2 E}{\partial Q \partial Q} + 0 + 0 = - \frac{\partial E}{\partial Q} \qquad [14]$$

$$B_0 \frac{\partial M_{1,1}}{\partial C_0} + B_1 \frac{\partial M_{1,1}}{\partial C_1} + \dots + B_F \frac{\partial M_{1,1}}{\partial C_F} + 0 + 0 + 0 = 1 - M_{1,1} \qquad [15.1]$$

$$B_0 \frac{\partial M_{2,1}}{\partial C_0} + B_1 \frac{\partial M_{2,1}}{\partial C_1} + \dots + B_F \frac{\partial M_{2,1}}{\partial C_F} + 0 + 0 + 0 = 2 - M_{2,1} \qquad [15.2]$$

Equations 11 and 12 can be written in the form of 15.1 and 15.2 because $M_{1,1}$ and $M_{2,1}$ are linear functions of the C_f. In equations 13.0 through 15.2, E, M_1, M_2, and their derivatives are to be evaluated at C_f and Q. Equation 10, or equations 13.0 through 13.F, represents F+2 equations in the F+4 unknowns P, B_f for f=0,1,...F, λ_1, and λ_2. Equations 11 and 12, or 15.1 and 15.2, represent two more equations in the same unknowns. This set of linear equations can be solved to get values for the B_f and P, which can then be used as corrections to the values of the C_f and Q. This process can be repeated, presumably with the corrections B_f and P becoming smaller with successive iterations until the values of C_f and Q converge. These converged values represent the values that are optimal in the sense that they minimize the total error E.

It is convenient to use the chain rule with the moments as intermediate variables in evaluating the derivatives in equations 10-15:

$$\frac{\partial E}{\partial Z} = W_1 \frac{\partial E_1}{\partial Z} + W_2 \frac{\partial E_2}{\partial Z} \qquad \text{for } Z = \text{any } C_f \text{ or } Q \qquad [16]$$

$$\frac{\partial E_1}{\partial C_f} = \int_Q^\infty \sum_{m=1}^{M} \frac{\partial D_1(x)}{\partial M_{1,m}} \frac{\partial M_{1,m}}{\partial C_f} dx = \sum_m \frac{\partial M_{1,m}}{\partial C_f} \int \frac{\partial D_1(x)}{\partial M_{1,m}} dx \qquad [17]$$

$$\frac{\partial^2 E_1}{\partial C_f \partial C_g} = \sum_m \sum_n \frac{\partial M_{1,m}}{\partial C_f} \frac{\partial M_{1,n}}{\partial C_g} \int \frac{\partial^2 D_1(x)}{\partial M_{1,m} \partial M_{1,n}} dx$$

$$+ \sum_m \frac{\partial^2 M_{1,m}}{\partial C_f \partial C_g} \int \frac{\partial D_1(x)}{\partial M_{1,m}} dx \qquad [18]$$

$$\frac{\partial E_1}{\partial Q} = -D_1(Q) \qquad [19.1]$$

$$\frac{\partial E_2}{\partial Q} = D_2(Q) \qquad [19.2]$$

$$\frac{\partial^2 E_1}{\partial Q^2} = - \frac{\partial D_1(Q)}{\partial Q} \qquad [20]$$

$$\frac{\partial^2 E_1}{\partial C_f \partial Q} = - \frac{\partial D_1(Q)}{\partial C_f} \qquad [21]$$

In this formalism, D_1 and D_2, their derivatives, and the integrals of these distribution functions and their derivatives, which appear in equations 17 - 21, all depend on the particular functional form chosen for the distribution functions that represent the histograms. The moments and their derivatives are independent of this choice of functional form:

$$M_{1,0} = \{ \sum_{s=1}^{S} \sum_{i,j} 1 \} \qquad [22]$$

The indexes i and j denote the i-th column and the j-th row in an image. The sum over i,j is to be taken over all the region 1 pixels for $M_{1,m}$ or all region 2 pixels for $M_{2,m}$. $M_{1,0}$ and $M_{2,0}$ are simply the number of pixels in all the training scenes in regions 1 and 2 respectively. The higher moments are normalized using these zeroth moments:

$$M_{1,1} = \frac{1}{M_{1,0}} \sum_{s} \sum_{i,j} \mathbf{R}_s(i,j)$$

$$= \sum_{f=0}^{F} C_f \{ \frac{1}{M_{1,0}} \sum_{s} \sum_{i,j} \mathbf{F}_{s,f}(i,j) \} \qquad [23]$$

$$M_{1,2} = \frac{1}{M_{1,0}} \sum_{s} \sum_{i,j} [\mathbf{R}_s(i,j)]^2$$

$$= \sum_{f} \sum_{g} C_f C_g \{ \frac{1}{M_{1,0}} \sum_{s} \sum_{i,j} \mathbf{F}_{s,f}(i,j)\mathbf{F}_{s,g}(i,j) \} \qquad [24]$$

and so on for the higher moments through m=M, with the obvious analogs for the region 2 moments. Note that these moments are taken about 0, not about the mean.

In practice, we need to go through all the feature images one time and calculate the sums that appear in braces { } in equations 22-24, and then we no longer need to access the images. It is important to access the images as infrequently and as efficiently as possible, because reading the images can represent a significant part of the time consumption in executing the program.

HARDWARE DESCRIPTION

The basic hardware concept for the ATR systems is the use of parallel processors. The three hardware systems discussed in this report – ATR1, ATR2, and ATR3 – are all variants of the general arrangement indicated in Figure 1.

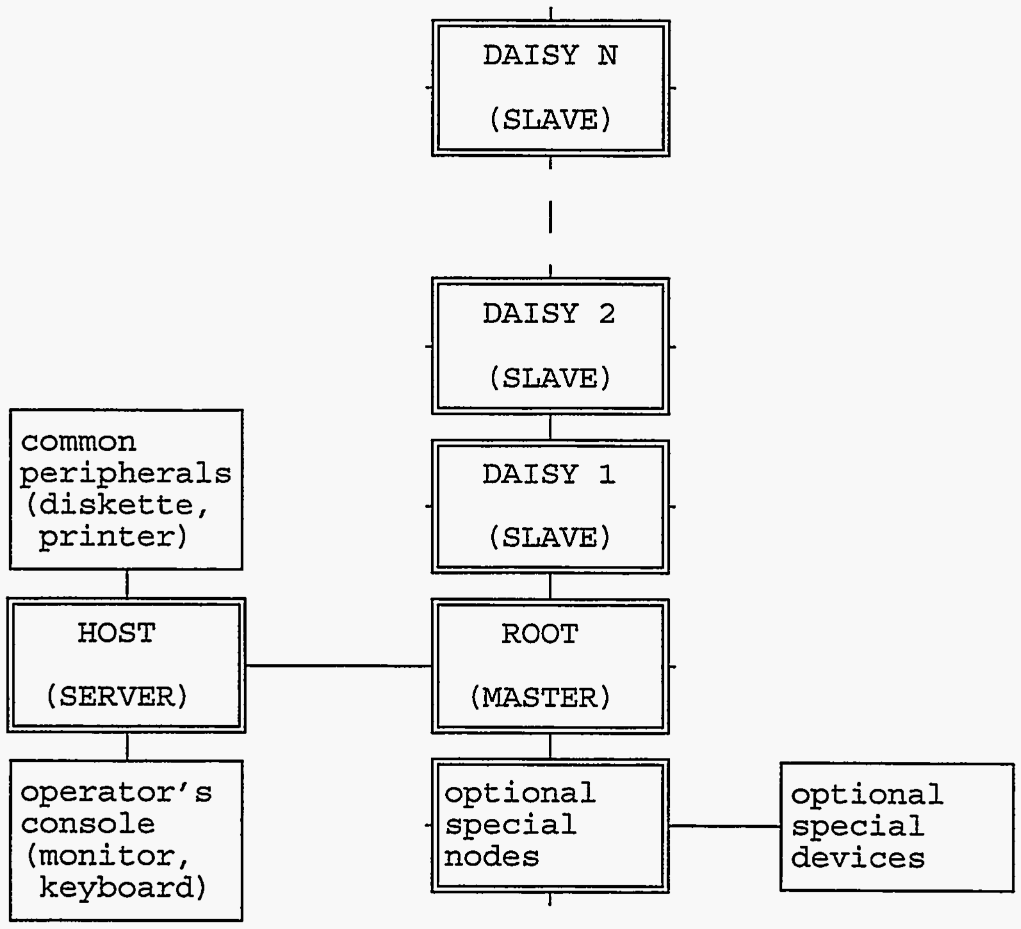


Figure 1: General ATR hardware diagram.

Parallel processor systems like the ATR systems comprise several, perhaps many, *nodes*. Each node includes one main central processor, and perhaps other devices such as memory and peripheral device interface hardware. In the ATR type of system, each node has its own memory; there is no shared memory, memory that is accessible by any of several processors. The ATR system nodes all exchange data via serial *links*, each of which is a two-way communication channel.

As is typical of parallel processor systems, each of the ATR systems has a *host* node, which we name HOST. The host node is a typical small computer, with the added feature that it has hardware and software that allow it to communicate with the rest of the parallel processor network. This host is the interface between the user and the parallel processor network; that is, it supplies an operator's console, a keyboard and a monitor. The host may also supply common peripheral devices, such as a diskette drive and a printer. In some of the ATR systems, the host also provides the essential high-capacity mass storage device, a mass data input/output device, and a high-resolution video monitor (separate from the operator's console monitor). The host function of allowing the parallel processor network to communicate with the operator and with data storage and input/output devices is referred to as a *server* function, and the program that runs on the host computer to communicate with the network is called a server program. For the ATR system, efficiency of the server function requires a fast data bus but does not require any significant computation power. Thus, the ATR system does not require a computationally powerful computer as a host. Prospective purchasers of parallel processor systems similar to those described here should be aware that hardware and software exists to allow any of a wide variety of computers to be used as hosts for any of the parallel processor networks described here.

In the ATR systems, only one parallel processor node is directly connected to the host. This node is called the *root* node. Connected to the root node is a daisy chain of nodes called *daisies* (not a standard term). With this simple linear configuration, the calculation speed of the system can easily be increased by simply adding more nodes to the end of the chain. Adding more nodes also increases the total memory of the system, since each node has its own memory. This arrangement allows the user to easily tailor the ATR hardware system to suit his own calculation speed requirements and budget constraints. In ATR1 and ATR2, the root node is named ROOT, and the daisies are named DAISY 1, DAISY 2, etc. In ATR3, because of the way the nodes are used, no node is named ROOT; the root node is DAISY 1.

Each of the ATR systems includes a high-resolution RGB monitor for displaying images, separate from the standard operator's console monitor. In ATR1, there is a special node that is an interface between the parallel processor network and the RGB monitor. In ATR2, the root node includes display interface hardware. In ATR3, the RGB monitor is a peripheral device on the host computer. ATR1 also has another special node, an interface between the parallel processor network and a SCSI bus which can be connected to mass storage devices such as a disk or a tape drive. The purpose of the special nodes is to allow large amounts of data to move between the parallel processors and the RGB monitor or the mass storage devices without the potential bottleneck of a relatively slow host computer bus.

The three existing ATR networks are diagrammed in Figures 2, 4, and 6, in which each node is represented by a box with a double-line border, with the node name near the bottom left of the box. The name in parentheses at the bottom left of each node box indicates the program that runs on that node, as described in the software documentation. These three systems are described separately in the following sections.

ATR1 Hardware

In ATR1, the root node and the 10 daisy nodes are CTRAMs (plug-in Computation TRAnsputer Modules), each with an INMOS T805 transputer (see Reference 3) running at 30 MHz and 4 Mbyte of memory. A *transputer* is a computer processor designed to work well with other transputers in a parallel processing system. Each transputer includes its own logical and numerical processors, a limited amount of fast (on-chip) memory, and four serial links to allow communication with other transputers or other devices. Each CTRAM link transmits data at about 1.8 Mbyte/second in either direction. In addition to the CTRAMs, ATR1 includes two special purpose nodes, one to interface with the high-resolution RGB monitor and the other to interface with a SCSI bus which gives access to mass storage devices, an Exabyte 8mm tape drive and a 1 Gbyte disk. Each of the two special nodes includes a transputer, and each runs a special program appropriate to its peripheral interface function. These two special nodes are used to avoid the potential bottleneck of the host computer bus when transferring large amounts of data to the mass storage and display devices. The host computer is an IBM compatible system with an Intel 80486DX processor operating at 33MHz, an ISA bus, an 80 Mbyte hard disk, diskette drives, keyboard, and monitor. The parallel processor nodes are mounted on two expansion boards, transputer mother boards, in the host computer, with the 10 physically small daisy nodes on one board and the other 3 larger nodes on the other board. The SCSI disk and tape drives are external, but could be mounted in the main computer case in future systems.

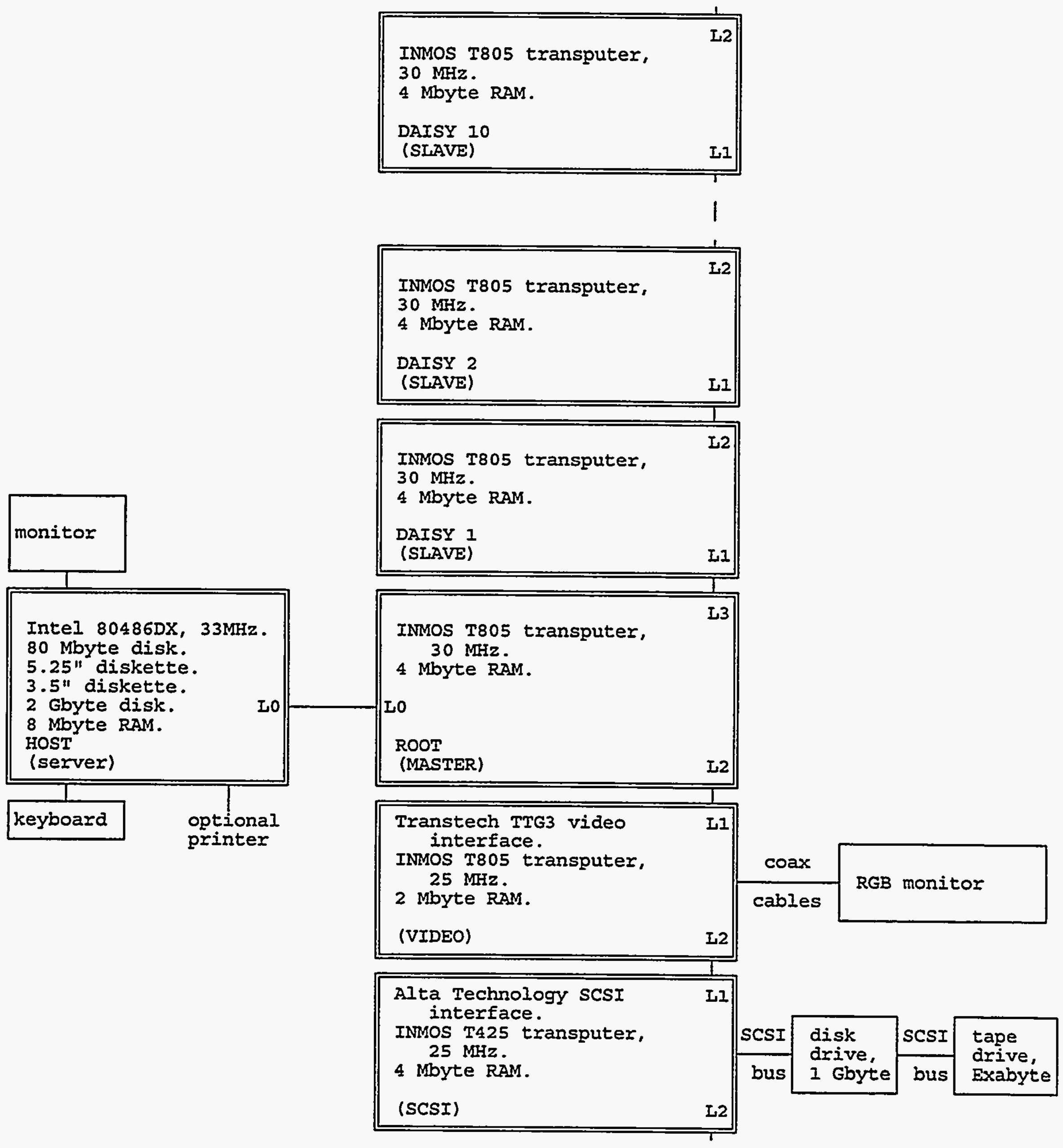
The **check¦mtest** software gives a listing indicating some transputer characteristics and network connections for ATR1, which is augmented here with the names of the nodes:

```
check 2.52 | mtest 2.52
    # Part rate Mb Bt [ Link0 Link1 Link2 Link3 ] RAM,cycle
    0 T805d-30 1.36 0 [  HOST   ...    1:1    2:1 ] 4K,1+4096K,3;  ROOT
    1 T805d-25 1.79 1 [  ...    0:2    3:1    ... ] 4K,1+2048K,4;  VIDEO
    2 T805d-30 1.80 1 [  ...    0:3    4:1    ... ] 4K,1+4096K,3;  DAISY  1
    3 T425b-25 1.75 1 [  ...    1:2    ...    ... ] 4K,1+4096K,3;  SCSI
    4 T805d-30 1.77 1 [  ...    2:2    5:1    ... ] 4K,1+4096K,3;  DAISY  2
    5 T805d-30 1.77 1 [  ...    4:2    6:1    ... ] 4K,1+4096K,3;  DAISY  3
    6 T805d-30 1.77 1 [  ...    5:2    7:1    ... ] 4K,1+4096K,3;  DAISY  4
    7 T805d-30 1.77 1 [  ...    6:2    8:1    ... ] 4K,1+4096K,3;  DAISY  5
    8 T805d-30 1.77 1 [  ...    7:2    9:1    ... ] 4K,1+4096K,3;  DAISY  6
    9 T805d-30 1.80 1 [  ...    8:2   10:1    ... ] 4K,1+4096K,3;  DAISY  7
   10 T805d-30 1.77 1 [  ...    9:2   11:1    ... ] 4K,1+4096K,3;  DAISY  8
   11 T805d-30 1.77 1 [  ...   10:2   12:1    ... ] 4K,1+4096K,3;  DAISY  9
   12 T805d-30 1.80 1 [  ...   11:2    ...    ... ] 4K,1+4096K,3;  DAISY 10
```

This list includes one line for each node except the host. The line including "DAISY 8", for example, indicates that the node that we call DAISY 8 was labeled number 10 by the **check** program, the processor is a T805 transputer running at 30 MHz, 1.77 Mbyte/second are transferred along link number 1 which is the boot link for this node, this node's link 1 is connected to node 9 link 2, this node's link 2 is connected to node 11 link 1, and this node has 4 kbytes of 1-cycle memory and 4096 kbytes of 3-cycle memory.

This ATR1 system (as well as ATR2 and ATR3) should work with a larger number of daisy nodes, and in fact this system has been used with 20 instead of 10 daisy nodes. However, the 20-daisy system did not work reliably, because of inadequate cooling air flow.

Figure 2 is a diagram of the ATR1 network, with each node represented as a double-line rectangle. Each node except the host is a TRAM (TRAnsputer Module) that plugs into a personal computer expansion board (transputer mother board, TRAM holder) in the host personal computer. L0, L1, L2, and L3 indicate link connections 0, 1, 2, and 3. In Figure 3 the two large rectangles represent two transputer mother boards and the double lines within the large rectangles represent jumpers.



L2
INMOS T805 transputer, 30 MHz. 4 Mbyte RAM.
DAISY 10 (SLAVE)
L1
L2
INMOS T805 transputer, 30 MHz. 4 Mbyte RAM.
DAISY 2 (SLAVE)
L1
L2
INMOS T805 transputer, 30 MHz. 4 Mbyte RAM.
DAISY 1 (SLAVE)
L1
monitor
Intel 80486DX, 33MHz. 80 Mbyte disk. 5.25" diskette. 3.5" diskette. 2 Gbyte disk. 8 Mbyte RAM. HOST (server)
L0
L3
INMOS T805 transputer, 30 MHz. 4 Mbyte RAM.
L0
ROOT (MASTER)
L2
keyboard
optional printer
Transtech TTG3 video interface. INMOS T805 transputer, 25 MHz. 2 Mbyte RAM.
(VIDEO)
L1
L2
coax cables
RGB monitor
Alta Technology SCSI interface. INMOS T425 transputer, 25 MHz. 4 Mbyte RAM.
(SCSI)
L1
L2
SCSI bus
disk drive, 1 Gbyte
SCSI bus
tape drive, Exabyte

Figure 2: ATR1 parallel processor network.

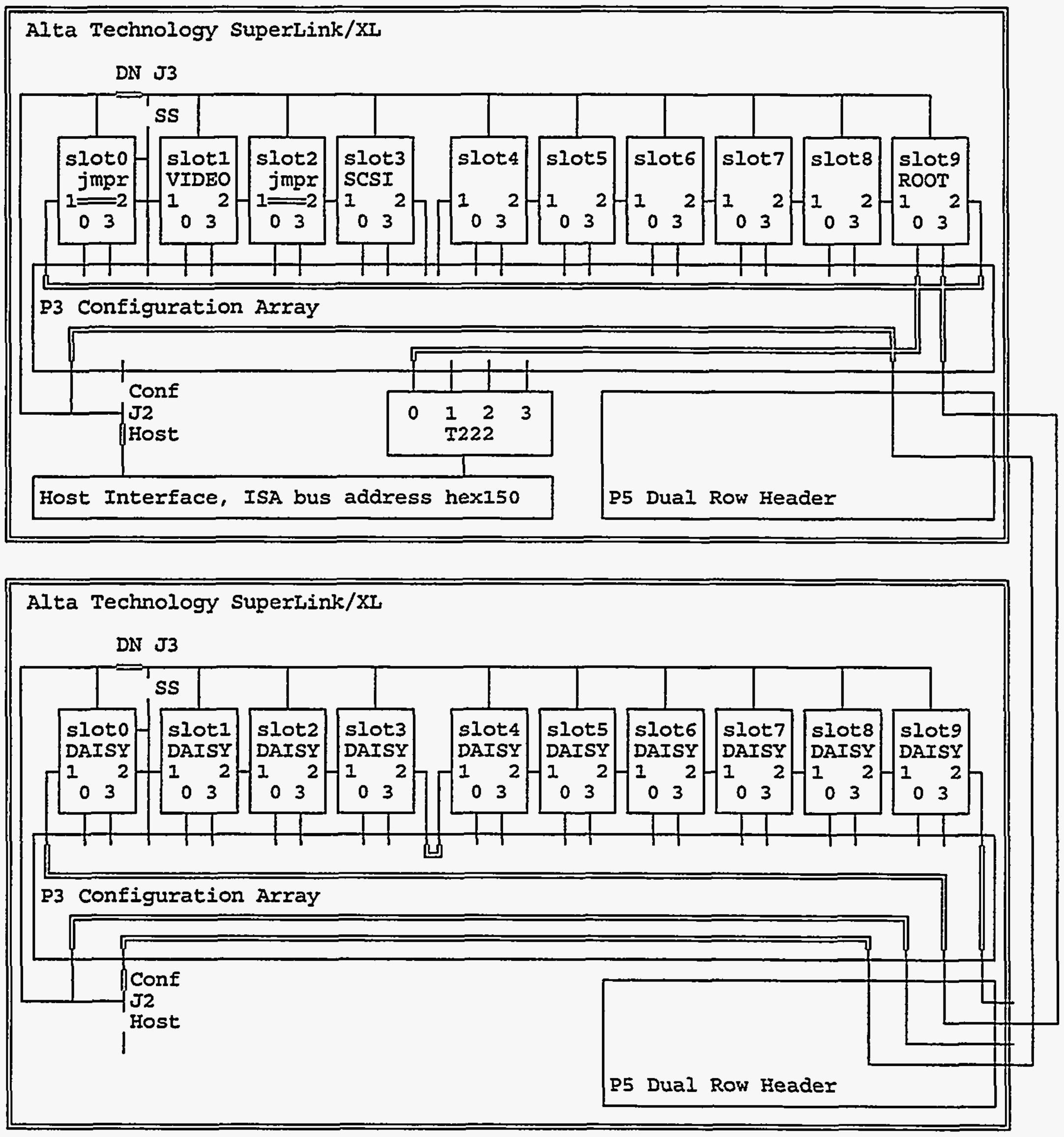


Figure 3: ATR1 transputer mother board connections.

ATR2 Hardware

In ATR2, each of the 3 daisy nodes uses a T9000 transputer (see Reference 4) running at 20 MHz, with 8 Mbyte of memory. The root node was designed as a video interface node and includes a T9000 transputer and 4 Mbyte of DRAM along with additional memory and processors for video display. The root node serves as both the master node and as a high-speed video interface. The T9000 transputers in ATR2 are the gamma E03 release with known disadvantages compared to the intended final production version, the most notable of which is the clock speed of 20 MHz instead of the originally intended 50 MHz. The link speeds in ATR2 are set to 100 Mbit/second, which implies a one-way data transfer rate of about 10 Mbyte/second. The parallel processor nodes are HTRAMs, which mount no more than 2 on each PC expansion board (HTRAM mother board). In ATR2, the host system disk is also used as the primary mass storage device, so bulk data transfers must go through the host ISA bus, but this is not believed to be a significant bottleneck in this case. A SCSI interface allows access to an external tape drive and other mass data transfer devices.

The annotated listing from the **t9spy** software indicates the network connections, the names and types of the nodes, and the software that runs on each node:

```
---------------- T9SPY --------------
|Device |Link 0|Link 1|Link 2|Link 3|  node   node      program
-------------------------------------  type   name
|0:T9000| EDGE | .... | 1: 1 | .... |  QT9D  ROOT      MASTER+VIDEO
|1:T9000| .... | 0: 2 | 2: 1 | .... |  B927  DAISY 1   SLAVE
|2:T9000| .... | 1: 2 | 3: 1 | .... |  B927  DAISY 2   SLAVE
|3:T9000| .... | 2: 2 | .... | .... |  B927  DAISY 3   SLAVE
-------------------------------------
```

Figure 4 is a diagram of the ATR2 network, and Figure 5 indicates the connections on and between the personal computer expansion boards (B108 HTRAM mother boards) that support the HTRAM nodes. The L0, L1, and L2 symbols in Figure 4 designate link connectors, and the mother board jumpers required are shown as double lines in Figure 5.

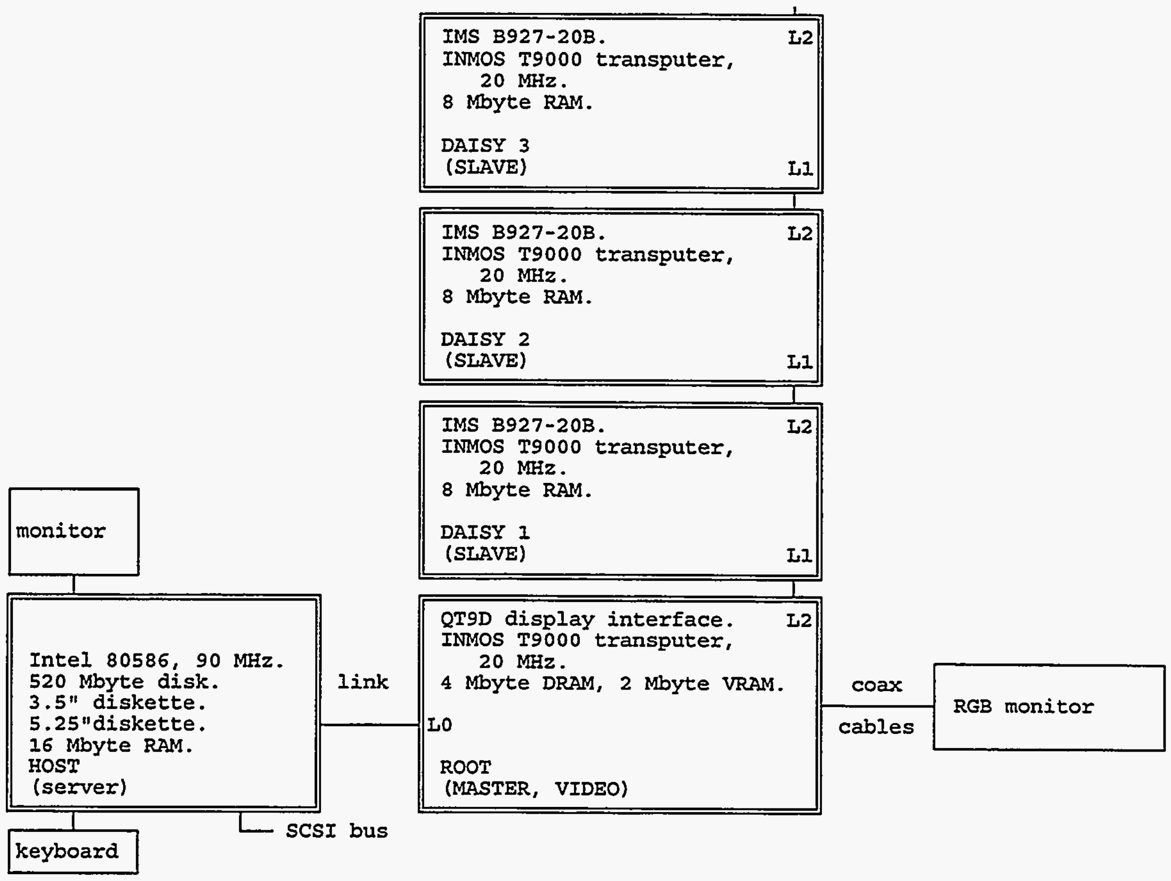


Figure 4: ATR2 parallel processor network.

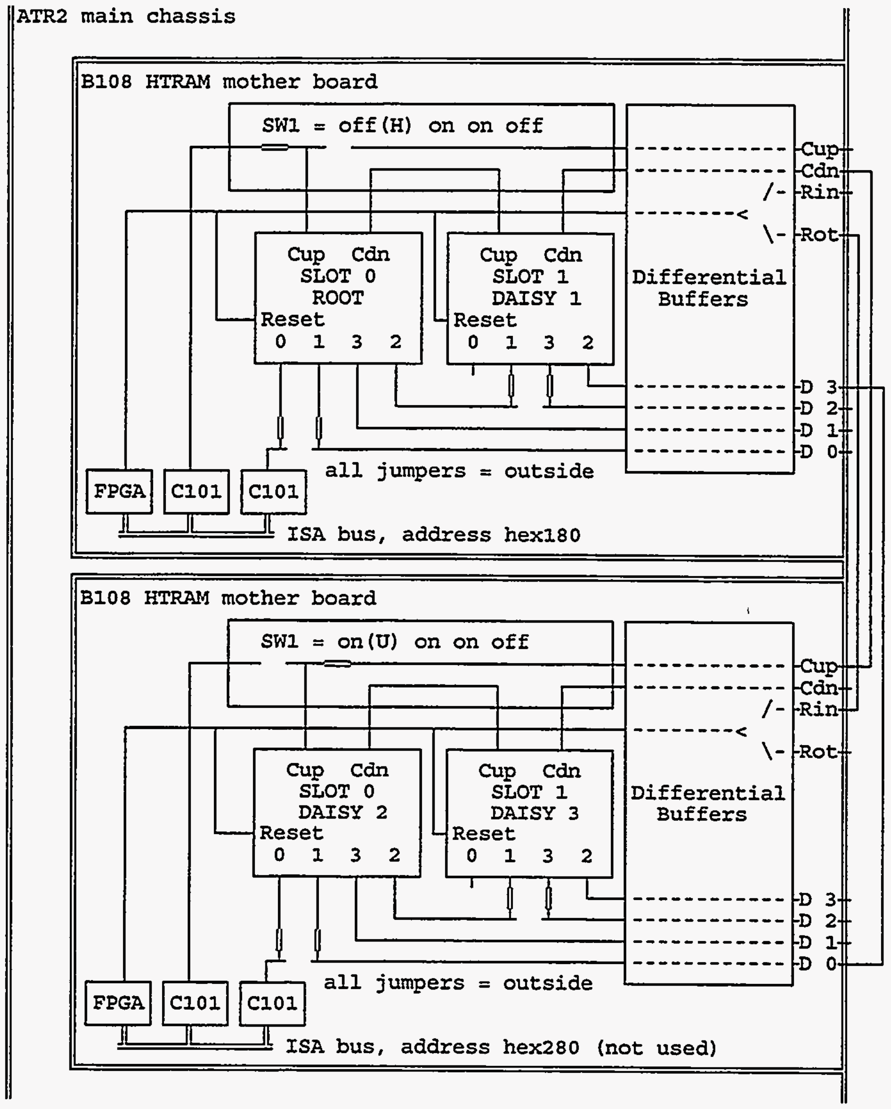


Figure 5: ATR2 transputer mother board connections.

ATR3 Hardware

The ATR3 host computer is a DEC 2000 Model 300 AXP. A VT510 monochrome monitor and keyboard, connected to a serial port, are used as the operator's console. With the ATR software in its normal mode, the high-resolution RGB monitor is used solely for displaying the images being processed. When the ATR software is being debugged with TCP/IP support, the RGB monitor is used with the second keyboard and the mouse to manipulate and observe the parallel processor nodes.

Each ATR3 parallel processor node is an Alta Technology AL/V66 which includes a DEC Alpha 21066 processor and 16 Mbyte of memory. Each node also includes a T425 transputer to facilitate inter-node communication and control operations such as booting the network, but the transputer's operation is transparent and the user does not need to explicitly address it. The nodes can communicate via the standard four 20 Mbit/second (about 1.8 Mbyte/second in either direction) transputer links, or T-links, and also via four 250 Mbit/second (17 Mbyte/second) A-links. The parallel processor nodes can be accessed using the TCP/IP model, in which case each node's T-links 2 and 3, and one host T-link, are used by the system and are not available explicitly to the user; the other T-links and the A-links are available at the user's discretion. This TCP/IP access is very useful for debugging the software that runs on the parallel processor nodes, as it allows each node to be monitored in a separate window of the user's display. The parallel processor network can also be used, more simply and efficiently, without TCP/IP support, in which case all the links are at the user's disposal. (T-links cannot be connected to A-links.) The parallel processor nodes are mounted in a VME chassis, separate from the host. The ATR3 vendor recommends leaving one or two vacant slots between nodes in the VME chassis.

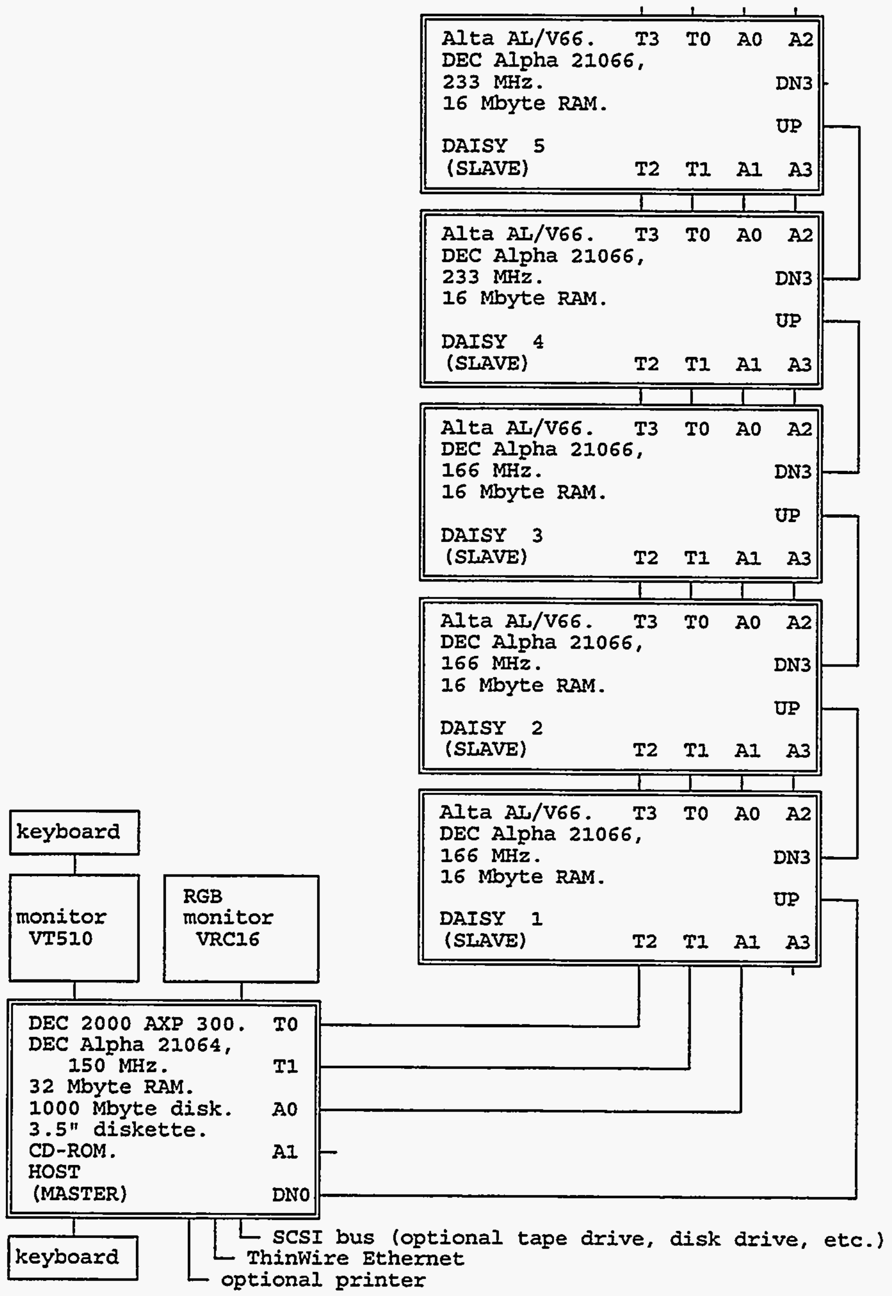
For historical reasons, three of the five nodes in this particular system have a clock speed of 166 MHz, and two run at 233 MHz. The current version of the ATR software cannot allocate computation tasks to the nodes differently depending on their different speeds. Therefore, the system speeds that are discussed later are determined by the speed of the 166 MHz nodes. A system using exclusively 233 MHz nodes would presumably be proportionately faster.

The ATR3 architecture would allow using several links in parallel; for example, we could communicate one message via the A2 and A3 links and another message via the A0 and A1 links at the same time, and thereby double the potential inter-node data transfer rate. However, this would substantially complicate the inter-node communication part of the ATR software, and the overall ATR software is such that it could not significantly benefit from the additional parallel links. ATR3 normally uses the A0 and A1 links for inter-node communication.

The **check** software prints the following information about the network, indicating the T-link but not the A-link connections:

```
Using /dev/hsi12 check 2.52
      #   Part rate Mb Bt  [ Link0 Link1 Link2 Link3 ]
      0   T425c-25 1.65 2  [  1:1    ...   HOST   1:2 ]
      1   T425c-25 1.75 1  [  2:1    0:0    0:3   2:2 ]
      2   T425c-25 1.74 1  [  3:1    1:0    1:3   3:2 ]
      3   T425c-25 1.75 1  [  4:1    2:0    2:3   4:2 ]
      4   T425c-25 1.74 1  [  ...    3:0    3:3   ... ]
```

Figure 6 is a diagram of the ATR3 network. In this figure, T0, T1, T2, and T3 are T-link connectors, and A0, A1, A2, and A3 are A-link connectors. The UP and DOWN connections are always required for system operation.



Alta AL/V66. DEC Alpha 21066, 233 MHz. 16 Mbyte RAM. DAISY 5 (SLAVE)
T3 T0 A0 A2 DN3 UP T2 T1 A1 A3
Alta AL/V66. DEC Alpha 21066, 233 MHz. 16 Mbyte RAM. DAISY 4 (SLAVE)
Alta AL/V66. DEC Alpha 21066, 166 MHz. 16 Mbyte RAM. DAISY 3 (SLAVE)
Alta AL/V66. DEC Alpha 21066, 166 MHz. 16 Mbyte RAM. DAISY 2 (SLAVE)
Alta AL/V66. DEC Alpha 21066, 166 MHz. 16 Mbyte RAM. DAISY 1 (SLAVE)
keyboard
monitor VT510
RGB monitor VRC16
DEC 2000 AXP 300. DEC Alpha 21064, 150 MHz. 32 Mbyte RAM. 1000 Mbyte disk. 3.5" diskette. CD-ROM. HOST (MASTER)
T0 T1 A0 A1 DN0
keyboard
SCSI bus (optional tape drive, disk drive, etc.)
ThinWire Ethernet
optional printer

Figure 6: ATR3 parallel processor network.

SOFTWARE DESCRIPTION

There are three programs, called E, F, and G. The three programs have many functions in common, but they also have some differences associated with the different operations they perform. The ATR1 system also includes the program TSCSI for manipulating the devices attached to the SCSI interface node. The user's manual[1] lists the user functions currently included in the ATR software.

A fundamental aspect of the current ATR software is that each image in memory is divided among the several daisies (slave nodes). This has important implications in terms of the speeds of the three hardware systems, which will be discussed later. A new version of the ATR software will be developed, in which images will not be distributed among several nodes. This new version will be described in a separate report later.

Mask Creation Program G

Program G is an interactive (keyboard-driven) program which can be used as a general purpose image manipulation program and also as a tool to create masks that are needed in the trainable mode of this ATR system. For each training scene, a skilled operator must create a mask that indicates which parts of the scene are targets, which parts are background, and which parts are not designated as either target or background regions. The program G is a tool to help the operator do this task. G can read a scene image from a file specified by the operator, and display the image on the screen. The operator can then use a cursor to draw polygons on the image and can designate the areas inside of each polygon to be whichever region (target, background, or unspecified) he wishes. Regions not marked by the operator are left as "unspecified". G creates a mask, an image that contains the operator's designations, and saves the mask to the file of the operator's choice. The original scene image file is not changed. The training program F later reads the mask file along with the scene image files. Program G is interactive in that it takes commands one at a time from the keyboard, not from a script file as do programs F and E.

Training Program F

The training processes is implemented in program F. (For general purpose, script-driven image manipulation applications, program E is better than program F.) The training program F calculates feature images according to an operator-supplied list of instructions (a script file), and finds the optimum values of coefficients to be used with these features in the surveillance program E.

Feature calculation typically starts with scene images and calculates several feature images. This may involve arithmetic or logical operations on a pixel-by-pixel basis; convolution in two dimensions with a template that is typically much smaller than the scene image; morphological operations; local order sorting operations, such as are done in a median filter; local co-occurrence matrix calculation; and others, limited only by the imagination of the operator. Of course, a given computer program can implement only a finite number of operations. The operations currently available in this F program are listed in the user's manual[1]. These built-in operations can of course be combined to form very complicated feature calculation procedures. The software can be expanded reasonably easily to incorporate additional operations, if that should become necessary.

Program F calculates the optimum values of the coefficients for the complete operator-specified feature set and for certain feature subsets. It will often happen that a substantially smaller subset will give error rates that are essentially the same as the error rate for the full original feature set. It is important to know this, because efficiency of the surveillance process (the analysis of images in the field) is greater for smaller feature sets. Therefore, the F program is arranged to look for optimal feature subsets. For each case, F calculates the optimal coefficient value for each included feature, and F also calculates the importance of keeping each feature included in that case. Then, a new case is formed by deleting the least important feature from the old set. This process of forming successively smaller subsets by deleting one feature at a time does, in effect, yield a graph of total error rate versus number of features, like the example in Figure 7. The operator can then see how many features he must include in the surveillance process to attain the desired error rate, and he can judge the trade-off between increased error rate and increased number of features.

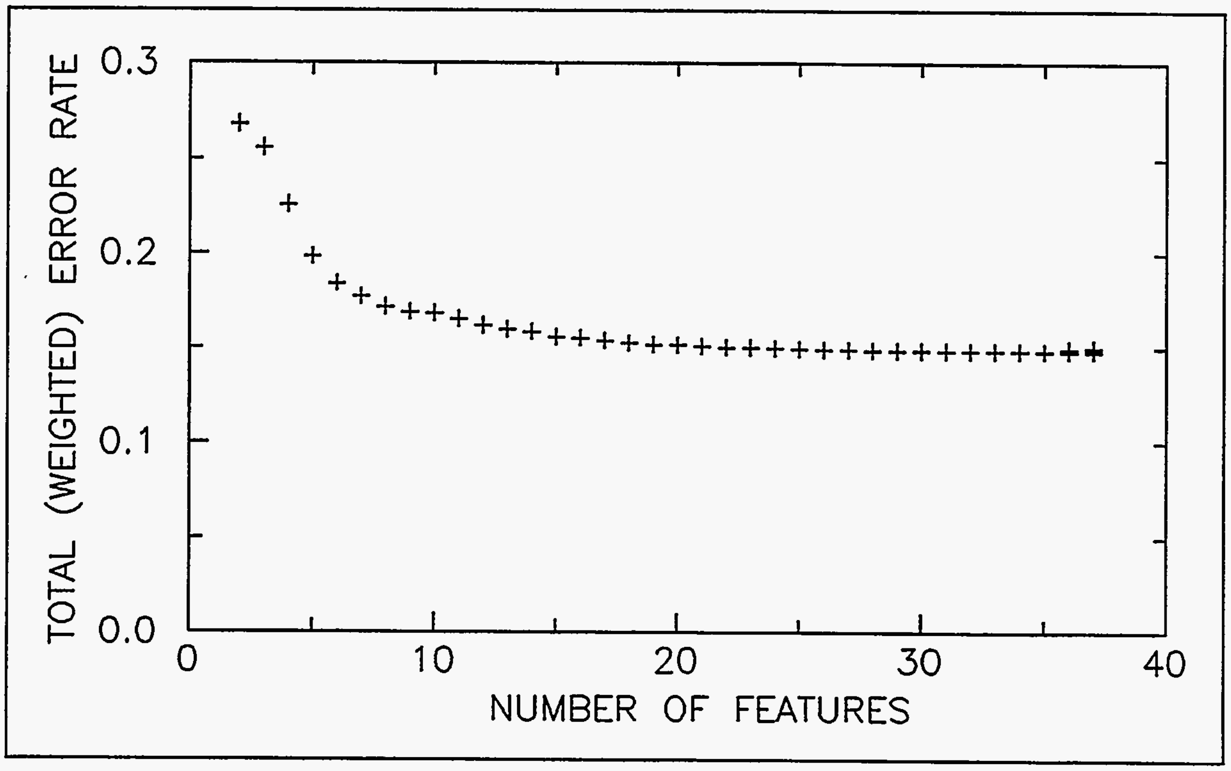


Figure 7: Typical example of total (weighted sum) error rate versus number of features. The optimization calculation tried several times, with slightly different resulting error rates, for some numbers of features; this is apparent from the double points plotted for 36 and 37 features.

For each case, the optimum coefficient values are determined by minimizing the total error rate. Program F does this minimization by using a linear approximation of the non-linear equations. As is usually the case in this type of calculation, it is possible that the process will not find the desired absolute minimum in the error rate, but will converge to a substantially different relative minimum point. Sometimes this is evidenced by a smaller subset of included features giving a smaller error rate than a larger, previously calculated feature set. When this occurs, program F uses the coefficient values from the smaller subset, smaller error rate case as a first guess and restarts with the full original set of features included. This procedure sometimes leads to the discovery of optimal points with error rate minima that are substantially smaller than those associated with the first-indicated optimal point for a given number of featues.

Surveillance Program E

Program E is a general-purpose, script-driven image manipulation program, and the script can be arranged to use program E in the trainable mode of this ATR system in which the optimized coefficients from program F are used. In the trainable mode, the surveillance program E is used in the field to analyze a set of scene images (no mask) and report on whether there is a target present in the scene. Program E uses the values of the coefficients determined by program F. Program E inputs scene images and normally creates a result image. If the value of any pixel in the result image is greater than Q, a value supplied by the training program F, that pixel is interpreted as an indication of the presence of a target. The result image can be written to a disk file with the Q value embedded in the file so that the image can be displayed later with obvious indications of any targets that may be present, and of course the result image can be displayed on the monitor immediately when it is calculated. Program E takes commands from a script file, not from the keyboard.

ATR1 SCSI Software

In the ATR1 system, a SCSI bus is interfaced directly to the transputer network, not to the host computer. One result of this is that the SCSI devices (disk drive and tape drive) are not controlled by the host DOS operating system. In this ATR application, separate file handling software is used for the SCSI devices. The SCSI disk may be used by programs E and F and G very much as if it were a DOS disk, for reading image files. A separate program, TSCSI, was written to allow other types of access to the SCSI devices. This program allows such operations as copying files to and from the SCSI devices, reading disk directories, deleting files, TAR file extraction, and certain diagnostic operations, listed in the user's manual[1]. Program TSCSI takes commands from the keyboard.

SYSTEM TESTS

The three programs E, F, and G have been verified as being functional on all three ATR hardware systems.

An important aspect of this ATR systems is its speed, primarily the speed of the parallel processor network. We have done direct measurements of speeds for certain typical image analysis operations, and we have compared the three ATR systems which use essentially the same software on different hardware systems. The results of the measurements reported here are in some cases different from similar measurements reported earlier, primarily because of improvements in the ATR3 system software and also because of changes in our own software. For these timing measurements, ATR3 was used with only 3 daisy nodes, run at 166 MHz, to make it more directly comparable with the three-node ATR2 system. The times quoted here were measured with the echo turned off, using images with 512 columns by 480 rows of pixels. Some explanation of these times follows:

COMMAND, comments

ATR1 seconds   ATR2 seconds   ATR3 seconds

This is the format for the following entries. The command, capitalized, and a comment in italics are in the first line; the execution times for each of the three hardware systems, in seconds, are in the second line; discussion is in following lines. The ± values given with the times are not statistical uncertainties; they are absolute limits on the accuracy of the time measurements, imposed by the limited resolution of the accessible clock.

;remark, remark with leading semicolon

< 0.0001   < 0.0001   < 0.0001

This is the time required for the master to read a command from the feature calculation file (script file) and dispose of it without any attempt at command interpretation. The times listed here are for a remark of minimal length, ";" with no real remark text. These times are strongly affected by the time required to read a line of ASCII characters from a disk file; the times will be greater for longer remarks.

REM, remark without echo

0.0003 ± 0.0001   0.0001 ± 0.0001   < 0.0001

This is the time required for the master to read a command from the feature calculation file and do minimal command interpretation. Command interpretation times are different for different commands and they are generally greater for greater numbers of parameters. The time will of course increase if the echo is turned on. The times listed here are for a remark of minimal length, "REM" with no real remark text.

SET_AB, set scale factors

0.0076 ± 0.0001   0.0031 ± 0.0001   0.0003 ± 0.0001

The master sets the values of two variables in a table in the master program, without interacting with the slaves.

CLEAR, clear memory, with no memory de-allocation

0.0019 ± 0.0001	0.0008 ± 0.0001	0.0017 ± 0.0001

The master writes values to a table in memory and sends a short message to all the slaves (daisy chain nodes), so this example includes minimal one-way master-to-slave communication. Each slave also writes a table in its own memory and, if images or kernels (operators) are defined, frees those memory blocks. No memory de-allocation was done in this test.

READY, wait until slaves are finished, with slaves not busy

0.0019 ± 0.0001	0.0004 ± 0.0001	0.0059 ± 0.0001

The master sends a short message to all the slaves. Each slave responds separately, and the master waits for all the slaves to respond. Thus, this command involves brief two-way communication between master and slave, with essentially no other operation.

For the three commands SET_AB, CLEAR, and READY, the ATR3 times are in the order expected when inter-node communication time dominates calculation time. For ATR1 and ATR2, the SET_AB command apparently requires more time to do less than the CLEAR and READY commands. This is believed to be because the SET_AB command requires a function call with three parameters, whereas the other two commands use function calls with no parameters. Times being dominated by calculation rather than communication would also account for the CLEAR time being greater than the READY time for ATR2.

DEFKERN, define a kernel (operator)

0.0132 ± 0.0001	0.0048 ± 0.0001	0.0100 ± 0.0001

This operation allocates memory for a local convolution type of kernel (operator), with the whole kernel being stored in each slave's memory. This operation does not set the values of the kernel. The master sends a short message to all the slaves, and writes a few values to a table in memory. Each slave writes a few values to memory, allocates a memory block, and sends a short message back to the master. If the kernel was already defined, the slave also de-allocates the old memory block; this de-allocation was done in this test.

DEFIMG, define an image, with no overlap rows

0.0114 ± 0.0001	0.0040 ± 0.0001	0.0098 ± 0.0001

DEFIMG, define an image, with 10 overlap rows

0.0115 ± 0.0001	0.0040 ± 0.0001	0.0098 ± 0.0001

This operation allocates memory for an image, with different parts of the image being stored in different slave's memories. This operation does not set any pixel values in memory. The master sends a short message to all the slaves and writes a few values to a table in memory. Each slave calculates which part of the image it should store in its own memory, allocates a memory block, and sends a short message to the master. If the image was already defined, as it was in this test, the slave also de-allocates the old memory block. Unlike an earlier version of this software, no significant difference in time is expected when the number of overlap rows is changed. This DEFIMG operation sends slightly shorter inter-node messages than the DEFKERN operation.

ZEROIMAGE, zero an image, with no overlap rows		
0.0250 ± 0.0010	0.0580 ± 0.0010	0.0045 ± 0.0005
	calcs=1.44	calcs=18.52
ZEROIMAGE, zero an image, with 10 overlap rows		
0.0350 ± 0.0010	0.0640 ± 0.0010	0.0045 ± 0.0005
ADD, add two images		
0.043 ± 0.001	0.124 ± 0.001	0.038 ± 0.001
	calcs=1.16	calcs=3.77
MUL, multiply two images		
0.049 ± 0.001	0.117 ± 0.001	0.038 ± 0.001
	calcs=1.40	calcs=4.30
DIV, divide two images		
0.239 ± 0.001	0.351 ± 0.001	0.066 ± 0.001
	calcs=2.27	calcs=12.07
SQRT, square root of an image		
0.288 ± 0.001	0.286 ± 0.001	0.080 ± 0.001
	calcs=3.36	calcs=12.00
ABS, absolute value of an image		
0.105 ± 0.001	0.159 ± 0.001	0.014 ± 0.001
	calcs=2.20	calcs=25.00
MAXCON, clip low end values of an image, if all pixels are clipped		
0.055 ± 0.001	0.134 ± 0.001	0.014 ± 0.001
	calcs=1.37	calcs=13.10
MAXCON, clip low end values of an image, if no pixels are clipped		
0.052 ± 0.001	0.119 ± 0.001	0.014 ± 0.001
	calcs=1.46	calcs=12.38
ADDCON, add a constant to an image		
0.037 ± 0.001	0.096 ± 0.001	0.012 ± 0.001
	calcs=1.28	calcs=10.28
MULCON, multiply an image by a constant		
0.042 ± 0.001	0.104 ± 0.001	0.012 ± 0.001
	calcs=1.34	calcs=11.67
DIVCON, divide an image by a constant		
0.044 ± 0.001	0.120 ± 0.001	0.012 ± 0.001
	calcs=1.22	calcs=12.22

For each of these simple arithmetic image manipulation operations, the master sends a short message to all the slaves. Each slave does the indicated pixel-by-pixel operation for its part of the specified images. There is no inter-slave communication. The bulk of the time is for the actual image manipulation. These times should be inversely proportional to the number of slave nodes, except for the ZEROIMAGE operation with overlap rows.

CONVOLVE, correlate an image with a kernel (operator), for a 5x5 pixel kernel

1.56 ± 0.01	3.09 ± 0.01	0.30 ± 0.01
	calcs=1.68	calcs=17.33

The master sends a short message to all the slaves. Each slave does the convolution calculation directly (no Fourier transform) for its part of the image. This operation normally must be preceded by the OVERLAP operation to set the values of the pixels in the overlap rows. This time should be inversely proportional to the number of slave nodes and roughly proportional to the size of the kernel domain.

SSQ, local sum of squares, for 5x5 pixel kernel

1.97 ± 0.01	3.43 ± 0.01	0.37 ± 0.01
	calcs=1.91	calcs=17.75

The master sends a short message to all the slaves. Each slave calculates a weighted sum of squares of values of the pixels in the local region indicated by the domain of the specified kernel, with weights equal to the kernel values. This operation normally must be preceded by the OVERLAP operation to set the values of the pixels in the overlap rows. This time should be inversely proportional to the number of slave nodes and roughly proportional to the size of the kernel domain.

OVERLAP, set pixel values in overlap rows, for 2 overlap rows

0.0169 ± 0.0001	0.0076 ± 0.0001	0.0300 ± 0.0010
	comms=2.22	comms=0.56

OVERLAP, set pixel values in overlap rows, for 10 overlap rows

0.0604 ± 0.0001	0.0303 ± 0.0001	0.0800 ± 0.0010
	comms=1.99	comms=0.755

The master sends a short message to all the slaves. The slaves exchange the values of the pixels in the overlap rows; thus, there is a lot of inter-slave communication, with one long message in each direction for each overlap row. This time is expected to be roughly proportional to the number of overlap rows and independent of the number of slave nodes.

SMTHX, smooth in X direction, for 5 passes

0.720 ± 0.010	1.140 ± 0.010	0.105 ± 0.001
	calcs=2.11	calcs=22.86

The master does a short calculation and sends a short message to all the slaves. Each slave smooths its own part of the indicated image without communication with other slaves. The smoothing function is more Gaussian for more passes, with 5 passes being a good approximation to a real Gaussian and 1 pass having a sharp cusp in the smoothing function. The time is proportional to the number of passes and inversely proportional to the number of slave nodes.

SMTHY, smooth in Y direction, for 5 passes

1.87 ± 0.01	2.08 ± 0.01	9.10 ± 0.10

This is like SMTHX, X direction smoothing, except that this Y direction smoothing requires significant inter-slave communication. Two short messages per image column per pass are sent by each slave. This time should be roughly inversely proportional to the number of slave nodes only if the calculation time dominates the communication time. The disproportionately long time required by ATR3 for this operation is believed to be due to communication overhead.

MEDIAN, two-dimensional median filter, for a 21-pixel window

11.7 ± 0.1	17.9 ± 0.1	1.8 ± 0.1
	calcs=2.18	calcs=21.67

The master sends a short message to all the slaves. Each slave does the calculation for its part of the image without inter-slave communication. This operation normally must be preceded by the OVERLAP operation to set the values of the pixels in the overlap rows. This time should be inversely proportional to the number of slave nodes.

UNDERSAMPLE, decimate (undersample) an image, for decimation by a factor of 2 in each direction with a pre-defined destination image

0.019 ± 0.001	0.114 ± 0.001	0.010 ± 0.001
	calcs=0.555	calcs=6.33

The master sends a short message to all the slaves. If the destination image is not already defined, this operation defines it; in this test, the destination image was already defined. This time should be inversely proportional to the number of slave nodes. For unknown reasons, ATR2 seems to have an anomalously low speed for this operation.

SCALE, scale an image

0.175 ± 0.001	0.354 ± 0.001	0.037 ± 0.001

The master sends a short message to all the slaves. Each slave scans its part of the specified image for the maximum value, and then the slaves daisy-chain communicate the maximum value to the master. The master then sends a short message to all the slaves, and each slave multiplies its part of the image by a constant. Thus, there is some master-slave and inter-slave communication involved in this scaling operation. This time should be approximately inversely proportional to the number of slave nodes.

READIMAGE, read an image

1.50 ± 0.10	1.10 ± 0.10	1.00 ± 0.10
	comms=1.36	comms=1.5

WRITEIMAGE, write an image

2.40 ± 0.10	1.85 ± 0.05	1.25 ± 0.05
	comms=1.30	comms=1.92

These times are for the host system disk. Accessing the ATR1 SCSI disk takes a little longer. These times should be essentially independent of the number of slave nodes. They depend on external factors such as the disk speed, the bus speed, and perhaps the host operating system or other data transfer software.

QUADUV, fit every local region with a quadratic polynomial, for a 21-pixel kernel

108.0 ± 1.0	147.0 ± 1.0	41.0 ± 1.0
	calcs=2.45	calcs=8.78

This requires a substantial amount of calculation, including a 6-parameter least squares fit, for every pixel. This time should be inversely proportional to the number of slave nodes.

BRANCH, in a minimal loop		
0.0218 ± 0.0002	0.0086 ± 0.0001	0.0012 ± 0.0001
JUMP, in a minimal loop		
0.0222 ± 0.0002	0.0088 ± 0.0001	0.0013 ± 0.0001

The times for BRANCH and JUMP commands vary greatly, depending mainly on the length of the feature command file (the user command script), particularly that part of the file before the BRANCH or JUMP command. The times will also depend on disk file access times or disk caching capabilities. These times are for unrealistically small feature command files. These times depend strongly on the speed of the host computer for command files of significant size.

In systems of this type, two different speeds are of interest: the speed of calculations within a node, and the speed of communication between nodes. Which is more important in limiting the overall speed of the system depends strongly on the particular application. In the above timing measurements, some commands are dominated by calculation time, and some by communication time.

For those timing measurements that are almost totally indicative of calculation time, a value is given for "calcs" for ATR2 and ATR3. This is the relative calculation speed per node, relative to ATR1. It is calculated by

$$\text{calcs} = \frac{(\text{ATR1 time}) * (\text{number of ATR1 nodes})}{(\text{ATR2 time}) * (\text{number of ATR2 nodes})}$$

for ATR2, with the obvious analog for ATR3. The values for this speed ratio vary significantly for different types of operations, because of the internal differences in the processors. There is a tendency toward a greater speed advantage for ATR2 and ATR3 for the more complicated calculations. We use as the representative calculation-intensive operation the two-dimensional median filter, MEDIAN. For this operation, ATR2 has a relative speed per node of 2.18, and ATR3 has a relative speed per node of 21.67. The important relative calculation speed per cost, which is simply the relative speed per node divided by the relative cost per node, is about 1.18 for ATR2 and 2.32 for ATR3.

This software package contains no operations that are purely communication. Even such operations as reading an image from a disk file involve a significant amount of calculation, in calculating array indexes and converting from 8-bit integers to 32-bit floating point values. One indication of relative communication speed is the OVERLAP operation, in which 512 floating point values (2048 bytes) per data transfer are exchanged between neighboring processors without type conversion and the only arithmetic is array index calculation. For this, like most inter-node communications, it is not reasonable to speak of speed per node, because the time of the overall operation usually does not depend significantly on the number of nodes. In the typical inter-node communication, there is in effect a parallelism with many pairs of nodes communicating simultaneously, so that the overall time is essentially the time required for any one pair of nodes to communicate. Thus, we represent the relative communication speed by the ratio of the times:

$$\text{comms} = \frac{(\text{ATR1 time})}{(\text{ATR2 or ATR3 time})} .$$

This relative communication speed, as measured for the OVERLAP operation for 10 overlap rows, is 1.99 for ATR2 and 0.755 for ATR3. The communication slowness of ATR3 is surprising, in light of the raw A-link speed of 250 Mbit/second compared with the ATR1 T-link speed of 20 Mbit/second. The explanation is that the ATR3 system has a substantial per-message overhead time, a time required to prepare for inter-node communication before the transmission actually takes place. This is apparent in the very long time required for ATR3 to do the SMTHY operation, which involves many short messages exchanged between neighboring nodes. A comparison of the SMTHY and SMTHX operations (SMTHX requires no inter-node communication, but is otherwise very similar to SMTHY) gives a quantitative estimate of the overhead time per message pair, one message arriving and one message leaving via a different link. These times for the three systems are included in Table 1. These values may not be precise, but they are believed to be at least reasonable approximations.

For typical image analysis or target search applications of the ATR software, ATR1 and ATR2 seem reasonably balanced, with neither calculation speed nor communication speed being a dominant factor in limiting the overall system speed. However, for ATR3 the communication slowness is an obvious, major limit to overall system speed. In fact, some applications run substantially faster on a single processor (the ATR3 host) than on the set of 5 ATR3 parallel processors. This obviously makes the use of parallel processors absurd, for this particular situation. To take advantage of the potential speed advantage of parallel processing in the ATR3 system, a change is required in the basic approach used in the ATR software. The present software distributes each image among the several daisy nodes; in ATR3, for example, 1/5 of each image resides on each of the daisies. This requires some inter-node communication for certain image analysis operations. Those operations that use many short messages between nodes cause the large loss of speed in the ATR3 system. In some cases, it would be possible to consolidate many short messages into one long message and thereby eliminate a large part of the message set-up overhead time. However, this is not possible for some image analysis operations.

The only reasonable approach to circumventing the ATR3 communication time problem seems to be a change in the basic approach used in the ATR software: instead of dividing each image among the several daisy nodes, each scene should be analyzed on a single node. This approach has its own problems. First, it imposes a more restrictive limit on how big a scene can be analyzed in one step, with the memory of one node instead of the combined memory of all the nodes being the new limiting factor. This is not a very serious problem, since it is usually not too difficult to divide a large scene into smaller scenes for separate analysis. Second, if only one scene is available at any one time, only one node will be used and all the other nodes will be wasted. This is not a problem if there are many scenes waiting for analysis. It seems that the new software approach should be tried.

CONCLUSIONS

This Automatic Target Recognition system can be very effective for rapid screening of large amounts of multi-image data. This system is versatile and should be broadly applicable. It incorporates image data fusion and uncertainty estimation. The software is easily expandable to incorporate new capabilities, and the hardware is easily expandable to increase the speed. This ATR system also informs the user of the importance of each feature used in the analysis, so that the user can make the process more efficient by eliminating unimportant features from the calculations and perhaps eliminating unimportant sensors from the multisensor data acquisition system. The software and the three hardware systems have been evaluated and found fully functional.

For the most advanced hardware system (ATR3), with the greatest calculation speed per parallel processor node, the technology of fast calculation has outpaced the technology of fast inter-node communication. Taking full advantage of this calculation speed will require substantial revision of the ATR software. This will be undertaken and reported separately.

REFERENCES

1. G. D. Lassahn, *Automatic TLI Recognition System, User's Guide*, INEL/EXT-97-00004, February 1997.

2. G. D. Lassahn, *Automatic TLI Recognition System, Programmer's Guide*, INEL/EXT-97-00005, February 1997.

3. *Transputer Databook*, second edition, INMOS/SGS-THOMSON, 1000 East Bell Road, Phoenix, AZ 85022, 1989.

4. *T9000 Transputer Hardware Reference Manual*, INMOS/SGS-THOMSON, 1000 East Bell Road, Phoenix, AZ 85022, 1993.

APPENDIX A

EXAMPLE 1: Median Filter Testing

EXAMPLE 1: Median Filter Testing

This example is presented to illustrate very simply what the ATR system does. In this example, we use only one training scene, and we have only one image per scene. The goal in this application is to find medium-sized lettering and reject large lettering, small lettering, and other kinds of texture. This example was actually intended as a test of the one-dimensional median filter calculation subroutines, but it does demonstrate some ATR system capabilities.

The training scene image is shown in Figure A1, color coded to indicate the target and background regions specified by the user with program G. The green regions are designated as targets for the training program. They include only the medium-sized lettering, in the top left and the bottom right corners of the scene. The red regions, designated as background, include the larger lettering, in the top right corner and at the left edge; some of the small lettering, at the bottom of the scene; and, a large amount of the picture engraving. The blue regions are not designated as either target or background, and are ignored by the training process. Note that it is not necessary to mark all of the small lettering, for example, as background; if some of it is marked as background and none of it is marked as target, that should be sufficient.

Two different values of the type 1 error weight coefficient W_1 were tried: 0.5 and 0.9. The latter, larger value gives more weight to errors in which the ATR system incorrectly classifies a background pixel as a target pixel, whereas the W_1=0.5 value gives equal weight to this type 1 error and type 2 errors in which a target pixel is incorrectly classified as a background pixel. The two different values of W_1 give significantly different results. Figure A2 shows the error rate versus number of features for both of two training program runs. The error rate values above 0.15 are for the first run, and the values below 0.05 are for the second run. That is, we get a smaller total (weighted sum) error rate if we attach more importance to type 1 errors, for this training data.

In this example, the coefficients chosen for use in the surveillance process are those for the second training calculation, with W_1=0.9. These coefficients were installed in the surveillance program data files. The surveillance program was then used to analyze four scenes, looking for medium-sized lettering. The four scenes, and the surveillance program result images obtained for each scene, are shown in Figures A3-A6. One of these surveillance scenes was also used as a training scene. Of course, when the image is used as a surveillance scene, we do not tell the ATR system which parts are target and background; we let the ATR computer tell us. All of these images were originally the same size as the training image of Figure A1, but they were reduced in size for easier presentation here. Unfortunately, the images loose a little quality and detail in the reduction and reproduction for printing. Nevertheless, the results clearly illustrate how the ATR system works.

In the result images, the left half of each of Figures A3-A6, the bright, green to pale green to white, regions indicate what the ATR system classified as targets; the dark, red to black, regions indicate background. The lighter regions are stronger indications of target.

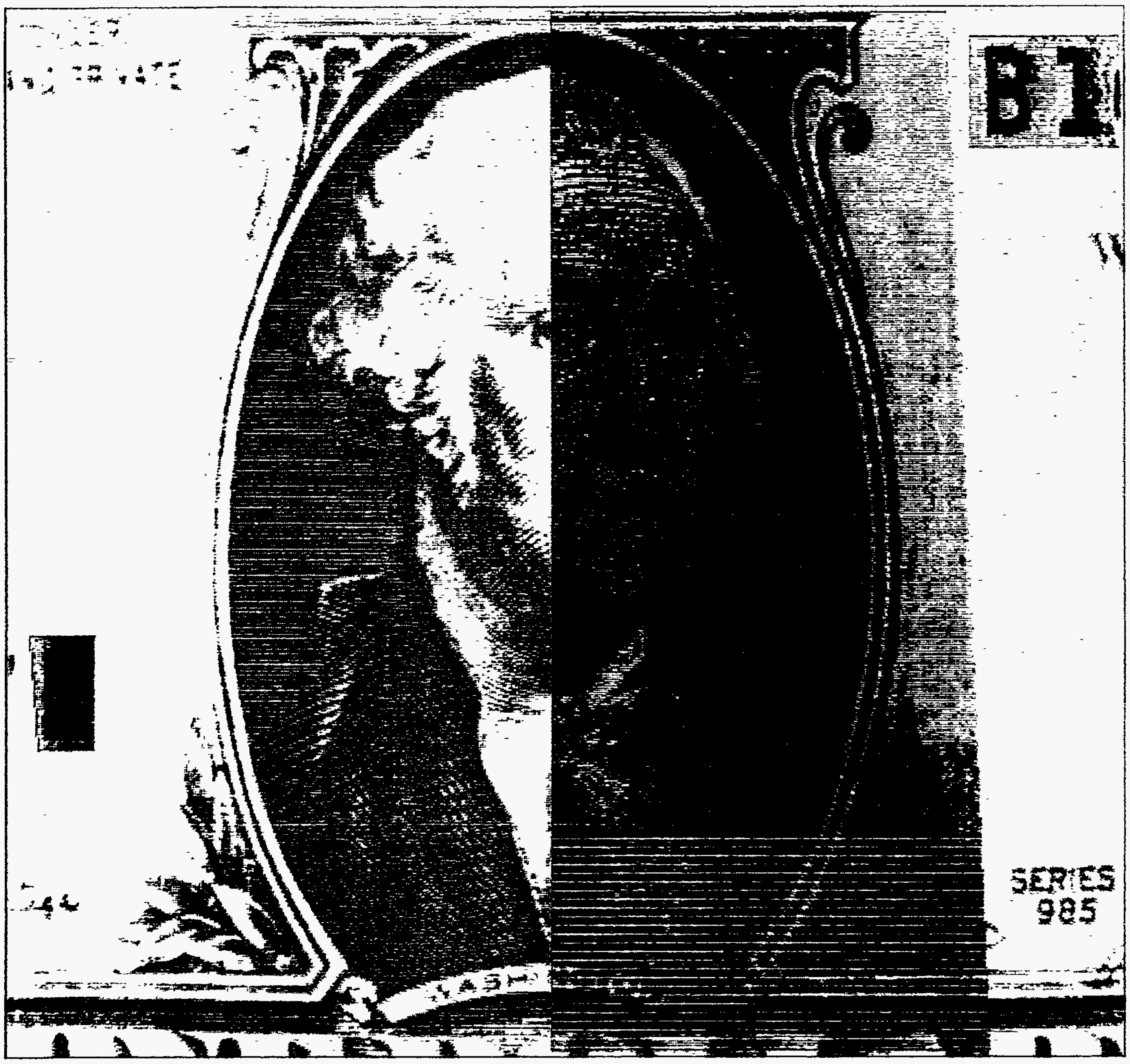


IMAGE FILE: **\img\DEMO4.img**
MASK FILE: **\img\M4D.img**

Figure A1: Training scene image with color-coded target (green), background (red), and unspecified (blue) regions.

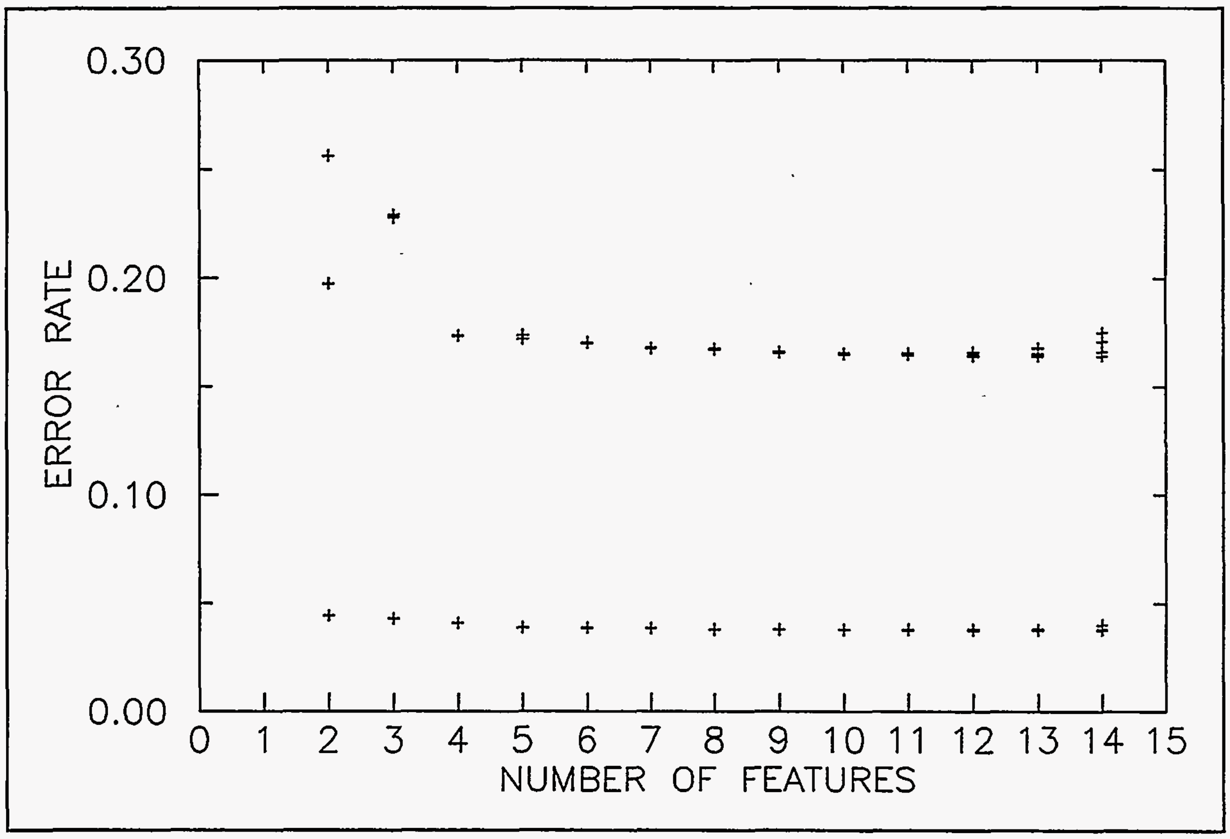


Figure A2: Total (weighted sum) error rate versus number of features for two training program calculations.

Figure A6 shows that the ATR system did correctly designate the medium-sized lettering as target, and it did correctly reject the larger letters as background. The results are mixed for the small letters at the bottom center of the scene, some being correctly rejected as background and some being incorrectly identified as targets. A few other regions in the Figure A6 scene, as well as a few small regions in the other scenes, are incorrectly identified as targets by this ATR system. These incorrect identifications simply mean that this ATR system, with the particular features which were chosen for this application, cannot distinguish between medium-sized lettering and whatever is shown in the scene images at those regions incorrectly identified as targets. Presumably, the operator could choose a better set of features and reduce the incidence of errors in this ATR application. Note, however, that the relatively small set of relatively simple features used in this simple example did quite well: there are essentially no type 1 errors (incorrect designation of targets as background), and there are not many type 2 errors (incorrect designation of background as target). The absence of type 1 errors is largely due to the large value (0.9) chosen for the type 1 error weight in the F optimization calculation.

Conclusion

As expected, with a limited feature set (one-dimensional median filters only) and limited training (only one training scene), the ATR system is quite effective but not perfect at distinguishing targets from background.

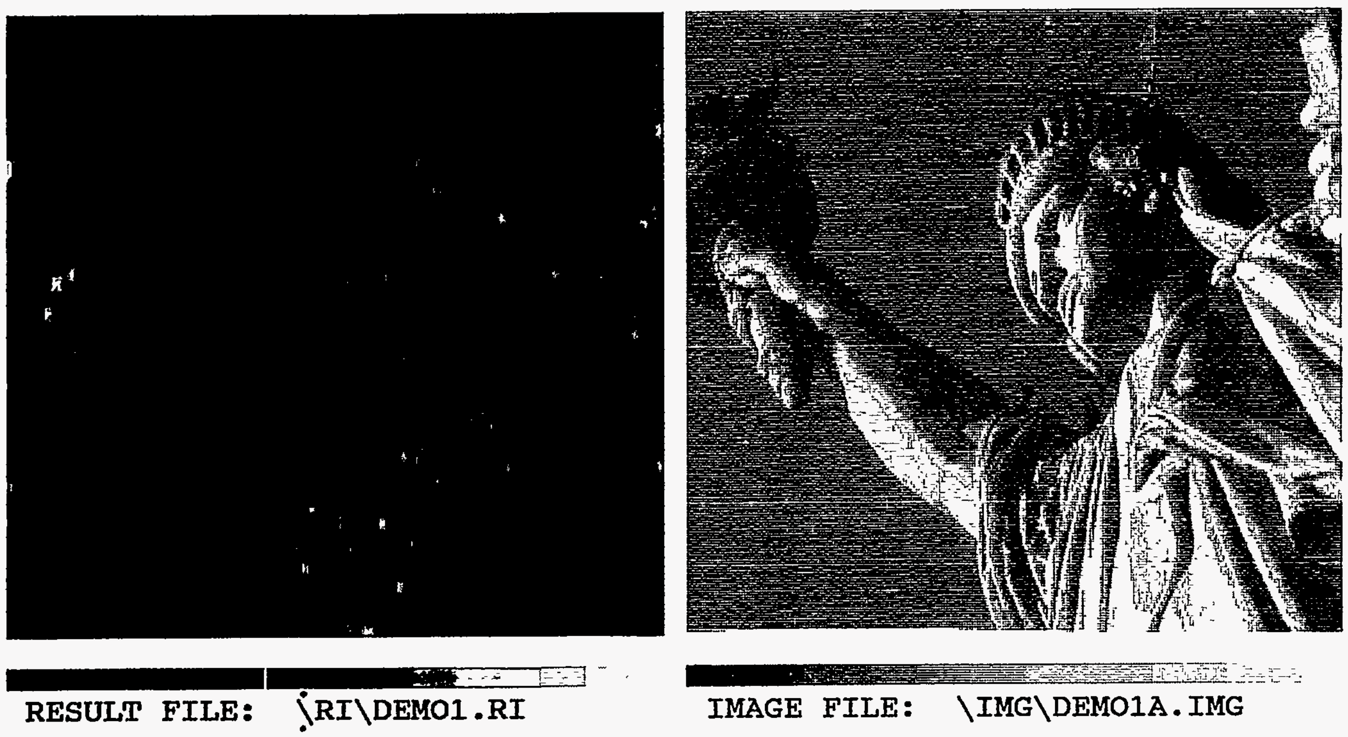


Figure A3: Result image and scene image for DEMO1.

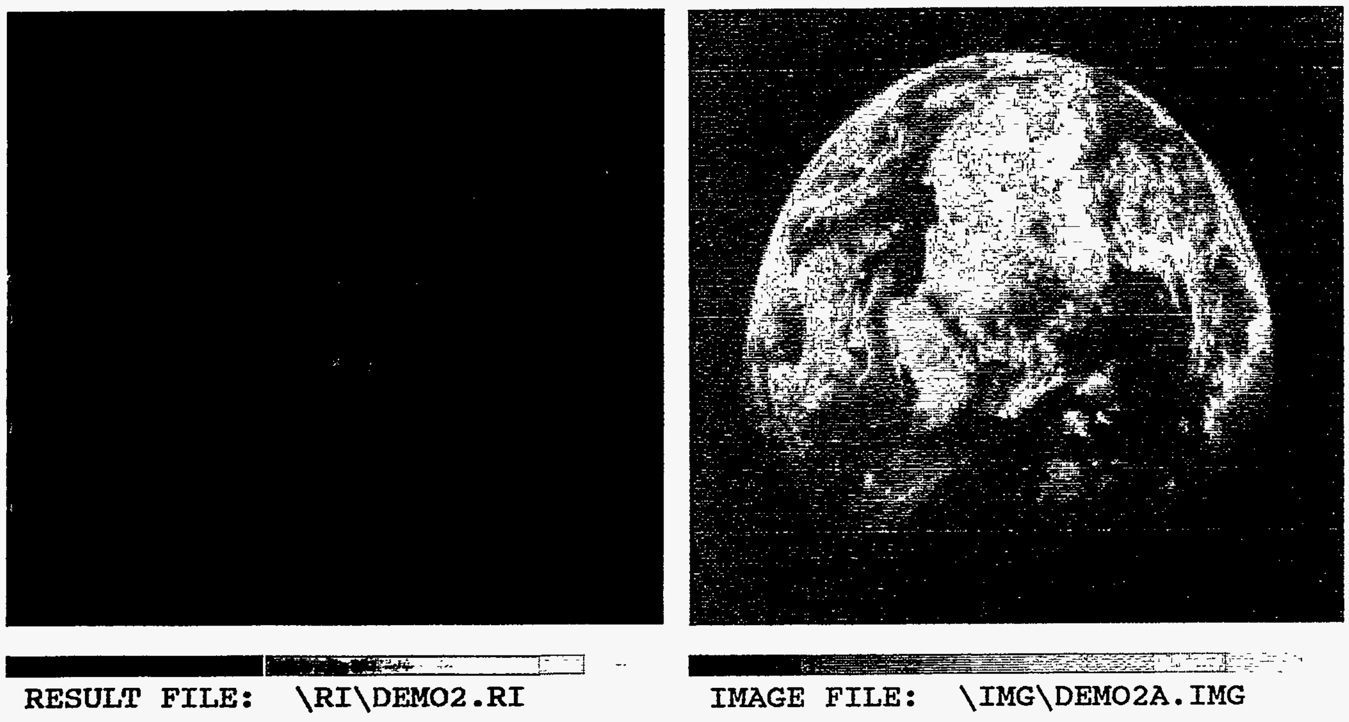
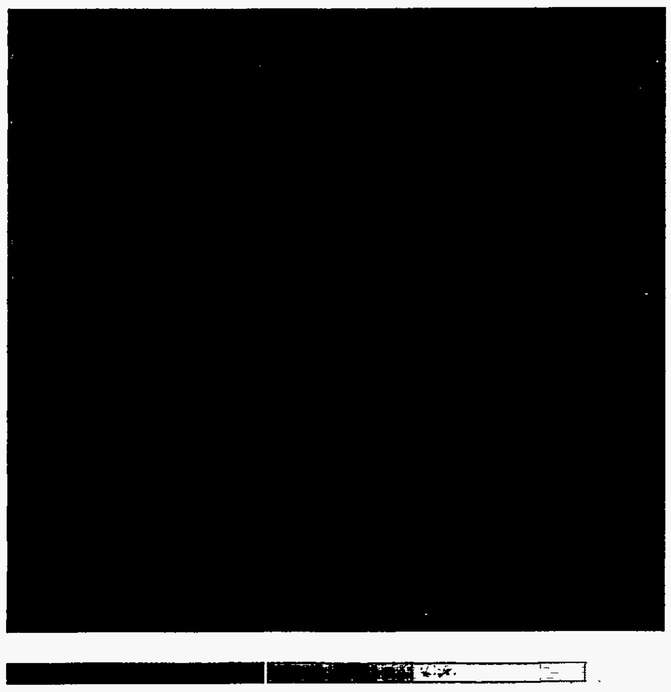


Figure A4: Result image and scene image for DEMO2.


RESULT FILE: \RI\DEMO3.RI

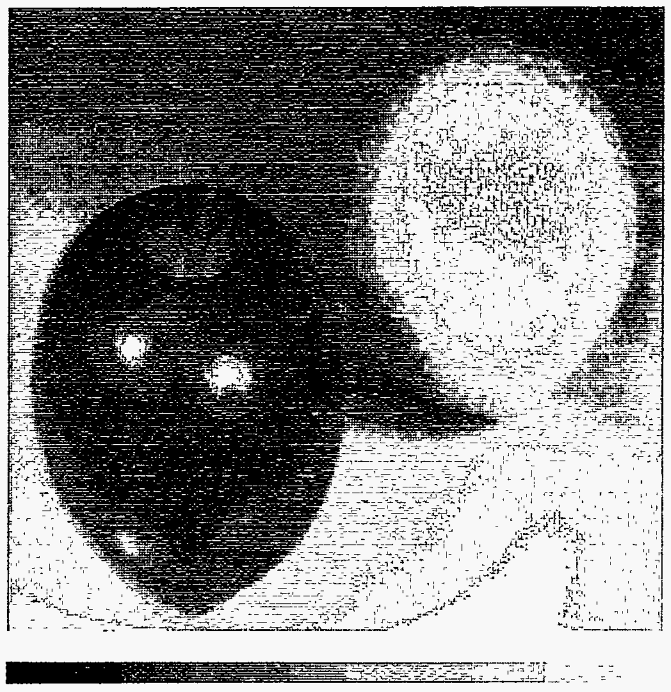
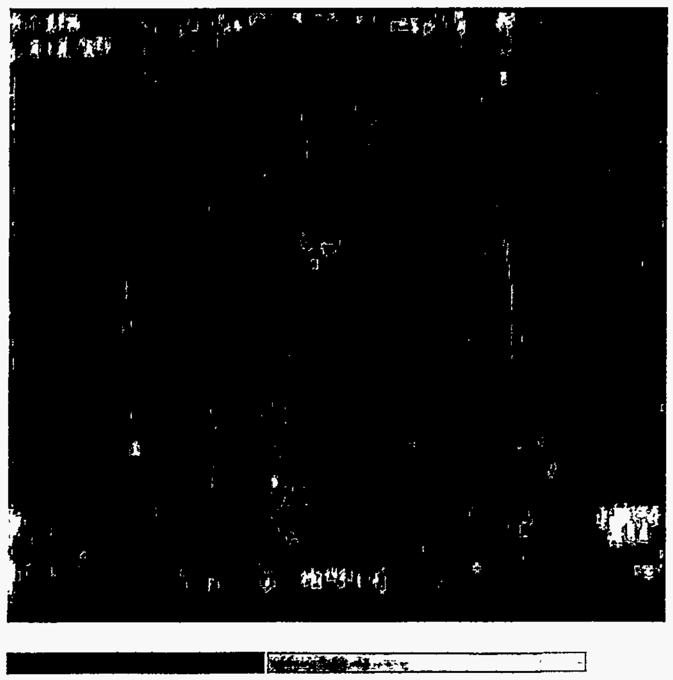


IMAGE FILE: \IMG\DEMO3A.IMG

Figure A5: Result image and scene image for DEMO3.


RESULT FILE: \RI\DEMO4.RI

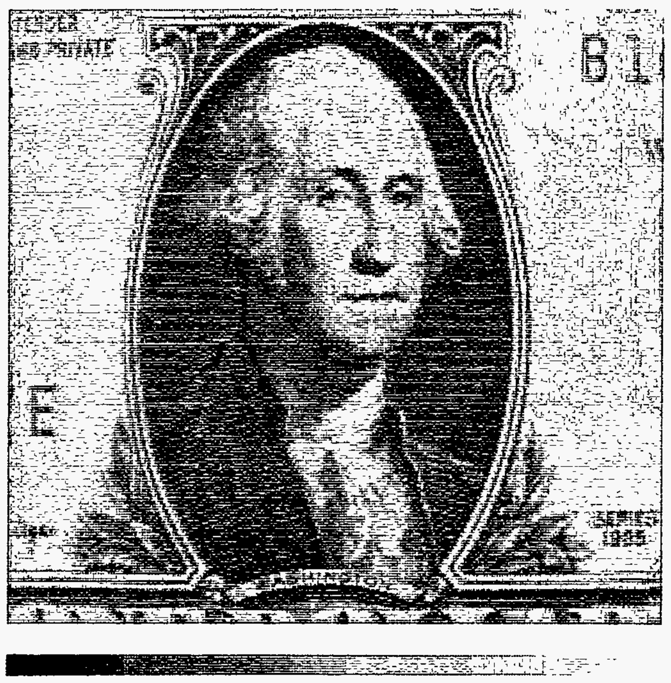


IMAGE FILE: \IMG\DEMO4A.IMG

Figure A6: Result image and scene image for DEMO4.

APPENDIX B

EXAMPLE 2: Roads and Riverbanks

EXAMPLE 2: Roads and Riverbanks

This simple example illustrates fusion of image data. We have two images of the same scene, one visible light and one infrared image. (These images are part of a set supplied by Karen Steinmaus of Battelle, Pacific Northwest Laboratories, one of the participants in the Department of Energy's Airborne Multisensor Pod System project.) The visible light image (Figure B1) shows roads quite clearly, but it also shows riverbanks and it is difficult to distinguish between the two features in this image. The infrared image (Figure B2) does not show the roads very well, but it clearly indicates where the river is. The two images together should allow us to find roads and reject riverbanks. Note, however, that this cannot be done by looking for roads in each of the two images separately and then simply adding or averaging the two results; a more sophisticated approach to image data fusion, such as that used in this ATR system, is required.

For this illustration, we do a very simple analysis using only 4 features. For the first feature, we do a convolution of the visible light image with a 13x13 pixel kernel in which the pixel values are proportional to X^2 with the mean subtracted out, clip the result to keep only negative values, and take the absolute value. This first feature indicates the presence of both vertical roads and vertical riverbanks.

For the third feature, we do a convolution of the infrared image with a 13x13 pixel kernel in which the pixel values are proportional to X, and square the result. This third feature is insensitive to roads, but shows vertical riverbanks very well.

The second and fourth features are analagous to the first and third, using kernels with dependence on the Y coordinate instead of the X coordinate to detect roads and riverbanks that run horizontally in the images instead of vertically.

Figures B1 and B2 both show the operator-selected target (green) and background (red) regions used in the training process. Note that these regions do not need to be marked on both of the scene images separately; it is sufficient to use either one of the scene images with the program G to mark the target and background regions. The training process was done with these two images (one scene) as the training data. The optimized coefficients were put into the surveillance program file, and the surveillance program was then used to analyze this same scene. The result image is shown in Figure B3. In this result image, the white is the strongest indication of roads, darker green is a weaker indication of roads, black is the strongest indication of background (non-road), and lighter red is a weaker indication of background. This result does distinguish clearly between road and riverbank, thus satisfying the goal for this illustration: the two images analyzed jointly give a clear indication of a result that is not obvious in either image separately. This result indicates horizontal roads more strongly than vertical roads; this is because the horizontal roads appear narrower in these images, and the particular convolution calculations used here are more sensitive to narrower features. This result also indicates as roads some regions that are neither road nor riverbank; this is not surprising, since no significant effort was made to exclude miscellaneous clutter from being identified as target in this very simple example.

Conclusion

This minimal example demonstrates image fusion: neither image alone allows finding roads and distinguishing them from riverbanks, but the two images analyzed jointly, with a very simple algorithm, accomplish the goal very easily.

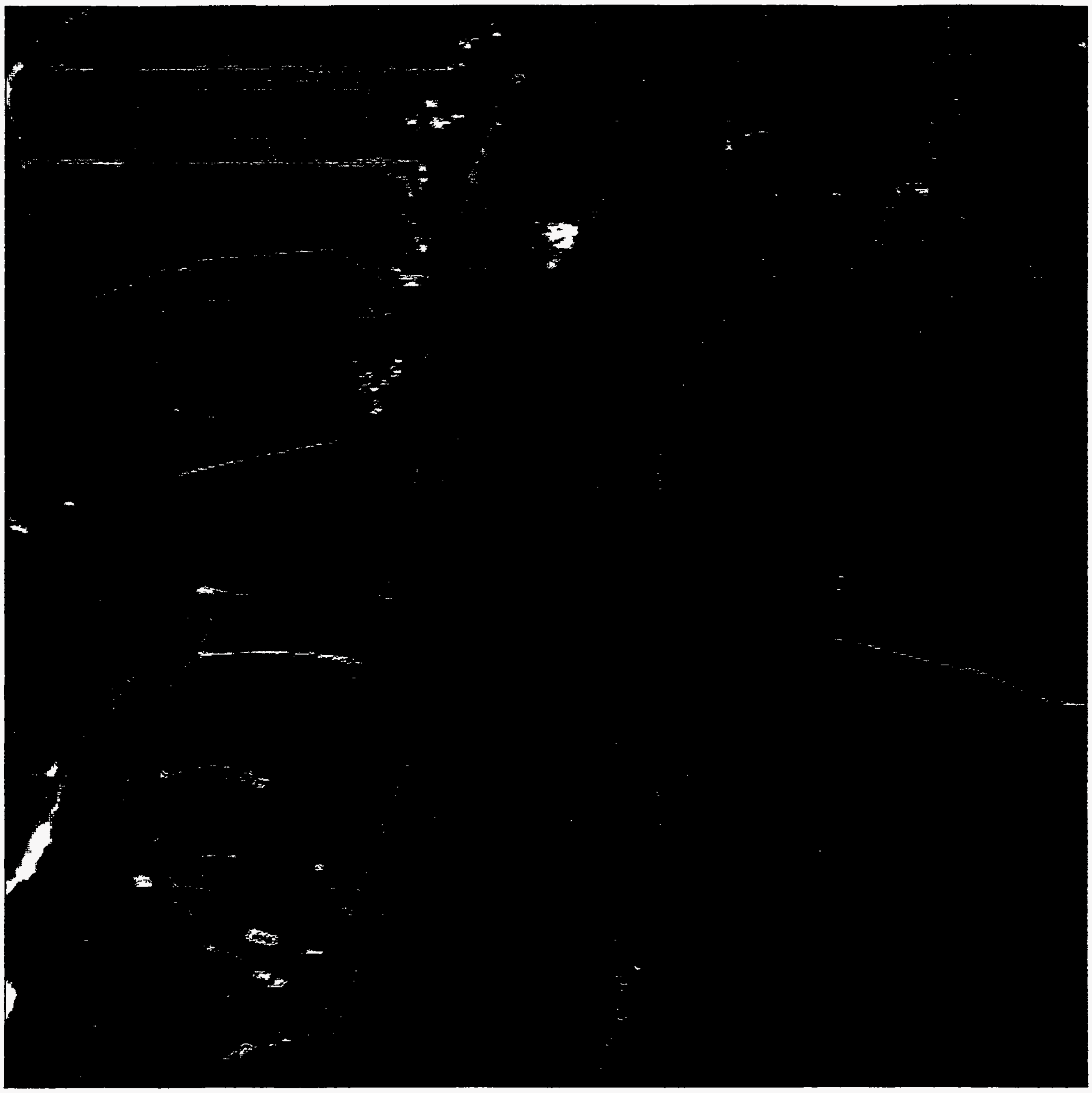
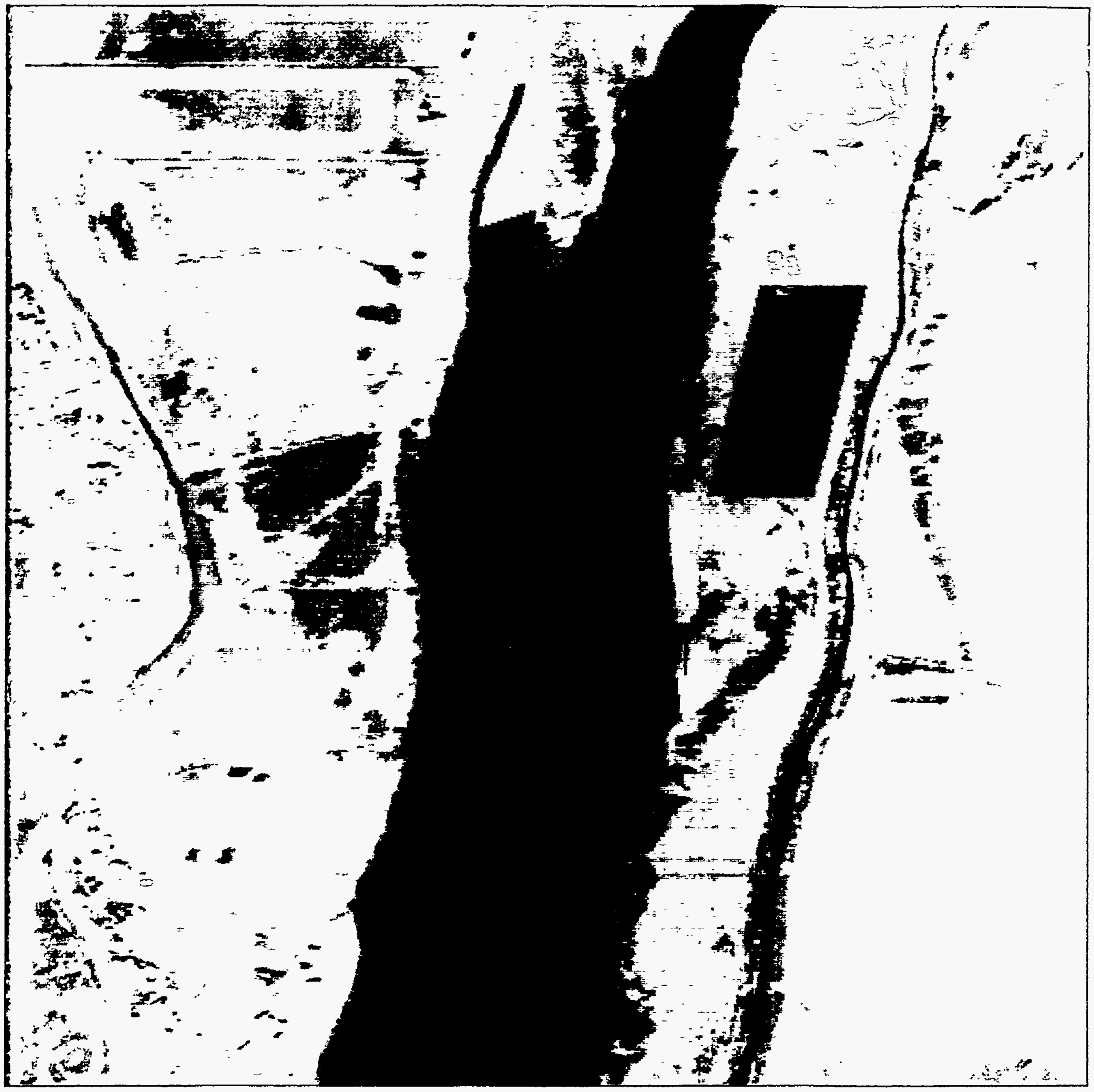


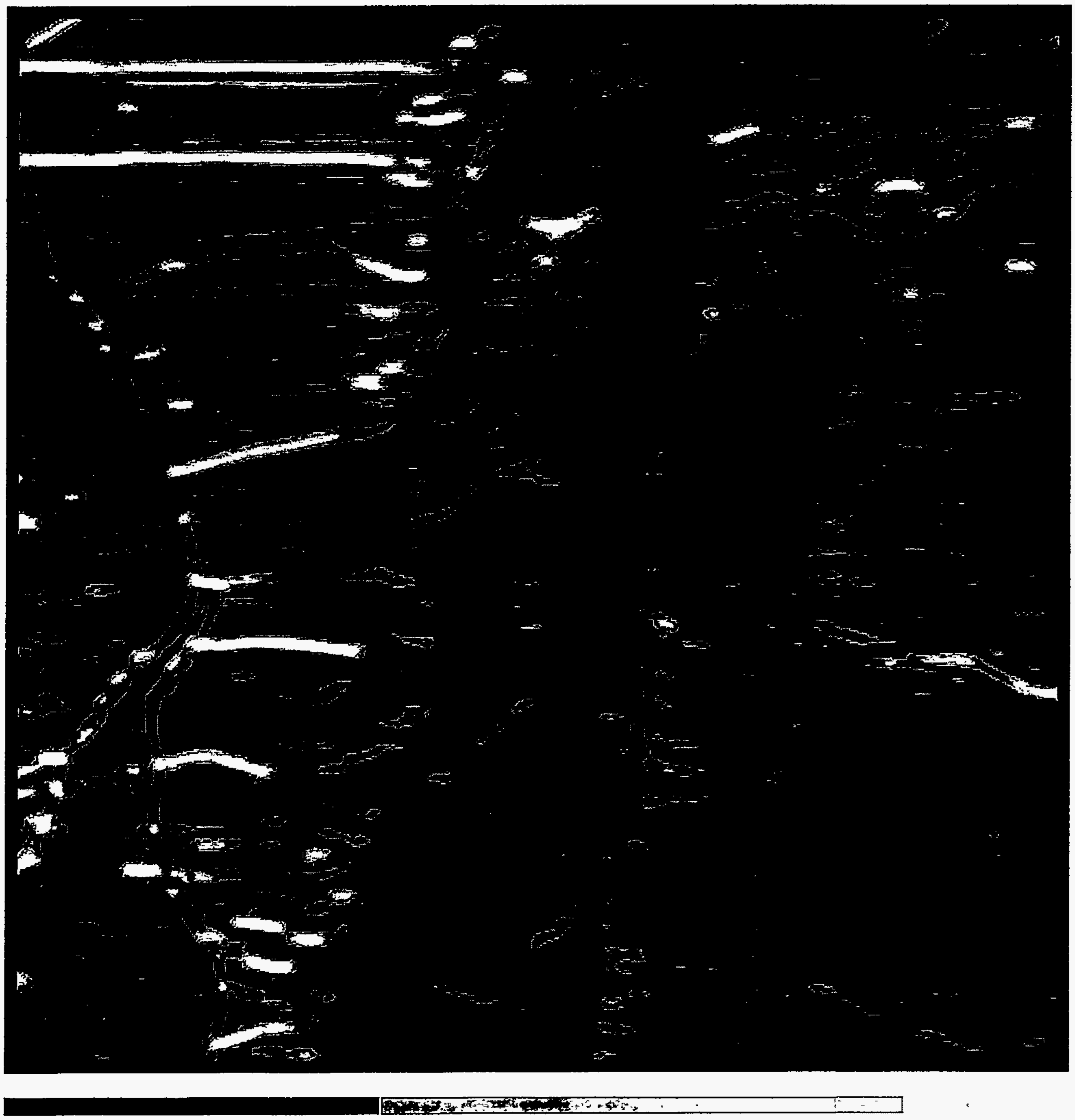
IMAGE FILE: image1b.img
MASK FILE: PNLmsk03.img

Figure B1: Visible light image, color coded to indicate operator-selected target (green) and background (red) regions.



```
IMAGE FILE:  image6b.img
 MASK FILE:  PNLmsk03.img
```

Figure B2: Infrared image, color coded to indicate operator-selected target (green) and background (red) regions.



```
RESULT FILE:  epnl2b.ri
```

Figure B3: Result image.

APPENDIX C

EXAMPLE 3: Buried Waste Location

EXAMPLE 3: Buried Waste Location

This example illustrates the use of the ATR system with data that is not standard image data, and the use of a completely naive but still quite effective analysis method in which the ATR system's capabilities replace operator understanding of the data. The primary purpose of this study was to assess the utility of this ATR system in buried waste recovery operations. This data is from the Department of Energy's Buried Waste Integrated Demonstration project at the Idaho National Engineering Laboratory. The images in the figures in this example should be regarded as maps of areas from which buried waste is being excavated.

In this application, as many as 7 sensors acquired different types of data. This data is not image data in the traditional sense. Rather, each "image" is merely a set of values measured at a two-dimensional array of points on the surface of the ground. Sensor S1A is the vertical component of the earth's magnetic field, and S1B is the gradient (derivative with respect to vertical position) of the vertical component of the magnetic field. S2A, S3A, S4A, and S5A are measurements of the electrical conductivity of the soil, like eddy current measurements, taken with different combinations of field orientation and phase shift. S6A is a volatile chemical sensor. Most scenes include measurements from the first six sensors. Because data from the seventh sensor S6A was available for only 3 of the 35 scenes, and for one of these three scenes no other sensor data was available, this seventh sensor data was not used in this brief study.

There were 5 experiments, referred to as E1, E2, E3, E5, and E6. The different experiments represent data recorded over five different areas with different buried objects. The five rectangles on the left of Figure C1 indicate the ***approximate*** sizes and locations of buried objects in the five experiments, as seen from above. In these sketches, the solid objects are magnetic, and the objects drawn with outlines only are, at least mostly, not magnetic. In E2, for example, there are a magnetic barrel (object 7) and a wooden box (object 6). Objects 9 and 10 in E3 are two boxes, one above the other, with some magnetic material in the top box. The top box was removed between L3 and L4 (to be described later). Similarly, object 20 (a vertical barrel) in E5 was removed between L2 and L3, and object 27 (a vertical steel pipe) in E6 was removed between L2 and L3.

For each experiment, there are several different levels or vertical positions of the sensors. The number of different levels is not the same for all experiments. The several levels are referred to as L0, L1, ... L0 is the highest level, and the sensors are 6 inches lower for each successive level. In some cases, layers of soil were removed between successive measurement levels. In these analyses, any one level of any one experiment is treated as a separate *scene*. Different scenes are expected to give different results for any measurement, because they contain different objects or because the objects are at different distances from the sensors. There are a total of 34 usable scenes in this data set.

The scene images in this data have 43 columns (43 x values, spaced 3 inches apart) and 25 rows (25 y values, spaced 6 inches apart). Some of the data sets did not have this many rows or columns, and some were missing a few data points from what was expected to be a regularly-spaced array. In all of these cases, the missing data points were filled in using a linear interpolation or extrapolation procedure so that each scene image used in this study had a full 43x25=1075 data points.

Perhaps the simplest analysis we can use with this ATR system for a set of data with 6 sensors is a linear combination of the 6 raw data images (augmented with the "constant" image which is always included for mathematical completeness). This simple linear analysis was tried with the buried waste data. The training was done with 3 scenes: E2L7, E5L5, and E6L5. The masks used were M2A, M5B, and M6A, shown in Figure C1. In this figure, the red (actually more brown) areas of the masks are designated as background, the green areas are target, and the black areas are not used in the training process. These masks mark barrels as targets and other regions as background; that is, this analysis is a search for barrels. The results of this training process were used to analyze 30 scenes from experiments E2, E3, E5, and E6; experiment E1 was not included in this analysis because this analysis uses 6 sensors and only 4 sensors were used in experiment E1. The results are indicated in Figure C2. In this figure, as in any of the result images, lighter regions are stronger indications of targets, and darker regions are stronger indications of background. Green and white (that is, very light green) indicate what the ATR system classifies as targets. Red and brown and black (dark red) indicate what the ATR system classifies as background. Figure C2 shows that this analysis did correctly identify the barrel (object 7) in E2 and another barrel (object 17) in E5 as targets when the excavation was deep enough so that the sensors were close to the barrels. This process also incorrectly gave small target indications at the top of E5L5 and in E6L6. These are ferromagnetic objects other than barrels, so it is not too surprising that this simple linear combination algorithm did not distinguish these objects from magnetic barrels. The process failed to detect a barrel at the center of E5, in levels L0-L2 (this barrel was removed between L2 and L3). The reason for this failure is not known. This barrel shows rather weakly in the raw data, suggesting the possibility that it may have been buried too deep to be detected by this algorithm. Although this algorithm for finding barrels did not yield perfect results, it did quite well, especially considering the simplicity of the algorithm and the fact that the algorithm required absolutely no understanding of the measurements or the physical processes involved.

This processes was repeated with an additional training scene, E5L2 with mask M5A, which specifies that the troublesome barrel (object 20) is designated as a target in the training process. The result of this analysis is shown in Figure C3. This process did locate object 20, but it also incorrectly identified a box (object 6) and other objects as targets. This rather poor performance is merely an indication that this simple linear combination algorithm cannot distinguish reliably between the objects designated as targets and those designated as background in this training set. The problem here is that object 20, which we believe to be a barrel, looks to the sensors like some other (background) object more than a target. This is a situation in which we must recognize that the algorithm we are using will not work well with this type of data. The obvious procedure is to try a better algorithm.

The next obvious level of complication in analysis is to add quadratic terms to the linear analysis, adding 21 product terms to the linear and constant terms in the previous analysis. This was tried using the same 4-scene training data set described for the linear case, with the result shown in Figure C4. As would be expected, this quadratic analysis is better than the linear analysis at distinguishing barrels from background, but it is still not perfect.

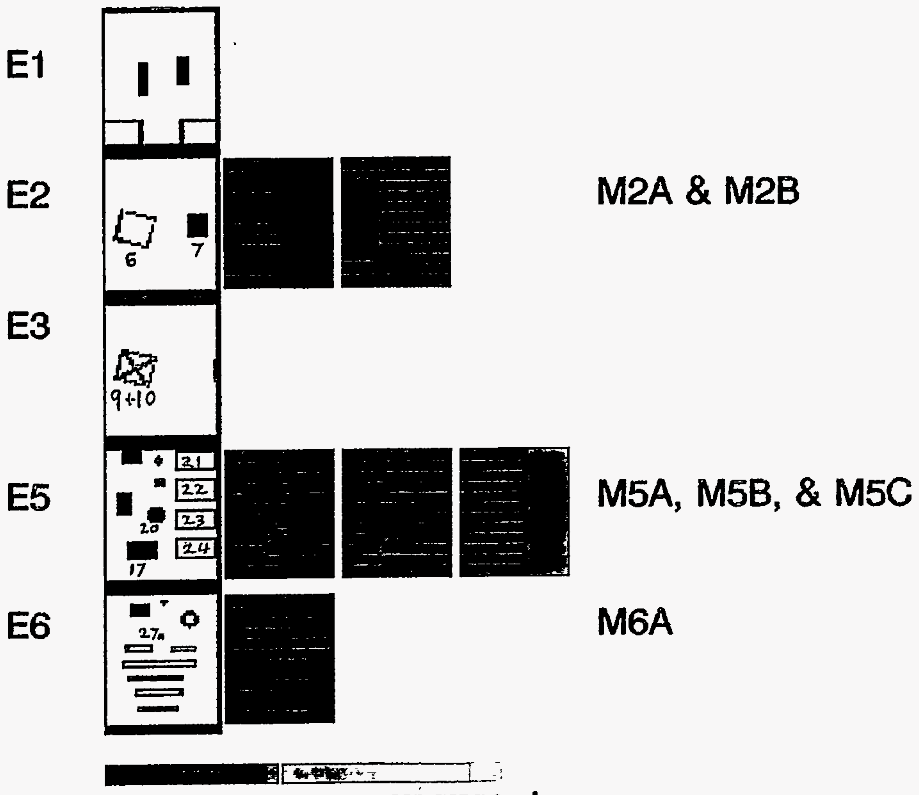
The same quadratic analysis procedure was used to try to find boxes instead of barrels. In this analysis, the training process used two training scenes, E5L5 with mask M5C and E2L7 with mask M2B. These masks select only boxes without ferrous metal contents as targets. The results of this analysis for 30 scenes are shown in Figure C5. This analysis was repeated using only the first training scene, with results shown in Figure C6. Obviously, the results of this type of analysis can depend significantly on the choice of training data. In both of these two analyses, boxes (objects 6, 9, 10, 21, 22, 23, and 24) were correctly identified as targets, and a few other objects were incorrectly identified as targets, with the single-training-scene result a little better than the two-training-scene result. This result is reasonably good.

An interesting aspect of these linear and quadratic analyses is that they require no understanding of the measurements.

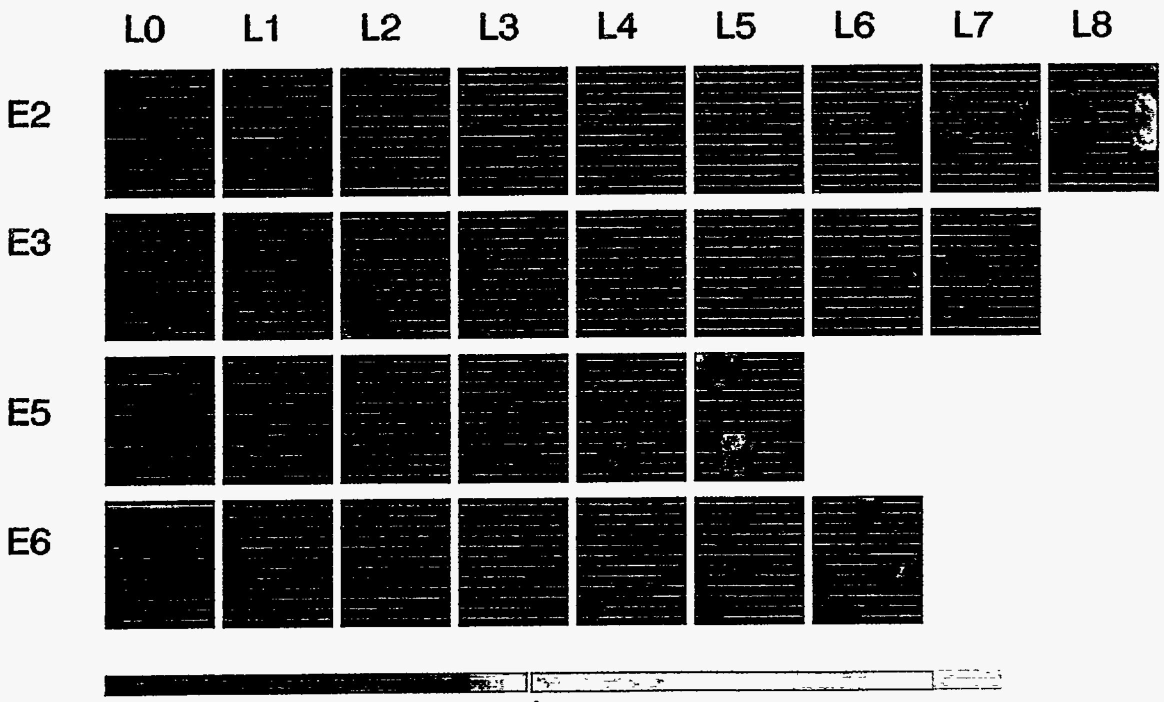
In an effort to more accurately identify barrels, we tried a more sophisticated analysis that made use of some understanding of the measurement processes. In this analysis, we made an effort to find peaks in the magnetic field and its gradient, and to discriminate on the basis of the widths of the peaks, on the assumption that barrels should cause peaks of a certain width in magnetic field measurements. This analysis used only sensors S1B and S1A; the other sensor data was ignored. The training process used the same training scenes as were used for the quadratic function search for barrels. The results of this analysis are shown in Figure C7. Note that experiment 1 is included, since this analysis does not require the sensors that are not included in experiment 1. This analysis was repeated with a different set of training data, using scenes E2L8, E5L2, E5L5, and E6L6 with the same masks as before. This gave slightly better results, shown in Figure C8. These results generally show the barrels as targets, which is good. They also show other ferromagnetic objects as targets, which we had hoped to avoid. However, it is not surprising that we are not able to distinguish well between barrels and other magnetic objects, since the peak width which we tried as the distinguishing feature is not really unique to barrels. Surprisingly, this analysis also shows as targets some presumed non-magnetic objects, such as a box (object 6) in E2. This is interpreted as a fairly strong indication that these boxes do in fact include some ferromagnetic material.

Conclusion

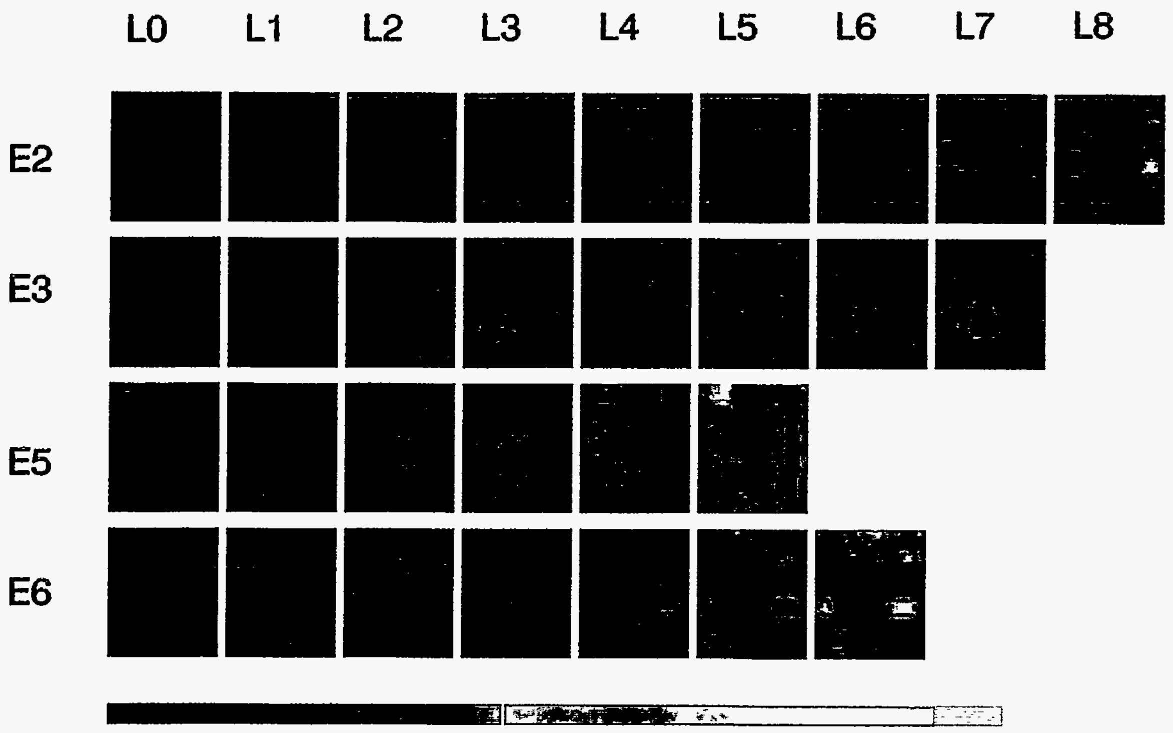
This ATR system is easily applicable to areas other than the usual airborne surveillance and target searches. Specifically, this example indicates that the ATR system used with this data can do quite well at finding buried objects, and it can give some indication of the nature of the object (magnetic or not) and the depth of the object. Failure to do better at indicating object type or depth is primarily due to limitations in the type of data available. This example also illustrates the importance of having a good training set, knowing exactly what background and target objects the training data represents, and making the training set self-consistent.



RESULT FILE: MASKS3.img
Figure C1: Buried objects, and masks used in training.

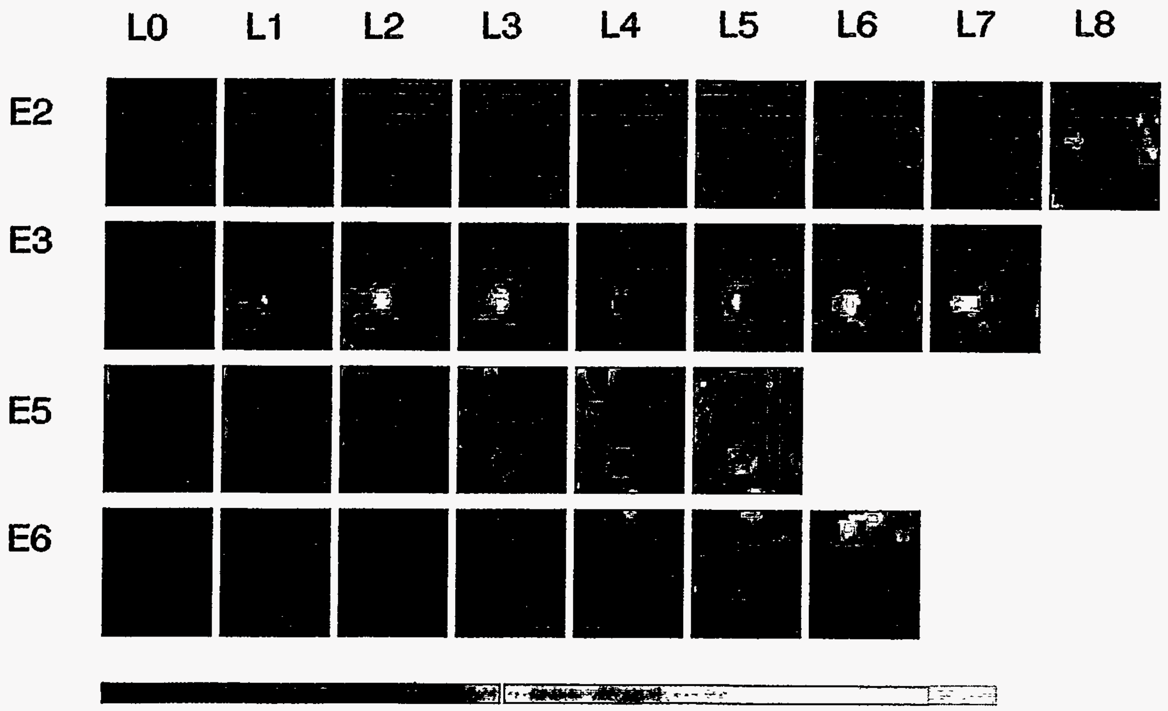


RESULT FILE: com1A2.img
Figure C2: Search for barrels, linear combination, three training scenes.



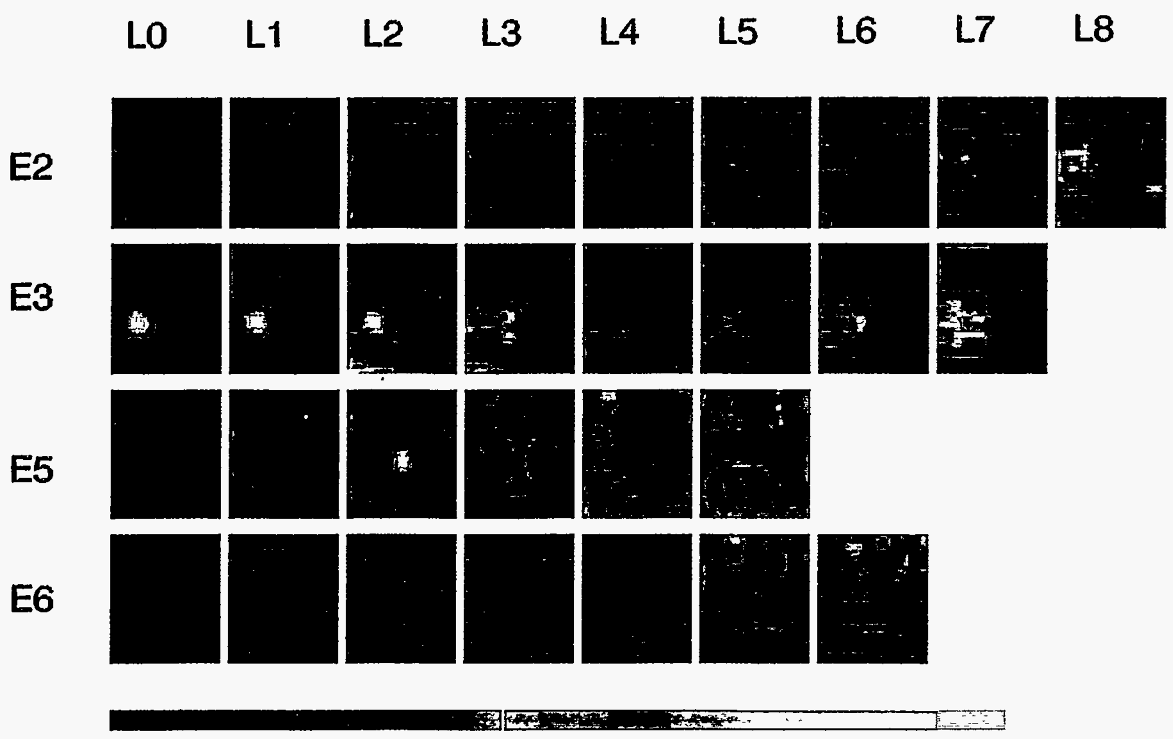
RESULT FILE: com4G2.img

Figure C3: Search for barrels, linear combination, four training scenes.



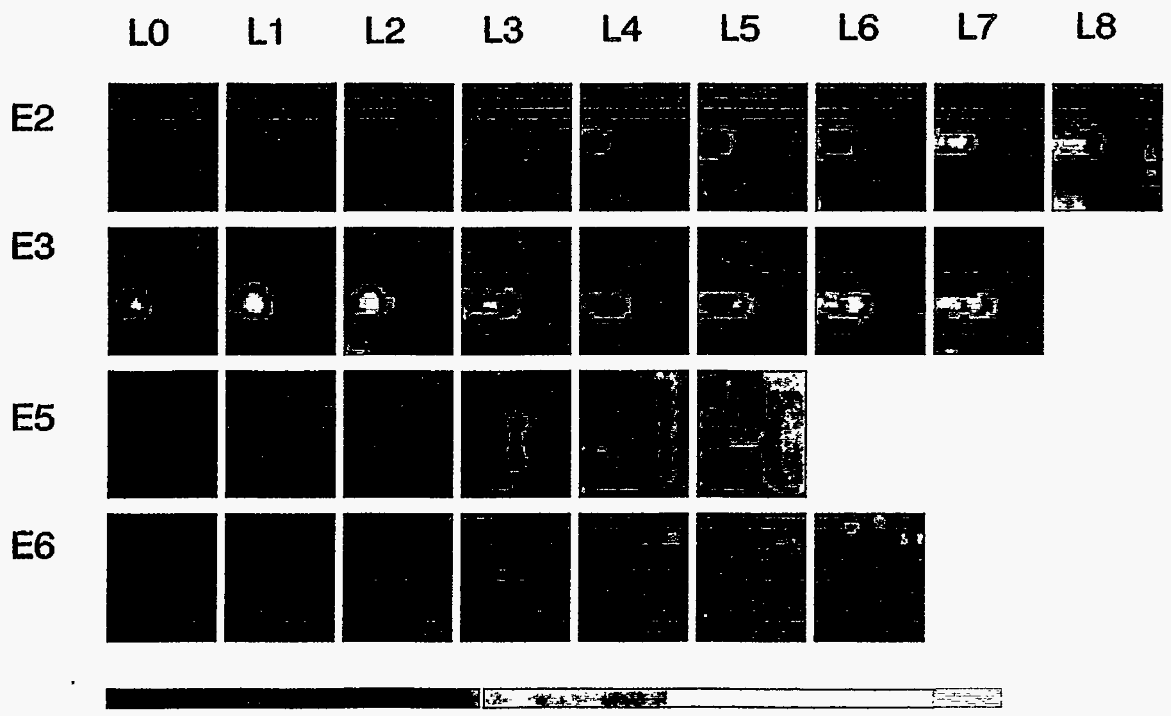
RESULT FILE: com4I2.img

Figure C4: Search for barrels, quadratic combination.



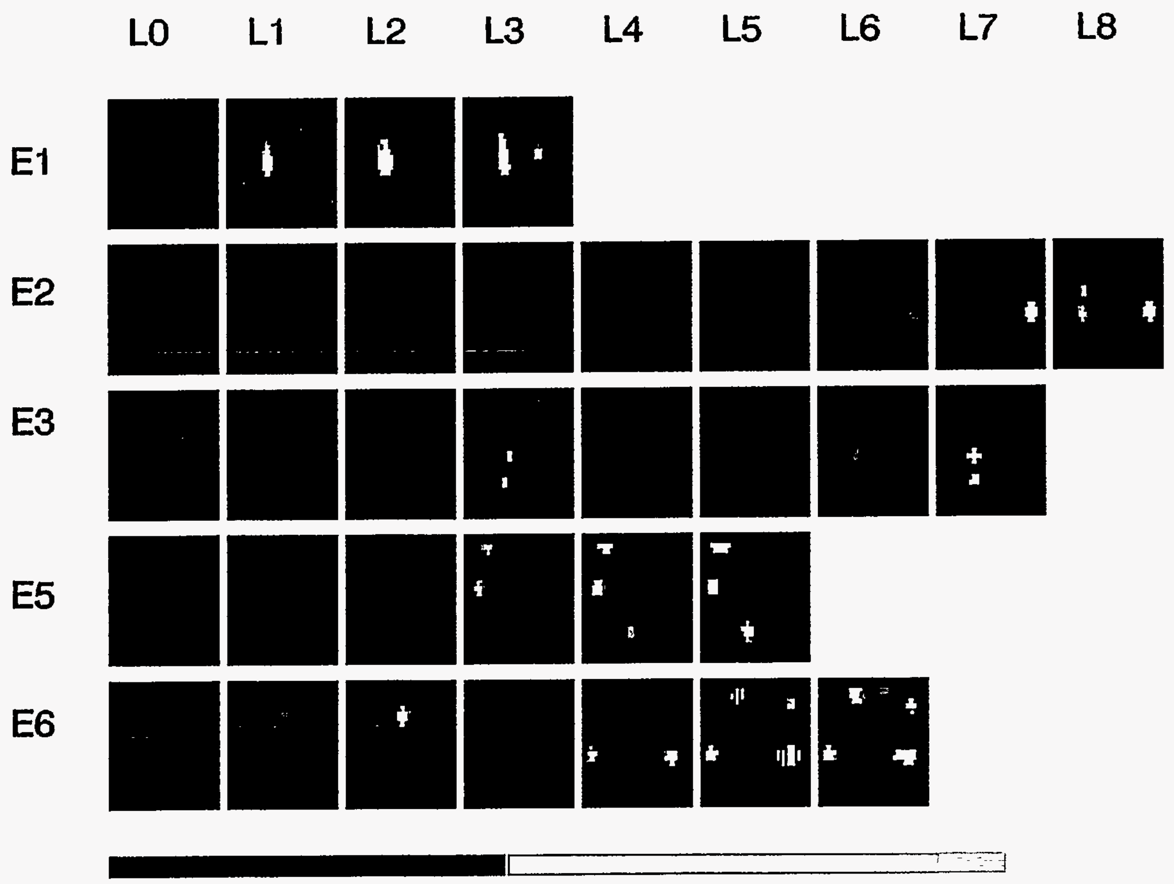
RESULT FILE: com4M22.img

Figure C5: Search for boxes, quadratic, two training scenes.



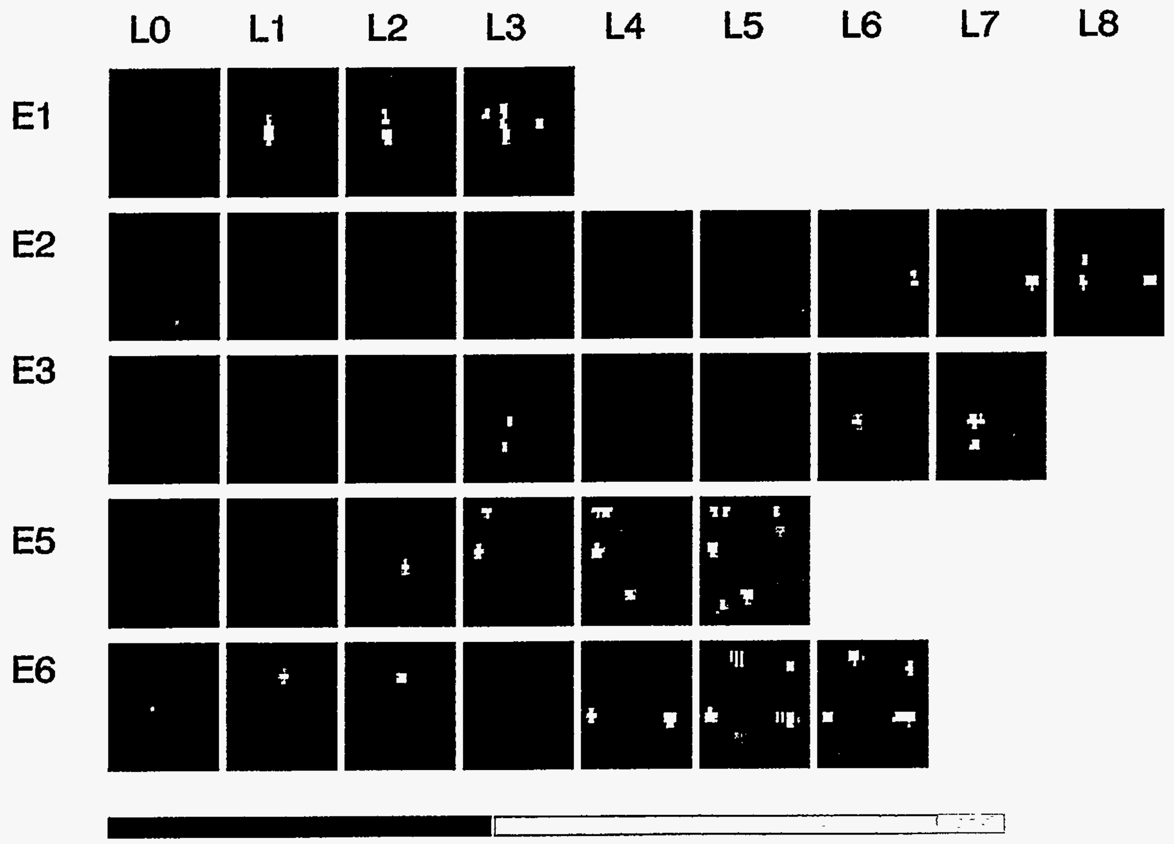
RESULT FILE: com4M21.img

Figure C6: Search for boxes, quadratic, one training scene.



RESULT FILE: com5K2.img

Figure C7: Search for barrels, peak width, first training set.



RESULT FILE: com502.img

Figure C8: Search for barrels, peak width, second training set.

APPENDIX D

EXAMPLE 4: Finding Airplanes

EXAMPLE 4: Finding Airplanes

This example illustrates the ability of the ATR system to function with poor data and in spite of errors or ignorance on the part of the operator. The goal is to find airplanes in pictures of airports, like the scene shown in Figure D1. There is an elegant approach to this problem, based on the assumption that the airplanes have the general shape of crosses or +'s. We can define a pair of kernels or operators which, together, find this general cross shape using the convolution operation. After some experimentation, this approach was used along with some very basic operations including looking for the correct size of darker area on a lighter background. For the training process, the scene was marked with two targets indicated in green (the airplane at the bottom center, and the airplane closest to the center of the image) and several background areas indicated in red in Figure D2. The training program gave the surprising result that searching for the cross shape did not significantly improve the ability of the system to find airplanes. The reason for the ATR system's not using the cross feature is obvious from an examination of the details of the image. The human vision system, seeing a "good" image like Figure D1 (perhaps in effect enhanced by the printing process) and knowing what to look for, is very good at idealizing shapes and filling in missing details. However, as Figures D3 - D5 show for the 7 airplanes in the top half of the scene, the airplanes comprise a surprisingly small number of pixels and they do not have much of the assumed cross shape. This image is actually of rather poor quality in terms of representing airplanes. It is no longer surprising that the ATR system did not give much weight to the cross feature in searching for airplanes. The interesting point is that the ATR system automatically discounted the unproductive search for crosses and used only the simpler features, even though this was contrary to the operator's expectations.

The result of using this very simple procedure is shown in Figure D6, in which the computer has drawn circles around each area it interprets as a target. The ATR system with this very simple scheme did correctly find all 9 airplanes, and it mistakenly marked a lot of background areas as targets. This general behavior, finding all the real targets and some false targets, is generally expected in a screening operation in which the system is optimized to not miss any real targets. However, the number of false targets indicated in Figure D6 is probably unacceptably high for most applications. We can improve on this result by placing more stringent requirements on the sizes of the dark spots that are called targets. Using this modified simple approach gives the result shown in Figure D7. This result correctly indicates all the airplanes as targets, and it incorrectly indicates two background areas, one of them so small as to be immediately discounted, as targets. This result is not perfect, but it is regarded as quite good, especially considering the poor detailed definition of the airplanes in the scene image and the simplicity of the search scheme.

Conclusions

This ATR system can correct some operator's mistaken assumptions, and can do a surprisingly good job with poor data and simple analysis schemes.

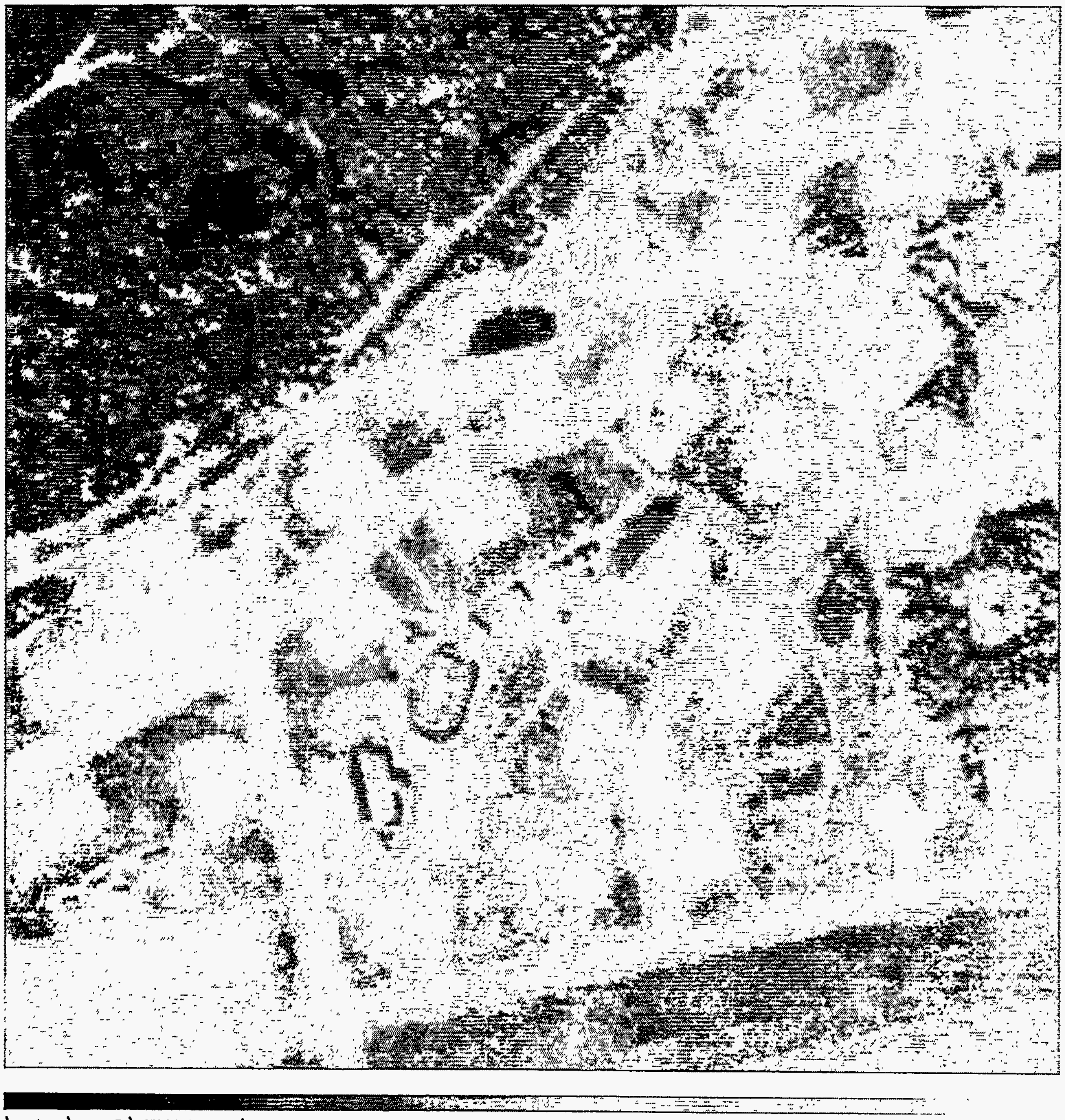

\atr\ex3\INPUT.img

Figure D1: A scene including 9 airplanes.

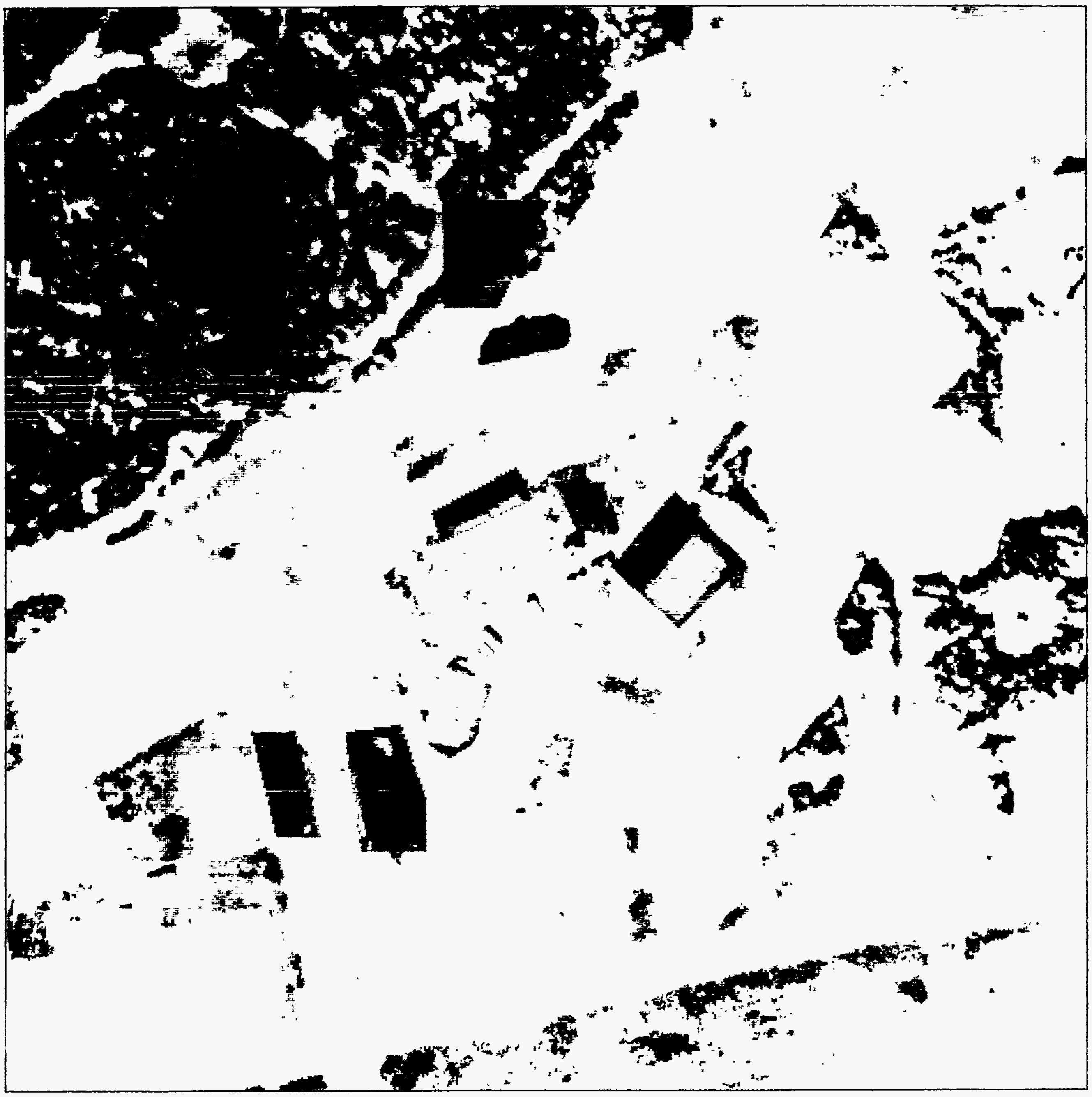


IMAGE FILE: INPUT.img
MASK FILE: MASKD.img

Figure D2: The scene marked with two target (green) and seven background (red) regions for training.

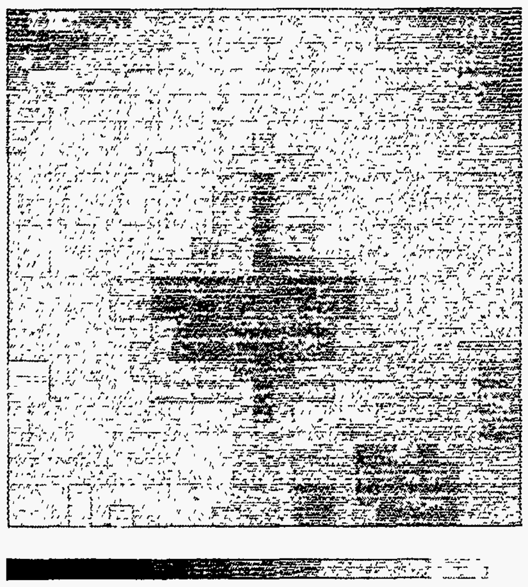


IMAGE FILE: detail1A.img

IMAGE FILE: detail1B.img

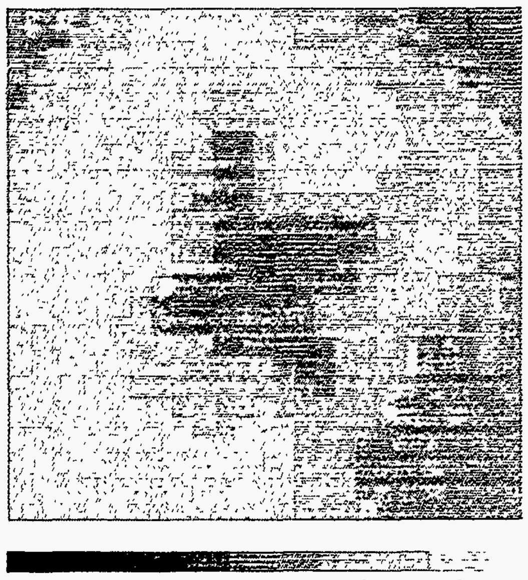


IMAGE FILE: detail1C.img

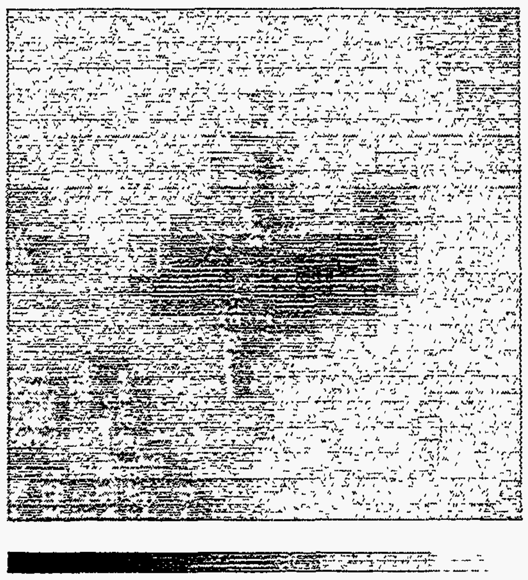


IMAGE FILE: detail1D.img

Figure D3: Details of the four airplanes closest to the top of the scene.

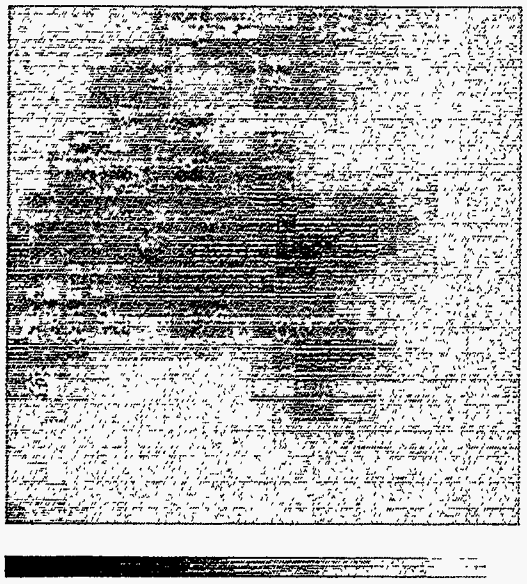


IMAGE FILE: detail1E.img

IMAGE FILE: detail1F.img

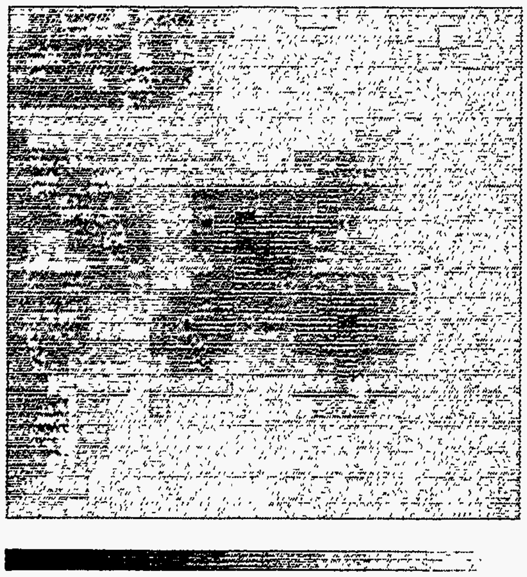


IMAGE FILE: detail1G.img

IMAGE FILE: detail1.img

Figure D4: Details of the next three airplanes.

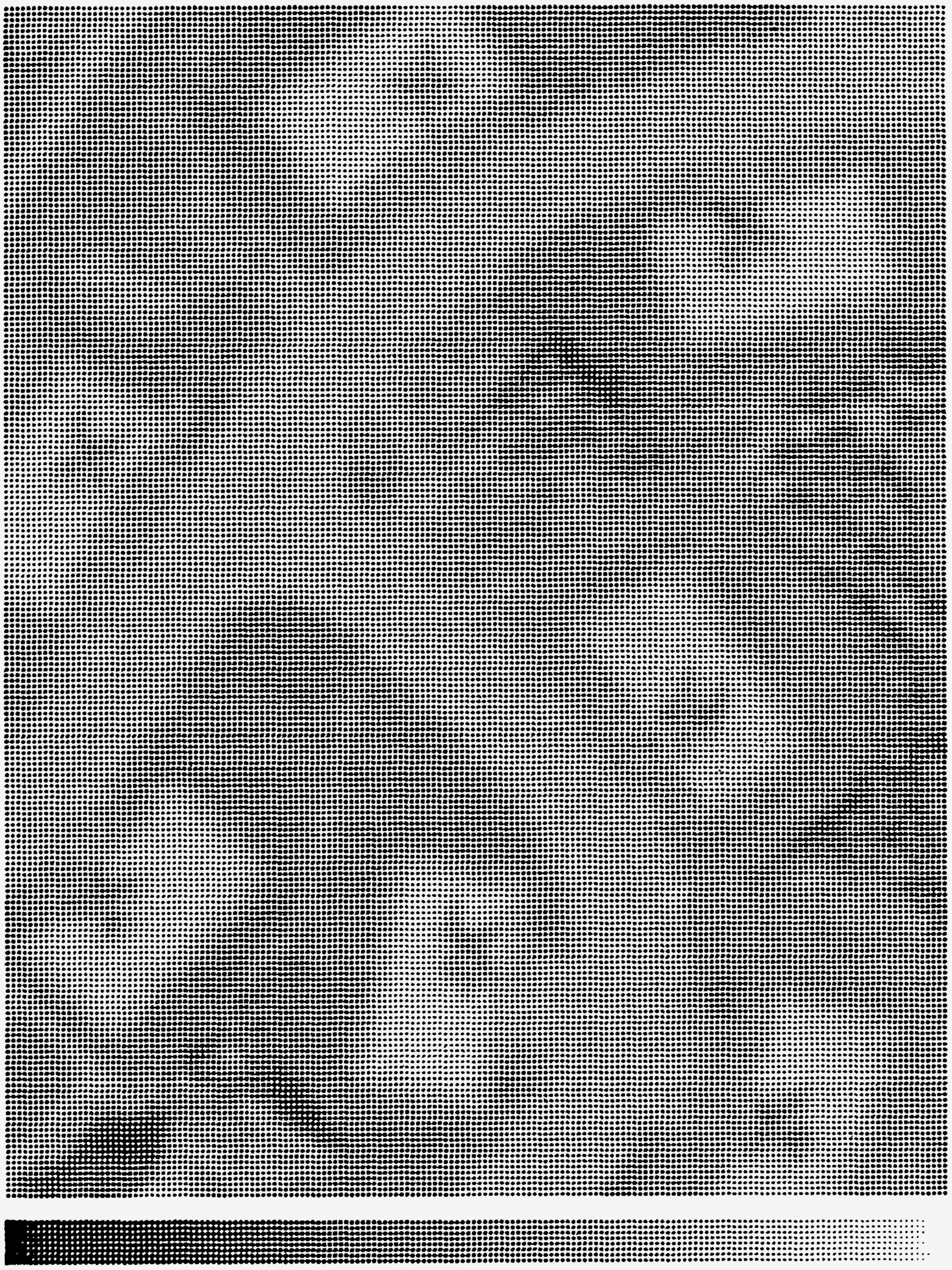
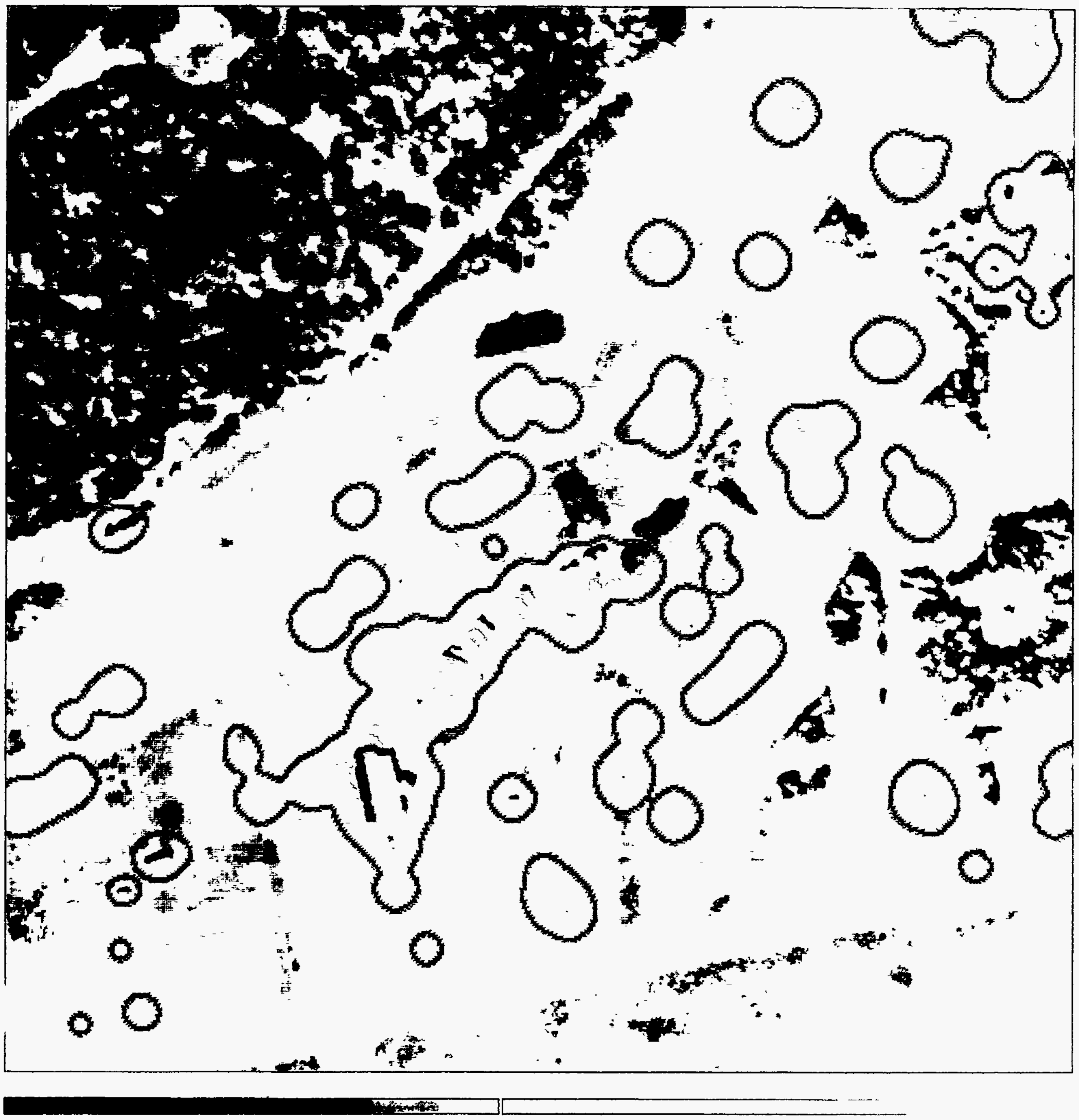
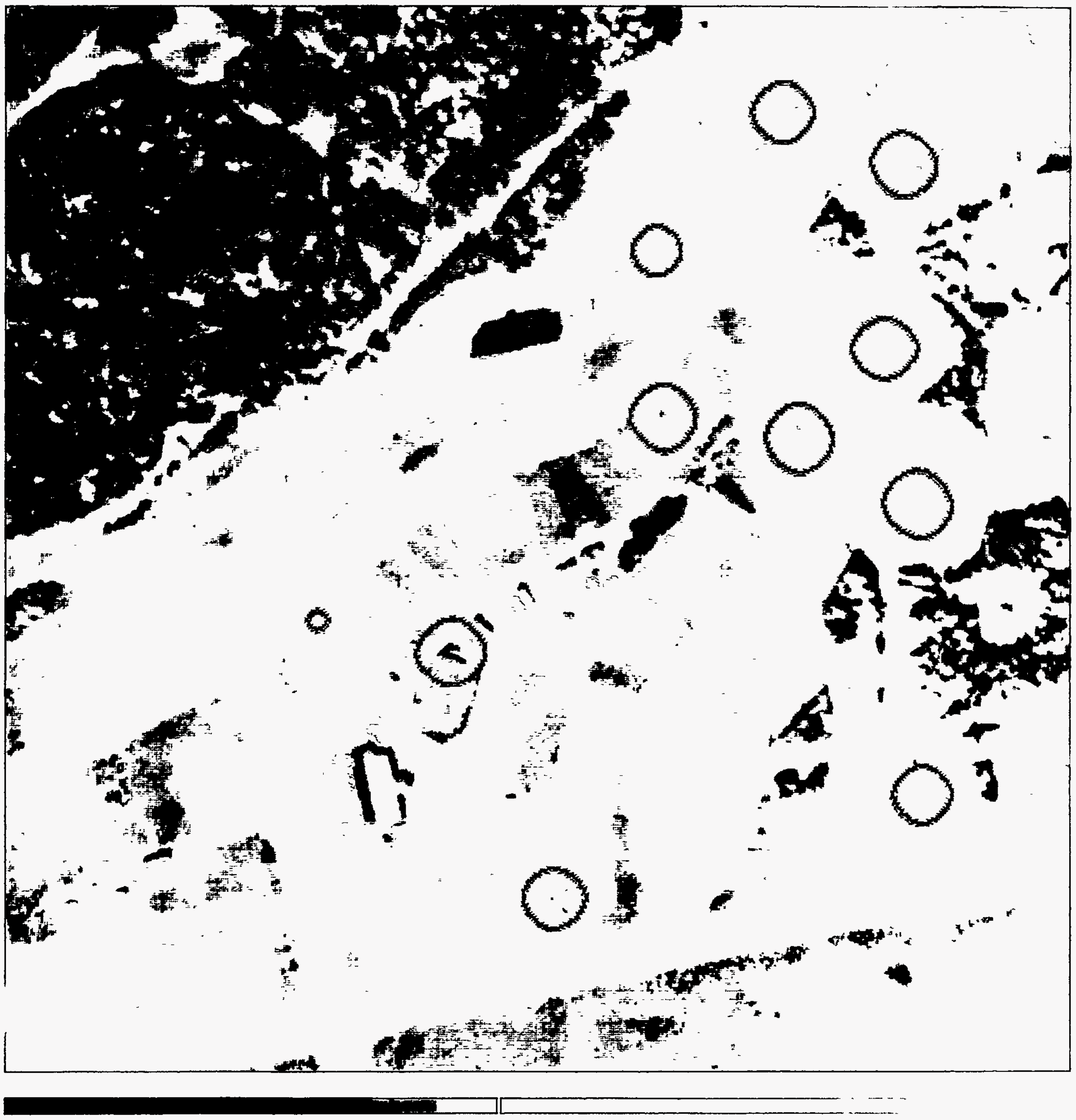


Figure D5: Detail image printed with one dot per pixel.



RESULT FILE: E9Xres.img

Figure D6: Result of the simplest search scheme, showing a number of false target indications.



RESULT FILE: E9Nres.img

Figure D7: Result of a more restrictive search scheme, showing only two false target indications.

APPENDIX E

EXAMPLE 5: Stack Shadows

EXAMPLE 5: Stack Shadows

This example is a more realistic target recognition application, and it includes an example of using the ATR system without the training process. The targets here are gas effluent stacks, such as a factory smoke stacks or chemical processing plant stacks for dispersing gaseous waste. The image data is multispectral, comprising images in 3 wavelength bands at about 1, 8, and 12 microns, acquired with the Daedalus imager carried by the AMPS aircraft. The images are large, with a typical scene containing ten million pixels for each of the three wavelengths.

As seen from above, the stacks themselves are not at all obvious or easily identifiable. However, if the sun is shining, stacks cast a quite distinctive shadow in the visible and near infrared spectral regions. Hence, this search for stacks is actually a search for stack shadows. Searching for stack shadows instead of stacks themselves does have some disadvantages, in that there are other objects that look very much like stack shadows. The procedure used here is a two-step process. The first step is to search the near infrared image to find shadows of the right size, shape, and orientation. This first step does not require any training process, since we know from other sources what the correct shadows should look like. The second step is to use all three of the available spectral bands with a training process in the hope that the extra spectral information will provide enough information to distinguish between stack shadows and other similar-looking objects, even though the nature of this multispectral information may not be at all intuitive to the human observer.

Figure E1 shows an example of the results of the stack shadow search. The first step of the search identified the four shadows marked in Figure E1. The top two of these really are stack shadows; the bottom two are shadows of the support structures of a water tower. The first step also incorrectly identified some other objects that are almost invisible in this printing of the image, including dirt roads, fences, and pond edges, near the top left corner of Figure E1. Some of these false indications of stack shadows could be eliminated by setting more stringent requirements on the orientation of acceptable shadows; however, for purposes of testing and demonstration, we left the angle criterion rather loose and used the multispectral information instead.

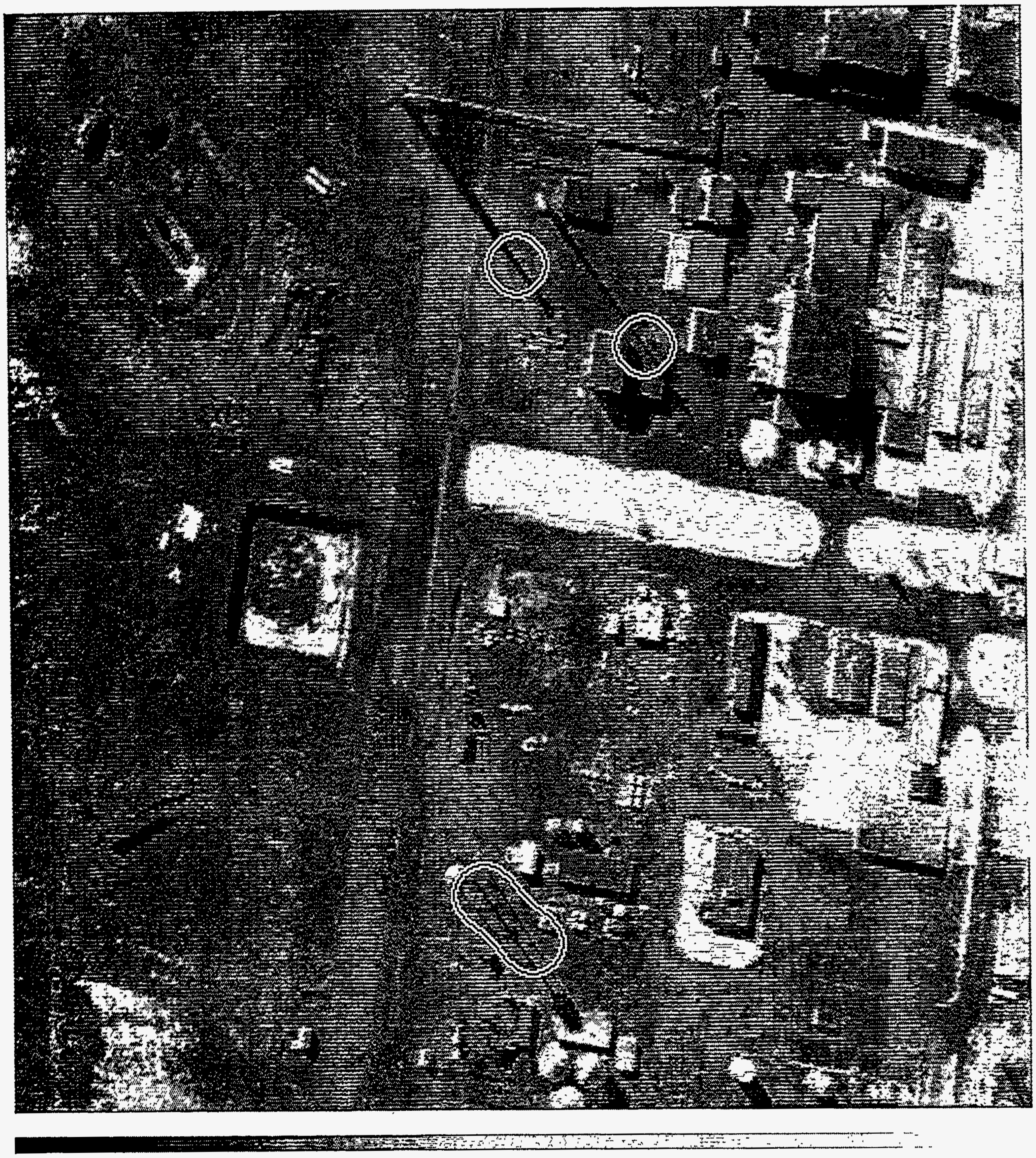
To distinguish real from false stack shadows, we added 7 features to be evaluated: (1) the intensity of the shadow in the (slightly smoothed) 1 micron image; (2) the contrast between the shadow and its surrounding area in the 1 micron image; (3) the "brightness" parameter that the program uses to describe how well the shadow fits the mathematical idealization of a straight line segment; (4) and (5) the intensity of the shadow in the 8 and 12 micron images; (6) and (7) the products (3) $*$ (2) and (3) $*$ (1). Training the system and using these features does indeed remove many of the false indications of stack shadows, giving the result shown in the Figure E1.

The total two-step search process was applied to 13 large scenes containing six known stacks. Five of the six stacks were correctly located; the sixth was not identified as a target, for two reasons, either of which would have been sufficient by itself. First, the sixth stack was larger than the parameters we specified as representing an acceptable stack; that is, it was larger than the stacks we were looking for. Second, the shadow of the sixth stack was interrupted by some buildings that added their own shadows and changed the simple straight line stack shadow into a more complicated, unacceptable shape. This second point is an unavoidable disadvantage of trying to identify stacks by finding their shadows. Figure E2 shows the missed sixth stack, along with another smaller stack that was correctly identified.

This stack search did wrongly identify some other objects as stacks, which might be expected. One of these objects was the already-mentioned water tower, which appears near the bottom of Figure E1 and also in three other scenes. Depending on the sun angle and how the shadows of the different support structures of the water tower aligned at different times of the day, the water tower shadows were sometimes indistinguishable from stack shadows. This of course is not surprising, since the water tower supports and a gas effluent stack are very similar structures, casting nearly identical shadows. Another stack-like structure that was incorrectly identified as a stack was a tower supporting a major electrical power line, two instances of which are shown in Figure E3. Figure E4 shows an agricultural area in which a segment of an irrigation canal was incorrectly identified as a stack shadow. This type of error occurred several times in the set of 13 scenes. As has been mentioned, the rate of occurrence of this type of error could be reduced by more stringent geometrical tests, but even the relaxed test conditions used here gave an acceptably low false positive error rate.

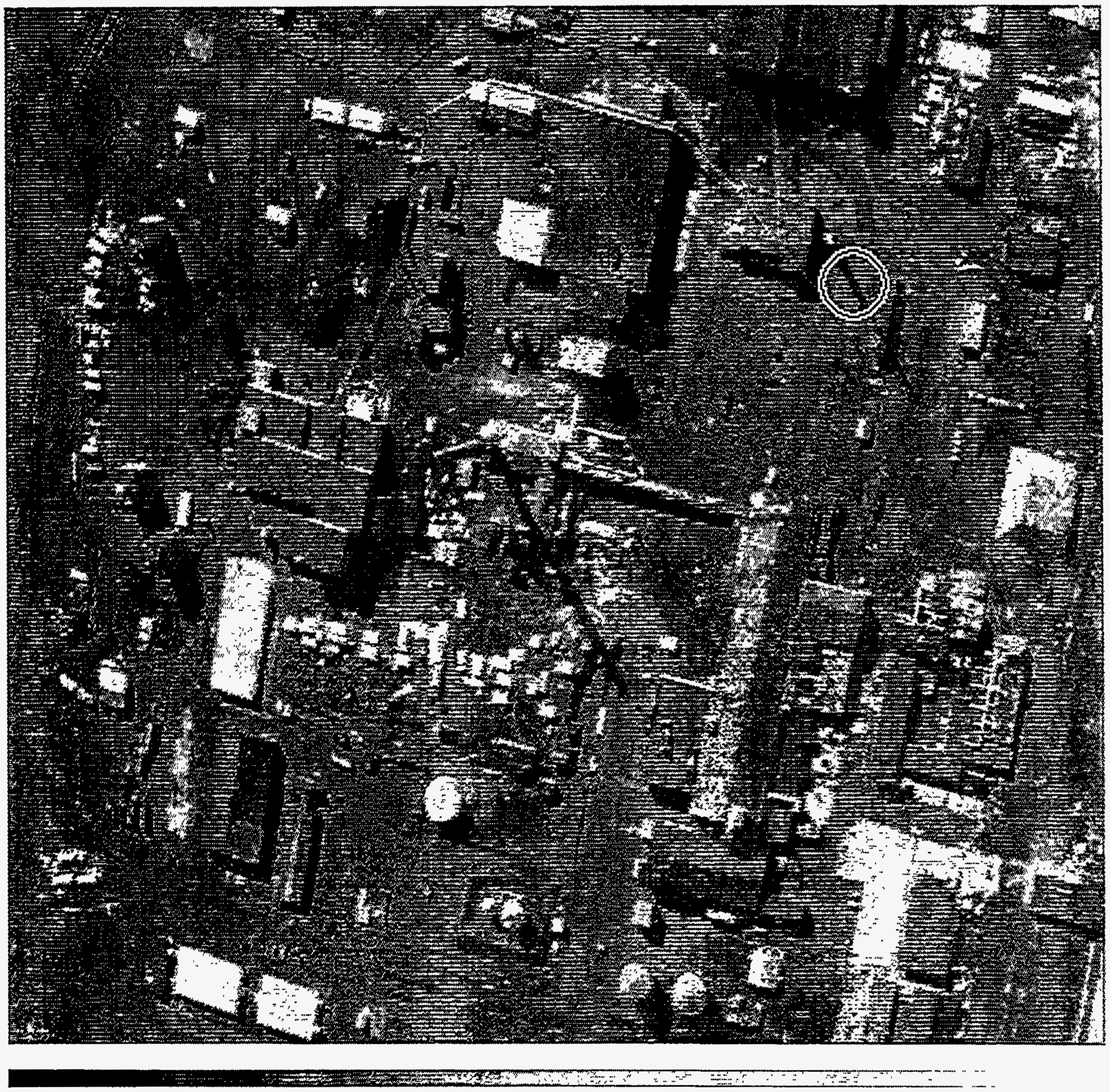
Conclusions

In a realistic application with large images, the ATR system performed well, finding all the known targets with a few false positive target indications.



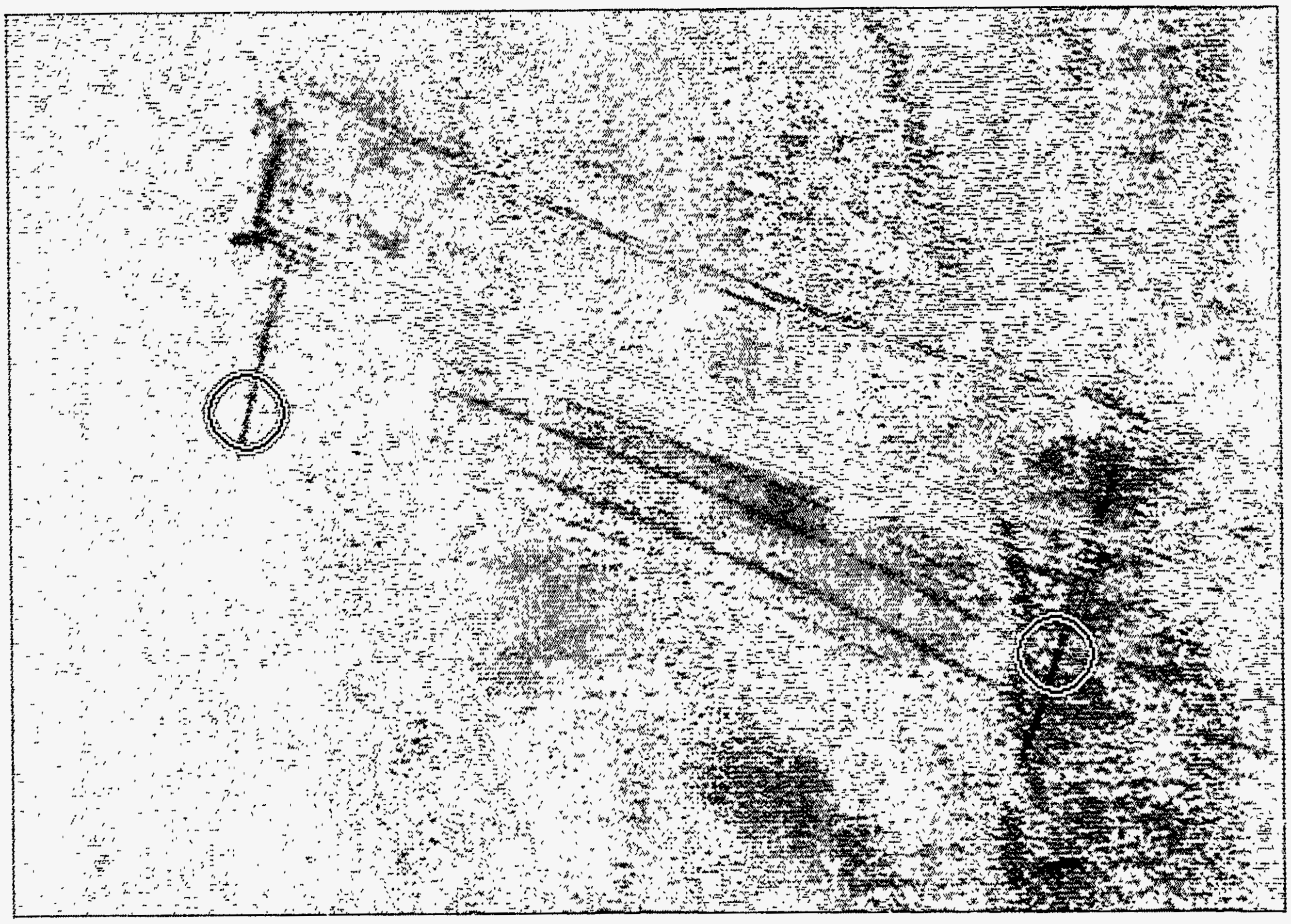
\atr\TRA01.img

Figure E1: Two correctly identified stack shadows (near top of image) and a pair of incorrect indications from water tower shadows (bottom of image).



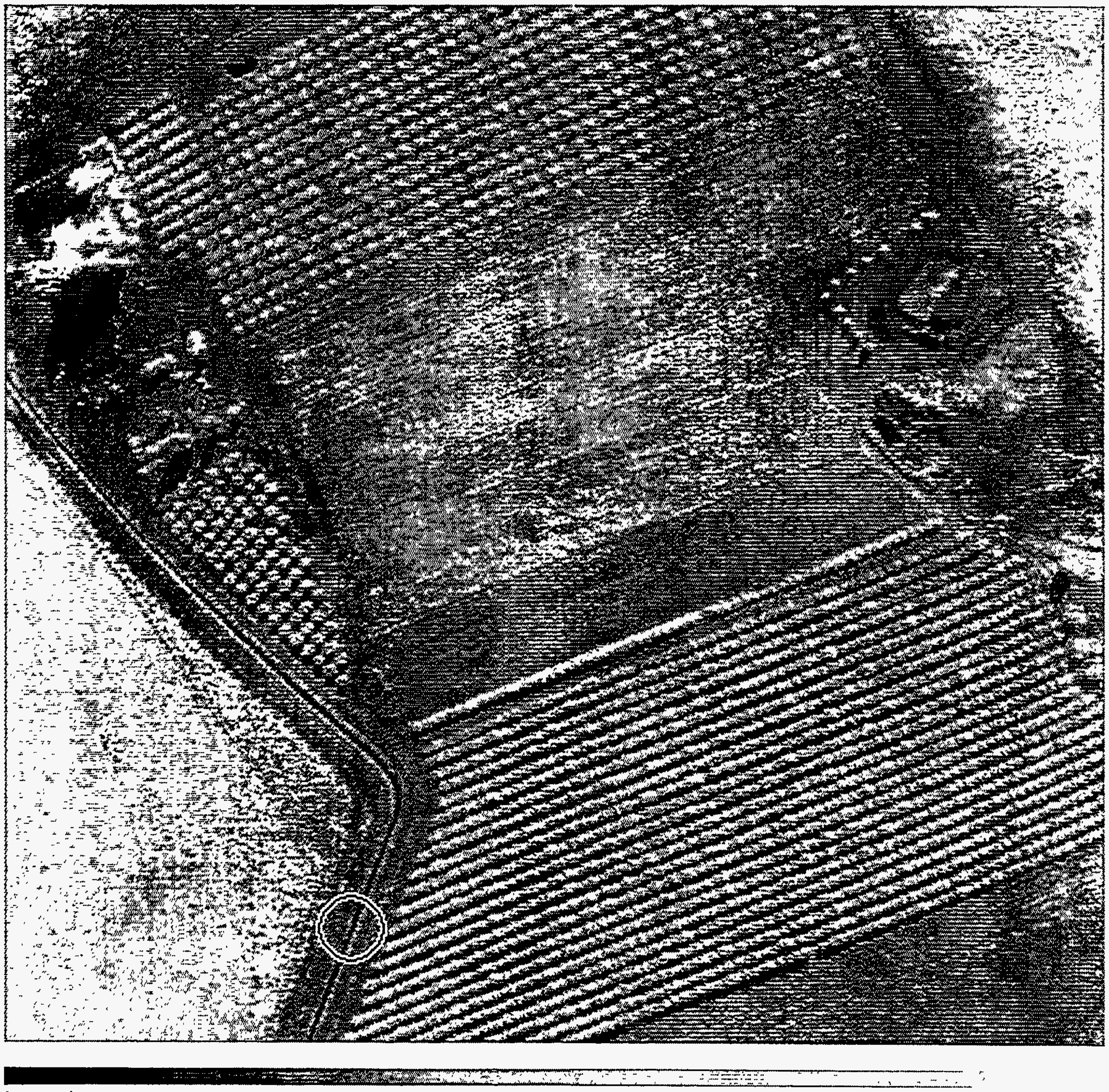
\atr\CPP01.img

Figure E2: One correctly identified stack shadow, and another that was too large to be included by the size parameters specified for this search.



\atr\WPPSS3a.img

Figure E3: Two power line support towers incorrectly reported as stacks. Some of the electric cables are visible as light lines, and their shadows are more reliably visible as dark lines in this image. This image was greatly contrast-enhanced for this printing.



\atr\E5.img

Figure E4: A segment of an irrigation canal which was incorrectly reported as a stack shadow.