NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A REAL-TIME IMAGE UNDERSTANDING SYSTEM FOR AN AUTONOMOUS MOBILE ROBOT

by

Leonard Vince Remias

March 1996

Thesis Advisor: Yutaka Kanayama

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1996	Master's Thesis

4. TITLE AND SUBTITLE
A Real-Time Image Understanding System For An Autonomous Mobile Robot (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Remias, Leonard Vince

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Yamabico-11 is an autonomous mobile robot used as a research platform with one area in image understanding. Previous work focused on edge detection analysis on a Silicon Graphics Iris (SGI) workstation with no method for implementation on the robot. *Yamabico*-11 does not have an on-board image processing capability to detect straight edges in a grayscale image and a method for allowing the user to analyze the data. The approach taken for system development is partly based on edge extraction and line fitting algorithms of [PET92] with a 3-D geometric model of the robot's world [STE92]. Image grabbing routines of [KIS95] were used to capture images with the robot's digital output camera and processed using image understanding routines developed for a SGI workstation. The routines were modified and ported onto the robot. The new method of edge extraction produces less ambient noise and more continuous vertical line segments in the gradient image which enhances pattern matching analysis of the image. *Yamabico*-11's computer system can capture an image with a resolution of 739 x 484 active picture elements. Edge detection analysis is performed on the robot which generates a list structure of edges and stored in the robot's memory for user analysis.

14. SUBJECT TERMS	15. NUMBER OF PAGES
robotics, computer vision, edge extraction, obstacle avoidance, image understanding	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

A REAL-TIME IMAGE UNDERSTANDING SYSTEM FOR AN AUTONOMOUS MOBILE ROBOT

Leonard V. Remias
Lieutenant, United States Navy
B.S. Old Dominion University, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1996

ABSTRACT

Yamabico-11 is an autonomous mobile robot used as a research platform with one area in image understanding. Previous work focused on edge detection analysis on a Silicon Graphics Iris (SGI) workstation with no method for implementation on the robot. *Yamabico*-11 does not have an on-board image processing capability to detect straight edges in a grayscale image and a method for allowing the user to analyze the data.

The approach taken for system development is partly based on edge extraction and line fitting algorithms of [PET92] with a 3-D geometric model of the robot's world [STE92]. Image grabbing routines of [KIS95] were used to capture images with the robot's digital output camera and processed using image understanding routines developed for a SGI workstation. The routines were modified and ported onto the robot.

The new method of edge extraction produces less ambient noise and more continuous vertical line segments in the gradient image which enhances pattern matching analysis of the image. *Yamabico*-11's computer system can capture an image with a resolution of 739 x 484 active picture elements. Edge detection analysis is performed on the robot which generates a list structure of edges and stored in the robot's memory for user analysis.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

First and foremost, I must acknowledge the unfailing and unconditional support I have received from from my lovely wife, Lerma, without which this work could never have been completed. Her positive attitude and understanding while dealing with the arduous task of child rearing were remarkable as they were essential to my success.

I also wish to express my deepest gratitude to Professor Yutaka Kanayama whose support and guidance have been a constant inspiration to me. A special thanks goes to Major Khaled Morsy for his invaluable assistance especially in helping me implement the C code used in this thesis.

Lastly, I would like to recognized Mike Williams and Rosalie Johnson for their tremendous support in resolving the hardware and software system problems encountered during this thesis.

I. INTRODUCTION

A. BACKGROUND

Image understanding developments in the field of computer vision has provided advanced research capabilities for image processing applications in medicine, cartography, industry, manufacturing, printing and publishing, and numerous scientific fields. In all cases image processing is concerned with the computer processing of pictures or images that have been converted to a numeric form. The ability to process images in this format is the fundamental study and framework for image understanding of a vision system for an autonomous mobile robot.

The study of robot vision has been of significant research at the Naval Postgraduate School in the last five years. While many of the basic approaches have undergone progressive refinement, numerous new directions continue to be pursued in both general and system specific applications. A vision system for an autonomous vehicle may be employed for a variety of uses including navigation, object recognition, and environmental mapping [DEC93].

In previous image understanding research, techniques were developed to implement a working vision based navigation control mechanism [PET92], to provide a capability for object recognition [DEC93], and integration of self contained image understanding subsystem independent of a unix workstation [KIS95].

B. OVERVIEW

Yamabico-11, is an autonomous mobile robot used as a test platform for research in motion planning, obstacle avoidance, environment exploration, path tracking, and image understanding. The ability to process images instantaneously or in real-time is crucial for navigating in a dynamic world. The ultimate goal of the image processing system in *Yamabico*-11 must be able to assess the environment and navigate with a safe and smooth

motion. This research develops an edge region finding algorithm and implements this system into the board of the *Yamabico*-11.

C. PROBLEM STATEMENT

The major problems addressed in this research is how to develop and implement an image understanding system previously written for a Silicon Graphics Iris (SGI) workstation in an autonomous mobile robot. The resulting image understanding system should be a part of the total intelligent autonomous robot and should provide functionality that will allow the robot to process images on board in real-time.

II. YAMABICO-11 ROBOT

Yamabico-11 shown in Figure 1, is an autonomous mobile robot used as a test platform for research in motion planning, obstacle avoidance, environment exploration, path tracking, and image understanding.

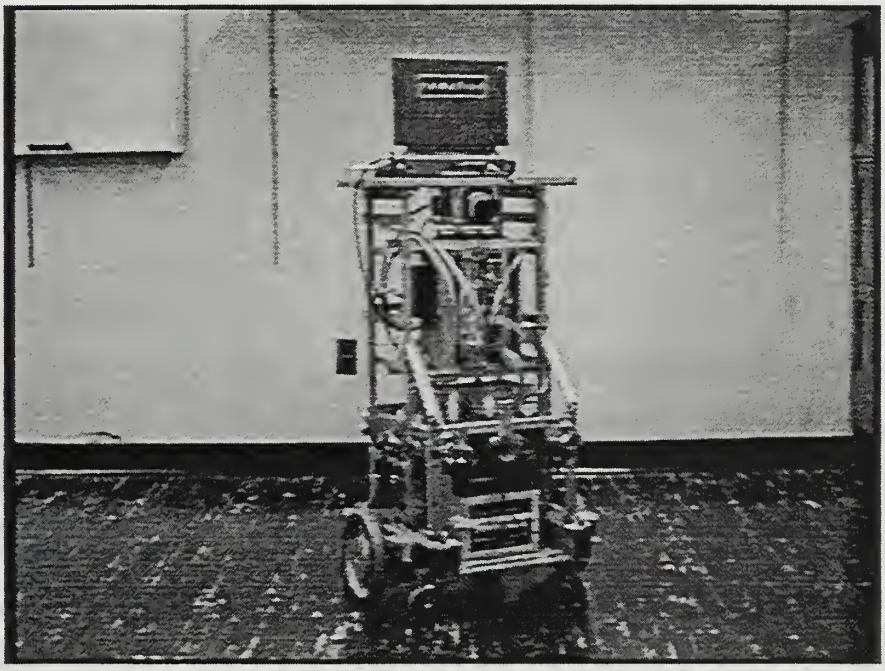

Figure 1. *Yamabico*-11 Autonomous Mobile Robot.

A. HARDWARE DESCRIPTION

Yamabico-11 is powered by two 12-volt wheelchair type batteries and is driven on two wheels by DC motors which drive and steer the robot while four spring-loaded caster wheels provide balance.

The master processor is an Ironic's SPARC-4 processor equivalent to a CPU Sun-3 workstation. This processor has 16 megabytes of main memory and runs with a clock speed of 33MHz on a VME bus.

All programs on the robot are developed using a Sun 3/60 workstation and UNIX operating system. These programs are first compiled and then downloaded to the robot via a RS-232 link at 9600 baud rate using a PowerBook™ 145 computer for the communication interface.

Twelve 40 kHz ultrasonic sensors are provided as the primary means by which the robot senses its environment. The sonar subsystem is controlled by an 8748 micro-controller. Each sonar reading cycle takes approximately 24 milliseconds.

The visual images from the robot are generated by a COHU solid state camera. The camera is mounted along the center line of the robot at a height of 34 inches. This camera provide black and white video to a display monitor and to the IMS board for processing and further display on the NEC multisync monitor shown in Figure 2.

The JVC TK870U CCD camera equipped with a FUJINON TV zoom lens provides a NTSC standard RGB video image through a video framer attached to a Silicon Graphics Iris™ (SGI) workstation shown in Figure 2. This video image is used to process images on the SGI only. The framer digitizes the sync and composite signals for storage on the SGI and also passes the signal to a high definition monitor. The video signal is transmitted via standard coaxial video cable to the image processing hardware.

The Cohu4110 from Cohu Inc. [COH90] is a digital output camera with a 1/2 inch format Charge Couple Device (CCD) image sensor. The area is 6.4 mm x 4.8 mm and 739 x 484 picture elements. The camera transfers in parallel one eight bit pixel which represents 256 shades of gray. The digital output eliminates the need for special hardware to convert the cameras analog signal into digital. This design also isolates sensitive analog circuits away from the host computer by putting them into the camera itself.

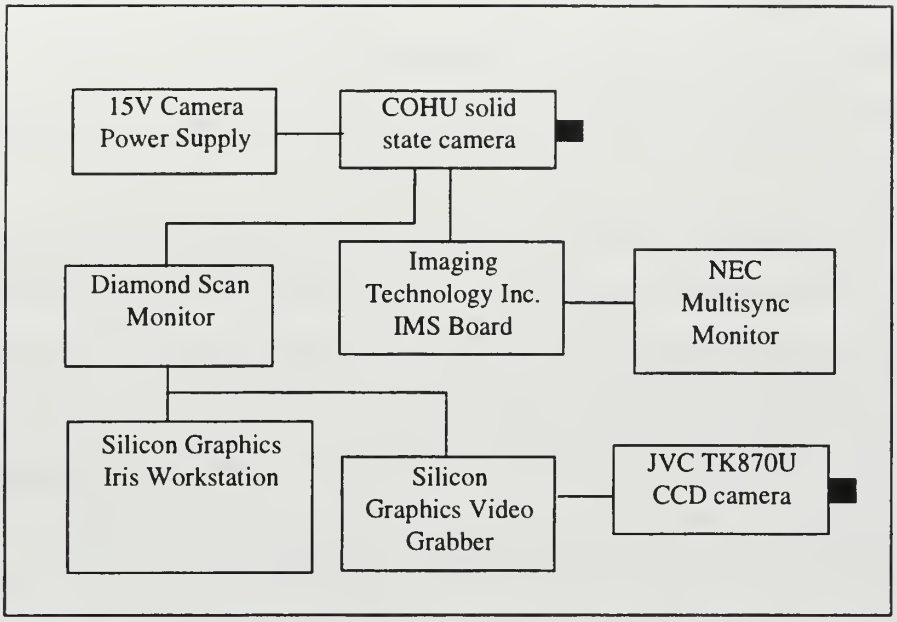

Figure 2. Vision system hardware components.

B. SOFTWARE DESCRIPTION

The software system consists of a kernel and a user program. The kernel is approximately 82,000 bytes and only needs to be downloaded once during the course of a given experiment. The user's program can be modified and downloaded quickly to support rapid development.

Motion and sonar commands are issued by the user in MML, the model-based mobile robot language. While the previous version of MML was based on point-to-point tracking, the current version being integrated into *Yamabico*'s control structure relies on a 'path tracking' approach. While MML provides the capability to define path types which include parabolic and cubic spiral, the most fundamental 'path' for the robot to follow is a line which is defined by a curvature (κ) and a location and orientation in two-dimensional space described in x, y, and theta. With $\kappa=0$, the line is straight, and $\kappa\neq0$ produces a circle of radius $1/\kappa$ ($\kappa<0 \equiv$ clockwise & $\kappa>0 \equiv$ counter-clockwise). The location and orientation can be the starting point of a semi-infinite line called a forward line (*fline*), the end point of

a semi-infinite line called a backward line (*bline*), or a point and direction along an infinite line (*line*). For all path types, once one has been specified and commanded, the robot performs the required calculations and adjusts the curvature of its motion as necessary. Additionally, transitions between successive paths are performed automatically and autonomously.

The functionality inherent in the MML plays a significant role in developing the capability for the robot to avoid obstacles. Consequently, a portion of this research effort was devoted to implementing some of the core functions in the newly developed 'path tracking' approach to motion control. This method allows for dynamic real-time specification of the proposed robot path based on sensory input and is especially well suited to employing the information generated from object recognition. Since the available information will include not only ranges (which is the sole data provided by sonar) but also dimensions, a complete avoidance maneuver can be determined.

As mentioned above, the sonar system is also controlled through the MML. Both raw sonar range returns as well as processed 'global' results incorporating least-squares line fitting are available to the user on board the robot. This capability should prove particularly useful in extending the environment in which the vision system can be applied, and its application is addressed in the discussion of the environmental model.

III. IMAGE DESCRIPTION

An image is a picture, photograph, display, or other form of visual representation of an object or scene. However, in digital image processing, it has another meaning: an image is a two dimensional array of numbers [NIB86].

A. MATRIX STRUCTURE

Since a digital image is similar to a matrix or array of numbers, the image can be represented by a structure called a bound matrix [DOU87]. The array type structure used to describe the images for segment extraction is shown in Figure 3 where:

- $P(i, j)$ indicates the light intensity of the picture element and is a non-negative value.
- $P(0, 0)$ is considered to be the origin.

$$\begin{bmatrix} P_{0,n-1} & P_{1,n-1} & \cdots & P_{m-1,n-1} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ P_{0,1} & P_{1,1} & \cdots & P_{m-1,1} \\ P_{0,0} & P_{1,0} & \cdots & P_{m-1,0} \end{bmatrix}$$

Figure 3. Bound matrix representation of image.

The numbers used for the light intensity of the pixel gives its level of darkness or lightness of the pixel area. Since each pixel is stored as a byte, 8 bits, the maximum pixel value is 255. This topic of pixel storage will be discussed in the next section on RGB Format. A higher number represents a lighter area with the maximum value 255 being white and 0 being black. All intermediate values will be shades of grey.

A 12 by 12 matrix shown in Figure 4 was used to represent an example image with different levels of light intensities. The array of numbers represented in the matrices were used to evaluate edge region generation and linear feature extraction discussed in the next section. The source code and results can be seen in Appendix A.

$$\begin{bmatrix}
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150
\end{bmatrix}$$

Figure 4. 12 x 12 matrix of pixel intensities of a square image.

B. RGB FORMAT AND GRAYSCALE CONVERSION

A single pixel is described in a RGB format which is comprised of 32 bits. The basic color components of each pixel in the image are red, green, and blue and are each represented by a byte, 8 bits. This gives an intensity value a range of 0 to 255 for each component. The alpha component, which represents the transparency of the pixel, is not considered when using the RGB format since all pixels are taken to be completely opaque. As was discussed earlier, an intensity value of zero represents a black pixel and an intensity value of 255 represents a white pixel. Figure 5 shows the bit placement for the RGB format.

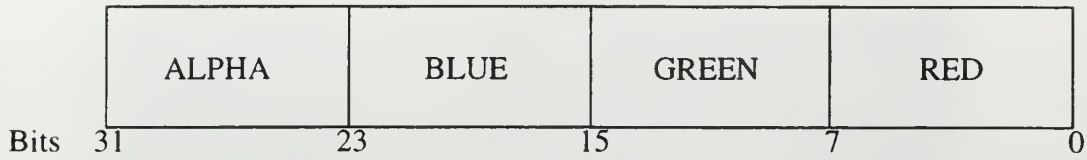

Figure 5. The RGB Format.

The data for all the pixels in an image is stored in a long, one-dimensional array. The ordering in the array with regard to position in the image is left to right, bottom to top so the lower left corner pixel would be the first element in the array while the upper right would be the last. Since the edge extraction process requires a black and white ('grayscale') representation for the pixels in an image, a conversion from the RGB values is necessary. According to the standard weighting factors set by the National Television Systems Committee (NTSC), a RGB color pixel is given an equivalent grayscale value by the following equation:

$$\text{GRAYSCALE} = [0.299 \;\; 0.587 \;\; 0.114] \begin{bmatrix} \text{Red Intensity} \\ \text{Green Intensity} \\ \text{Blue Intensity} \end{bmatrix} \qquad \text{(Eq 1)}$$

C. GRADIENT IMAGE

The previous example of the matrix image in Figure 4 of different light intensities were simplified in order to evaluate the region finding and linear fitting algorithms. Using an array of numbers in the matrix, we could represent different light intensities called grayscale values. The gradient image would simulate contrasting regions between the areas of similar light intensities within the image. This simplification allowed for the evaluation of the region finding and line extraction algorithms prior to testing them in a real and more complex image such as Figure 6. Figure 7 shows the gradient image as a result of applying the edge extraction algorithm to the input image in Figure 6.

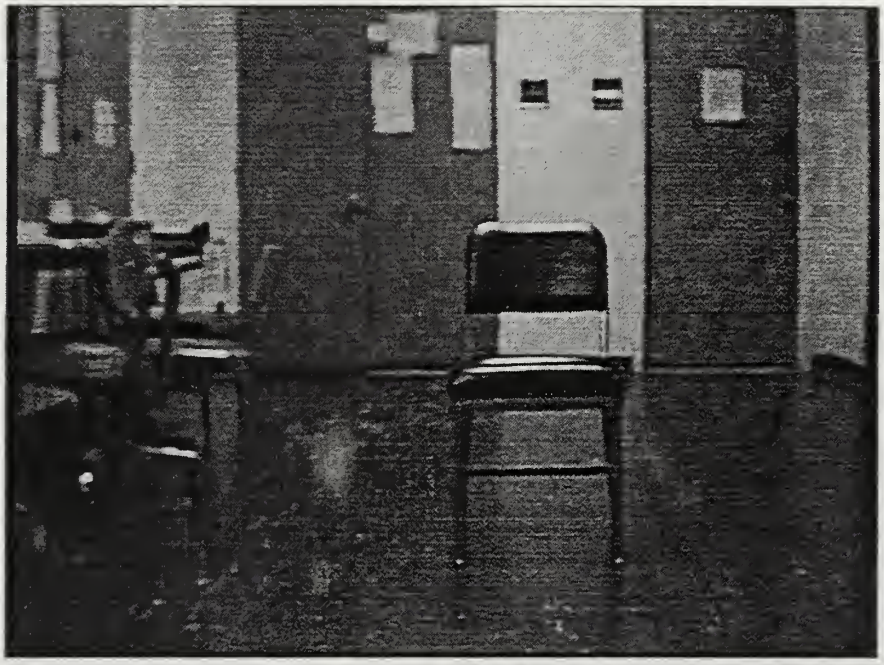

Figure 6. Grayscale image of hallway with chair.

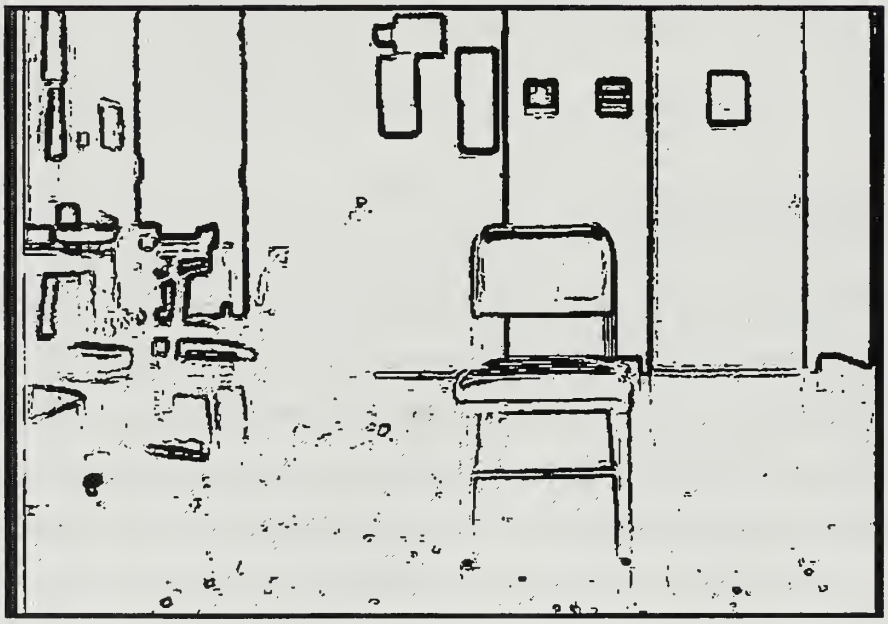

Figure 7. Gradient image of hallway with chair.

IV. EDGE EXTRACTION

A. GRADIENT-TYPE EDGE DETECTORS

The gradient-type edge detectors are used to determine pixel gradients. Two partial difference operators, one for determining the change of pixel intensities in the horizontal (*dx*) direction and another for the vertical direction, must be specified. Gradient-type edge detectors are square matrices of weights, mapped onto a group about a central pixel or point. The weights are multiplied with the intensities of the eight surrounding pixels and then summed to provide values for the intensity changes in the horizontal and vertical axes. Commonly used gradient-type edge detectors are Prewitt, Sobel, and Roberts gradients [BAL82] shown in Figures 7 through 10.

-1	0	1
–1	0	1
-1	0	1

dx

1	1	1
0	0	0
-1	–1	-1

dy

Figure 8. Prewitt gradient edge detectors.

-1	0	1
–2	0	2
-1	0	1

dx

1	2	1
0	0	0
-1	–2	-1

dy

Figure 9. Sobel gradient edge detectors.

−1	0
0	−1

dx

0	1
−1	0

dy

Figure 10. Roberts gradient edge detectors.

A modified Sobel gradient edge detector with values of $\sqrt{2}$ vice 2 for weights of the non-diagonal pixels in Figure 10 is used to compensate for a two-dimensional plane of pixels evenly spaced in both horizontal and vertical directions. The modified Sobel gradient tends to reduce the effects of noise.

-1	0	1
$-\sqrt{2}$	0	$\sqrt{2}$
-1	0	1

dx

1	$\sqrt{2}$	1
0	0	0
-1	$-\sqrt{2}$	-1

dy

Figure 11. Modified Sobel gradient edge detector.

1. Pixel Gradient Computation

To compute the pixel gradient magnitude, we let P be the set containing the eight surrounding pixels P with indices (i,j) known as location $P(i,j)$ shown in Figure 11.

$$\begin{bmatrix} P_{i-1,j+1} & P_{i,j+1} & P_{i+1,j+1} \\ P_{i-1,j} & P_{i,j} & P_{i+1,j} \\ P_{i-1,j-1} & P_{i,j-1} & P_{i+1,j-1} \end{bmatrix}$$

Figure 12. The set P of all eight pixels surrounding pixel $P(i,j)$.

We define the bi-directional gradients gx(i,j) and gy(i,j) by multiplying the corresponding weights in the Sobel edge detector matrix by the corresponding pixel intensity values in P.

$$\begin{aligned} g_x(i,j) &= -P(i-1,j-1) - 2P(i-1,j) - P(i-1,j+1) \\ &\quad + P(i+1,j-1) + 2P(i+1,j) + P(i+1,j+1) \end{aligned} \qquad \text{(Eq 2)}$$

$$\begin{aligned} g_y(i,j) &= P(i-1,j+1) + 2P(i,j+1) + P(i+1,j+1) \\ &\quad -P(i-1,j-1) + -2P(i,j-1) - P(i+1,j-1) \end{aligned} \qquad \text{(Eq 3)}$$

The bidirectional gradients gx(i,j) and gy(i,j) represent the change in pixel intensities in the horizontal direction (dx) and the vertical direction (dy) at pixel location $P(i,j)$. With these gradient components, the pixel Gradient Magnitude G(i,j) is calculated by

$$G(i,j) = \sqrt{(g_x(i,j))^2 + (g_y(i,j))^2} \qquad \text{(Eq 4)}$$

and the Gradient Direction $\Phi(i,j)$ is calculated by

$$\Phi(i,j) = \text{atan2}\,(g_y(i,j), g_x(i,j)) \qquad \text{(Eq 5)}$$

where atan2 is the subroutine function defined in Appendix A.

B. PROPERTIES OF AN EDGE REGION

We define the edge region as that boundary where the intensity level of the pixels is changing rapidly or whose pixels has a significant gradient magnitude. A pixel that has been included in the edge region must be significant, i.e. have a gradient magnitude greater than the specified threshold value. Threshold value can be determined dynamically by scanning the image once and computing the average weight of pixel intensities or by maintaining a histogram of previous images' average pixel intensities. For any two pixels to belong to an edge region, the pixels must be adjacent to each other and the pixels must have *close* gradient direction angles. We define closeness as the difference between the two pixel's gradient direction angles as being less than some constant angle ϕ .

A portion of an image with the edge region R between two distinct areas of common light intensity (areas A and B) is shown in Figure 12. Pixels included in the edge region R will have an average gradient direction angle close to the normal of the line segment describing that region.

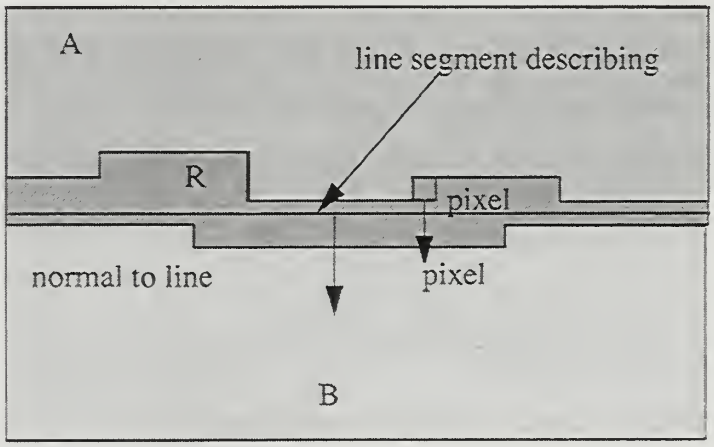

Figure 13. Edge region between two areas of common light intensity.

A pixel will not be included in the edge region if its gradient angle is not close to the region's average gradient angle Φavg. Thus, determining the pixels gradient angle and comparing the regions average gradient angle is crucial to the initial process of edge detection.

Another important property of the edge region is the number of pixels that have been included in the region. Regions that consists of relatively small amounts of pixels produces a line extraction image with many broken line segments. One cause for generating regions with a relatively small amount of pixels can be attributed to random variations in the image, termed noise [BAL82].

C. EDGE REGION ALGORITHM

1. Scanning Image

The image is scanned starting from the origin at pixel location (0,0) and continues to be scanned from left to right, bottom to top as shown in Figure 13.

2. Adjacency Test

Once the pixel's gradient magnitude $G(i,j)$, gradient direction $\Phi(i,j)$, and significance have been determined, the pixel is then examined to be included in a region. The adjacency test evaluates the current pixel at location (i, j) and compares its gradient direction with each of four adjacent pixels to determine gradient direction closeness. The adjacent pixels are the pixel to the left $(i-1, j)$, the pixel below $(i, j-1)$ and its left $(i-1,j-1)$ and right $(i+1, j-1)$ neighbors as shown in Figure 13.

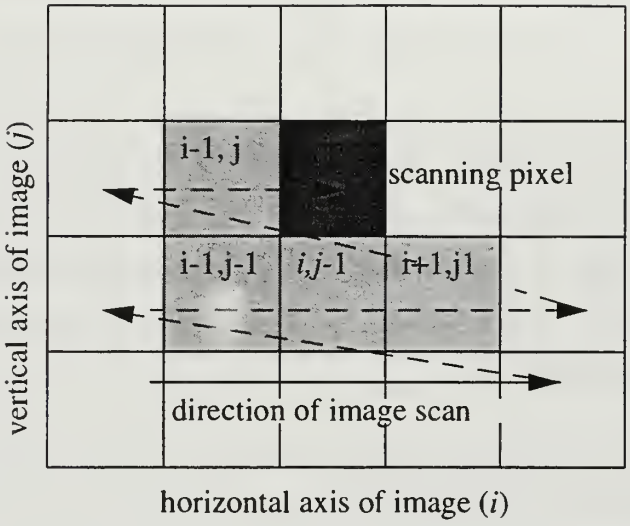

Figure 14. Direction of scanning an image.

3. Pixel Inclusion In a Region

The pixel being evaluated is included in a region of existing pixels of adjacent neighbors that have close gradient directions and are significant. If the pixel does not meet the criteria to be included in a region, a new region is created with this pixel as the start of the new region.

For any two pixels (i,j) to be included in one region, R, the following conditions must exist:

- Pixels (i,j) and R must be adjacent.

- The difference between the gradient direction $\Phi(i,j)$ of pixel (i,j) and the average gradient direction Φave of the region R is less than some specified angular difference ϕ .

If a pixel is adjacent to more than one significant pixels that are included in the same region, the new pixel will be included in that region if it meets the above conditions, otherwise it will be part of a new region.

D. LINEAR FEATURE EXTRACTION

Once the gradient image is constructed, we must provide a method for recognizing the sets of data points or pixels which form the linear feature of the region and a method for finding and describing the line segment that best fits these sets of data points. The method used for determining a line segment from the two-dimensional region of data points is by least squares fitting. In the gradient image, the moments of pixel locations (x, y pixel coordinates of the image) for all pixels in the region are computed to obtain the line segment for the regions's major axis. This line segment continues to grow until certain measures of the line segment indicate that the line segment should be ended and a new one started. We use an implementation of least squares fitting described by [KAN90].

1. Least Squares Fitting

Suppose we have collected n consecutive valid data points in a local coordinate system, $(p_1,\ldots, p_n)$, where $p_i = (x_i, y_i)$ for $i = 1,\ldots,n$. We obtain the moments m_{jk} of the set of points

$$m_{jk} = \sum_{i=1}^{n} x^j{}_i y^k{}_i \qquad (0 \le j, k \le 2, \text{ and } j + k \le 2) \qquad \text{(Eq 6)}$$

Notice that $m_{00} = n$. The *centroid* C is given by

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \equiv (\mu_x, \mu_y) \qquad \text{(Eq 7)}$$

The secondary moments around the centroid are given by

$$M_{20} \equiv \sum_{i=1}^{n} (x_i - \mu_x)^2 = m_{20} - \frac{(m_{10})^2}{m_{00}} \qquad \text{(Eq 8)}$$

$$M_{11} \equiv \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) = m_{11} - \left(\frac{m_{10} m_{01}}{m_{00}} \right) \qquad \text{(Eq 9)}$$

$$M_{02} \equiv \sum_{i=1}^{n} (y_i - \mu_y)^2 = m_{02} - \frac{(m_{01})^2}{m_{00}} \qquad \text{(Eq 10)}$$

We adopt the parametric representation (r,α) of a line with constants r and α. If a point $p = (x,y)$ satisfies an equation

$$r = x\cos\alpha + y\sin\alpha \qquad (-\pi/2 < \alpha \leq \pi/2) \qquad \text{(Eq 11)}$$

then the point p is on a line L whose normal has an orientation α and whose distance from the origin is r as shown in Figure 14. This method has an advantage in expressing lines that are horizontal to the X axis. The point-slope method, where $y = mx + b$, is incapable of representing such a case ($m = \infty$, b is undefined).

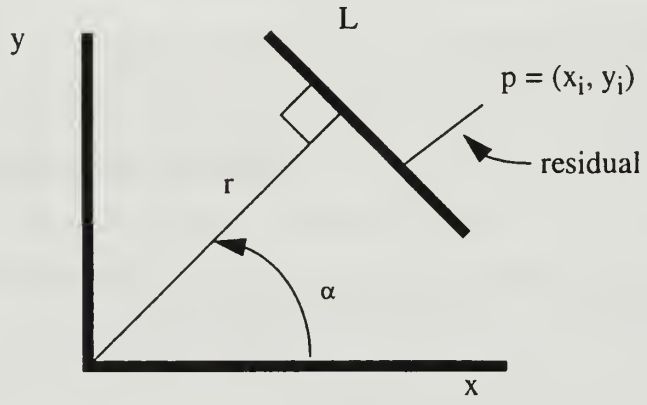

Figure 15. Representation of a line L using r and α.

The residual of point $p_i = (x_i, y_i)$ and the line L = (r,α) is $x_i\cos\alpha + y_i\sin\alpha - r$. Therefore, t the sum of the squares of all residuals is

$$S = \sum_{i=1}^{n} (r - x_i\cos\alpha - y_i\sin\alpha)^2 \qquad \text{(Eq 12)}$$

The line which best fits the set of points is supposed to minimize S. Thus the optimum line (r,α) must satisfy

$$\frac{dS}{dr} = \frac{dS}{d\alpha} = 0 \qquad \text{(Eq 13)}$$

Thus,

$$\frac{dS}{dr} = 2\sum_{i=1}^{n}(r - x_i\cos\alpha - y_i\sin\alpha) \quad \text{(Eq 14)}$$

$$= 2\left(r\sum_{i=1}^{n}1 - \left(\sum_{i=1}^{n}x_i\right)\cos\alpha - \left(\sum_{i=1}^{n}y_i\right)\sin\alpha\right) \quad \text{(Eq 15)}$$

$$= 2\,(rm_{00} - m_{10}\cos\alpha - m_{01}\sin\alpha) \quad \text{(Eq 16)}$$

$$= 0$$

and

$$r = \frac{m_{10}}{m_{00}}\cos\alpha + \frac{m_{01}}{m_{00}}\sin\alpha = \mu_x\cos\alpha + \mu_y\sin\alpha \quad \text{(Eq 17)}$$

where r may be negative. Substituting r in Equation (12) by Equation (17) ,

$$S = \sum_{i=1}^{n}\left((x_i - \mu_x)\cos\alpha + (y_i - \mu_y)\sin\alpha\right)^2 \quad \text{(Eq 18)}$$

Finally,

$$\frac{dS}{d\alpha} = 2\sum_{i=1}^{n}\left(\left(x_i - \mu_x\right)\cos\alpha + \left(y_i - \mu_y\right)\sin\alpha\right)\left(-\left(x_i - \mu_x\right)\sin\alpha + \left(y_i - \mu_y\right)\cos\alpha\right) \quad \text{(Eq 19)}$$

$$= 2\sum_{i=1}^{n}\left(\left(y_i - \mu_y\right)^2 - \left(x_i - \mu_x\right)^2\right)\sin\alpha\cos\alpha + 2\sum_{i=1}^{n}\left(x_i - \mu_x\right)\left(y_i - \mu_y\right)(\cos^2\alpha - \sin^2\alpha) \quad \text{(Eq 20)}$$

$$= (M_{02} - M_{20})\sin 2\alpha + 2M_{11}\cos 2\alpha \quad \text{(Eq 21)}$$

$$= 0$$

Therefore

$$\alpha = \frac{\operatorname{atan}\left(2M_{11}/(M_{02} - M_{20})\right)}{2} \quad \text{(Eq 22)}$$

Equation (17) and Equation (22) are the solutions for the line parameters generated by a least squares fit.

2. Finding Endpoints

The residual of a point $p_i = (x_i, y_i)$ is

$$\delta_i = (\mu_x - x_i)\cos\alpha + (\mu_y - y_i)\sin\alpha \qquad \text{(Eq 23)}$$

Therefore, the projection, p'_i of the point p_i onto the major axis of the distribution Ellipse is

$$p'_i = (x_i + \delta_i\cos\alpha, y_i + \delta_i\sin\alpha) \qquad \text{(Eq 24)}$$

We will use p'_1 and p'_n as estimates of the endpoints of the line segment L obtained from the set p of data points.

3. Line Segment Validity Test Of Edge Region

The equivalent ellipse of inertia in Figure 15 for the edge region, R, will have the same moments about the centroid (M_{20}, M_{11}, and M_{02}) as R. M_{major} and M_{minor}, the moments about the major axes of the ellipse, are defined as:

$$M_{major} = \frac{(M_{20} + M_{02})}{2} - \sqrt{\left(\frac{M_{02} - (M_{20})}{2}\right)^2 + M_{11}{}^2} \qquad \text{(Eq 25)}$$

$$M_{minor} = \frac{(M_{20} + M_{02})}{2} + \sqrt{\left(\frac{M_{02} - (M_{20})}{2}\right)^2 + M_{11}{}^2} \qquad \text{(Eq 26)}$$

The lengths of the major and minor axes, d_{major} and d_{minor} are:

$$d_{major} = 4\sqrt{\left(\frac{M_{major}}{m_{00}}\right)} \qquad \text{(Eq 27)}$$

$$d_{major} = 4\sqrt{\left(\frac{M_{minor}}{m_{00}}\right)} \qquad \text{(Eq 28)}$$

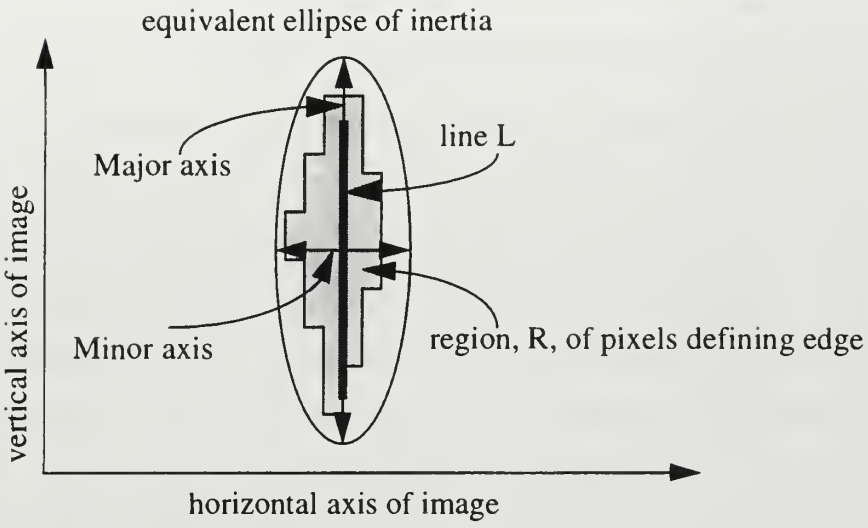

Figure 16. Equivalent ellipse of inertia.

We consider the major axis to be the line segment L, for the edge region. The endpoint of the major axis will be the endpoints for the desired line segment. The projections of the first and last pixels, p'_1 and p'_n associated with the region R, will therefore be used as estimates for the two endpoints of L. A ratio, ρ, of the axes length can then be used to describe the *thinness* of the ellipse.

$$\rho = \sqrt{\frac{d_{minor}}{d_{major}}} \qquad \text{(Eq 29)}$$

Using the values for number of pixels (m_{00}), major axis length (d_{major}), and the ellipse thinness (ρ) as parameters for comparison to edge region significance, edge length and how much the edge region resembles a line can be simply tested as long as the least squares fits moments are maintained for every pixel included in an edge region. Let C_3($0 \le C_3 < 1$) be a constant for the maximum allowable ratio ρ that can be used to describe a line. Let C_4 be a specified constant for the minimum number of pixels and C_5 for the minimum line length. Three requirements can therefore be specified for the line testing of an edge region.

1. The ratio of axes length is less than the maximum rho $(\rho < C_3)$,

2. The number of pixels is greater than the specified minimum $(m_{00} > C_4)$, and

3. The line length of the region is greater than the minimum length $(d_{major} > C_5)$.

The ratio ρ proves to be the most significant measurement. For ρ to equal 1.0 means that the length of the minor axis is equal to the length of the major axis, representing an edge region that resembles a circular blob. Therefore, ρ can be compared to a maximum ratio, C_3, specified by the user. C_3 equal to 1.0 allows all edge regions to be considered thin enough to represent a line segment. A value of 0.1 seems to work well.

The two other tests can be used to trim down the number of smaller, less significant line segments. The regions that meet these requirements will be saved for the desired output as the found line segments and all the other regions will be discarded.

E. EDGE EXTRACTION RESULTS

The edge region algorithm and the linear feature extraction described in the previous sections were implemented in the *testedge* algorithm detailed in Appendix A. The results of the input matrix images are shown in the following figures. Figure 16 shows the generation of edge regions from the square matrix image previously shown in Figure 4. Figure 17 is the line segment extraction of the square image in Figure 16.

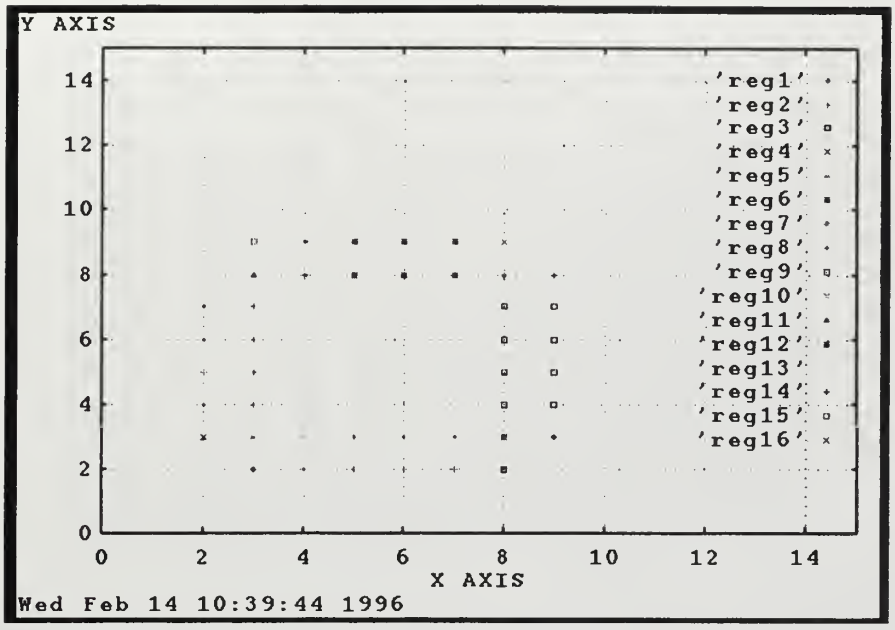

Figure 17. Edge region generation of matrix square image.

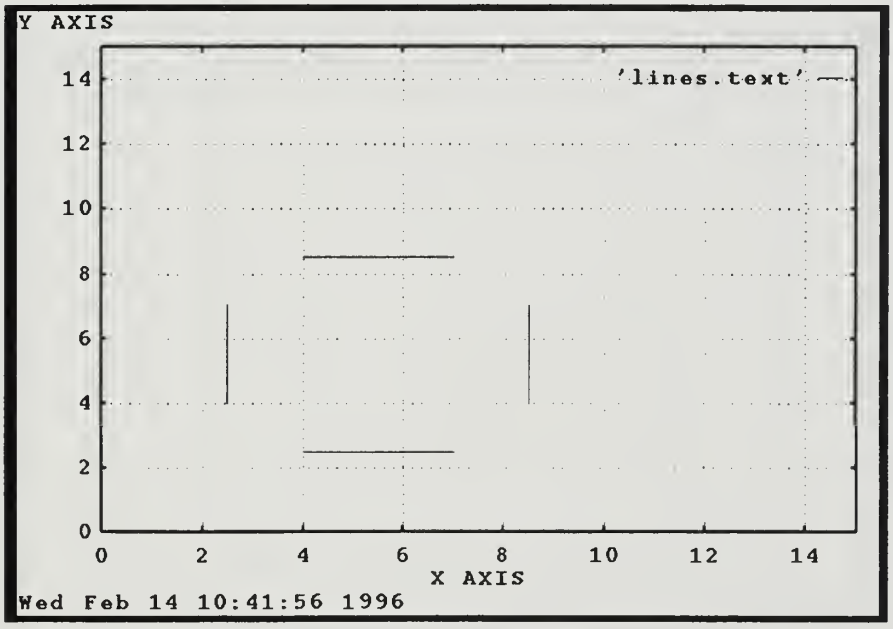

Figure 18. Line segment extraction of edge region of square image.

Figure 18 through Figure 23 show the matrix images and line segment results of more sample images.

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 150 & 0
\end{bmatrix}$$

Figure 19. Two light intensity values of 12 by 12 matrix sample image.

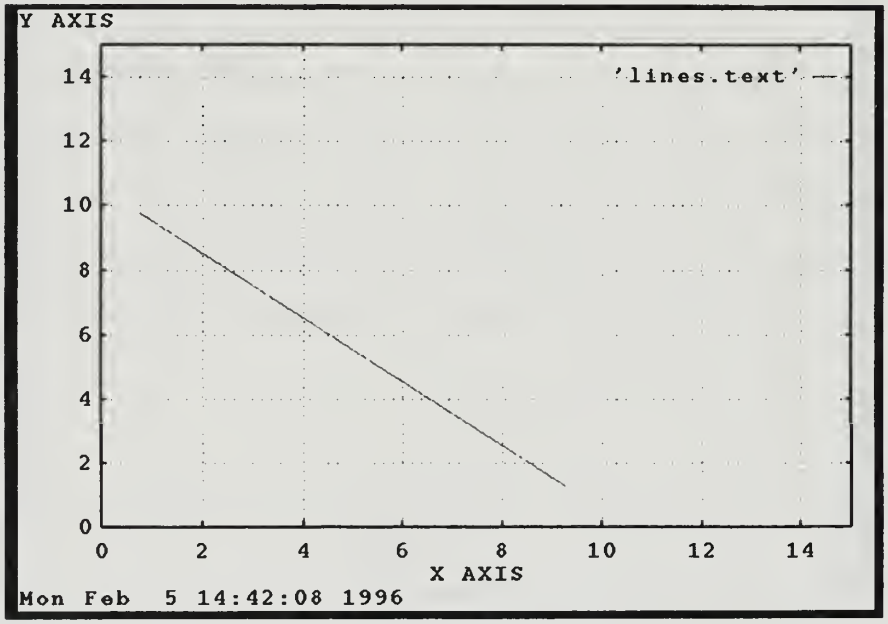

Figure 20. Line segment extraction of above image.

0	0	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	0	0	0	255
150	150	0	0	0	0	0	0	0	0	255	255
150	150	150	0	0	0	0	0	0	255	255	255
150	150	150	150	0	0	0	0	255	255	255	255
150	150	150	150	150	0	0	255	255	255	255	255
150	150	150	150	150	150	255	255	255	255	255	255
150	150	150	150	150	150	255	255	255	255	255	255
150	150	150	150	150	150	255	255	255	255	255	255
150	150	150	150	150	150	255	255	255	255	255	255
150	150	150	150	150	150	255	255	255	255	255	255
150	150	150	150	150	150	255	255	255	255	255	255

Figure 21. Three light intensity values of 12 by 12 matrix sample image.

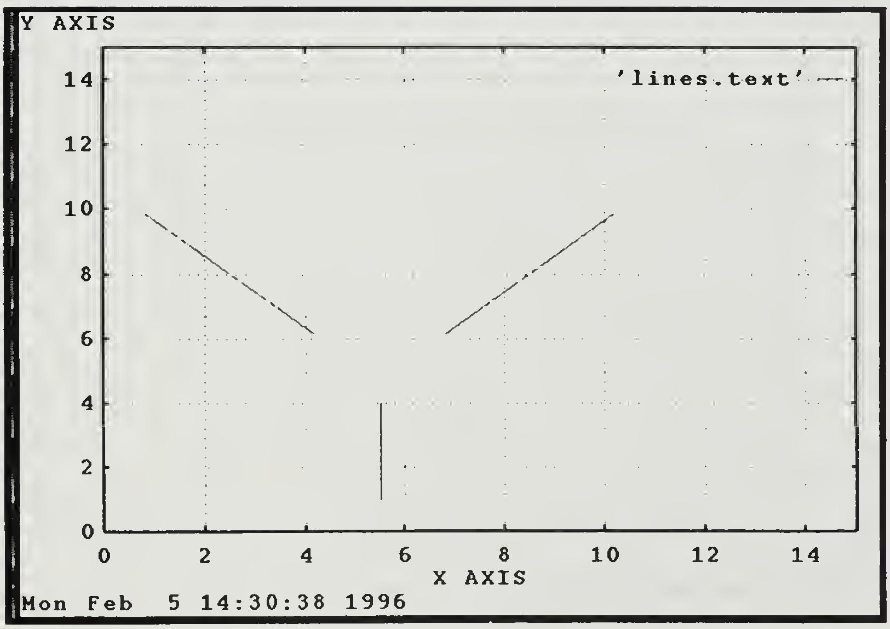

Figure 22. Line segment extraction of above image.

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 & 150 & 150 & 150 \\
0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 & 150 & 150 & 150 \\
0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 & 150 & 150 & 150 \\
0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 & 150 & 150 & 150 \\
0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 & 150 & 150 & 150 \\
0 & 0 & 0 & 0 & 0 & 0 & 150 & 150 & 150 & 150 & 150 & 150 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0 \\
150 & 150 & 150 & 150 & 150 & 150 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 23. Two light intensity values of 12 by 12 matrix sample image.

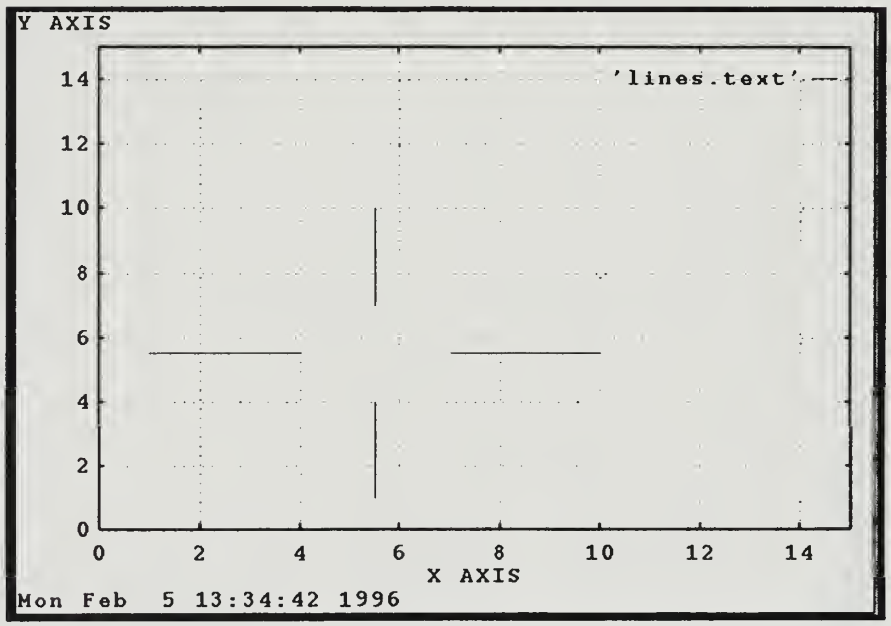

Figure 24. Line segment extraction of above image.

The edge region algorithm and the linear feature extraction described in the previous sections were implemented in the *findedge* algorithm detailed in Appendix B. The results of an actual input images shown in the following figures. The images shown in Figure 24 and 27 produces the gradient images shown in Figures 25 and 28 respectively.

The extracted line segment features are shown in Figures 26 and 29. The five parameters for gradient threshold and line segment properties C_1 through C_5 were determined by trial and error to achieved the best line segment extraction.

Figure 25. Input image of hallway.

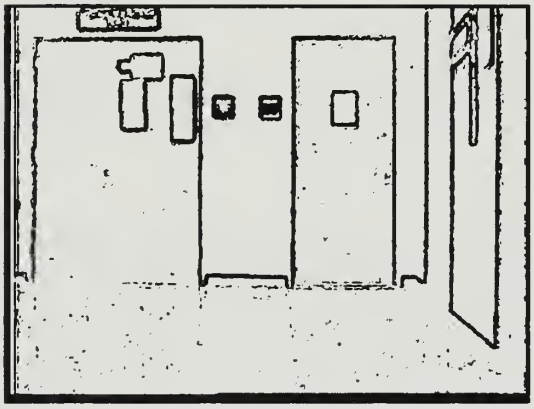

Figure 26. Gradient image of hallway. $C_1 = 80.0$, $C_2 = 15.0$degrees.

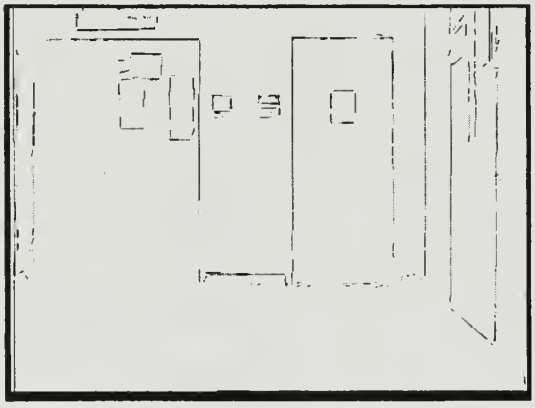

Figure 27. Line segment extraction via *findedge* implementation. $C_3 = 1.0$, $C_4 = 10$ pixels, $C_5 = 10.0$.

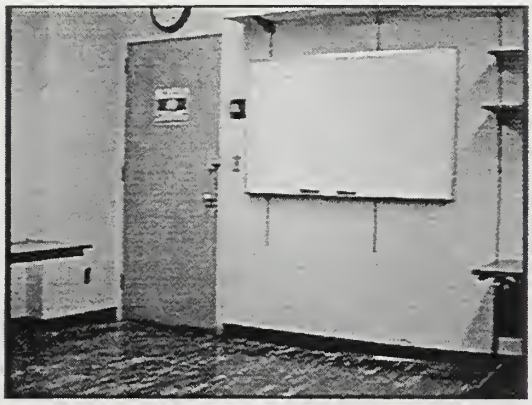

Figure 28. Input image of room corner.

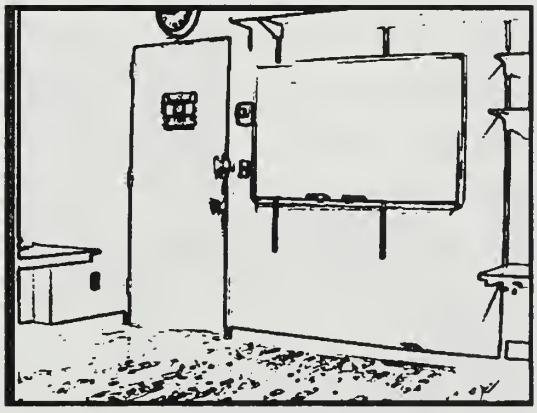

Figure 29. Gradient image of room corner. $C_1 = 80.0$, $C_2 = 20.0$ degrees.

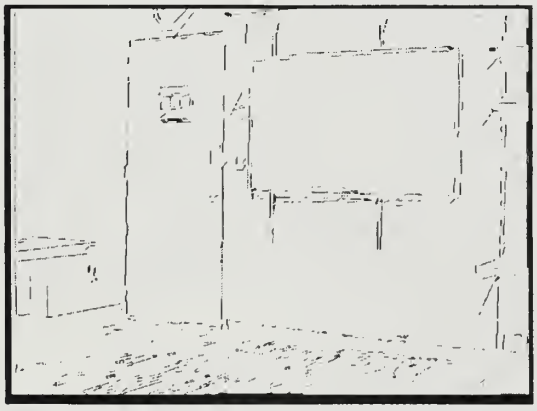

Figure 30. Line segment extraction via *findedge* implementation. $C_3 = 0.1$, $C_4 = 7$ pixels, $C_5 = 7.0$.

The results of both line segment extractions can be refined by varying the parameters to find the best line segment output. However, this method is tedious and not ideal for implementation on board a robot operating in a dynamic environment. For this reason, we assume that the interior lighting level will remain relatively constant while the robot traverses its environment. An input image taken by the robot will be processed with a common set of edge region and linear feature parameters. These parameters could be established based on heuristics of various input images the robot would see in its environment. The parameters used for this test were varied slightly to improve the line segment extraction. The parameters were also left the same for all input images and the results were compared with the images in which the parameters were varied. However, no significant changes were seen in the line segment extraction of the images. Examination of the line segment extraction results are consistent and enables further processing for pattern matching and pose determination [PET92].

Figures 30 through 32 demonstrate the ability of the *findedge* algorithm to discriminate details of an input image. This ability is ideal for object identification. Extracting enough information from an image could provide additional three-dimensional data to enhance the "intelligence" of the robots's obstacle avoidance maneuvers.

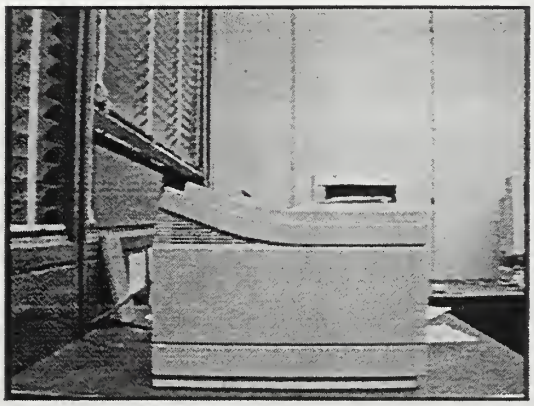

Figure 31. Input image of a printer.

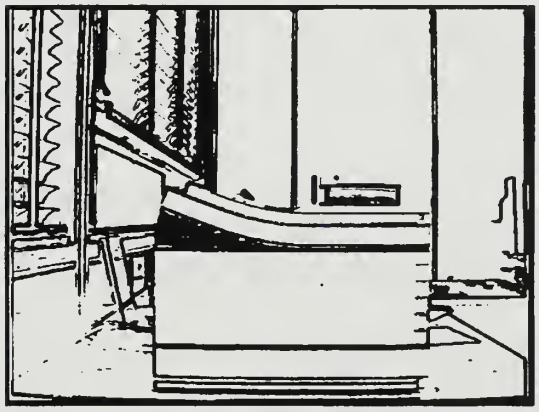

Figure 32. Gradient image of printer. $C_1 = 70.0$, $C_2 = 15.0$ degrees.

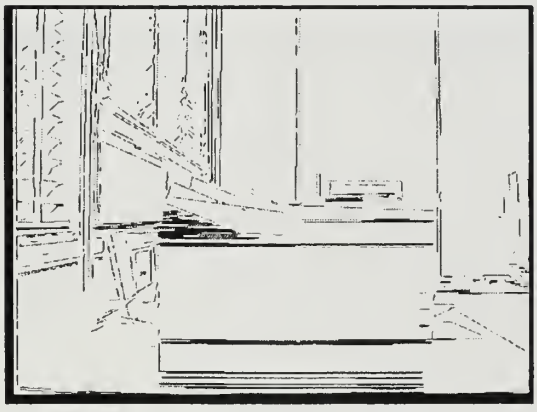

Figure 33. Line segment extraction via *findedge* implementation. $C_3 = 1.0$, $C_4 = 7$ pixels, $C_5 = 7.0$.

An implementation of object identification on *Yamabico* is the alignment method [ULL91]. The basic premise of this approach is that given a known set of feature points for a known object and the same points on an unknown object, it is possible to map the two sets via constant coefficient linear equation if they are alike. The powerful aspect of this relationship is the fact that it is valid regardless of rotational/and or translational differences, permitting direct analysis of image objects according to the object database. [DEC93]. Once classifying an object, the ability to obtain object depth allows for obstacle avoidance measures based on image processing.

V. IMAGE TO SPARC BOARD INTERFACE

A. SPARC BOARD DESCRIPTION

Yamabico's IV-SPRC-25A CPU board has 16Mbytes of DRAM, located in the lower portion of the 4Gbyte address space. The address space above this 16Mbytes is devoted to VME bus use. It contains several memory regions as shown in Figure 33. The address space above these VMEbus regions is reserved for EPROM and board configuration registers.There are three logical mappings to the VME bus: the cluster-internal virtual bus, the synchronous Mbus and the asynchronous T-bus. All of *Yamabico's* address space has been mapped using the T-bus.

The VMEbus interface which resides on the T-bus allows operations with Motorola 68020 type protocols. A T-bus master may only access other T-bus devices and local DRAM. T-bus features are:

- Fully asynchronous bus
- Separate 32-bit address and data buses
- Four Gigabytes of physical memory address space

The T-bus 32-bit address space is conceptually divided into seven regions. Some of these regions are fixed and others have programmable sizes. Figure 33 shows the default address space which was used in this system. The space is initialized by resident initialization code in the Ironics SPARC CPU card. There are two regions which involve the *Yamabico*'s image subsystem: Region 3 and A24 space [KIS95].

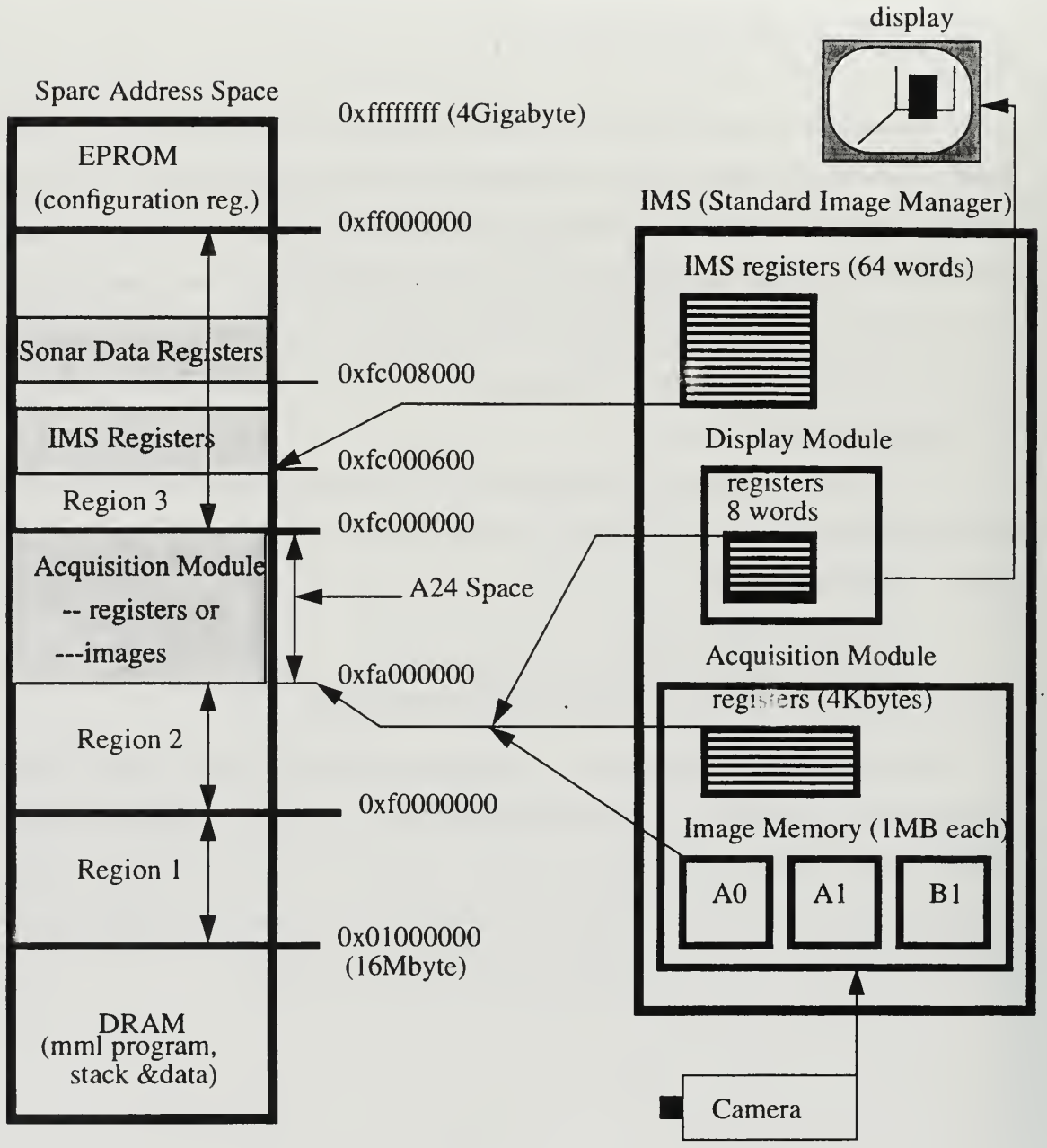

Figure 34. VME mapping of image modules into SPARC-4 address space [KIS95].

1. Region 1 Address Space

Region 1 supports the same three addressing modes as region 3. It can deliver data as 16 or 32 bit blocks. The region starts at the end of local DRAM space and extends through the address specified in the Boundary 2 address register (0xfffd0600). The attributes of this region are set in the Region 1 Attributes register (0xfffd0900). The Ironics initialization code maps region 1 to the VMEbus as A32/D32.

2. Region 2 Address Space

Region 2 supports the same addressing/data modes as region 1 and 3. It starts at the end of region 1 extending up to the start of region 3.

3. Region 3 Address Space

Region 3 starts at the end of region 2 and extends to the bottom of the EPROM address space (0xff000000). The attributes of this region are set in the Region 3 Attributes Register (0xfffd0b00). The Ironics initialization code initially sets up this region for addressing and data which contain 32 bits (A32/D32). To avoid conflicting with the address space used by the sonar, the image board can be offset from the base address of this region up to 0xff00 in 0x100 intervals. The default offset, 0x0600, was chosen for *Yamabico* since it does not conflict with the VME address space for the sonar or the dual axis controller registers. The sonar registers are mapped to 0xfc008000 and the image board is mapped to 0xfc000600 and uses 64 words (0x00-0x7f).

The image board addresses in this region are used to set initial configuration of the image board. This includes setting up address space for the actual images and additional modules such as the acquisition module and the display module. Configuration information stored in this region includes:

Size mapped into VME memory space (image memory)
Address space desired (24 bit of 32 bit)
Input frame masking bits
Page selection (image, acquisition module, display module

4. A24 Space

The A24 space contains the actual images plus the acquisition module registers and display module registers. This region can overlay all three regions previously discussed. It can start anywhere from the top of local DRAM up to the bottom of EPROM address space. *Yamabico*'s image subsystem uses the Ironics default for this region which is from 0xfa000000 to 0xfbffffff. The default setting also assumes a 16 bit data width for the slave board. This default setting is also used, although data can be up to 32 bits wide.

B. IMAGE BOARD DESCRIPTION

1. Standard Image Manager (IMS)

After setting up the A24 Space attributes some IMS configuration registers must be set to allow the acquisition module registers to map to this region [ITIIMS93]. These include the IMS configuration register, and page select registers. The IMS status register can also be checked to confirm that the IMS board is present. The configuration register contains bits that set the amount of memory to be mapped to VME address space and allow selection between standard (A24) or extended (A32) addressing. In the A32 mode, the high byte of the configuration register configures the upper 8-bits of the memory base address.

As shown in Figure 33, the acquisition module is mapped to 0xfa000000, which is the beginning of A24 space. Standard addressing (24 bits) was selected with a map size of 1MB. Module register access is not affected by this mapping size. With a map size of 1 Mbyte, the three images, the acquisition module registers and display module registers can be mapped to VME address space using the page register. The page register allows the setting of pixel size, choice of enabling or disabling VME memory access and the choice of selecting either one of three memory pages or the acquisition and display modules. Initially the page register is set to access the acquisition module registers through VME memory addresses. Figure 34 shows the flow of pixel data through the image manager.

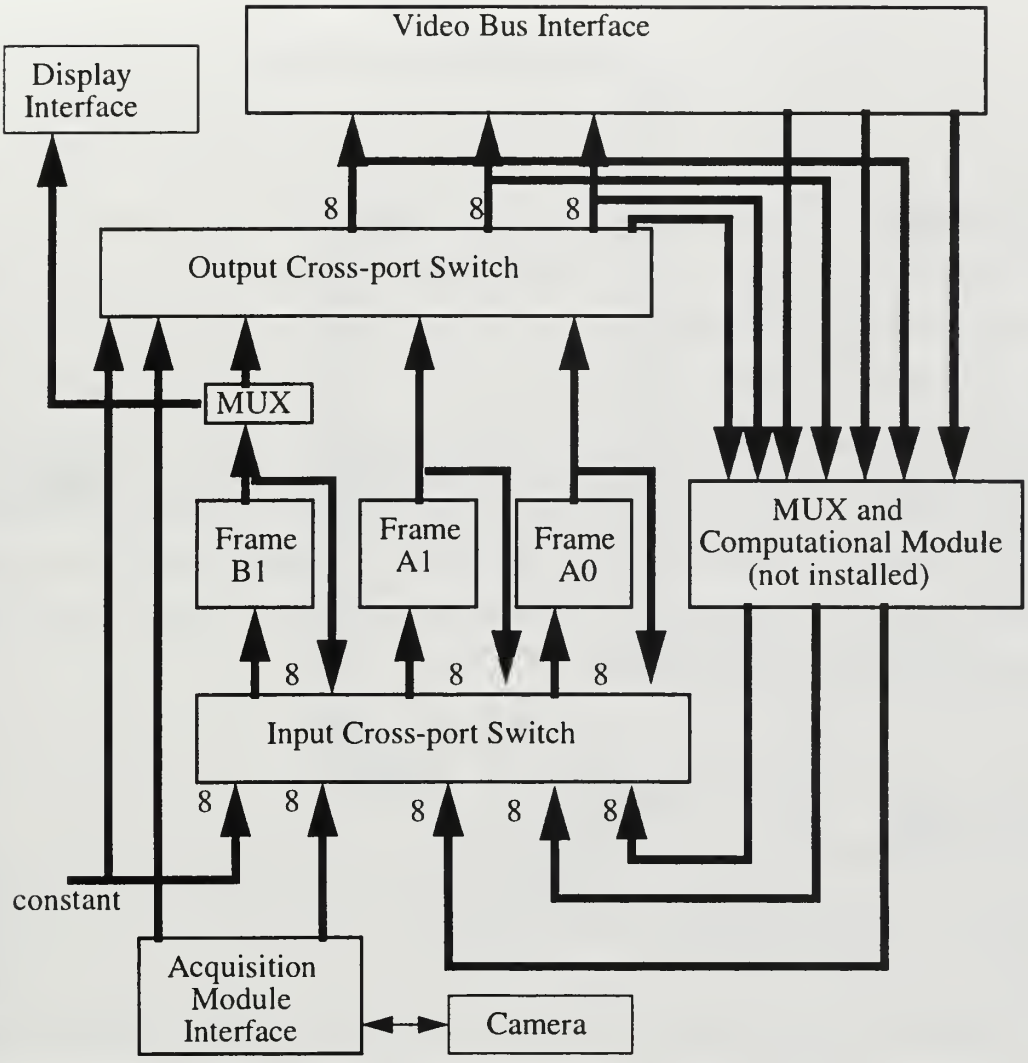

Figure 35. Standard Image Manager (IMS) [ITIIMS93].

2. Acquisition and Display

The IMS board is supported by two smaller plug-in modules: the Acquisition Module and the Display module. The Acquisition module provides an interface between the camera and the IMS board. The display module is described in the next section and interfaces with a display monitor.

a. Acquisition Module (AM)

The acquisition module is produced by the same company as the main image board (Imaging Technology Inc.)[ITIAM93]. It is designed to interface with many different cameras and operate in a variety of modes. This is both an advantage and a hinderance since the 17 registers augmenting the 43 registers on the main image board must all be programmed. The module itself can receive up to 24 bits in parallel, but the camera chosen is only an 8 bit grayscale. Our 8 bit version uses a RS-422 for data input. The camera's 8 bit pixel data is first stored into a 4k by 8 bit FIFO queue and then transferred to the motherboard. In the 8 bit version of ITI's acquisition module the 8 bit camera data is passed directly to a 4K by 8 bit FIFO queue. The 8 bits are duplicated on three output channels. Since our camera is an area scan device, the acquisition module outputs horizontal and vertical frame timing to the mother board based on the line enable, frame enable and pixel clock inputs from the camera [KIS95].

b. Display Module (DM)

The DM-PC Pseudocolor Display Module (DM) is a plug-module for the image manager. It provides a medium resolution pseudocolor RGB display for many types of monitors including 1024 by 768 non-interlaced and up to 1024 by 1024 interlaced monitors. It receives 8 bits of image data from the mother board (IMS module) and converts it into RGB pseudocolor. Overlay memory is supported for applications requiring graphics to overlay images. The heart of the display module is the Texas Instruments TMS34010 Graphics System Processor (GSP) which controls all graphics and image display functions. All display module registers are in-turn mapped to the GSP registers. The pseudocolor transformation is performed by a Bt478 RAM digital-to-analog converter (RAMDAC). All options for displaying are software programmable. The route of data through these components is shown in Figure 35.

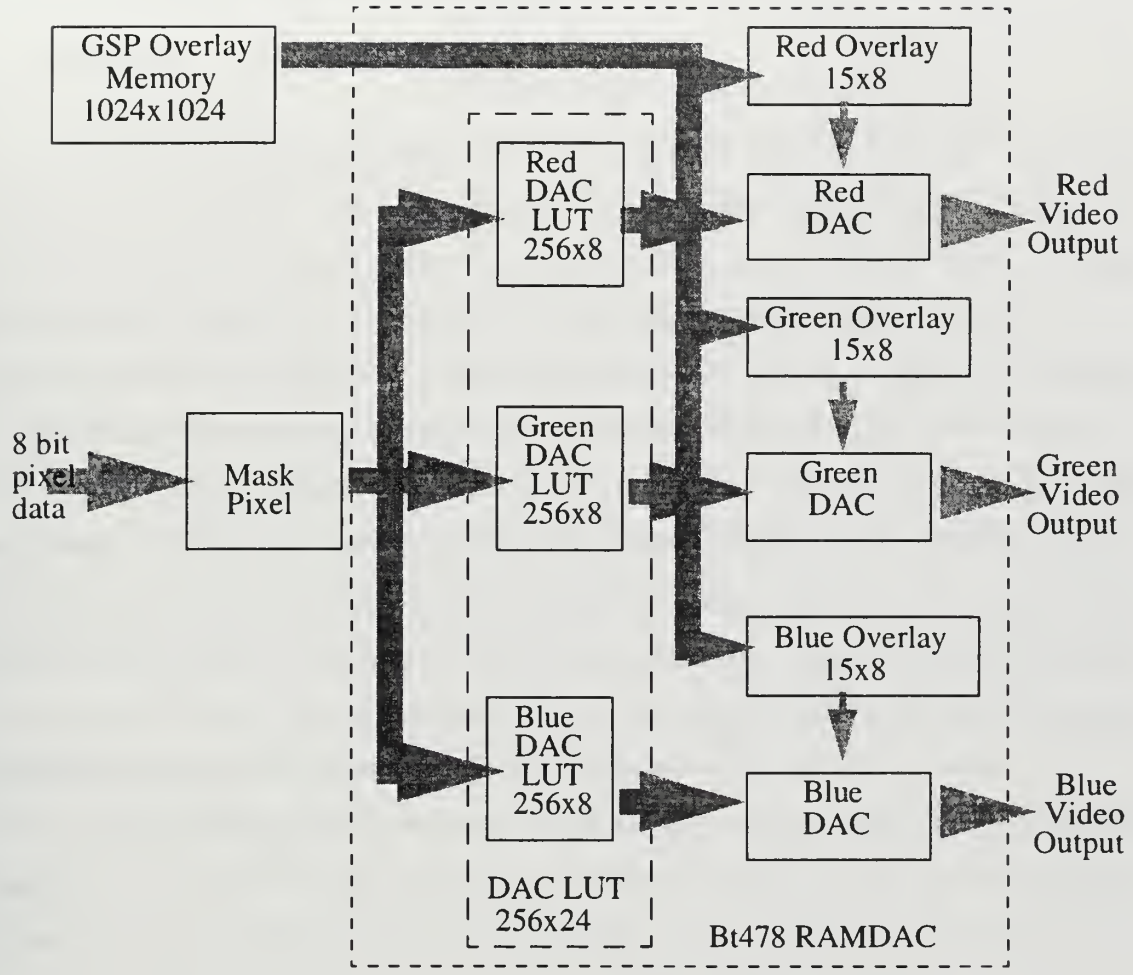

Figure 36. Display Module analog conversion block diagram [ITIDM92].

The display module displays an image that is stored in frame B1 of the IMS mother board. Therefore it does not do any processing on the image, other than the mapping that is done to display the image in a RGB format. The DM does support an overlay which can be programmed to display menus or text. This feature may be used in later research to display lines generated by edge finding software.

The RAMDAC uses a look-up table (LUT) for the mapping of the 8 bit grayscale image on to a 24 bit RGB display image. There is also program memory available on the GSP which can be used to store operation code, such as the TIGA™ graphical user

interface. There are three main hardware components for converting eight bit digital data into an analog video output:

- Three Digital to Analog Converters (DAC)
- DAC LUT (look-up table)
- Overlay Color Table
- Pixel Mask

Figure 35 is a block diagram which shows the flow of data within the display module. The 8 bit image first passes through the pixel mask which allows the programmer to strip bits from each pixel. This could be used to limit the number of values used for each shade of gray and therefore, certain thresholds of differences between grayscales. For example, if only two values were required for further image processing, the pixel mask could be set to 1000 0000 binary. This would map the grayscale input to either 00 or 80 hexadecimal and any pixel with a grayscale of 127 or less would be displayed as white and any pixel with a value greater than 127 would become gray.

After exiting the pixel mask, the pixel data is transformed into a pseudocolor image with the DAC LUT. The DAC LUT consists of red, green and blue triplet bytes that contain the conversion value for each color. Because we want to see a grey scale representation of the image, we have coded the DAC LUT to echo the value of the pixel for all three colors. The DAC LUTs could be used to accent a certain value, perhaps a threshold value of interest. In that case when a pixel with the threshold value was received, the green and blue DAC LUTs could output a 0x00 will the red outputs 0xff. This would cause all pixels with the value of interest to be displayed in red.

The final processing that takes place prior to the image being displayed is combining an overlay with the image. The overlay memory consists of 1024x1024x4 bits of data. When the value of the 4 bits is zero, no overlay is displayed. The other 15 values can be obtained from the 4 bits and will override the image data with colors from a overlay color table. This table is accessed the same way the DAC LUT does with red, green and blue data obtained for each pixel [KIS95].

VI. IMAGE UNDERSTANDING SYSTEM ON YAMABICO-11

A. OVERALL FUNCTIONALITY

The image understanding system on Yamabico-11 incorporates the real-time processing capabilities of the Single Board Super Computer System, SPARC™ CPU, and the image processing system of the Standard Image Manager (IMS) VME Board described in Chapter V. The recent installation of a COHU digital output camera coupled with the image board yields a higher system resolution and NTSC grayscale compatibility than the previous implementation of the JVC CCD camera described in Chapter II.

The image understanding system is partly based on edge extraction and line fitting algorithms of [PET92] with a 3-D geometric model of the robot's world [STE92]. By using image grabbing routines of [KIS95], images are captured with the digital output camera installed on the robot. An image recording feature called the image log, similar to that of the motion and sonar log, was developed to store the captured image in the robot's memory for downloading to a workstation for further analysis.

By porting image understanding routines previously developed for a Silicon Graphics Iris (SGI) workstation and modifying the routines for compatibility with *Yamabico's* image memory structure, the robot now has the ability to process images on-board.

B. INITIALIZING IMAGE MANAGER

The standard image manager's initialization routine first sets the paging register to allow selection of frame memory, acquisition registers or display registers. The display initialization routine is called first. It is followed by the acquisition initialization routine. The frame masks are then cleared to allow 8 bits of pixel data to pass to each frame. Two control registers are then set. The first is for the image manager, which sets the clock frequency, along with enabling display and acquisition. More information on image initialization and routines can be found in [KIS95].

C. ACQUIRING IMAGE WITH SNAP COMMAND

1. setInputPath

This routine selects the camera input as the source of the image and puts the image in frames A0, A1, B1, where it will reside:

- setInputPath(A0, CAMERA);
- setInputPath(A1, CAMERA);
- setInputPath(B1, CAMERA);

2. setFrameAcquire

This routine sets up frames to receive a single image. The image operation is actually executed by the "acqEnable" command.

- setFrameAcquire(A0, SNAP);
- setFrameAcquire(A1, SNAP);
- setFrameAcquire(B1, SNAP);

3. acqEnable

This routine initiates the snap operation specified. Prior to this operation, frame B1 is waiting and ready to receive the image. This command starts the acquisition. The command has no function call parameters:

- acqEnable();

D. FORMATTING IMAGE FOR PROCESSING

Combining the operations from the previous section, allows the formation of the primitives needed to format an image. The support routines used are described in [KIS95].

1. Log and snap an image in frame B1 for processing

- setInputPath(B1, CAMERA);
- ImageLog(NULL, 0)
- setFrameAcquire(B1, SNAP)

- acqEnable();
- copyImageToMemory(B1);

The following code segment copies the image to memory as a two-dimensional array. This was needed for compatibility of the edge extraction and linear fitting algorithms.

```
YAMAIMAGE b1Image;
void
copyImageToMemory(IMSPage frameNumber)
{

unsigned long page1Index = 0xfa000000; /* First empty line for pg2 pixels */
unsigned long page2Index = 0xfa07a400; /* First 1K of pixels on page 2*/
unsigned long page1Source = 0xfa000000;
unsigned long page2Source = 0xfa07a400;
int i,j;
int xSize = 732;
int ySize = 476;
 b1Image.xSize = xSize;
 b1Image.ySize = ySize;

  for(i=0; i<238; i+=2)
  {
    for (j=0; j<366; j+=2) /*732 bytes but 2 bytes per access*/
    {
        b1Image.image[i][j] = *(BYTE*)page1Source;
        page1Source += 2;
        b1Image.image[i+1][j] = *(BYTE*)page2Source;
        page1Source += 2;
}
    page1Index += 0x800;
    page2Index += 0x800;
```

```
        page1Source = page1Index;
        page2Source = page2Index;
    }
        LogImage(b1Image);
}
```

E. IMAGE UNDERSTANDING FOR MOTION PLANNING

Sonar research [MAC93], [LOC94], has been the basis of motion planning for *Yamabico*-11 and has provided improvements in robot positioning accuracy. However, the integration of an image understanding system enhances the overall sensor capabilities of *Yamabico*-11 and is the first step towards achieving complete and/or integrated visual navigation control. Various methods for implementation of a visual navigation system utilizing a single image have been pursued by [STE92], [PET92], and [DEC93]. Another approach of incorporating stereo vision for navigation is addressed by [KRI89].

One method is to store a model line segment image in the robot and match it with the actual line segment extraction data computed via the robot image processing system. By conducting pattern matching of both images, the capabilities for robot position correction, vehicular navigation, as well as object identification and recognition exists thus enhancing the visual navigation system.

An example of a path generation [DEC93] problem can be used to execute the following algorithm:

- Snap an image while traveling along a particular path. The robot sees the actual image in Figure 34.

- Perform image analysis on the image. A model line segment image shown in Figure 35 has been downloaded into the robot and compared with the computed line segment extraction performed on the robot shown in Figure 36.

- If no object is detected, continue on the current path. Shown in Figure 36

- If an object is detected, analyze object for range and dimension information.

- Determine the 'safest' and least significant maneuver and compute the required distance to shift left or right.

- Define a new path based on the above input and transition to it or return to the original path once past the object, as detected by side-looking sonar. Shown in Figure 37.

Box object in hallway

Figure 37. Hallway image with object.

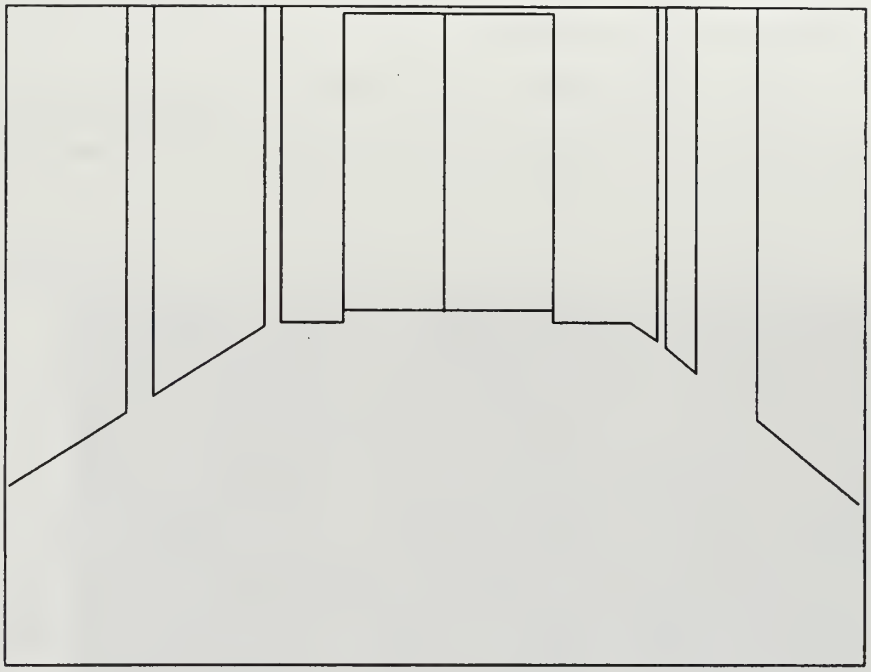

Figure 38. Line segment model data structure.

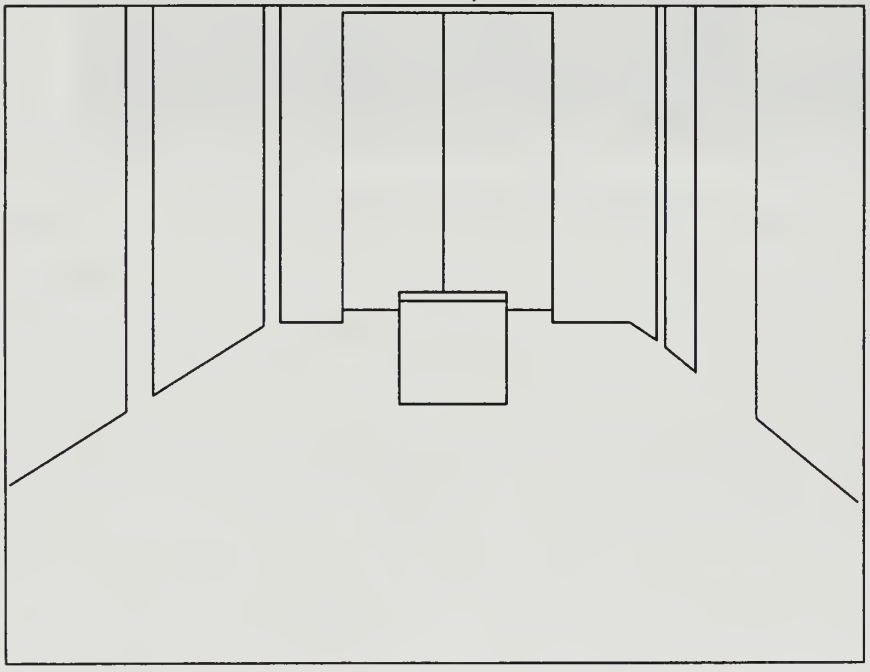

Figure 39. Model line segment extraction with box object.

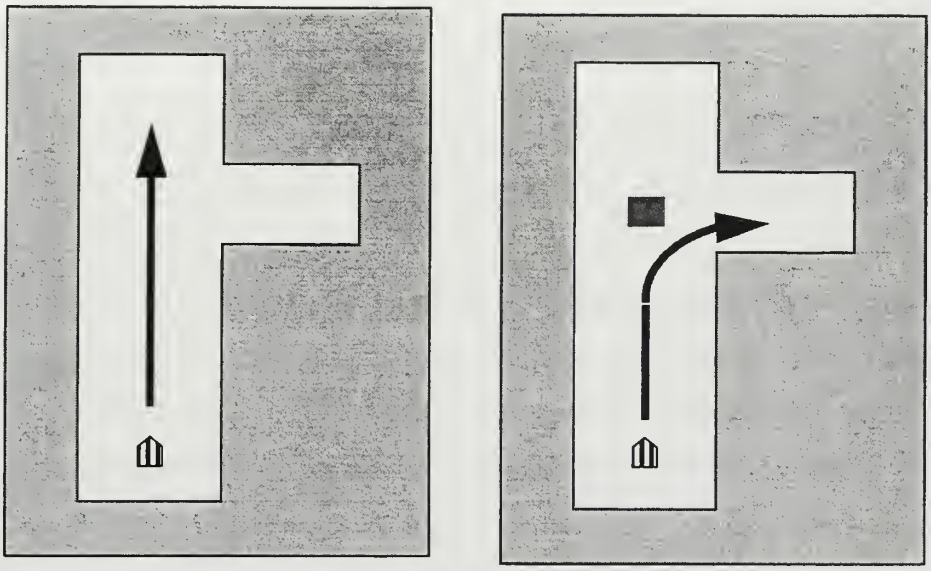

Figure 40. Yamabico-11 image motion planning.

These elements for path generation provide a richer sensing capability but may be limited due to physical hardware and software limitations of the robot. One software problem discovered by [KIS95] was the impact of the image system on the operation of other systems. The integration of the image understanding system into the motion control and sonar control could have resulted in several changes to MML but was avoided to permit development and testing of both images routines and MML separately. Other studies for combining vision with motion planning can be seen in [LUM88].

The capability of combining motion and sonar control by embedding image commands in the user.c routine and processing the image while the robot traverses its environment provides the framework for future image understanding needs for motion control.

VII. CONCLUSION

A. OVERALL RESULTS

The overall result of this research effort is an autonomous mobile robot that can now successfully capture and process an image on-board by finding straight edges in a grayscale (black and white) image.

An image log was developed to record image data and provide downloading of the image from the robot to a workstation for further processing. By analyzing the image data, previously developed Silicon Graphics image understanding routines were able to be modified for compatibility with the robot's memory structure. These routines were ported onto the robot's Single Board Super Computer System and provides *Yamabico-11* with the on-board, real-time image processing capabilities.

These features provide the foundation for future developments of a real time dynamic visual navigation system for an autonomous mobile robot.

B. FOLLOW-ON WORK

Although this research effort has improved the image processing capabilities of *Yamabico*-11, there remains much work to be done. The following are some possible areas of follow-on work.

1. Sensor Integration

The combination of sonar and vision capabilities can provide *Yamabico* with even greater information of its environment. Integrating the two sensors can improve the robot's obstacle detection, position determination, and object recognition abilities.

2. Image Processing Algorithm

In order for *Yamabico* to be successful in a dynamic environment, the vision system must provide the robot with the ability to quickly assess its surroundings and navigate with safe and smooth motions. The governing principle behind *Yamabico's* ability to react is the algorithm used to compute the edge extraction. The method used in this work for scanning an image is cumbersome and increases processing time. Other methods such as arbitrarily picking a pixel location in an image based on random sampling is currently being pursued among the vision research group. This method appears logically to be more efficient and suitable for a robot operating in a dynamic environment.

3. Embedding Image Understanding Software in MML

Embedding image understanding routines in *Yamabico's* MML system can enhance the image understanding capabilities of the robot. Newly developed path planning software may allow the robot to operate independently for longer periods of time. An interrupt driven image understanding routine could continually look for obstacles independent of the other systems.

4. Color Image Analysis

Yamabico's black and white camera was chosen to allow a grasslike computation of edge and line segment extraction routines which are more easier to compute than a colored image. However, the additional information in a colored image may increase the robot's ability to recognize edges in an image. The detection of a change in the surface's RGB value may allow the robot to find more significant edges.

APPENDIX A - TESTEDGE ROUTINES

The following routines provide implementation for the edge region extraction and line segment extraction. Given an input matrix image the algorithm determines the edge regions for the gradient image and produces the line segment extraction.The files included are the following: types.h, linesupport.h, testedge.c. They are located in the ~yamabico/vision95 directory.

```
/*********************************************************************/
/*file types.h
/*
/*This file holds the structure definitions for edge extraction and linear
/*fitting routines.
/*Types defined:EDGE_REGION_TYPE
/*               PIXEL_INFO
/*               POINT_TYPE
/*               LINE_TYPE
/*********************************************************************/

typedef struct edge_region_type

{
  int first_pixel;
  int last_pixel ;
  double avg_phi;
  double sum_phi;
  int  reg_num ;
  double m00 ;
  double m10 ;
  double m01 ;
  double m11 ;
  double m20 ;
  double m02;
  struct edge_region_type *next ;
} REGION;


 typedef struct pixel_info
 {
  double phi;
  REGION *r;
  int significant;
 } PIXEL ;

struct point_type
{

  double x,y; /* x,y coordinates of the pixel endpoints */
```

```
;
typedef struct point_type POINT;


typedef struct match_type
{

  /*  LINE *line;*/    /* ptr to LINE type (from Jim Stein's "graphics.c")
*/

  double angle_view_diff; /* angular difference between image and model
                                          lines in the image */
  float conf;     /* confidence value for the match */

 float dist;     /* distance between the *match LINE and IMG_LINE */

 float scale;    /* ratio IMG_LINE->dmajor / MODEL_LINE->length */

 struct match_type *next;

 MATCHTYPE;


typedef struct line_type
{

  char name[3];
  POINT p1, p2;            /* the 2 endpoints for the line */
  struct line_type *next; /* ptr to the next IMG_LINE in the image */

  /* Least Squares Fit momments: ---------------------------------- */
  int m00;     /* Number of pixels */
  double m10;     /* Sum x */
  double m01;     /* Sum y */
  double m11;     /* Sum x*y */
  double m20;     /* Sum x*x */
  double m02;     /* Sum y*y */
  double phi;     /* Calculated normal orientation of IMG_LINE */
  double dmajor; /* Length of major axis of equivallent ellipse */
  double dminor; /* Length of minor axis of equivallent ellipse */
  double rho;     /* Ratio dminor/dmajor */

  /* Pattern Matching Information: -------------------------------- */
  double    angle_to_image_center;
  MATCHTYPE *matchlist; /* List of matches to LINE types */
  MATCHTYPE *pm;    /* present match being considered */

} IMG_LINE;
```

```
/*************************************************************************/
/*file atan2.c
/*
/*This file provides a definition of the atan2() function.
/*************************************************************************/
#define PI 3.141592653589793

double atan2(y,x)
  double x, y;
{
  if (x > 0.0) return (arctan(y/x));
    else if ((x < 0.0) && (y > 0.0)) return (arctan(y/x)+ PI);
    else if ((x < 0.0) && (y < 0.0)) return (arctan(y/x)- PI);
    else if ((x < 0.0) && (y == 0.0)) return (PI);
    else if ((x == 0.0) && (y > 0.0)) return (PI/2.0);
    else if ((x == 0.0) && (y < 0.0)) return (-PI/2.0);
    else return (0.0);
}


/*************************************************************************/
/*FILENAME: linesupport.h
/*AUTHOR: Leonard V. Remias and Khaled Morsy
/*DATE: 01 October 1995
/*
/*DESCRIPTION: Collection of region finding functions.
/*
/*IMG_LINE *create_line (REGION *r, double M20, double M11, double M02,
/*                       double Dmajor, double Dminor, double Rho)
/*   void fatal(char message)
/*   void line_test (REGION *r)
/*************************************************************************/
int xdim;/*width of input image (nr pixels)*/
int ydim;/*width of input image (nr pixels)*/

int Linecount = 0;   /*counter for number of IMG_LINEs made*/

IMG_LINE *Line_list_head = NULL;


/*-----------------------------------------------------------------*/
/* void fatal (char message)
/*
/*   Prints error message and exits out of the program.
/*-----------------------------------------------------------------*/
fatal(message)
     char *message;
{
  fprintf(stderr, "Fatal ERROR: ");
  perror(message);
  exit(-1);       /* exit by failure */
```

```
}


/*-------------------------------------------------------------------*/
/* IMG_LINE *create_line (REGION *r, double M20, double M11, double M02,
/*                    double Dmajor, double Dminor, double Rho)
/*
/*   Returns pointer to newly instantiated IMG_LINE with variables set
/*   according to moments described by REGION r, secondary moments
/*   M20, M11, and M02, axis lengths Dmajor, Dminor, and ratio of
/*   axis lengths Rho.
/*-------------------------------------------------------------------*/
IMG_LINE *create_line(r,M20,M11,M02,Dmajor,Dminor,Rho)
REGION *r;
double M20,M11,M02,Dmajor,Dminor,Rho;

{

  IMG_LINE *l;

  long  x1, y1, x2, y2;
  double Phi, delta1, delta2;
  int negative_phi;

  /*intf("\nenter create_line");  */

  /* r->first_pixel mapped onto the IMG_LINE will be endpoint p1
     r->last_pixel mapped onto the IMG_LINE will be endpoint p2*/

         x1 = r->first_pixel % (xdim),
         y1 = r->first_pixel / (xdim),
             x2 = r->last_pixel % (xdim),
             y2 = r->last_pixel / (xdim);

 /* Calculate the normal orientation of the IMG_LINE by atan2()
function.*/

      Phi = atan2(-2*M11,M02-M20)/2.0;

  /* Delta1 and delta2 are the offsets used to calculate the endpoints
     for the IMG_LINE segment based upon values x1,y1 and x2,y2.*/


      delta1 = (r->m10/r->m00 - (double)x1)*cos(Phi) +
                          (r->m01/r->m00 - (double)y1)*sin(Phi);
      delta2 = (r->m10/r->m00 - (double)x2)*cos(Phi) +
                          (r->m01/r->m00 - (double)y2)*sin(Phi);

  /* Phi = atan2(-2*M11,M02-M20)/2.0) always returns positive result to
Phi.
```

```
    Therefore, negative_phi = (r->avg_phi < 0.0) is necessary.*/

     negative_phi = (r->avg_phi < 0.0);

  /* Allocate memory for IMG_LINE l.*/
  if((l = (IMG_LINE *)malloc(sizeof(IMG_LINE))) == NULL) {
    fatal("create_line: malloc\n");
  }

  /* Calculate x,y coordinates for endpoints p1 and p2.*/
  l->p1.x = (double)x1 + delta1*cos(Phi);
  l->p1.y = (double)y1 + delta1*sin(Phi);
  l->p2.x = (double)x2 + delta2*cos(Phi);
  l->p2.y = (double)y2 + delta2*sin(Phi);


  /* Copy least squares fit moments.*/
  l->m00 = r->m00;
  l->m10 = r->m10;
  l->m01 = r->m01;
  l->m11 = r->m11;
  l->m20 = r->m20;
  l->m02 = r->m02;


  /* Phi is positive, but -pi < r->avg_phi < pi.*/
  if(negative_phi) l->phi = -Phi;
  else l->phi = Phi;

  /* Update rest of IMG_LINE values.*/
  l->next   = NULL;
  l->dmajor = Dmajor;
  l->dminor = Dminor;
  l->rho    = Rho;

  l->angle_to_image_center = 0.0; /* default values for LINE matching*/
  l->matchlist  = NULL;
  l->pm         = NULL;

  ++Linecount;      /* Increment global variable, Linecount.*/
  strcpy(l->name,"   ");
  sprintf(l->name, "%d", Linecount);

  return(l);        /* Return IMG_LINE l.*/

}

/*

/*-------------------------------------------------------------------
/*  void write_all_lines (long x, long y)
```

```
/*
/*    Write to output file "lines.text".
/*-------------------------------------------------------------------
write_all_lines(x,y)
     long x,y;       /* the x,y dimensions of the image

{
  IMG_LINE *l = Line_list_head;
  FILE *lines_file;

  lines_file = fopen("lines.text","w");

  fprintf(lines_file,"--- DATA FOR EXTRACTED LINE SEGMENTS ---\n");
   fprintf(lines_file,"Image size: nr pixels x axis = %d, nr pixels y axis
=  %d\n",x,y);
  fprintf(lines_file,"Extracted line segments listed in order by
length.\n\n");

  printf("\nenter write all lines\n");


while(l!=NULL)
    {

      fprintf(lines_file,"%s> length = %.4f, thinness = %.4f,
orientation = %.4f, m00 = %d\n",l->name, l->dmajor, l->rho,l->phi, l-
>m00);
      fprintf(lines_file,"endpoints: (%.2f %.2f)  (%.2f %.2f)\n\n",
l->p1.x,l->p1.y,l->p2.x,l->p2.y);

      l = l->next;
    }
  fclose(lines_file);

  printf("  lines found in image written to: 'lines.text'\n");
}
*/


/*-------------------------------------------------------------------*/
/*  void write_all_lines (long x, long y)
/*
/*    Write to output file "lines.text".
/*-------------------------------------------------------------------*/
write_all_lines(x,y)
     long x,y;       /* the x,y dimensions of the image */

{
  IMG_LINE *l = Line_list_head;
  FILE *lines_file;
```

```
  lines_file = fopen("lines.text","w");

while(l!=NULL)
    {
    fprintf(lines_file,"%.2f %.2f\n%.2f %.2f\n\n",
                l->p1.x,l->p1.y,l->p2.x,l->p2.y);

      l = l->next;
    }
  fclose(lines_file);

  printf("  lines found in image written to: 'lines.text'\n");
}


#include <stdio.h>
#include <math.h>
#include "types.h"
#include "linesupport.h"

#define MAX_DELTA_PHI  deg_to_rad(15)
#define PI      3.14159265

/* Constants for function:  void line_test (REGION *r) */
#define MIN_PIXELS_PER_LINE   4   /*minimum pixels allowed for a IMG_LINE
*/
#define MIN_DMAJOR            4.0 /* minimum major axis length allowed */
#define MAX_RHO              1.0  /* maximum ratio (Rho=Dminor/Dmajor) */

/*FILE *reg_file;*/

int gl[144]=  {150,150,150,150,150,150,150,150,150,150,150,150,
               150,150,150,150,150,150,150,150,150,150,150,150,
               150,150,150,150,150,150,150,150,150,150,150,150,
               150,150,150,  0,  0,  0,  0,  0,  0,150,150,150,
               150,150,150,  0,  0,  0,  0,  0,  0,150,150,150,
               150,150,150,  0,  0,  0,  0,  0,  0,150,150,150,
               150,150,150,  0,  0,  0,  0,  0,  0,150,150,150,
               150,150,150,  0,  0,  0,  0,  0,  0,150,150,150,
               150,150,150,  0,  0,  0,  0,  0,  0,150,150,150,
               150,150,150,150,150,150,150,150,150,150,150,150,
               150,150,150,150,150,150,150,150,150,150,150,150,
               150,150,150,150,150,150,150,150,150,150,150,150};


/*int gl[144]=  {150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
```

```
                    0,  0,  0,  0,  0,  0,150,150,150,150,150,150,
                    0,  0,  0,  0,  0,  0,150,150,150,150,150,150,
                    0,  0,  0,  0,  0,  0,150,150,150,150,150,150,
                    0,  0,  0,  0,  0,  0,150,150,150,150,150,150,
                    0,  0,  0,  0,  0,  0,150,150,150,150,150,150,
                      0,  0,  0,  0,  0,  0,150,150,150,150,150,150};*/


/*int gl[144]=  {150,150,150,150,150,150,150,150,150,150,150,  0,
                 150,150,150,150,150,150,150,150,150,150,  0,  0,
                 150,150,150,150,150,150,150,150,150,  0,  0,  0,
                 150,150,150,150,150,150,150,150,  0,  0,  0,  0,
                 150,150,150,150,150,150,150,  0,  0,  0,  0,  0,
                 150,150,150,150,150,150,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,150,  0,  0,  0,  0,  0,  0,  0,
                 150,150,150,150,  0,  0,  0,  0,  0,  0,  0,  0,
                 150,150,150,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                 150,150,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                 150,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}; */
int gl[144]=  {150,150,150,150,150,150,255,255,255,255,255,255,
               150,150,150,150,150,150,255,255,255,255,255,255,
               150,150,150,150,150,150,255,255,255,255,255,255,
               150,150,150,150,150,150,255,255,255,255,255,255,
               150,150,150,150,150,150,255,255,255,255,255,255,
               150,150,150,150,150,150,255,255,255,255,255,255,
               150,150,150,150,150,  0,  0,255,255,255,255,255,
               150,150,150,150,  0,  0,  0,  0,255,255,255,255,
               150,150,150,  0,  0,  0,  0,  0,  0,255,255,255,
               150,150,  0,  0,  0,  0,  0,  0,  0,  0,255,255,
               150,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,255,
                 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0};


double dx,dy,threshold = 90;
double orientation, min;

long xdim=12;
long ydim=12;

int neighbor_included;
int i;
int region_found_on_the_row;
int neighbor_included;
int x, y;

PIXEL current[12], prev[12];

REGION *first_region ;  /* head of the linked list of regions */
REGION *last_region ;   /* last region in the list so far */
REGION *reg;
```

```
/*************************************************************************
   FUNCTION : deg_to_rad()
   PURPOSE  : convert degree to radian
*************************************************************************/

double deg_to_rad(theta)
double theta;
{
  return theta * PI/180;
}


/*************************************************************************
   FUNCTION : normalize()
   PURPOSE  : return the normalized value of orientation
*************************************************************************/

double normalize(o)
double o;
{
  int d = 0;
  d=(o+PI)/(2*PI);

  if (d>=0)
    d=d+1 ;
  o = o -( 2 *PI *(d-1)) ;
  return (o);
}

/*************************************************************************/
void update(current,x,y)
PIXEL current[];
int x,y;
{
 REGION *reg;
 double o;

 o=current[x].phi;
 reg=current[x].r;
 reg->last_pixel = i;
 reg->sum_phi+=o;
 ++reg->m00;
 reg->m10+=x;
 reg->m01+=y;
 reg->m11+=x*y;
 reg->m20+=x*x;
 reg->m02+=y*y;
 reg->avg_phi = reg->sum_phi / reg->m00;
```

```
 /*fprintf(reg_file,"\n%d %d %d",x,y,reg);*/

}

/*********************************************************************/
void compare1(current, x, y)
PIXEL current[];
int x,y;
{
  double t;
  REGION *current_region;
  t= fabs(normalize(current[x].phi-current[x-1].phi));

  if ((t<= MAX_DELTA_PHI )&&(current[x-1].significant == 1))
   {
      min = t;
      current[x].r = current[x-1].r;
      neighbor_included = 1;
   }
}


/*********************************************************************/
void compare2(current, prev, x1, x2, y)
PIXEL current[], prev[];
int x1, x2, y;
{
  double t;
  REGION *current_region;
  t=fabs(normalize(current[x1].phi - prev[x2].phi));

 if ((t<= MAX_DELTA_PHI )&&(prev[x2].significant == 1))
 {
     if (t<min)
     {
         min = t;
         current[x1].r = prev[x2].r;
             neighbor_included = 1;
     }
 }
}

/*********************************************************************/
REGION *create_region(x,y,o)
int x,y;
double o;
{

 REGION *reg;

 if((reg=(REGION *) malloc(sizeof(REGION)))== NULL)
```

```
    printf("create region");

 reg->first_pixel = i;
 reg->last_pixel =  i;
 reg->avg_phi = o;
 reg->sum_phi = o;
 reg->m00 = 1;
 reg->m10 = x;
 reg->m01 = y;
 reg->m11 = x*y;
 reg->m20 = x*x;
 reg->m02 = y*y;
 reg->next = NULL;

 if(first_region == NULL)
    first_region = reg;
 if(last_region != NULL)
    last_region->next=reg;
 last_region=reg;

 return(reg);
}

/*********************************************************************/
void pixel_membership(current, prev, x, y, o)
PIXEL current[], prev[];
int x, y;
double o;
{
  REGION *reg;

          if (x > 1)
              {
            compare1(current, x, y);
                  if (y > 1)
                 compare2(current, prev, x, x-1, y);
          }
           if (y > 1)
               {
            compare2(current, prev, x, x, y);
            compare2(current, prev, x, x+1, y);
           }

        if (neighbor_included==0)
        {
          reg =  create_region(x,y,o);

              /* fprintf(reg_file,"\n%d %d %d", x, y, reg);*/
          current[x].r = reg;
        }
             if (neighbor_included ==1)
```

```
          update(current,x,y);
}

/*-----------------------------------------------------------------*/
/* void line_test (REGION *r)
/*
/*   Determines if REGION r meets three requirements to be a IMG_LINE:
/*     (1) The number of pixels in REGION r (r->m00) be greater than
/*         MIN_PIXELS_PER_LINE.
/*     (2) The ratio (Rho) of the length of major and minor axes of the
/*         REGION be less than MAX_RHO.
/*     (3) The length of the major axis (Dmajor) be greater than
/*         MIN_DMAJOR, the minimum IMG_LINE length allowed.
/*   If all three conditions are met, a new IMG_LINE type is created and
/*   appended to the Line_list in order of significance (in this case
/*   Dmajor).
/*-----------------------------------------------------------------*/
line_test(r)

  REGION *r;
{

  IMG_LINE *l, *insert_pt = Line_list_head;
  double M20,M11,M02;
  double Ma,Mb,Mmajor,Mminor,Dmajor,Dminor,Rho;

  /* First test -- A IMG_LINE must have a required minimun number of
pixels.*/

  if(r->m00 > MIN_PIXELS_PER_LINE)
  {

      /* Calculate secondary moments by least squares fit.*/
      M20 = r->m20 - ((r->m10*r->m10)/r->m00);
      M11 = r->m11 - ((r->m10*r->m01)/r->m00);
      M02 = r->m02 - ((r->m01*r->m01)/r->m00);

      /* Calculate major and minor axis lengths, Dmajor and Dminor.*/
      Ma = (M20+M02)/2.0;
      Mb = sqrt( ((M02-M20)*(M02-M20)/4.0) + (M11*M11) );
      Mmajor = Ma - Mb;
      Mminor = Ma + Mb;
      Dmajor = 4.0*sqrt(Mminor/r->m00);
      Dminor = 4.0*sqrt(Mmajor/r->m00);

      /* Calculate ratio Rho.*/
      Rho = Dminor/Dmajor;

      /* Second & Third tests
          -- Ratio Rho must represent a line, not a blob.
          -- IMG_LINE must be at least a certain length.*/
```

```
      if((Rho < MAX_RHO) && (Dmajor > MIN_DMAJOR))
      {
          /* The REGION passed the three requirments to be a line.*/

              l = create_line(r,M20,M11,M02,Dmajor,Dminor,Rho);


              /* Add new IMG_LINE to IMG_LINE list in order by IMG_LINE
length, dmajor.*/
              if (Line_list_head == NULL)
              {
                 Line_list_head = l;

              }
              else if (l->dmajor > Line_list_head->dmajor)
              {
             l->next = Line_list_head;
                 Line_list_head = l;
              }
              else
              {
                while((insert_pt->next != NULL) &&
                          (l->dmajor < insert_pt->next->dmajor))
                {
                           insert_pt = insert_pt->next;
                }
                   l->next = insert_pt->next;
                   insert_pt->next = l;
              }

            } /* end if second and third tests*/
    }     /* end if first test */

  }


/*********************************************************************/
/* main function */
/*********************************************************************/
main()
{
   PIXEL current[12], prev[12];
   int UL,U,UR,L,R,DL,D,DR,c;
   /*   reg_file = fopen("regions.txt","w");*/


   for (y=1; y < ydim ; ++y)
   {
     for (x=1; x < xdim ; ++x)
```

```
  {
    i = (y * xdim) + x;
    neighbor_included =0;
    min = PI;

    UL=i+xdim-1; U = UL+1 ; UR = U+1;
    L= i-1 ; R = i+1;
    DL=i-xdim-1 ; D=DL+1; DR = D+1;

   if(x == xdim-1)
   {
       for (c=0; c<xdim-1; c++)
           {
         prev[c]=current[c];
         current[c].phi=0.0;
         current[c].r = NULL;
         current[c].significant = 0;
       }
   }

     if((x != xdim-1) && (y != ydim-1))

         {

           /* calculate dx,dy via sobel operator for pixel i */
           dx = -gl[UL] + gl[UR]
                -2*gl[L] + 2*gl[R]
                -gl[DL] + gl[DR];

           dy = gl[UL]+2*gl[U]+gl[UR]
               -gl[DL]-2*gl[D]-gl[DR];

          if((dx*dx)+(dy*dy) > threshold)
              {
              orientation = atan2(dy,dx);
                      current[x].phi = orientation;
              current[x].significant = 1;
                  pixel_membership(current,prev,x,y,orientation);
              }
     }
  }
}

/*     fclose(reg_file);*/

   /* Check remaining REGIONs for lines: */

    reg = first_region;

    while(reg != NULL)
    {
```

```
        line_test(reg);
        reg = reg->next;
       }

    /* Write the lines list to file "lines.text". */

         write_all_lines(xdim, ydim);
          printf("\nLines found in image written to:'lines.text'\n");

}
```

APPENDIX B - FINDEDGE ROUTINES

```
/*********************************************************************/
/*   FILENAME:findedge.c
/*   AUTHOR: Leonard V. Remias and Khaled Morsy
/*           some fragments from Peterson
/*   DATE: 18 January 1996
/*
/*DESCRIPTION: An image gradient program incorporating edge-finding.
/*Displays edge-gradient image and associated lines.
/*
/*This application is designed for use on a Silicon-/*Graphics Iris (r)
/*workstation utilizing a.sgi or similar rgb formatted image.
/*RGB values are of type LONG in the form AABBGGRR where:
/*               AA - alpha value, 0-255
/*               BB - blue component, 0-255
/*               GG - green component, 0-255
/*               RR - red component, 0-255
/*
/*SiliconGraphics graphics library functions used within the
/* display_bw_and_gradient_images() routine:
/*               qdevice(), winset(), c3f(), move2(), draw2(),
/*               swapbuffers(), reshapeviewport(), winclose(),
/*               and lrectwrite().
/*
/*NPSIMAGE function routines borrowed courtesy of M.Zyda:
/*read_sgi_rgbimage(), get_empty_rgba_npsimage(),
/*get_empty_rgb_npsimage(), and rgbalong_to_bwlong().
/*
/*Least Squares Fit method for line-finding from
/*"Sonar Data Interpretaion for Autonomous Mobile Robots"
/*by Y.Kanayama, T.Noguchi, & B.Hartman, 1990.
/*********************************************************************/
#include <gl.h>               /* SiliconGraphics (r) graphic library   */
#include <gl/image.h>       /* SGI image structure library           */
#include <device.h>          /* Machine-dependent device library      */
                             /* for keys and mouse-buttons            */
#include <stdio.h>          /* C standard i/o library                */
#include <math.h>           /* C math library for atan2()            */

#include "image_types.h"      /* Type definitions for NPSIMAGE, etc.   */
#include "edge_types.h"       /* Type definitions for EDGE, LINE, etc  */
#include "npsimagesupport.h" /* Some NPSIMAGE functions               */
#include "edgesupport.h"      /* EDGE and IMG_LINE building functions  */
#include "displaysupport.h"   /* Graphics display functions            */
```

```
int neighbor_included;

main(argc, argv)
     int argc;
     char *argv[];
{
  NPSIMAGE *img1, /* input file_name.rgb color image */
           *img2, /* black&white image */
           *img3; /* gradient image */

/* pointers to RGBA longs ("bitsptr"s) of respective NPSIMAGEs */
long     *ptr1, *ptr2, *ptr3;
double    dx, dy, Th = THRESHOLD*THRESHOLD;
int z = 0;  /*counter for pixels in gradient image */
REGION *reg;
PIXEL current[646],prev[646];
double orientation;
int UL,U,UR,L,R,DL,D,DR,c;
int i;
long     x, y;
long  ptr4[313956];  /*646 x 486 */

reg_file = fopen("regions.txt","w");

  if(argc != 2) fatal("usage: findedge filename\n");

  /* Read in input rgb image */
  img1 = read_sgi_rgbimage(argv[1]);
  if(img1 ==(NPSIMAGE *) NULL)
    {
      fatal("File %s is a NULL image.\n",img1->name);
    }
   Xdim = img1->xsize;                /* else set global Xdim and ptr1 */
  Ydim = img1->ysize ;

  ptr1 = img1->imgdata.bitsptr;
 /* printf("findedge:> %s xsize= %d ysize= %d pixels= %d\n",
             img1->name,img1->xsize,img1->ysize, (img1->xsize*img1-
>ysize));*/

   /* Declare new NPSIMAGEs */
  if((img1->type == RGBAWITHALPHA) || (img1->type == RGBA))
    {
     img2 = get_empty_rgba_npsimage(Xdim,img1->ysize,img1->name);
      img3 = get_empty_rgba_npsimage((Xdim-2),img1->ysize-2,"findedge");
    }
  else
    {
      fatal("Unknown or c-mapped image type: %d.\n",img1->type);
    }
  ptr2 = img2->imgdata.bitsptr;
```

```
ptr3 = img3->imgdata.bitsptr;

/* The scan of an RGB image is from the bottom row -> up,
   traversing the rows left to right. */

/* In order for the Sobel operator to be calculated for a specific pixel,
    all eight surronding pixels must have an absolute (black & white)
    light intensity calculated.  Function rgbalong_to_bwlong() performs
   this task. */

/* Due to the nature of the Sobel operator, the pixels in the top and
    bottom rows as well as pixels in the leftmost and rightmost columns
    of the input image will not be calculated.  In order to start cal-
   culating the Sobel operators, the first 2 rows of the input image
  must be converted to black & white light intensity values. */

/* Calculate bw values for the entire input image. */

for(i=0; i< ((Xdim * Ydim)-1);++i)
  {
    rgbalong_to_bwlong( ptr1[i],&ptr2[i]);
             ptr4[i] = gray;
  }

 for (y=0; y < Ydim; ++y)
 {
   for (x=0; x < Xdim; ++x)
   {
          i = (y * Xdim) + x;
          neighbor_included = 0;
          min = PI;

          UL=i+(Xdim-1); U = UL+1 ; UR = U+1;
          L= i-1 ; R = i+1;
          DL=i-(Xdim-1); D=DL+1; DR = D+1;

          if(x == (Xdim-1))
         {
            for (c=0; c<(Xdim-1); c++)
            {
             prev[c]=current[c];
             current[c].phi=0.0;
             current[c].r = NULL;
             current[c].significant = 0;
           }
   }

          if((y == 0) || (y == (Ydim-1)) || (x == 0) || (x == (Xdim-1)))
         {
             set_pixel_black(&ptr3[i]);
         }
```

```
            else
            {
            /* Calculate dx,dy via Sobel operator for pixel i. */

               dx = - (ptr4[i+(Xdim-1)]) + (ptr4[i+(Xdim+1)])
                    -  2 *( ptr4[i-1])+ 2 *(ptr4[i+1])
                    - (ptr4[i-Xdim-1]) + (ptr4[i-Xdim+1]);
                   dy =  (ptr4[i+Xdim-1]) + 2*(ptr4[i+Xdim])
                        +(ptr4[i+Xdim+1]) -(ptr4[i-Xdim-1])
                        - 2*(ptr4[i-Xdim]) - (ptr4[i-Xdim+1]);

                if( ((dx*dx)+(dy*dy)) > Th)
                {
                        orientation = atan2(dy,dx);
                        current[x].phi = orientation;
                        current[x].significant = 1;
                        pixel_membership(current,prev,x,y,orientation);
                        set_pixel_black(&ptr3[z]);
                }
               else
               {
                        set_pixel_white(&ptr3[z]);
                }

                        ++z;
                }
    } /* endfor x */
  } /* end for y */

  fclose(reg_file);

    reg = first_region;

            while (reg != NULL)
    {
            reg->m10,reg->m01,reg->m11,reg->m20,reg->m02);*/
            line_test(reg);
            reg = reg->next;
    }

  /* Write the lines list to file "lines.text". */
  write_all_lines(argv[1],img3->xsize,img3->ysize);
  printf("\nNumber lines found in %s = %d", argv[1],Linecount);
  printf("\nNumber regions found in %s = %d\n", argv[1],reg_count);

  /* Display the black&white and gradient images on the screen. */
  display_bw_and_gradient_images(img2,img3,Line_list_head);
  display_line_image(img2,Line_list_head);

  printf("\nfindedge %s...done.\n",argv[1]);
}
```

```
/*************************************************************************/
/*file edge_types.h
/*
/*This file holds the structure definitions for edge extraction and linear
/*fitting routines.
/*Types defined:EDGE_REGION_TYPE
/*               PIXEL_INFO
/*               POINT_TYPE
/*               LINE_TYPE
/*************************************************************************/

struct point_type
{
  double x,y; /* x,y coordinates of the pixel endpoints */
};
typedef struct point_type POINT;



typedef struct line_type
{
  char name[3];  /* for troubleshooting */

  POINT p1, p2;  /* the 2 endpoints for the line */

  /* Least Squares Fit momments: */
  double m00;    /* Number of pixels should be long*/
  double m10;    /* Sum x */
  double m01;    /* Sum y */
  double m11;    /* Sum x*y */
  double m20;    /* Sum x*x */
  double m02;    /* Sum y*y */
  double phi;    /* Calculated normal orientation of IMG_LINE */
  double dmajor; /* Length of major axis of equivallent ellipse */
  double dminor; /* Length of minor axis of equivallent ellipse */
  double rho;    /* Ratio dminor/dmajor */

  /* Pattern Matching Information: ------------------------------- */
  double    angle_to_image_center;

  int *matchlist;/* Bogus pointers for IMG_LINE *create_line(EDGE *r,...)
*/
                 /* in 'edgesupport.c'. */
  int *pm;       /* These pointers are required for matching and are
                           declared as (MATCHTYPE *)s in 'match_types.h'
*/

  struct line_type *next; /* ptr to the next IMG_LINE in the image */

} IMG_LINE;
```

```
typedef struct edge_region_type
{
  long   first_pixel;
  long   last_pixel ;
  double avg_phi;
  double sum_phi;
  double  m00 ;
  double  m10 ;
  double  m01 ;
  double  m11 ;
  double  m20 ;
  double  m02;
  struct edge_region_type *next ;
} REGION;


 typedef struct pixel_info
 {
  double phi;
  REGION *r;
  int significant;
  } PIXEL ;


/*************************************************************************/
/*   FILENAME: edgesupport.h
/*   AUTHOR: Leonard V. Remias and Khaled Morsy
/*           (some parts from original version by Peterson)
/*   DATE: 19 January 1996
/*         with new changes
/*DESCRIPTION: Collection of edge finding functions.
/*
/*   void fatal(char message)
/*   int gradient_angles_close(double r,double s)
/*   int close_to_negative_pi(double phi)
/*   int horizontal(EDGE *r)
/*   REGION *create_region(int x, int y, double o)
/*   void add_pixel_to_edge (long z, double phi, EDGE *r)
/*   EDGE *combine_edges(EDGE *r1, EDGE *r2)
/*  IMG_LINE *create_line (REGION *r, double M20, double M11, double M02,
/*                         double Dmajor, double Dminor, double Rho)
/*   void line_test (REGION *r)
/*   void check_active_edges ()
/*   void rgbalong_to_bwlong (long rgbalong, long *bwlong)
/*   void pixel_membership (long z, double phi)
/*   void set_pixel_white (long *rgbalong)
/*   void set_pixel_black (long *rgbalong)
/*   void write_all_lines (long x, long y)
/*   IMG_LINE *fastlines (NPSIMAGE *img)
/*************************************************************************/
```

```
#define THRESHOLD 70.0

/* Gradient Angular Orientation */
/*#define MAX_DELTA_PHI  deg_to_rad(15) */
#define MAX_DELTA_PHI (PI*15/180)
#define PI                    3.14159265

/* Constants for function:  void line_test (REGION *r) */

#define MIN_PIXELS_PER_LINE  7  /* was 10 minimum pixels allowed for a
IMG_LINE */
#define MIN_DMAJOR           7.0  /*was 10.0 * minimum major axis length
allowed */
#define MAX_RHO             1.0   /* maximum ratio (Rho=Dminor/Dmajor) */


/* --- Global variables ----------------------------------------------
*/
 FILE *reg_file;
 long  Xdim;                /* width of input image (nr pixels) */
 long  Ydim;                /* width of input image (nr pixels */
 int Linecount = 0;           /* counter for number of IMG_LINEs made  */
 int reg_count = 0;         /* counter for number of regions */
 int    gray ;              /* added by khaled 1-8-95 */


/* Pointers to: first region , last region and IMG_LINE list*/

REGION *first_region=NULL;
REGION *last_region=NULL;
IMG_LINE   *Line_list_head = NULL;

int neighbor_included;
double min;


/*********************************************************************
   FUNCTION : deg_to_rad()
   PURPOSE  : convert degree to radian
*********************************************************************/

double deg_to_rad(theta)
double theta;
{
  return theta * PI/180;
}
```

```
/*********************************************************************
   FUNCTION : normalize()
   PURPOSE  : return the normalized value of orientation
*********************************************************************/
double normalize(o)
double o;
{ double h=o;
  int d = 0;
  if(o>=-PI && o <PI) return(o);

  d=(o+PI)/(2*PI)  ;
  if (d>=0 && o > 0.0)
    d=d+1 ;
  o = o -( 2 *PI *(d-1)) ;

/*  while(o < -PI)
     o = o + PI + PI;
  while(o >= PI)
     o = o-PI-PI;*/

  return (o);
}


void update(current,x,y)
PIXEL current[];
long x,y;
{
 REGION *reg;
 double o;

 o=current[x].phi;
 reg=current[x].r;
 reg->last_pixel = (Xdim * y + x);
 reg->sum_phi+=o;
 ++reg->m00;
 reg->m10  += x;
 reg->m01  += y;
 reg->m11  += x*y;
 reg->m20  += x*x;
 reg->m02  += y*y;
 reg->avg_phi = (reg->sum_phi / reg->m00);
}
```

```
/*************************************************************************/
void compare1(current,x,y)
PIXEL current[];
 long x,y;
{
  double t;
  REGION *current_region;
  t= fabs(normalize(current[x].phi-current[x-1].phi));

  if ((t<= MAX_DELTA_PHI )&&(current[x-1].significant == 1))
   {


     min = t;
     current[x].r = current[x-1].r;
    /*current[x].region_number = current[x-1].region_number ;*/
     neighbor_included = 1;
     }
}


/*************************************************************************/
void compare2(current,prev,x1,x2,y)
PIXEL current[],prev[];
long x1,x2,y;
{
  double t;
  REGION *current_region;
  t=fabs(normalize(current[x1].phi - prev[x2].phi));

 if ((t<= MAX_DELTA_PHI )&&(prev[x2].significant == 1))
   {
     if (t<min)
       {
            min = t;
            current[x1].r = prev[x2].r;
            /*current[x1].region_number = prev[x2].region_number ;*/
            neighbor_included = 1;
       }
   }
}
```

```
/*********************************************************************/
REGION *create_region(x, y, o)

long x,y;
double o;
{
 REGION *reg;

 if((reg=(REGION *) malloc(sizeof(REGION)))==NULL)
  {
    printf("create edge");
  }
  ++reg_count;
 reg->first_pixel =(Xdim * y + x); if( reg->first_pixel < 0)
                        printf("ERROR");
 reg->last_pixel = (Xdim * y + x);
 reg->avg_phi = o;
 reg->sum_phi = o;
 reg->m00 = 1.0;
 reg->m10 = x;
 reg->m01 = y;
 reg->m11 = x*y;
 reg->m20 = x*x;
 reg->m02 = y*y;
 reg->next = NULL;

  if (first_region == NULL)
      first_region = reg;
  if (last_region != NULL)
      last_region->next = reg;
  last_region=reg;

 return(reg);
}


/*********************************************************************/
void pixel_membership(current,prev,x,y,o)
PIXEL current[],prev[];


 long x,y;
 double o;
{
  REGION *reg;

          if (x > 1)
            {
            compare1(current,x,y);
                        if (y > 1)
                        {
```

```
                                    compare2(current,prev,x,x-1,y);
                        }
                }
                if (y > 1)
                {
                compare2(current,prev,x,x,y);
                compare2(current,prev,x,x+1,y);
               }

        if (neighbor_included == 0)
        {
            reg =  create_region(x,y,o);
            fprintf(reg_file,"\n%d %d %u", x, y, reg);
            current[x].r = reg;
        }
                if (neighbor_included == 1)
                {
                update(current,x,y);
                }
}


/*-----------------------------------------------------------------*/
/* void fatal (char message)
/*
/*   Prints error message and exits out of the program.
/*-----------------------------------------------------------------*/
fatal(message)
     char *message;
{
  fprintf(stderr, "Fatal ERROR: ");
  perror(message);
  exit(-1);       /* exit by failure */
}


/*-----------------------------------------------------------------*/
/* IMG_LINE *create_line (REGION *r, double M20, double M11, double M02,
/*                        double Dmajor, double Dminor, double Rho)
/*
/*   Returns pointer to newly instantiated IMG_LINE with variables set
/*   according to moments described by REGION r, secondary moments
/*   M20, M11, and M02, axis lengths Dmajor, Dminor, and ratio of
/*   axis lengths Rho.
/*-----------------------------------------------------------------*/
IMG_LINE *create_line(r,M20,M11,M02,Dmajor,Dminor,Rho)
     REGION *r;
     double M20,M11,M02,Dmajor,Dminor,Rho;
{
  IMG_LINE   *l;
```

```
  long x1,  y1,  x2,  y2;
  double Phi,  delta1,  delta2;
  int negative_phi;

  /* r->first_pixel mapped onto the IMG_LINE will be endpoint p1
     r->last_pixel mapped onto the IMG_LINE will be endpoint p2 */


        x1 = r->first_pixel%(Xdim),
        y1 = r->first_pixel/(Xdim),
        x2 = r->last_pixel%(Xdim),
        y2 = r->last_pixel/(Xdim);

  /* Calculate the normal orientation of the IMG_LINE by atan2() function.
*/

        Phi = atan2(-2*M11,M02-M20)/2.0;

  /* Delta1 and delta2 are the offsets used to calculate the endpoints
     for the IMG_LINE segment based upon values x1,y1 and x2,y2. */

        delta1 = (r->m10/r->m00 - (double)x1)*fcos(Phi) +
                          (r->m01/r->m00 - (double)y1)*fsin(Phi);
        delta2 = (r->m10/r->m00 - (double)x2)*fcos(Phi) +
                          (r->m01/r->m00 - (double)y2)*fsin(Phi);

  /* Phi = atan2(-2*M11,M02-M20)/2.0) always returns positive result to
Phi.
     Therefore, negative_phi = (r->avg_phi < 0.0) is necessary. */

        negative_phi = (r->avg_phi < 0.0);

  /* Allocate memory for IMG_LINE l. */

  if((l = (IMG_LINE *)malloc(sizeof(IMG_LINE))) == NULL) {
    fatal("create_line: malloc\n");
  }

  /* Calculate x,y coordinates for endpoints p1 and p2. */
  l->p1.x = (double)x1 + delta1*fcos(Phi);
  l->p1.y = (double)y1 + delta1*fsin(Phi);
  l->p2.x = (double)x2 + delta2*fcos(Phi);
  l->p2.y = (double)y2 + delta2*fsin(Phi);

  /* Copy least squares fit moments. */
  l->m00 = r->m00;
  l->m10 = r->m10;
  l->m01 = r->m01;
  l->m11 = r->m11;
  l->m20 = r->m20;
  l->m02 = r->m02;
```

```
  /* Phi is positive, but -pi < r->avg_phi < pi. */

  if(negative_phi) l->phi = -Phi;
  else l->phi = Phi;

  /* Update rest of IMG_LINE values. */
  l->next   = NULL;
  l->dmajor = Dmajor;
  l->dminor = Dminor;
  l->rho    = Rho;
  l->angle_to_image_center = 0.0; /* default values for LINE matching */
  l->matchlist  = NULL;
  l->pm         = NULL;

  ++Linecount;      /* Increment global variable, Linecount. */
  strcpy(l->name,"   ");
  sprintf(l->name, "%d", Linecount);

  return(l);        /* Return IMG_LINE l. */
}


/*-------------------------------------------------------------------*/
/* void line_test (REGION *r)
/*
/*   Determines if REGION r meets three requirements to be an IMG_LINE:
/*     (1) The number of pixels in REGION r (r->m00) be greater than
/*         MIN_PIXELS_PER_LINE.
/*     (2) The ratio (Rho) of the length of major and minor axes of the
/*         EDGE be less than MAX_RHO.
/*     (3) The length of the major axis (Dmajor) be greater than
/*         MIN_DMAJOR, the minimum IMG_LINE length allowed.
/*   If all three conditions are met, a new IMG_LINE type is created and
/*   appended to the Line_list in order of significance (in this case
/*   Dmajor).
/*-------------------------------------------------------------------*/
line_test(r)
  REGION *r;
{
  IMG_LINE   *l, *insert_pt = Line_list_head;
  double M20,M11,M02;
  double Ma,Mb,Mmajor,Mminor,Dmajor,Dminor,Rho;


/*********************************************************************/
/* First test -- A IMG_LINE must have a required minimun number of pixels.
*/
/*********************************************************************/

  if(r->m00 > MIN_PIXELS_PER_LINE)
```

```
{
    /* Calculate secondary moments by least squares fit.*/
    M20 = r->m20 - ((r->m10*r->m10)/r->m00);
    M11 = r->m11 - ((r->m10*r->m01)/r->m00);
    M02 = r->m02 - ((r->m01*r->m01)/r->m00);

    /* Calculate major and minor axis lengths, Dmajor and Dminor.*/
    Ma = (M20+M02)/2.0;
    Mb = sqrt( ((M02-M20)*(M02-M20)/4.0) + (M11*M11) );
    Mmajor = Ma - Mb;
    Mminor = Ma + Mb;
    Dmajor = 4.0*sqrt(Mminor/r->m00);
    Dminor = 4.0*sqrt(Mmajor/r->m00);


    /* Calculate ratio Rho.*/
    Rho = Dminor/Dmajor;

  /* Second & Third tests -- Ratio Rho must represent a line, not a blob.
                    -- IMG_LINE must be at least a certain length.  */



 if((Rho < MAX_RHO) && (Dmajor > MIN_DMAJOR))
 {
            /* The REGION passed the three requirments to be a line.*/

             l = create_line(r,M20,M11,M02,Dmajor,Dminor,Rho);

            /* Add new IMG_LINE to IMG_LINE list in order by IMG_LINE
length, dmajor. */
            if(Line_list_head == NULL)
            {
               Line_list_head = l;
 }
            else if(l->dmajor > Line_list_head->dmajor)
            {
      l->next = Line_list_head;
               Line_list_head = l;
            }
            else
            {
              while((insert_pt->next != NULL) &&
                            (l->dmajor < insert_pt->next->dmajor))
              {
                            insert_pt = insert_pt->next;
              }
                   l->next = insert_pt->next;
                   insert_pt->next = l;
            }
```

```
            }   /* end if second and third tests*/
  }     /* end if first test*/
}



/*-----------------------------------------------------------------*/
/* void rgbalong_to_bwlong (long rgbalong, long *bwlong)
/*
/*   Converts color to black/white for rgba formatted pixels.
/*   The weights assigned for each color are television standards.
/*   This function courtesy of M. Zyda.
/*-----------------------------------------------------------------*/

rgbalong_to_bwlong(rgbalong,bwlong)

             long rgbalong;  /* input color rgbalong */
     long *bwlong;   /* output b/w rgbalong  */

{
  unsigned char red, green, blue, alpha;
  unsigned bw;

  /* Use bit masks to get RGB and alpha values from input rgbalong. */
  red   =  rgbalong & 0x000000ff;
  green = (rgbalong & 0x0000ff00) >> 8;
  blue  = (rgbalong & 0x00ff0000) >> 16;
  alpha = (rgbalong & 0xff000000) >> 24;
/*  if ( i== 4000  || i== 5000) printf("\n%d  :alpha = %d" , i, alpha);*/
  /* Calculate the black&white intesity using NTSC standard.
     intensity = 0.299(red) + 0.587(green) + 0.114(blue)        */
  bw = (0.299*red) + (0.587*green) + (0.114*blue);

  /* Save the black&white intensity in bwlong. */
  *bwlong = (alpha<<24)|(bw<<16)|(bw<<8)|bw;
  gray=bw;


}


/*-----------------------------------------------------------------*/
/* void set_pixel_white (long *rgbalong)
/*
/*   Sets the specified long integer pointed to by rgbalong to be
/*   white by setting all RGB bits to ff (255 dec -> max intensity).
/*-----------------------------------------------------------------*/
set_pixel_white(rgbalong)
     long *rgbalong;
{
```

```
  *rgbalong = 0xffffffff;
}


/*-----------------------------------------------------------------*/
/* void set_pixel_black (long *rgbalong)
/*
/*   Sets the specified long integer pointed to by rgbalong to be
/*   black by setting all RGB bits to 00 (0 dec -> min intensity).
/*-----------------------------------------------------------------*/
set_pixel_black(rgbalong)
     long *rgbalong;
{
  *rgbalong = 0xff000000;
}


/*-----------------------------------------------------------------*/
/*  void write_all_lines (long x, long y)
/*
/*    Write to output file "name.text".
/*-----------------------------------------------------------------*/
write_all_lines(x,y)
     long x,y;       /* the x,y dimensions of the image */
{
  IMG_LINE *l = Line_list_head;
  FILE *lines_file;

  lines_file = fopen("lines.text","w");
  fprintf(lines_file,"--- DATA FOR EXTRACTED LINE SEGMENTS ---\n");
   fprintf(lines_file,"Image size: nr pixels x axis = %d, nr pixels y axis
=  %d\n",x,y);
  fprintf(lines_file,"Extracted line segments listed in order by
length.\n\n");

  while(l!=NULL)
    {
     fprintf(lines_file,"%s> length = %.4f, thinness = %.4f, orientation
= %.4f, m00 = %d\n",
                 l->name, l->dmajor, l->rho, l->phi, l->m00);

     fprintf(lines_file,"endpoints: (%.2f %.2f)  (%.2f %.2f)\n\n",
                 l->p1.x,l->p1.y,l->p2.x,l->p2.y);
      l = l->next;
    }
 /* fclose(lines_file);*/

  printf("\nLines found in image written to: 'lines.text'");
}
```

APPENDIX C - IMAGE UNDERSTANDING ROUTINES

The following routines provide implementation for the imageTest.c routine.
The files included are the following:
user.h, displayCtrl.h, displayCtrl.c, acquisitionCtrl.h,
acquisitionCtrl.c, imsControl.h, imsControl.c, imagelog.h, imagelog.c.
They are located in the ~yamabico/vision96 directory.

```
/*************************************************************/
}
FILENAME:          imageTest.c
DESCRIPTION:       This file contains image test routines
REVISION HISTORY: Leonard V. Remias and Khaled Morsy, Winter 96'
                   some fragments from Kisor
*************************************************************/

/*****************************
To shorten the descriptions below several abbreviations are used:
AM: Acquisition module
DM: Display module

A0: Frame A0
A1: Frame A1
B1: Frame B1
Acq: acquisition (as in waitAcq, waitAquisition)
IMS Ref Man: Imaging Technology Inc's IMS (Standard Image Manager)
               reference Manual for series 150/40
*****************************/

#include "user.h"
#include "displayCtrl.h"
#include "acquisitionCtrl.h"
#include "imsControl.h"


/**** Local Prototypes *****/

int
getChoice(void);
/* Gets the menu choice that is desired
*/

void
interlacePage2(void);
/* Copies the page 2 part of an interlaced image into between the lines
** of page one of the interlaced image to make a truely interlaced picture
*/
```

```
void
copyImageToMemory(IMSPage frameNumber);


void
copyImageToFrame(IMSPage frameNumber);



int
user(){  /* lvr test program for grabbing/snapping an image */


BOOLEAN readyToExit = FALSE; /* The exit flag */


   while (!readyToExit)
      switch (getChoice()) {
         case 1: /* Start grabbing images */
            setInputPath(A1, CAMERA);
            setInputPath(B1, CAMERA);
            setInputPath(A0, CAMERA);
          puts("Input paths set: Camera->A0, Camera->A1 and CAMERA->B1");

            setFrameAcquire(A0, GRAB);
            setFrameAcquire(A1, GRAB);
            setFrameAcquire(B1, GRAB);
            puts("Set up to grab images into frames A0, A1 and B1");

            cycleThroughLEDs();
            acqEnable();
            puts("Frame has been grabbed");
            break;

          case 2: /* Stop grabbing images */
            setInputPath(A1, CAMERA);
            setInputPath(B1, CAMERA);
            setInputPath(A0, CAMERA);
          puts("Input paths set: Camera->A0, Camera->A1 and CAMERA->B1");

            setFrameAcquire(A0, FREEZE);
            setFrameAcquire(A1, FREEZE);
            setFrameAcquire(B1, FREEZE);
            puts("Stop grabbing images into frames A0, A1 and B1");
            break;

          case 3: /*Snap an image */
                setInputPath(A1, CAMERA);
            setInputPath(B1, CAMERA);
                setInputPath(A0, CAMERA);
          puts("Input paths set: Camera->A0, Camera->A1 and CAMERA->B1");
```

```
            setFrameAcquire(A0, SNAP);
          setFrameAcquire(A1, SNAP);
            setFrameAcquire(B1, SNAP);
          puts("Set up to Snap an image into frames A0, A1 and B1");

          cycleThroughLEDs();
          acqEnable();
          puts("Frame has been snapped");
          break;

           case 4: /* Zeroize (clear) pixel data in all frames B1 */
          setInputPath(A1, CONSTANT);
          setInputPath(B1, CONSTANT);
          setInputPath(A0, CONSTANT);
         puts("Input paths set: Constant->A0, Constant->A1 & Constant-
>B1");

          setFrameAcquire(A0, SNAP);
          setFrameAcquire(A1, SNAP);
          setFrameAcquire(B1, SNAP);
          puts("Set up to Snap an image into frames A0, A1 and B1");

          cycleThroughLEDs();
          acqEnable();
          puts("Frame has been snapped");
          break;

           case 5: /* Zeroize (clear) 1 MByte frame page A0 */
          interlacePage2();
          break;


           case 6: /* Zeroize (clear) 1 MByte frame page A0 */
          setFirstKToConstant(A0, 0x0000);
          break;

           case 7: /* Zeroize (clear) 1 MByte frame page A1 */
          setFirstKToConstant(A1, 0x0000);
          break;

           case 8: /* Zeroize (clear) 1 MByte frame page B1 */
          setFirstKToConstant(B1, 0x0000);
          break;

       case 9:
          selectPage(A0);
          puts("Frame A0 selected");
          readyToExit = TRUE;
          break;
```

```
            case 10:
               selectPage(A1);
               puts("Frame A1 selected");
               readyToExit = TRUE;
               break;

            case 11:
               selectPage(B1);
               puts("Frame B1 selected");
               readyToExit = TRUE;
               break;

            case 12:
               selectPage(AM);
               puts("Acquisition module selected");
               readyToExit = TRUE;
               break;

            case 13:
               selectPage(DM);
               puts("Display module selected");
               readyToExit = TRUE;
               break;

            case 14:
                   ImageLog(NULL,0);
               copyImageToMemory(B1);
               readyToExit = TRUE;
               break;

            case 15:
               copyImageToFrame(B1);
               break;

            default:
               puts("Error:main - Illegal option selected");
               rexit();
      }
}


int
getChoice(void)
/* Gets the menu choice that is desired
*/ {

   puts("\n\nWhat would you like to see?");
   printf("\n Enter  1 Start grabbing images.");
   printf("\n Enter  2 Stop  grabbing images.");
   printf("\n Enter  3 Snapshot (snap an image)");
   printf("\n Enter  4 Zeroize (clear) pixel data in all frames.");
```

```
   printf("\n Enter  5 Interlace the image.");
   printf("\n Enter  6 Zeroize (clear) 1 MByte frame page A0");
   printf("\n Enter  7 Zeroize (clear) 1 MByte frame page A1");
   printf("\n Enter  8 Zeroize (clear) 1 MByte frame page B1");
   printf("\n Enter  9 Exit with frame A0 selected");
   printf("\n Enter 10 Exit with frame A1 selected");
   printf("\n Enter 11 Exit with frame B1 selected");
   printf("\n Enter 12 Exit with Acquisition Module registers selected");
   printf("\n Enter 13 Exit with Display Module registers selected");
   printf("\n Enter 14 Copy image from IMS frame B1 to Sparc Memory");
   printf("\n Enter 15 Copy image from Sparc memory to IMS frame B1 ");

   printf("\n\n The choice is : ");
   return( GetInt() );


 }


void
interlacePage2(void)
/* Copies the page 2 part of an interlaced image into between the lines
** of page one of the interlaced image to make a truely interlaced picture
*/ {
int i, j;
unsigned long page1Index = 0xfa000000; /* First empty line for pg2 pixels
*/
unsigned long page2Index = 0xfa07a400; /* First 1K of pixels on page 2*/
unsigned long page1Destination = 0xfa000800;
unsigned long page2Source = 0xfa07a400;

   selectPage(B1);

   for (i=0; i<236; i++) {
printf("The indexes are now %x and %x.\n",page1Index, page2Index);
      for (j=0; j<366; j++) { /*732 bytes but 2 bytes per access */
         *(WORD*)page1Destination = *(WORD*)page2Source;
         page1Destination += 2;
         page2Source += 2;
      }
      page1Index += 0x800;
      page2Index += 0x800;
      page1Destination = page1Index;
      page2Source = page2Index;

   }


}
```

```
YAMAIMAGE b1Image;

BYTE image[476][732];

void
copyImageToMemory(IMSPage frameNumber)
/*  Put comments here
*/
{

unsigned long page1Index = 0xfa000000; /* First empty line for pg2 pixels
*/
unsigned long page2Index = 0xfa07a400; /* First 1K of pixels on page 2*/
unsigned long page1Source = 0xfa000000;
unsigned long page2Source = 0xfa07a400;


int i,j;

int xSize = 732;
int ySize = 476;


   b1Image.xSize = xSize;
   b1Image.ySize = ySize;

   printf("IM IN copyImageToMemory function\n");

   for(i=0; i<238; i+=2)
   {
      for (j=0; j<366; j+=2) /*732 bytes but 2 bytes per access*/
      {

            b1Image.image[i][j] = *(BYTE*)page1Source;
            page1Source += 2;

            b1Image.image[i+1][j] = *(BYTE*)page2Source;
            page1Source += 2;

      }

      page1Index += 0x800;
      page2Index += 0x800;
      page1Source = page1Index;
      page2Source = page2Index;

   }
         LogImage(b1Image);
}
```

```
/***********
static int GetWheelEncoder(LONG HighWordAddress, LONG LowWordAddress)
{
   LONG Wheel;

   Wheel = *(WORD*)HighWordAddress;
   Wheel = Wheel << 16;
   Wheel += *(WORD*)LowWordAddress;

   return Wheel;
}

*********/

void
copyImageToFrame(IMSPage frameNumber)
/*  Put comments here
*/ {

unsigned long page1Index = 0xfa000000; /* First empty line for pg2 pixels
*/
unsigned long page2Index = 0xfa07a400; /* First 1K of pixels on page 2*/
unsigned long page1Source = 0xfa000000;
unsigned long page2Source = 0xfa07a400;

int errorAddress = 0;

int i, j;


   for(i=0; i<476; i++) {
      for (j=0; j<732; j++) { /*732 bytes but 2 bytes per access */
             *(BYTE*)page1Source = b1Image.image[i][j];
            page1Source++;

      }
      page1Index += 0x400;
      page1Source = page1Index;
   }
   if (errorAddress)
      printf("There are differences in the frame and sparc memory!\n");
}
```

```
/*************************************************************************/
/*file user.h
/*This files includes the various files for integrating the image routines
with MML and for setting up the system to upload/download image data
/*********************************************************/
#include <math.h>
#include "definitions.h"
#include "stdiosys.h"
#include "serial.h"
#include "trace.h"
#include "geometry.h"
#include "time.h"
#include "seqcmd.h"
#include "system.h"
#include "memsys.h"
#include "imagelog.h"

/*************************************************************
FILENAME:          displayCtrl.h
DESCRIPTION:       This file contains prototypes for the low
                   level image routines that manipulate the
                   display module registers.
REVISION HISTORY:  jck: LT John Kisor USN Winter 95'
jck 950110
*************************************************************/
/***Notes******
940112 Find out how much of Sparc's memory is currently used.
***************/

typedef enum    { NI800by600,   /* 800x600 non-interlaced */
                  NI640by480,   /* 640x480 non-interlaced */
                  NI1024by768,  /* 1024x768 non-interlaced */
                  RS170,        /* RS170   512x480 interlaced */
                  CCIR,         /* CCIR    512x512 interlaced */
                  RS170SQ,      /* RS170sq 640x480 interlaced */
                  CCIRSQ1,      /* CCIRsq  768x512 interlaced */
                  CCIRSQ2,      /* CCIRsq  768x574 interlaced */
                  OneKbyOneK    /* 1024x1024 interlaced */
                } MONITORS;


void
initDisplay(MONITORS monitorType);
/* Configure the display module to display images on the
** selected monitor. See DM reference manual
*/


void
resetDMCtrl(void);
/* resets the communication between the host and the GSP
```

```
*/


void
initGSP(void);
/* write causes the GSP to enter a reset state. Reading causes the GSP to
** exit the reset state See sect 3.1.5 of the DM Ref Man
** Show Mike this!!!
*/

int
checkDMId(void);
/* checks that the correct version of the display module is present.
** See sect 3.1.6 of the DM Ref Man
*/


void
selectMonitor(MONITORS monitorType);
/* Initializes the display modules GSP for the monitor specified
** See sect 3.2.3 of the DM Ref Man
*/


void
writeDACLUT(void);
/* Writes a one to one coorespondence LUT for the output of RGB values
** An input of 0 sets R:0 G:0 & B:0, an input of 1 sets R:1, G:1 & B:1
etc...
**See sect 3.3.2, 3.3.3 and 3.3.4 of the DM Ref man.
*/



void
writeOverlay(void);
/* Sets al overlay color mappings to zero (clear)
** No overlays will be shown only the image.
**See sect 3.3.6, 3.3.7 and 3.3.8 of the DM Ref man.
*/




/************************************************************
FILENAME:          displayCtrl.c
DESCRIPTION:       This file contains the code to produce output from the
                   the ITI display module
REVISION HISTORY:    jck: LT John Kisor USN Winter 95'
jck 950110
************************************************************/
```

```
/*****************************
jck 950111
To shorten the descriptions below several abbreviations are used:
AM: Acquisition module
DM: Display module

A0: Frame A0
A1: Frame A1
B1: Frame B1
Acq: acquisition (as in waitAcq, waitAquisition)
IMS Ref Man: Imaging Technology Inc's IMS (Standard Image Manager)
              reference Manual for series 150/40
DM Ref Man: Imaging Technology INc's DM-PC Hardware Reference Manual
*****************************/



#include "definitions.h"
#include "system.h"
#include "stdiosys.h"

#include "imsControl.h"
#include "displayCtrl.h"


/* Local prototypes */

void setGSPAddress(WORD MSBofGSPAddress, WORD LSBofGSPAddress);
/* Sets up the address of that GSP that will either be written or read
** See sect 3.1 of the DM Ref Man
*/

void
writeToGSP(WORD gspData);
/* Writes some data to a GSP address
** See  section 3.1.1, 3.1.2 and 3.1.3 of DM Ref man
*/

WORD
readFromGSP(void);
/* Reads some data from a GSP address
** See  section 3.1.1, 3.1.2 and 3.1.3 of DM Ref man
*/

WORD
getGSPAddress(void);
/* Test routine to see what values are being set into the DACLUT
** write address
*/
```

```
void
zeroizeGSPMemory(void);
/* Zeroizes the program and overlay memory in the GSP
** See sect 3.2.1 and 3.2.2 in the DM Ref Man.
*/

void
initDisplay(MONITORS monitorType)
/* Configure the display module to display images on the
** selected monitor. See DM reference manual
*/ {

   cycleThroughLEDs();
   selectPage(DM);            /* Select the display  module */
   puts("Display Module Registers Selected");

   if (checkDMId())
      puts("Correct DM present");
   else
      puts("ERROR-- incorrect DM present");

   resetDMCtrl();
   puts("Communication between the host and GSP reset");

   initGSP();
   puts("Resets the GSP state");

   cycleThroughLEDs();
   selectMonitor( monitorType );

   writeDACLUT();
   puts("Writing to the DACs LUT");

   writeOverlay();
   puts("Writing zeros to overlay");

}


void
resetDMCtrl(void)
/* resets the communication between the host and the GSP
*/ {
const dmCtrlReg = 0xfa000006;

const zero = 0x0000;

   *(WORD*)dmCtrlReg = zero;
}
```

```
void
initGSP(void)
/* write causes the GSP to enter a reset state. Reading causes the GSP to
** exit the reset state See sect 3.1.5 of the DM Ref Man
** Show Mike this!!!
*/ {
const displayControl = 0xfa000006;
const GSPResetReg    = 0xfa000008;

const resetZero = 0x0000;
const setIncrements = 0x0000; /* NO increment after write */
const cacheFlush = 0x4000;    /* flush the old program from the cache */

WORD  contents;

   *(WORD*)displayControl = ( setIncrements | cacheFlush );

   *(WORD*)GSPResetReg = resetZero;
   contents = *(WORD*)GSPResetReg;

   zeroizeGSPMemory();

}


int
checkDMId(void)
/* checks that the correct version of the display module is present.
** See sect 3.1.6 of the DM Ref Man
*/ {

const modVersion = 0xfa00000c;

const correctVersion = 0xfff5;

int status;


   if ( *(WORD*)modVersion & correctVersion){
       puts("Correct display module installed for cohu4110");
       status = 1;
     }
   else {
       puts("Error - checkDMId():Incorrect display module installed");
       printf("%d does not equal %d\n", *(WORD*)modVersion,
correctVersion);
       status =0;
```

```
    }


   return status;
}



void
selectMonitor(MONITORS monitorType)
/* Initializes the display modules GSP for the monitor specified
** See sect 3.2.3 of the DM Ref Man
*/ {

const lowAddrReg  = 0xfa000000;
const highAddrReg = 0xfa000002;
const dataPortReg = 0xfa000004;

 WORD offset[12] = {0x0080, 0x00b0, 0x0090, 0x0000,
         0x0010, 0x0020, 0x0030, 0x0040, 0x0050, 0x0060, 0x0070, 0x0000};

int i;

WORD monitorInfo[1][12] = {
   {0xf010, 0x00cc, 0xc00c, /* 800x600 non-interlaced */
    0x0005, 0x0017, 0x007d, 0x0085, 0x000e, 0x0014, 0x026d, 0x0270,
0x0007}};

/********
   {0xf010, 0x00cc, 0xc00c, /* 640x480 non-interlaced
   0x0005, 0x0012, 0x0064, 0x006a, 0x000e, 0x0015, 0x01f6, 0x01f9, 0x002d}

   {0xf010, 0x00cc, 0xc00c, /* 1024x768 non-interlaced
    0x0005, 0x001e, 0x00a0, 0x00a7, 0x0000, 0x0014, 0x0315, 0x0318,
0x000b}};

   {0x9020, 0x00cc, 0xc00c,  /* RS170
    0x0005, 0x0005, 0x0047, 0x004f, 0x0003, 0x0011, 0x0102, 0x0106,
0x0247},

   {0x9020, 0x00cc, 0xc00c,  /* CCIR
    0x0009, 0x0006, 0x0048, 0x0055, 0x0001, 0x0014, 0x0115, 0x0153,
0x0a47},

   {0x9020, 0x00cc, 0xc00c, /* RS170sq
    0x0014, 0x0008, 0x005a, 0x006f, 0x0003, 0x0012, 0x0103, 0x0260,
0x064d},

   {0x9020, 0x00cc, 0xc00c, /* CCIRsq1 NO value given for [11]!!!!!!
   0x0009, 0x000d, 0x006f, 0x007b, 0x0001, 0x0013, 0x0134, 0x014c, 0xe5c},
```

```
   {0x9020, 0x00cc, 0xc00c, /* CCIRsq2
    0x0009, 0x000d, 0x006f, 0x007b, 0x0001, 0x0013, 0x0134, 0x014c,
0x0e5c},

   {0xb020, 0x00cc, 0xc00c, /* 1024x1024
    0x000d, 0x001d, 0x009f, 0x00a9, 0x0007, 0x0024, 0x0225, 0x022b,
0x010b}};
******/
   printf("Monitor 0 (1024x768) values are:\n (");
   for (i=0; i<12; i++)
      printf("%x, ",monitorInfo[0][i] );
   printf(")\n");

   for(i=0; i<11; i++){
      *(WORD*)lowAddrReg  = offset[i];        /* offset in GSP address
space */
     *(WORD*)highAddrReg = 0xc000;   /* high order word for GSP internal
Reg*/
      *(WORD*)dataPortReg =  monitorInfo[0][i];
      printf("Writing to:%x%x the value:%x\n" ,*(WORD*)highAddrReg,
                                               *(WORD*)lowAddrReg,
                                               *(WORD*)dataPortReg);
   }

   *(WORD*)lowAddrReg  = 0x0000;  /*0ffset for control register, in GSP
space*/
   *(WORD*)highAddrReg = 0x0100;    /* high order word for "local"
registers */
   *(WORD*)dataPortReg =  monitorInfo[0][11];
     printf("Writing to:%x%x the value:%x\n", *(WORD*)highAddrReg,
                                              *(WORD*)lowAddrReg,
                                              *(WORD*)dataPortReg);
}



void setGSPAddress(WORD MSBofGSPAddress, WORD LSBofGSPAddress)
/* Sets up the address of that GSP that will either be written or read
** See sect 3.1 of the DM Ref Man
*/ {

const lowAddrReg  = 0xfa000000;
const highAddrReg = 0xfa000002;

   *(WORD*)lowAddrReg  = LSBofGSPAddress;
   *(WORD*)highAddrReg = MSBofGSPAddress;
}
```

```
void
writeToGSP(WORD gspData)
/* Writes some data to a GSP address
** See  section 3.1.1, 3.1.2 and 3.1.3 of DM Ref man
*/ {

const dataPortReg = 0xfa000004;

   *(WORD*)dataPortReg = gspData;

}



WORD
readFromGSP(void)
/* Reads some data from a GSP address
** See  section 3.1.1, 3.1.2 and 3.1.3 of DM Ref man
*/ {

const dataPortReg = 0xfa000004;

WORD gspData;

   gspData = *(WORD*)dataPortReg;

   return gspData;
}



WORD
getGSPAddress(void)
/* Test routine to see what values are being set into the DACLUT
** write address
*/ {
   setGSPAddress(0x0100, 0x0080);
   return( readFromGSP() );
 }




void
writeDACLUT(void)
/* Writes a one to one coorespondence LUT for the output of RGB values
** An input of 0 sets R:0 G:0 & B:0, an input of 1 sets R:1, G:1 & B:1
```

```
etc...
**See sect 3.3.2, 3.3.3 and 3.3.4 of the DM Ref man.
*/ {

WORD i, z;
WORD LUTValue;

   setGSPAddress(0x0100, 0x00a0);  /* Address of mask register */
   writeToGSP(0xffff); /* Set to NO masking in DAC */


   setGSPAddress(0x0100, 0x0080);  /* GSP DAC LUT write address - 2 */
   writeToGSP(0x00ff);

   setGSPAddress(0x0100, 0x0090);
/*   readFromGSP();    /* Read RED value */
/*  readFromGSP();   /* Read GREEN value, now LUT has pre-read next RGB */

   for(i=0; i<256; i++) /* For each pixel grayscale value */
      for(z=0; z<3; z++){  /* for the red, green and blue DACs */
         writeToGSP(i);
         /* Address is self incrementing after 3 writes (RG and B DACs)*/
       }


/********* Debugging code from here down, remove later jck ****/
/*      This code prints out the values that are stored in the DAC LUT */
   setGSPAddress(0x0100, 0x00b0);
   writeToGSP(0x00ff);

   setGSPAddress(0x0100, 0x0090);

   printf("TESTING:Values of GSP DAC LUT are.....\n");
   for(i=0; i<256; i++)
      for(z=0; z<3; z++) {
          LUTValue = (readFromGSP() & 0x00ff);
         printf("%d, ",LUTValue);
            }

   printf("\nEnd of LUT values\n");
/*****************************/
}




void
zeroizeGSPMemory(void)
```

```
/* Zeroizes the program and overlay memory in the GSP
** See sect 3.2.1 and 3.2.2 in the DM Ref Man.
*/ {

const displayControl = 0xfa000006;
const incrementOnRW =     0x1800;   /* Auto increment on reads and writes
*/
const noIncrementOnRW =    0xe7ff;

WORD oldDisplayControl;
WORD i, j, z;


  oldDisplayControl = *(WORD*)displayControl;   /* turn on auto increment
*/
   *(WORD*)displayControl = ( oldDisplayControl | incrementOnRW );

   printf("Zeroizing GSP program and overlay memory");
   setGSPAddress(0x3800, 0x0000);
   for(z=0; z<8; z++){             /* from 38,00000 to 3f,fffff  */
      printf(".");
      for(i=0; i<1024; i++)        /* 1 meg (1024 1Ks) */
        for(j=0; j<1024; j++)    /*3800000 -> 3bffffff is program memory*/
          writeToGSP(0x0000);   /*3c00000 -> 3cffffff is overlay memory */
    }
    printf("\n");

  oldDisplayControl = *(WORD*)displayControl; /* turn off auto increment
*/
   *(WORD*)displayControl = ( oldDisplayControl & noIncrementOnRW );

}




void
writeOverlay(void)
/* Sets al overlay color mappings to zero (clear)
** No overlays will be shown only the image.
**See sect 3.3.6, 3.3.7 and 3.3.8 of the DM Ref man.
*/ {

WORD i, z;
WORD OverlayValue;

   setGSPAddress(0x0100, 0x00bf);  /* GSP DAC overlay write address */
   writeToGSP(0x00ff); /* Start at zero minus one */
```

```
   setGSPAddress(0x0100, 0x00d0);  /* GSP DAC overlay data address */
   readFromGSP();    /* Read RED value */
   readFromGSP();    /* Read GREEN value, now LUT has pre-read next RGB */

   for(i=0; i<256; i++) /* For each pixel grayscale value */
      for(z=0; z<3; z++){  /* for the red, green and blue DACs */
         writeToGSP(0x0000);
         /* Address is self incrementing after 3 writes (RG and B DACs)*/
       }


/********* Debugging code from here down, remove later jck ****/
/*      This code prints out the values that are stored in the DAC LUT */
   setGSPAddress(0x0100, 0x00f0);
   writeToGSP(0x00ff);

   setGSPAddress(0x0100, 0x00d0);

   printf("TESTING:Values of GSP DAC Overlay are.....\n");
   for(i=0; i<256; i++)
      for(z=0; z<3; z++) {
         OverlayValue = (readFromGSP() & 0x00ff);
         printf("%d, ",OverlayValue);
            }

   printf("\nEnd of Overlay values\n");


/*************************************************************
FILENAME:          acqTest.h
DESCRIPTION:       This file contains prototypes for the low
                  level routines for the setting up the acquisition module
REVISION HISTORY:  jck: LT John Kisor USN Winter 95'
jck 950110
*************************************************************/



int checkAMVer(void);
/* Checks for the correct acquistion module version (i.e. 8 bit RS422)
** See Section 3.2 in the AM Ref Man
*/


void
initAcqModule(void);
/* Initializes the aquisition module
*/


void
```

```
setCameraClockCtrl(void);
/* Sets the camera output clock frequency to the CCLK pins on the camera
** See Sect 3.3 in the AM Ref Man.
*/


void
setCameraTimingCtrl(void);
/* Sets up the timing interface to the camera
** See 3.4 of the AM Ref Man
*/


void
setExtSyncCtrl(void);
/* sets the EXSYNC frequency, length and polarity
** See sect 3.5 in AM Ref man
*/


void
setSyncTime(void);
/* sets the duration of the EXSYNC pulse output
** Ser Sect 3.6 in AM Ref Man
*/


void
setLutCtrl(void);
/* sets the page size and output data path
** See sect 3.7 of AM Ref Man
*/


void
setHorizOffset(void);
/* Sets the location of the first valid pixel of each line relative
** to the selected edge of the LEN input
*/


void
setHorizActive(void);
/* sets the number of valid pixels in each line. When horizontal offset
** reaches zero the AM enables the horizontal counter and tarts loading
** the line FIFOs. See sect 3.10 in AM Ref Man
*/


void
setVertOffset(void);
```

```
/* Sets the location of the first valid line of each frame relative to the
** selected edge of FEN input.When the AM offset counter reaches zero, the
** AM signals the mother board that the first line is ready.
** See sect 3.11 in the AM Ref Man
*/


void
setVertActive(void);
/* Sets the number of valid lines in each frame. When the vertical offset
** counter reaches zero, the AM starts unloading lines from the FIFO to the
** mother board. See sect 3.12 AM Ref Man
*/


void
initAMTrigger(void);
/* initializes the AM trigger register. See sect 3.13 in AM Ref Man
*/


int
enableAMTrigger(void);
/* Enables the AM trigger mode. This bit is cleared when a trigger occurs.
** It cannot be set while a trigger cycle is in progress.
** See sect 3.13.1 in AM Ref Man
*/


int
enableSoftTrigger(void);
/* Enables the use of a software trigger as opposed to a trigger from the
** camera. See Sect 3.13 in AM Ref man.
*/


void softTrigger(void);
/* Causes an external trigger pulse to the AM, if the software trigger is
** enabled. See sect 3.15 in AM Ref Man.
*/




/************************************************************
FILENAME:          acquisitionCtrl.c
DESCRIPTION:       This file contains the image code for the acquistion
module
REVISION HISTORY:    jck: LT John Kisor USN Winter 95'
jck 9501101
************************************************************/
```

```
/******jck Notes*************
FAX941028 refers to a fax received from ITI regarding the acquistion
registers.
    Most of the values for these routines were obtained from that fax.

*****************************/
/*****************************
jck 950111
To shorten the descriptions below several abbreviations are used:
AM: Acquisition module
DM: Display module

A0: Frame A0
A1: Frame A1
B1: Frame B1
Acq: acquisition (as in waitAcq, waitAquisition)
IMS Ref Man: Imaging Technology Inc's IMS (Standard Image Manager)
              reference Manual for series 150/40
AM Ref Man: Imaging Technology Inc's AM-DIG Hardware reference manual
*****************************/

#include "definitions.h"
#include "system.h"
#include "stdiosys.h"

#include "acquisitionCtrl.h"
#include "imsControl.h"


void
initAcqModule(void)
/* Initializes the aquisition module
*/ {

   selectPage(AM);            /* Select the acquistion module */
   puts("Acquisition Module Registers Selected");

   if (checkAMVer())
      puts("Correct AM present");
   else
      puts("ERROR-- incorrect AM present");

   setCameraClockCtrl();
   setCameraTimingCtrl();
   setExtSyncCtrl();
   setSyncTime();
   setLutCtrl();
   setHorizOffset();
   setHorizActive();
   setVertOffset();
```

```
   setVertActive();
   initAMTrigger();

/**** Want trigger to come from camera ********
   enableAMTrigger();
   enableSoftTrigger();
   softTrigger();
**********/
}
```

```
int checkAMVer(void)
/* Checks for the correct acquistion module version (i.e. 8 bit RS422)
** See Section 3.2 in the AM Ref Man
*/ {

const modVersion = 0xfa000002;

const correctVersion = 0xff03;

int status;

   if (  *(WORD*)modVersion & correctVersion){
       puts("Correct acquistion module installed for cohu4110");
       status = 1;
     }
   else {
       puts("Error - checkModuleVer():Incorrect acquistion module
installed");
       printf("%d does not equal %d\n", *(WORD*)modVersion,
correctVersion);
       status =0;
     }
   return status;
}
```

```
void
setCameraClockCtrl(void)
/* Sets the camera output clock frequency to the CCLK pins on the camera
** See Sect 3.3 in the AM Ref Man.
*/ {

const cohuClockCtrl = 0xfa000004;
```

```
const clockFreq    = 0x0000; /* set internal clock frequency to 14.32 MHz
*/
const clockDivisor = 0x0000; /* Divide that clock frequency by one */
const xillixMode   = 0x0000; /* not a Xillix camera */
const modeCtrl     = 0x0e00; /* Mode control bits SeeFAX941028*/


   *(WORD*)cohuClockCtrl = (clockFreq    |
                            clockDivisor |
                            xillixMode    |
                            modeCtrl);

}



void
setCameraTimingCtrl(void)
/* Sets up the timing interface to the camera
** See 3.4 of the AM Ref Man
*/ {

const cohuTimeCtrl = 0xfa000006;

const pixelClkPol    = 0x0001; /* Sample data on falling edge of PCLK */
const lineEnablePol  = 0x0002; /* Falling edge of LEN enables horiz
timing*/
const frameEnablePol = 0x0000; /* Falling edge of FEN enables vertical
timing*/
/*jck!!! There is a conflict here. The fax token is "FEN_NoInvert" this
     implies the rising edge of FEN will enable vertical timing, but the
     cooresponding comment reads://Falling Edge of FEN enables Vertical
Timing.
         See sections 2.2.1.2 and section 3.4.3 in AM Ref Man. */
const fieldPolarity  = 0x0000; /* Field input lwo defines even field */
const scanMode       = 0x0000; /* Area scan mode */
const interlaceMode  = 0x0020; /* Interlaced camera */
const LENSmall      = 0x0000; /* Normal, LEN inactive period is greater */
                              /* than one PCLK cycle jck???*/
const NoLineMiss     = 0x0000; /* Two lines inserted between line scan
frames*/
const lineVariable   = 0x0200; /* Reset FIFO every FEN */

   *(WORD*)cohuTimeCtrl = (pixelClkPol   |
                           lineEnablePol |
                           frameEnablePol|
                           fieldPolarity |
                           scanMode      |
                           interlaceMode |
                           LENSmall      |
```

```
                              NoLineMiss    |
                              lineVariable  );

}


void
setExtSyncCtrl(void)
/* sets the EXSYNC frequency, length and polarity
** See sect 3.5 in AM Ref man
*/ {

const cohuSyncCntrl = 0xfa000008;

const syncFreq    = 0x0005; /* ext clk frequency: 14.32/2048=7Khz
(143usec)*/
const syncMode    = 0x0000; /* ExSync triggers itself jck??? poss 0x0010*/
const syncEnable  = 0x0000; /* EXSYNC counter disabled */
const syncPol     = 0x0040; /* EXSYNC high when triggered,....*/
const intEnable   = 0x0000; /* programmable integration disabled jck??? */
const intPulsePol = 0x0100; /* PRI disabled - output always low */

   *(WORD*)cohuSyncCntrl = (syncFreq    |
                            syncMode    |
                            syncEnable  |
                            syncPol     |
                            intEnable   |
                            intPulsePol );
}


void
setSyncTime(void)
/* sets the duration of the EXSYNC pulse output
** Ser Sect 3.6 in AM Ref Man
*/ {

const cohuSyncTime = 0xfa00000a;

const hz2046 = 0x07fb;

   *(WORD*)cohuSyncTime = hz2046;

}




void
setLutCtrl(void)
/* sets the page size and output data path
```

```
** See sect 3.7 of AM Ref Man
*/ {

const cohuLutCtrl = 0xfa00000c;

const pixelSize  = 0x0000;
const muxSetting = 0x0000;

   *(WORD*)cohuLutCtrl = (pixelSize | muxSetting);

}


void
setHorizOffset(void)
/* Sets the location of the first valid pixel of each line relative
** to the selected edge of the LEN input
*/ {

const horizOffset = 0xfa000010;

const cohuOffset = 0x009a;

   *(WORD*)horizOffset = cohuOffset;

}


void
setHorizActive(void)
/* sets the number of valid pixels in each line. When horizontal offset
** reaches zero the AM enables the horizontal counter and tarts loading
** the line FIFOs. See sect 3.10 in AM Ref Man
*/ {

const horizActive = 0xfa000012;

const cohuHorizActive = 0x02d8;

  *(WORD*)horizActive = cohuHorizActive;

}



void
setVertOffset(void)
/* Sets the location of the first valid line of each frame relative to the
** selected edge of FEN input.When the AM offset counter reaches zero, the
** AM signals the mother board that the first line is ready.
** See sect 3.11 in the AM Ref Man
```

```
*/ {

const vertOffset = 0xfa000014;

const cohuVertOffset = 0x0001;


   *(WORD*)vertOffset = cohuVertOffset;
}


void
setVertActive(void)
/* Sets the number of valid lines in each frame. When the vertical offset
** counter reaches zero, the AM starts unloading lines from the FIFO to the
** mother board. See sect 3.12 AM Ref Man
*/ {

const vertActive = 0xfa000016;

const cohuVertActive = 0x01dc;

   *(WORD*)vertActive = cohuVertActive;
}


void
initAMTrigger(void)
/* initializes the AM trigger register. See sect 3.13 in AM Ref Man
*/ {

const triggerCtrl = 0xfa000018;

const disableTrigger = 0x0000;/*trigger enabled*/
const triggerSource  = 0x0000;/*External trigger from camera connector,
                                trigger on falling edge */

   *(WORD*)triggerCtrl =( disableTrigger | triggerSource);

}




int
enableAMTrigger(void)
/* Enables the AM trigger mode. This bit is cleared when a trigger occurs.
** It cannot be set while a trigger cycle is in progress.
** See sect 3.13.1 in AM Ref Man
*/ {
```

```
const triggerCtrl = 0xfa000018;

const enableTrigger = 0x0001; /* Trigger enabled */
const triggerNotActive = 0x0004; /*read-o, 1:No ext. trigger cycle in
progress*/

int status;

   if ( ( *(WORD*)triggerCtrl & triggerNotActive)){/*trigger not in
progress*/
      *(WORD*)triggerCtrl = enableTrigger;
      status = 1;
    }
   else
      status = 0;

   return status;
}


int
enableSoftTrigger(void)
/* Enables the use of a software trigger as opposed to a trigger from the
** camera. See Sect 3.13 in AM Ref man.
*/ {

const triggerCtrl = 0xfa000018;

const enableTrigger = 0x0001; /*Trigger enabled */
const triggerSource = 0x0002; /* Trigger comes from software trig.
register */
const triggerNotActive = 0x0004; /*read-only,1:No ext.trigger cycle in
progress*/

int status;

   if ( ( *(WORD*)triggerCtrl & triggerNotActive)){/* trigger not in
progress */
      *(WORD*)triggerCtrl = (enableTrigger | triggerSource);
      status = 1;
    }
   else
      status = 0;

   return status;

}



void
```

```
softTrigger(void)
/* Causes an external trigger pulse to the AM, if the software trigger is
** enabled. See sect 3.15 in AM Ref Man.
*/ {

const softwareTrigger = 0xfa00001c;

const goEnable = 0x0000;

   *(WORD*)softwareTrigger = goEnable;
}


/*************************************************************
FILENAME:          imsSystem.h
DESCRIPTION:       This file contains prototypes for the low
                   level image routines that interface with
                   the IMS and its modules
REVISION HISTORY:  jck: LT John Kisor USN Winter 95'
jck 950110
*************************************************************/
/***Notes******
940111 Have kelbe check code for style
940112 Have to use constants instead of defines for clear/setMask masks
940112 Find out how much of Sparc's memory is currently used.
***************/

typedef enum   {     A0,    /* image frame A0 */
                     A1,    /* image frame A1 */
                     B1,    /* image frame B1 */
                     AM,    /* acquistion module */
                     DM     /* display module */
               } IMSPage;


typedef enum   {     CAMERA,     /* Input will be the camera */
                     A0ZERO,    /* Input to A0 will be all zeros */
                  CONSTANT,/* input will be value set in constant mask */
                     A0ONES, /* input to A0 will be all ones */
                     INA0,     /* input will be image in frame A0 */
                     INA1,    /* input will be image in frame A1 */
                     INB1    /* input will be image in frame B1 */
               } INPUTPath;

typedef enum   {     GRAB,      /* Continuously grab an image */
                     SNAP,     /* Snap only one image */
                     FREEZE   /* Stop grabbing images */
               } ACQUIREType;


void
```

```
cycleThroughLEDs(void);
/* Visual positive reenforcement. Cycles through all the LEDS once
*/

void
setFirstKToConstant(IMSPage pageNumber, WORD filler);
/* Puts a selected bit pattern in the first 1K of a page so that
** the difference between the pattern and image will be more noticable.
*/


void
initIMS(void);
/* Initializes the IMS board to standard addressing w/1MB page size and
** offset of 0x00 from the default address that is the beginning of
** A24 space.(i.e. 0xfa000000) Then it selects the acquistion register
** using the page register
*/


void
clearMask(IMSPage frameNum);
/* Sets the mask to allow ALL bits to pass into the frame specified
** See IMS Ref man 3.3 and setMask
*/


void
setMask(IMSPage frameNum);
/* Sets the mask to allow NO bits to pass into the frame specified
** See IMS Ref man 3.3 and setMask
*/

void
setAOIControl(void);
/* Sets the video bus out of AOI mode.
** See sect 3.9.4 in IMS Ref man
*/

void
selectPage(IMSPage pageNumber);
/* Selects between the different image frames and modules on the IMS board
*/

void
setIMSControl(void);
/* Sets the IMS control register. See section 3.11 of the IMS hardware Ref
man
*/
```

```
void
setFrameIRQ(void);
/* The type of frame interrupt will be put here.
** Currently not used.
*/


void resetFrameCtrl(IMSPage frameNum);
/* Resets the controls for horizontal pitch, tiling and write zoom for
** frames A0, A1 and B1. See IMS Ref man 3.29, 3.34
*/


void
resetFramePanScroll(IMSPage frameNum);
/* Resets a frame for no (zero) pan and scroll. See 3.21 of the IMS ref man
*/

void
setXPortSwitch(void);
/* Sets the cross-port switch for the application bus width and direction
** This register must be set prior to setting any path registers
** See AM Ref man sect 3.43
*/

void
setInputPath(IMSPage selectedFrame, INPUTPath imageSource);
/* Selects the source of the image that will be put into a particular
** frame. See AM Ref Man 3.40, 3.41 and 3.42
*/

void
setFrameAcquire(IMSPage frameNum, ACQUIREType typeOfAcquire);
/* controls the acquistion to frame A0, B1 or A1
** The input cross port switch should be set prior to calling this func.
** Decision was made to have actual acquistions triggered by the
** acquisition enable register (3.19 Ref man)
*/



void
acqEnable(void);
/* This register controls the synchronous acquisition into the IMS
** frame stores (see IMS ref man 3.19) This enables acquisition into the
** frame stores when an acquire is pending in the IMS acquisition registers
** and the frames waitAcq is set
*/


/************************************************************
```

```
FILENAME:          imsControl.c
DESCRIPTION:       This file contains the main image board control
                   functions
REVISION HISTORY:    jck: LT John Kisor USN Winter 95'
jck 950110
************************************************************/

/*****************************
jck 950111
To shorten the descriptions below several abbreviations are used:
AM: Acquisition module
DM: Display module

A0: Frame A0
A1: Frame A1
B1: Frame B1
Acq: acquisition (as in waitAcq, waitAquisition)
IMS Ref Man: Imaging Technology Inc's IMS (Standard Image Manager)
               reference Manual for series 150/40
*****************************/

#include "definitions.h"
#include "system.h"
#include "stdiosys.h"

#include "displayCtrl.h"
#include "acquisitionCtrl.h"
#include "imsControl.h"




void
setFirstKToConstant(IMSPage pageNumber, WORD filler)
/* Puts a selected bit pattern in the first 1K of a page so that
** the difference between the pattern and image will be more noticable.
*/ {
unsigned long frame = 0xfa000000;

int i, z;

   selectPage(pageNumber);
   printf("Filling memory with 0x%x\n",filler);
   printf("Filling line #     ");
   for (i=0; i<450; i++){  /*only stuff we see, 1024 will clear entire
frame*/
      if (i < 10)
         printf("\b");
      else if (i < 100)
         printf("\b\b");
      else if (i < 1000)
```

```
            printf("\b\b\b");
         else
            printf("\b\b\b\b");
         printf("%d", i);
         for (z=0; z<1024; z++) {
            *(WORD*)frame = filler;
            frame += 2;
         }
      }
      printf("\n");
}



void
cycleThroughLEDs(void)
/* Visual positive reenforcement. Cycles through all the LEDS once
*/ {

int i, j;

   resetAllLEDs();

   LEDon(FIRST_LED);
   LEDoff(LAST_LED);

   j = 0;
   for(i=0;  i< 35000; i++)    /* Emotional positive feedback for jck */
      if ((i  %  5000) == 0) {
         changeLEDstate(FIRST_LED+j);
         if (j == (LAST_LED - 1)) {
            changeLEDstate(FIRST_LED);
            j = 0;
             }
         else {
            changeLEDstate(FIRST_LED+j + 1);
            j++;
              }

       }
}


void
initIMS(void)
/* Initializes the IMS board to standard addressing w/1MB page size and
** offset of 0x00 from the default address that is the beginning of
** A24 space.(i.e. 0xfa000000) Then it selects the acquistion register
** using the page register
*/ {
const imsConfigReg       = 0xfc000600;
```

```
const standardAddressing = 0x04;
const OneMegPage         = 0x00;
const baseAddressOffset  = 0x0000;

   *(WORD*)imsConfigReg = (standardAddressing | OneMegPage |
baseAddressOffset);
   puts("Setting IMS for A24/D16 with base address of 0x000 (fa000000)");

   initDisplay( NI800by600 );    /* 800x600 non-interlaced */

   initAcqModule();
   puts("Acquisition module registers are set.");

   selectPage(A0);   /* Select frame A0 */

   clearMask(A0);
   clearMask(A1);
   clearMask(B1);
   puts("Masks cleared for: A0, A1 and B1");

   setAOIControl();
   puts("Disabled AOI control.");


   setIMSControl();
   puts("Setting IMS control register");

   resetFrameCtrl(A0);
   resetFrameCtrl(B1);
   puts("Reset frame control registers for A0, A1 and B1");

   resetFramePanScroll(A0);
   resetFramePanScroll(A1);
   resetFramePanScroll(B1);
   puts("Set zero pan and scroll for frames A0, A1 and B1");

   setXPortSwitch();
   puts("Cross-port switch is set");

}



void
clearMask(IMSPage frameNum)
/* Sets the mask to allow ALL bits to pass into the frame specified
** See IMS Ref man 3.3 and setMask
*/ {

const frameAMask  =  0xfc000608;
```

```
const frameBMask  =  0xfc00060c;

const frameA0Mask =  0xff00;
const frameA1Mask =  0x00ff;
const frameB1Mask =  0xffff; /*This sets constant mask to 0x00 also */

WORD oldFrameAMask = *(WORD*)frameAMask;

   printf("Clearing mask for frame: ");
   switch (frameNum) {
      case A0:  *(WORD*)frameAMask = (oldFrameAMask | frameA0Mask);
                puts("A0");
                break;
      case A1:  *(WORD*)frameAMask = (oldFrameAMask | frameA1Mask);
                puts("A1");
                 break;
      case B1:  *(WORD*)frameBMask = (frameB1Mask );
                puts("B1");
                 break;
      default:
         puts("Error:clearMask - Illegal frame selected");
         rexit();
      }
}


void
setMask(IMSPage frameNum)
/* Sets the mask to allow NO bits to pass into the frame specified
** See IMS Ref man 3.3 and setMask
*/ {

const frameAMask  =  0xfc000608;
const frameBMask  =  0xfc00060c;

const frameA0Mask =  0xff00;
const frameA1Mask =  0x00ff;
const frameB1Mask =  0xff00;  /*This sets constant mask to 0xf also */

WORD oldFrameAMask = *(WORD*)frameAMask;

   printf("Protecting with mask frame: ");
   switch (frameNum) {
      case A0:  *(WORD*)frameAMask = (oldFrameAMask & frameA0Mask);
                puts("A0");
                break;
      case A1:  *(WORD*)frameAMask = (oldFrameAMask & frameA1Mask);
                puts("A1");
                break;

      case B1:  *(WORD*)frameBMask = (frameB1Mask );
```

```
                puts("B1");
                break;
      default:
         puts("Error:setMask - Illegal frame selected");
         rexit();
      }
}


void
setAOIControl(void)
/* Sets the video bus out of AOI mode.
** See sect 3.9.4 in IMS Ref man
*/ {

const AOIControl = 0xfc000617;

const disableAOI = 0x00;

   *(BYTE*)AOIControl = disableAOI;
}


void
selectPage(IMSPage pageNumber)
/* Selects between the different image frames and modules on the IMS board
*/ {

const imsPageReg     =  0xfc000620;

const selectFrameA0  =  0x0010;
const selectFrameA1  =  0x0011;
const selectFrameB1  =  0x0013;
const selectAM       =  0x0014;
const selectDM       =  0x0015;

   switch (pageNumber) {
      case A0: *(WORD*)imsPageReg = selectFrameA0;
               puts("Selecting Frame A0");
               break;
      case A1: *(WORD*)imsPageReg = selectFrameA1;
               puts("Selecting Frame A1");
               break;
      case B1: *(WORD*)imsPageReg = selectFrameB1;
               puts("Selecting Frame B1");
               break;
      case AM: *(WORD*)imsPageReg = selectAM;
               puts("Selecting Acquisition module\n");
               break;
      case DM: *(WORD*)imsPageReg = selectDM;
               puts("Selecting Display module\n");
```

```
                break;
        }

 }


void
setIMSControl(void)
/* Sets the IMS control register. See section 3.11 of the IMS hardware Ref
man
*/ {
const imsControl = 0xfc000624;

const FPSel      = 0x0000;  /* Sets the IMS to be master. */
const clockSpeed = 0x0004;  /*Sets clock speed to be 20Mhz */
const DispMode   = 0x00a0;  /*DISPOE, ETMODE, DMEN, CBMR */
       /* DISPOE(1): Enable display output. Not needed since DM is present
*/
         /* ETMODE(0): Define AM module trigger mode. */
         /* DMEN  (1): Enable image display */
         /* CBMR  (0): Do not clear the mask registers, This bit can be
                       used to protect all frame bits */

   *(WORD*)imsControl = (FPSel |clockSpeed | DispMode );
   puts("IMS set 20Mhz, IMS:master, Display:enabled and in AM trigger
mode\n");
 }


void
setFrameIRQ(void)
/* The type of frame interrupt will be put here.
** Currently not used.
*/ {
/*Empty function */
}


void resetFrameCtrl(IMSPage frameNum)
/* Resets the controls for horizontal pitch, tiling and write zoom for
** frames A0, A1 and B1. See IMS Ref man 3.29, 3.34
*/ {

const frameACtrl = 0xfc000650; /* Frame A0 and A1 control */
const frameBCtrl = 0xfc00065a; /* Frame B1 control */

/* Write (output) zoom */
const aUnPack         = 0x0000; /* 0:normal ops, no packing of bits in
A0,A1 */
const verticalZoom    = 0x0000; /* No zoom, but a good project for someone.
```

```
*/
const horizontalZoom  = 0x0000;

/* Read (input) zoom) */
const bNoHorizZoom    = 0x0000; /* No zoom. Avail: 1x, 2x and 4x */
const bNoVertZoom     = 0x0000;

/* Write (output) zoom */
const bUnPack       = 0x0000; /* 0:normal ops, no packing of bits in B1 */
const bWriteVertZoom  = 0x0000;
const bWriteHorizZoom = 0x0000;

   printf("Resetting frame: ");
   switch (frameNum) {
      case A0:
      case A1: *(WORD*)frameACtrl = (aUnPack |  verticalZoom |
horizontalZoom);
                  puts(" A0 and A1");
                  break;
      case B1:  *(WORD*)frameBCtrl = (aUnPack | bUnPack |
                                    bNoVertZoom  | bNoHorizZoom |
                                  bWriteVertZoom | bWriteHorizZoom  );
                  puts(" B1");
                  break;
      default:
        puts("Error:resetFramePanScrolACtrl - Illegal frame selected\n");
          rexit();
      }
}


void
resetFramePanScroll(IMSPage frameNum)
/* Resets a frame for no (zero) pan and scroll. See 3.21 of the IMS ref man
*/ {

const a0ReadPan      = 0xfc000640;
const a0ReadScroll   = 0xfc000642;
const a0WritePan     = 0xfc000644;
const a0WriteScroll  = 0xfc000646;

const a1ReadPan      = 0xfc000648;
const a1ReadScroll   = 0xfc00064a;
const a1WritePan     = 0xfc00064c;
const a1WriteScroll  = 0xfc00064e;

const b1ReadPan      = 0xfc000652;
const b1ReadScroll   = 0xfc000654;
const b1WritePan     = 0xfc000656;
const b1WriteScroll  = 0xfc000658;
```

```
const zeroPan        = 0x00;
const zeroScroll     = 0x00;

   printf("Setting pan and scroll to zero for frame: ");
   switch (frameNum) {
      case A0: *(WORD*)a0ReadPan     = zeroPan;
               *(WORD*)a0ReadScroll  = zeroScroll;
               *(WORD*)a0WritePan    = zeroPan;
               *(WORD*)a0WriteScroll = zeroScroll;
               puts("AO.");
              break;

      case A1: *(WORD*)a1ReadPan     = zeroPan;
               *(WORD*)a1ReadScroll  = zeroScroll;
               *(WORD*)a1WritePan    = zeroPan;
               *(WORD*)a1WriteScroll = zeroScroll;
               puts("A1.");
              break;
      case B1: *(WORD*)b1ReadPan     = zeroPan;
               *(WORD*)b1ReadScroll  = zeroScroll;
               *(WORD*)b1WritePan    = zeroPan;
               *(WORD*)b1WriteScroll = zeroScroll;
               puts("B1.");
              break;
      default:
         puts("Error:resetFramePanScroll - Illegal frame selected");
         rexit();
      }

}


void
setXPortSwitch(void)
/* Sets the cross-port switch for the application bus width and direction
** This register must be set prior to setting any path registers
** See AM Ref man sect 3.43
*/ {

unsigned int crossPortSwitch = 0xfc000670;

const portConstant = 0x0007;

int i;

   for (i=0; i<8; i++){ /* Go through 8 words, 0x70->0x7f */
      *(WORD*)crossPortSwitch = portConstant;
      crossPortSwitch += 0x00000002;
   }
}
```

```
void
setInputPath(IMSPage selectedFrame, INPUTPath imageSource)
/* Selects the source of the image that will be put into a particular
** frame. See AM Ref Man 3.40, 3.41 and 3.42
*/ {

const a0InputPath = 0xfc00066a;
const a1InputPath = 0xfc00066c;
const b1InputPath = 0xfc00066e;

const cameraInput = 0x00b8;
const a0ZeroInput = 0x00c0;
const constantInput = 0x0090;
const a0OnesInput = 0x00e0;
const a0Input     = 0x0288;
const a1Input     = 0x0298;
const b1Input     = 0x02b0;

   switch(selectedFrame) {
   case A0:
      switch(imageSource) {
      case CAMERA: *(WORD*)a0InputPath = cameraInput;
                   break;
      case A0ZERO: *(WORD*)a0InputPath = a0ZeroInput;
                   break;
      case CONSTANT: *(WORD*)a0InputPath = constantInput;
                   break;
      case A0ONES: *(WORD*)a0InputPath = a0OnesInput;
                   break;
      case INA0:  *(WORD*)a0InputPath = a0Input;
                   break;
      case INA1:  *(WORD*)a0InputPath = a1Input;
                   break;
      case INB1:  *(WORD*)a0InputPath = b1Input;
                   break;
      default:
       puts("Error:setInputPath - Illegal input for frame A0 selected.");
         rexit();
      }
      break;

   case A1:
      switch(imageSource) {
      case CAMERA:   *(WORD*)a1InputPath = cameraInput;
                     break;
      case CONSTANT: *(WORD*)a1InputPath = constantInput;
                     break;
      case INA0:     *(WORD*)a1InputPath = a0Input;
```

```
                    break;
        case INA1:  *(WORD*)a1InputPath = a1Input;
                    break;
        case INB1:  *(WORD*)a1InputPath = b1Input;
                    break;
        default:
         puts("Error:setInputPath - Illegal input for frame A1 selected.");
           rexit();
        }
        break;

    case B1:
        switch(imageSource) {
        case CAMERA:   *(WORD*)b1InputPath = cameraInput;
                    break;
        case CONSTANT: *(WORD*)b1InputPath = constantInput;
                    break;
        case INA0:  *(WORD*)b1InputPath = a0Input;
                    break;
        case INA1:  *(WORD*)b1InputPath = a1Input;
                    break;
        case INB1:  *(WORD*)b1InputPath = b1Input;
                    break;
        default:
         puts("Error:setInputPath - Illegal input for frame B1 selected.");
           rexit();
        }
        break;

        default:
         puts("Error:setInputPath - Illegal destination frame selected.");
           rexit();

    }
}



void
setFrameAcquire(IMSPage frameNum, ACQUIREType typeOfAcquire)
/* controls the acquistion to frame A0, B1 or A1
** The input cross port switch should be set prior to calling this func.
** Decision was made to have actual acquistions triggered by the
** acquisition enable register (3.19 Ref man)
*/ {

const frameA0Acq  = 0xfc000630;
const frameA1Acq  = 0xfc000632;
const frameB1Acq  = 0xfc000636;

const triggerMode = 0x00; /* No wait for trigger from AM */
```

```
const waitAcq     = 0x02; /* prevent acquisition till acqEnable is written
*/

WORD  acquireMode;

   switch(typeOfAcquire) {
      case SNAP:
       acquireMode = 0x0080; /*Make this a snap operation (one picture) */
         break;
      case GRAB:
         acquireMode = 0x00c0; /*Make this a grab operation (continuous
snap)*/
         break;
      case FREEZE:
         acquireMode = 0x0000; /* Stop taking pictures */
         break;
      default:
       puts("Error:setFrameAcquire - Illegal type of acquire selected. ");
         puts("Stopping grab or snap, if enabled\n");
         acquireMode = 0x0000;
         break;
   }

   printf("Setting Acquistion controls for ");
   switch (frameNum) {
      case A0:  while ( (*(WORD*)frameA0Acq & 0x0010))
                              ;
              *(WORD*)frameA0Acq = (triggerMode | waitAcq |acquireMode );
                puts("Frame A0 ");
                break;
      case A1:  while ( (*(WORD*)frameA1Acq & 0x0010))
                              ;
              *(WORD*)frameA1Acq = (triggerMode | waitAcq |acquireMode );
                puts("Frame A1 ");
                 break;
      case B1:  while ( (*(WORD*)frameB1Acq & 0x0010))
                              ;
              *(WORD*)frameB1Acq = (triggerMode | waitAcq |acquireMode );
                puts("Frame B1 ");
                 break;
      default:
         puts("Error:setFrameAcq - Illegal frame selected.");
         rexit();
      }
   puts("awaiting AM trigger and acquistion enable\n");
 }


void
acqEnable(void)
/* This register controls the synchronous acquisition into the IMS
```

```
** frame stores (see IMS ref man 3.19) This enables acquisition into the
** frame stores when an acquire is pending in the IMS acquisition registers
** and the frames waitAcq is set
** For system reset use: IV-SPRC > ex fffc00e3
**                       0xfffc00e3: 0xff ? f0
*/ {
const frameAcqEnable = 0xfc000634;

const enable = 0x0000;

   *(WORD*)frameAcqEnable = enable;
   puts("IMS enabled for image acquistion.\n");


}
/*****************************************************************/
/*   FILENAME:imagelog.h
/*   AUTHOR: Leonard V. Remias and Khaled Morsy
/*           some fragments from Kelbe
/*   DATE: 01March 1996

#ifndef __IMAGELOG_H
#define __IMAGELOG_H

/*
   Must be called prior to turning on image logging. This makes
   image logging two steps. This must be, since we cannot depend
   on static variables being properly initialized on Yamabico
   for subsequent runs.
*/
void     InitImagelog(void);


/*
    Function prepares the tracing system to log data.
     The tracing frequency specifies how many image control cycles are
     skipped before a data point is logged. A value of 1 or less causes
    the logging to occur each cycle.
      The filename is the name of the file to create on the host. If NULL
    is passed in, a default name is used.
      The buffer size specifies how many bytes of storage to allocate to
    save the data. If a value of 0 is specified, a default size is used.

    Tracing is automatically turned on after this call.

   The simplest way to call this function and have image logging is:

   InitImageLog(NULL, 0);

   This will use all of the default values, and should work fine for most
   cases.
```

```
*/
void     ImageLog(char *Filename, int BufferSize);
/*
    Function enables data logging. Frequency is used by the
    system's logging functions to determine the number of image
    control cycles between logged data. For example, if the
    frequency is 3, then data would be logged on every third call
    to LogTimedImage() or LogImageData(). Use this to change
    the logging frequency.
*/


/*************************************************************************/
/*   FILENAME:imagelog.c
/*   AUTHOR: Leonard V. Remias and Khaled Morsy
/*           some fragments from Kelbe
/*   DATE: 01March 1996
/*   PURPOSE: Prepare the system for image logging
*/

#include "definitions.h"
#include "imagelog.h"
#include "stdiosys.h"
#include "trace.h"


/***        Local variables  ***/

static IOhandle   LogHandle;
static int        Enabled;


/***        Code        ***/


void
InitImagelog(void)
{
        LogHandle = EOF;
        Enabled = 0;

}


void
ImageLog(char *filename, int bufSize)
{
   if (bufSize <= 0)
      bufSize = IMAGELOGSIZE;     /* if not specified, use the default */

   if (filename == NULL)
```

```
      filename = IMAGELOGFILE;


   LogHandle = IOopenimage(filename, bufSize);

   if (LogHandle == EOF)  {
      printf("\nError initializing image logging file %s\n", filename);
      return;
   }
   Enabled = 1;
}


void
LogImage(YAMAIMAGE b1image)

{

int i,j;

/* checking LogHandle here is not really necessary, but saves a function
call */

 if (Enabled)


      for(i=0; i<476; i++)
      {
        for(j=0; j<732; j++)
        {

         IOprintf(LogHandle,"%d\n", b1image.image[i][j]);

        }
      }

}
```

LIST OF REFERENCES

[BAL82] Ballard, Dana H., and Brown, Christopher M., *Computer Vision*, Prentice-Hall, Inc. Englewood Cliffs, NewJersey 1982.

[COH90] Cohu, Inc., "*Installation and Operation Manual for 4110 Digital Output Monochrome CCD Camera*", Technical Manual Code 6X-899, August 1990.

[DEC93] Declue, Mark J., *Object Recognition Through Image Understanding For An Autonomous Mobile Robot*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

[DOU87] Dougherty, Edward R., and Giardina, Charles R., *Matrix Structured Image Processing*, Prentice-Hall International, 1987.

[ITIAM93] Imaging Technology Inc., "*AM-DIG Hardware Reference Manual*" Document Number 47-H44002-00, October 1993.

[ITIDM92] Imaging Technology Inc., "*DM-PC Hardware Reference Manual*" Document Number 47-H44001-00, September 1992.

[ITIIMS93] Imaging Technology Inc., "*IMS Hardware Reference Manual*" Document Number 47-H54002-00, April 1993.

[KAN94] Kanayama, Y., "Mathematical Theory of Robotics: Introduction to 2D Spatial Reasoning", *Lecture Notes of the Advanced Robotics Course,* Department of Computer Science, Naval Postgraduate School, Winter Quarter 1994.

[KIS95] Kisor, John Carl, *Integration Of An Image Hardware/Software System Into An Autonomous Robot*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1995.

[KRI89] Kriegman, Triendl and Binford, *Stereo Vision and Navigation in Buildings for Mobile Robots*, IEEE Transactions on Robotics and Automation,Vol. 5, No. 6, pp 332-343, December 1989.

[LOC94] Lochner, J., "*Analysis and Improvement of an Ultrasonic Sonar System on an Autonomous Mobile Robot*" Master's Thesis, Naval Postgraduate School, Monterey, California, March 1994.

[LUM88] Lumelsky, Vladimir and Skewis, Tim, *A Paradigm For Incorporating Vision In The Robot Navigation Function*, IEEE Transactions on Robotics and Automation, 1988.

[MAC93] MacPherson, David L., *Automated Cartography by an Autonomous Mobile Robot*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, September 1993.

[NIB86] Niblack, Wayne, *An Introduction to Digital Image Processing*, Prentice-Hall International, 1986.

[PET92] Peterson, Kevin L., *Visual Navigation for an Autonomous Mobile Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

[STE92] Stein J., "*Modeling, Visibility Testing and Projection of an Orthogonal Three Dimensional World in Support of a Single Camera Vision System*", Master's Thesis, Naval Postgraduate School, Monterey, California, March 1994.

[ULL91] Ullman, Shimon and Basri, Ronen, *Recognition by Linear Combinations of Models*, IEEE Transactions on Pattern Analysis and Machine Intelligence,Vol. 13, No. 10, pp 992-1005, October 1991.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library — 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr Yutaka Kanayama, Code CS/KA — 3
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. LT Leonard V. Remias — 2
   2546 W. Azalea Point Road
   Norfolk, VA 23518