GENERATION OF A ROBOTIC CELL BASED ON TECHNIQUES OF IMAGING AND REPRODUCTION

A THESIS

Presented to the Department of

Mechanical Engineering

California State University, Long Beach

In Partial Fulfillment

of the requirements for the Degree

Master of Science

By Steven N. Corday

B.S., 1994 California State University of Long Beach

December 2002

WE, THE UNDERSIGNED MEMBERS OF THE COMMITTEE,

HAVE APPROVED THIS THESIS

GENERATION OF A ROBOTICS CELL BASED ON

TECHNIQUES OF IMAGING

AND REPRODUCTION

By

Steven N. Corday

COMMITTEE MEMBERS

[illegible]

Ortwin Ohtmer, Dr.-Ing. Mechanical Engineering

[illegible]

Hung V. Vu, Ph.D. Mechanical Engineering

[illegible]

Grote Karl, Dr.-Ing. Mechanical Engineering

ACCEPTED AND APPROVED ON BEHALF OF THE UNIVERSITY

[illegible]

Hamid Hefazi, Ph.D.
Department Chair, Department of Mechanical and Aerospace Engineering

California State University, Long Beach

December 2002

ABSTRACT

GENERATION OF A ROBOTIC CELL BASED ON

TECHNIQUES OF IMAGING

AND REPRODUCTION

By

Steven N. Corday

December 2002

Robots have been used in industry since 1965 to extend the capability of machines to perform certain tasks formerly done by humans, and to control sequences of operations without human intervention. A physical robot is normally composed of a main frame (manipulator) consisting of 3 linear axes (joints) that move in the x-y-z directions called a Cartesian Robot.

One model type is the Seiko D-TRAN XM5000 Cartesian Robot System donated to the School of Engineering at California State University Long Beach. The miscellaneous robotic parts, donated to the University, had to be assembled. A software program, DARLTalk, obtained by the Seiko vendor [1] was installed on a PC

and then interfaced with the robot's controller. After the robot's assembly was complete, this thesis student used the DARLTalk communication software as the primary means to program and operate the robotic cell.

The aim of this thesis is to study the adaptation of pattern recognition to a robotic cell. In order to accomplish the adaptation of pattern recognition a handwritten image is scanned using an off the shelf commercial scanner. The scanning process creates a bitmap file, an industry standard file format for graphic data. The next step is the conversion of the bitmap file to a postscript ASCII file of characters to represent the image. This ASCII file is used in the next major phase of the process, the pattern recognition phase. This process creates an output file that duplicates the image with the appropriate executable coordinates formatted for the robotic cell to read.

ACKNOWLEDGMENT

First, I would like to thank my advisor Dr. Ortwin Ohtmer for introducing me to this research topic, and more importantly, for his comments, guidance, and the effort spent to help make this thesis possible. Given the hurdles of setting up the robotic equipment, Dr. Ortwin Ohtmer was instrumental in providing additional school resources and funds to purchase additional parts to make the robotic cell operational for programming.

I would like to thank my former colleague Paul Doyle who works in the CAD support group at Boeing, Huntington Beach and Brian Wey, self-employed software engineer, who provided guidance. Their collective abilities in computer programming was instrumental in helping me rewrite and debug the problematic software of Bauman's written more than 10 years ago.

I would like to thank my daughter Lorraine and son Michael for their love. No matter what paths I take in life, I will always be a part of their lives too.

Finally, I would like to extend my heartfelt thanks to my wife, Dee Anna Escobedo-Corday, for her

understanding and patience and support. With her support and caring, impossible things were made possible. She is a wonderful friend and companion, and I dedicate this thesis to her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

LIST OF GRAPHS

LIST OF ABBREVIATIONS

ASCII	= American Standard Code for Information
CNC	= Computer-based Numerical Control
CPU	= Central Processing Unit
DARLTalk	= D-TRAN Assembly Robot Language
DCE	= Data Communication Equipment
DTE	= Data Terminal Equipment
FORTRAN	= FORmula TRANslation
I/O	= Input/Output
K	= Thousand (x1000)
kg	= Kilogram
kgf/cm^2	= Kilogram Force per Centimeter Square
lbf/in^2	= Pound force per Inch Square
mm	= Millimeter
PC	= Personal Computer
Twips	= Twentieth of a Point
VDC	= Volts, Direct Current

CHAPTER 1

INTRODUCTION

The proposed procedure of using a scanned or computer generated bitmap image and recreating this image with the Seiko D-TRAN robotic cell is outlined in this report. Core to this project is the expansion of the bitmap image to a postscript ASCII file. The postscript ASCII language remains one of the most powerful graphic design tools in the world that can easily handle graceful curves, fancy fonts, and tool paths to name a few.

The potential applications of the robotic cell centers on the postscript ASCII file handled by the new software program also introduced in this report. A new efficient Postscript File Scan program written for this project is based from several old DOS software programming modules written years ago back in 1989 by Ronald C. Bauman for the Department of Mechanical Engineering as a demonstration for Pattern Recognition. The software used by Ronald C. Bauman was based on a closed mesh detection frame program, written in FORTRAN, by Dr. Ortwin Ohtmer [3] and modified to run in a DOS environment.

By figuring out how the multi-stepped process of pattern recognition worked, I was able to recreate the main logic of the code by correcting the syntax problems and the numerous programming errors. Also, new advanced features were encoded to fix the shortcomings of the code to accept any size bitmap, automatically compute the detection frame and segment size (pixel). The format of the output files from the newly created program was tailored to the robotic cell.

Although the old programming modules had many errors in the source code and did not compile properly, the image transformation and the detection frame logic portion of the old programming modules were the only things borrowed, corrected, updated and incorporated into the new Postscript File Scan program for this thesis.

CHAPTER 2

RELATED WORK

There has been a variety of research work performed at the Department of Mechanical Engineering, California State University Long Beach, using different software platforms such as FORTRAN and "C" programming. Based on the research the program tools developed made it possible to merge 2D and 3D-modeling techniques by applying graphical pattern recognition implemented with lasers beams, video cameras and a 4-axis robot. For example, a paper presented by Dr. Ortwin Ohtmer [4] demonstrated the mathematical analysis/model of measuring points representing shell elements on a 3D-object compiled in a matrix notation and interactively transforming this data via software tools into a graphical representation onto an APOLLO workstation screen.

To perform this goal of capturing the analytical data points (e.g., x, y and z coordinates) the 3D-object had to be fixed onto a stable surface. A laser beam was then mounted on a hand attachment of the robot pointing at the 3D-surface creating a normal vector. A video camera mounted on the robot to positionally control movements normal to the surface, starts recording the multiple dots of light rays (as a normal vector) projected on the 3D-surface. The measured 3D surfaces represented by a Spline based on Graph Theory is recorded as digitized pixels. Through every programmed iteration or scanning process around the 3D-object, the 4-axis mechanical robot makes (incremental) radial sweeps to complete the topological measurements.

The mathematical measurements are described as follows:

1. A scale-factor "h" specified is the apex of the laser beam and center of the video camera lens/window frame intersecting forming an angle $\theta \leq 30^\circ$. To obtain a reasonable accuracy the angle θ should have a lower limit setting.
2. The picture point "A" represented as the laser beam dot on the 3D surface has an eccentric position "e" within the video camera lens/window frame. Eccentric value can be specified applying the software using Graphical Pattern Recognition techniques.

The normal distance "X" measured from the path of the laser beam, between the picture point "A" on the 3D-surface and laser beam dot can be calculated according to Equation (1).

$$x = h - e/\sin\theta \qquad \text{Equation (1)}$$

In Figure 1 a graphical representation demonstrates the mathematical modeling process. A video camera is capturing the graphic data of the spline surfaces generated by the recorded digitized images of the laser beam as pixels used to calculate the x, y, z coordinates of the 3D-object.

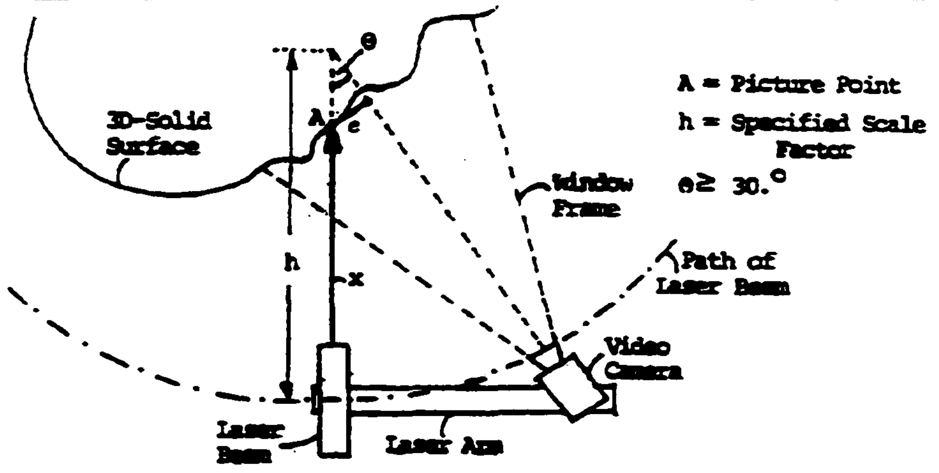


FIGURE 1. Mathematical modeling process of scanned 3D-objects

CHAPTER 3

ROBOTIC CELL DESCRIPTION AND SETUP

XM5000 D-TRAN Robotic Cell

The XM5000 D-TRAN Seiko is a Cartesian coordinate type robot designed for handling material and assembly work. The XM5000 Seiko robot consists of the following equipment:

1. Manipulator (robot)
2. Controller
3. Teach Terminal
4. Personal Computer (PC): Donated by the University Mechanical Engineering Department.
5. Cables: Not included. Purchased through outside vendor.
6. DARLTalk (D-TRAN Assembly Robot Language) Software Version 2.02A: Obtained from Epson America, Inc.

The Seiko robot equipment was donated to the University Engineering Department from Epson America, Inc. 18300 Central Avenue, Carson, California 90747, web site: www.robots.epson.com (formerly Seiko Instruments USA Inc.)

Seiko D-TRAN XM5000 Robotic Cell Assembled

The assembled 4-axis Robotic Cell (see Figure 2) is now set up for normal operation from the Teach-terminal or the host computer interface. The DARLTalk communication software loaded on to the PC-based environment allows for easy interfacing tailored for the operator to "drive" the Seiko D-TRAN robot system. Details of the DARLTalk Communication Software can be found in Appendix A. The operator will gain an appreciation of the Robot's straightforward architecture. This architecture allow more concentration on manufacturing and less on how to translate to desired positions which can be taught to the robot manually by the Teach Terminal or via DARLTalk with the power to write challenging software programs.

The programming language DARL (D-TRAN Assembly Robot Language) has a similar structure to BASIC: however, it is tailored to the Seiko D-TRAN robot to be easy to use in two-way communication. Using the Teach Terminal, or the optional host PC connection, the operator can determine the status of items such as the robot's position, inputs and outputs, coordinates position, frames of reference, jog-speed and program speed.

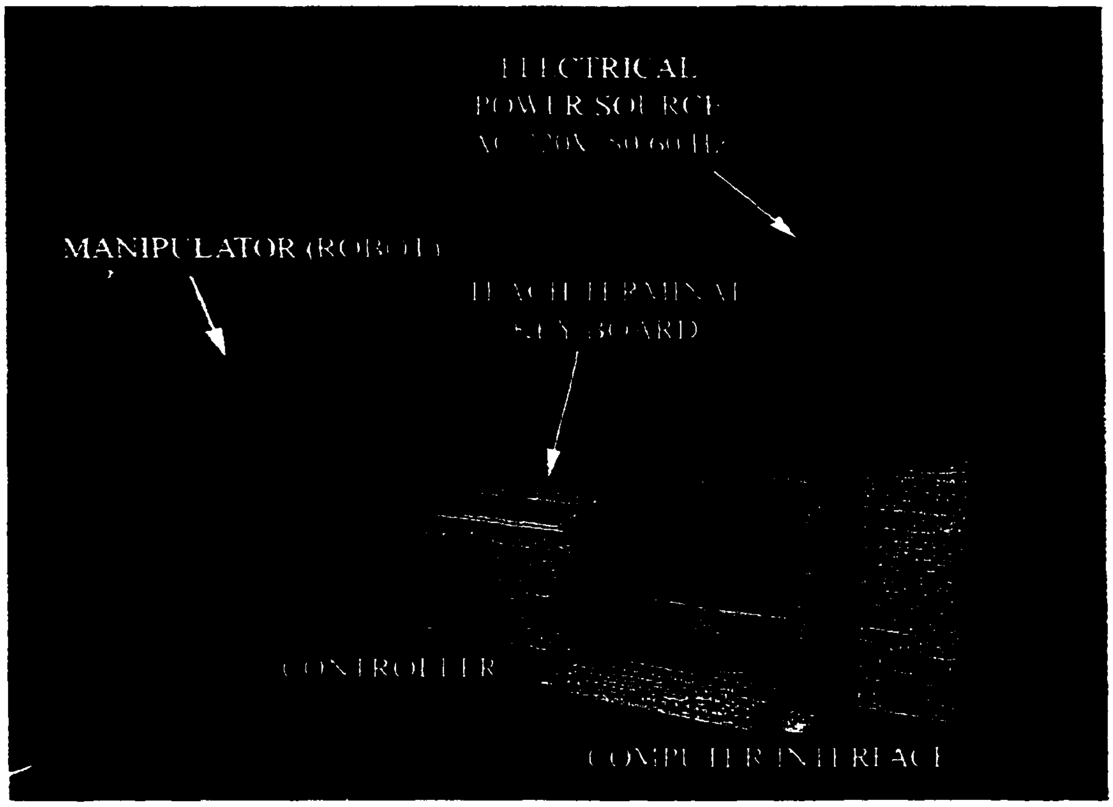


FIGURE 2. SEIKO D-TRAN robotic cell physical setup

Manipulator Setup

The Manipulator (robot) with shipping brackets had to be unpacked and positioned on to a stable work-cell base. Without the shipping brackets the manipulator weighs 118.1 kg. Next, the 2 manipulator "footprint" mounting pads had to be well secured to the work-cell base (table) with 4 bolts each and tightened to a torque of 440kfg-cm.

<u>Manipulator Specifications</u>

The stroke of each coordinate for the XM5000 Manipulator is as follows: (See Figure 3.)

Y stroke = 600 mm
X stroke = 400 mm
Z stroke = 100 mm
A-Axis = +/- 999 Degrees (rotation)
Note: the A-Axis (Unit part number AU200) has an allowable maximum torque value of 38kgf.

The forces and accelerations that the XM5000 Manipulator will impart on a work-cell base (table) can be calculated using the following table.

TABLE 1. Manipulator Specification

AXIS	FORCE (kgf)	ACCELLERATION (G)
Y	(0.0194 x Xmm) + (0.0043 x Zmm) + 28.4	0.51
X	(0.0043 x Zmm) +14.3	0.51
Z	(0.004 x Zmm) + 6.2	0.33

Note: "Zmm" means the Z stroke in millimeters (mm) and so forth.

Example: What is the maximum force (kgf) that can be generated by the manipulator Y-axis that the work-cell base will experience.

Kgf = (0.0194 x 400mm) + (0.0043 x 100mm) + 28.4 = 36.59

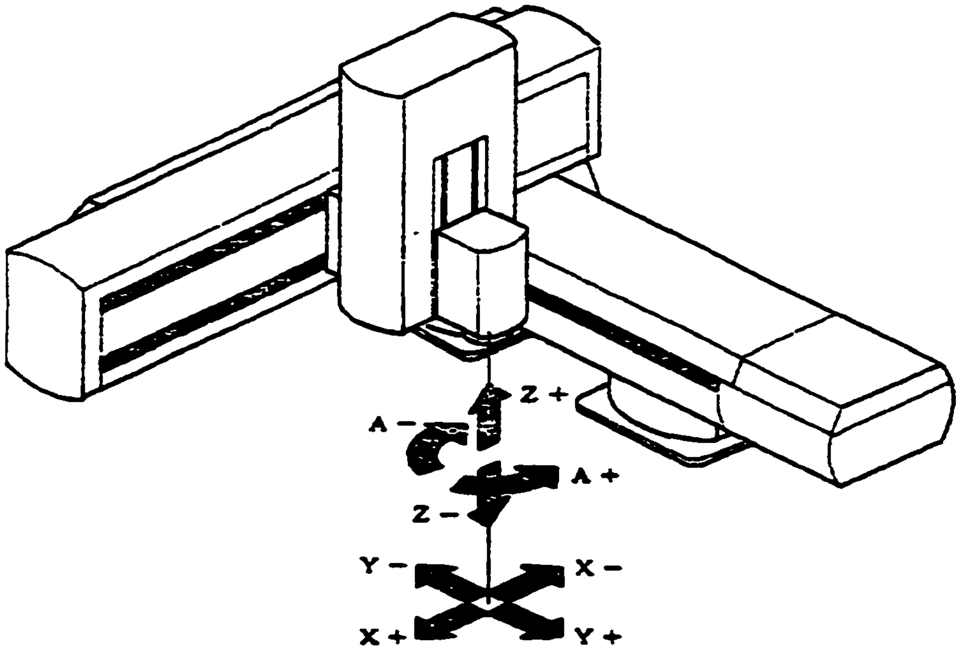


FIGURE 3. Coordinate system of (XM5000) robot

A-Axis Moment of Inertia

Consideration should be given to the moment of inertia (see Figure 4) when the bottom of the A-axis/end effector flange is attached with a gripper tool and rotates. The payload distance from the center of the A-axis when picked from the jaws of a gripper tool should not be too great as to overload the A-axis flange drive motor. When operating the XM5000 D-TRAN Robot at maximum speed make sure the design of the gripper and moment of the inertia of the payload is within 105gf.cm.sec^2 (7.59471bf.ft.sec^2).

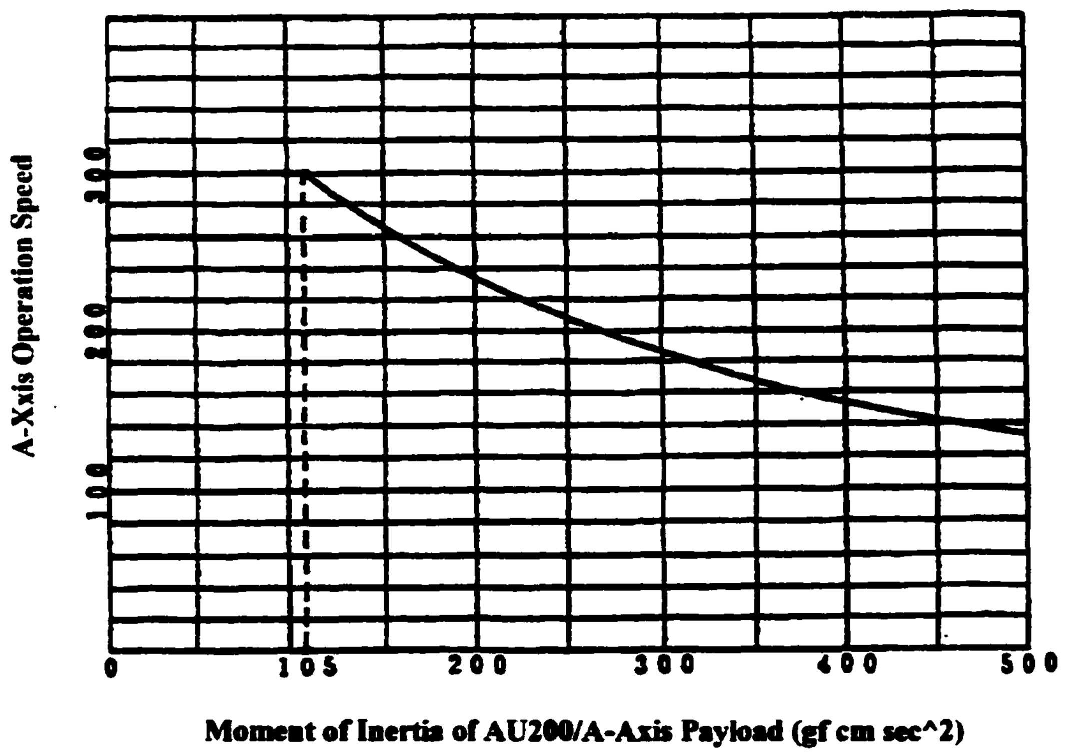


FIGURE 4. Limitation in moment of inertia and A-axis operation speed

Vise Grip Tool

A new vise grip tool was developed and fabricated for the Robot end-effector for this thesis project. This tool is bolted to the bottom of the end-effector to hold small objects such as a pen for tracing (see Figure 5).

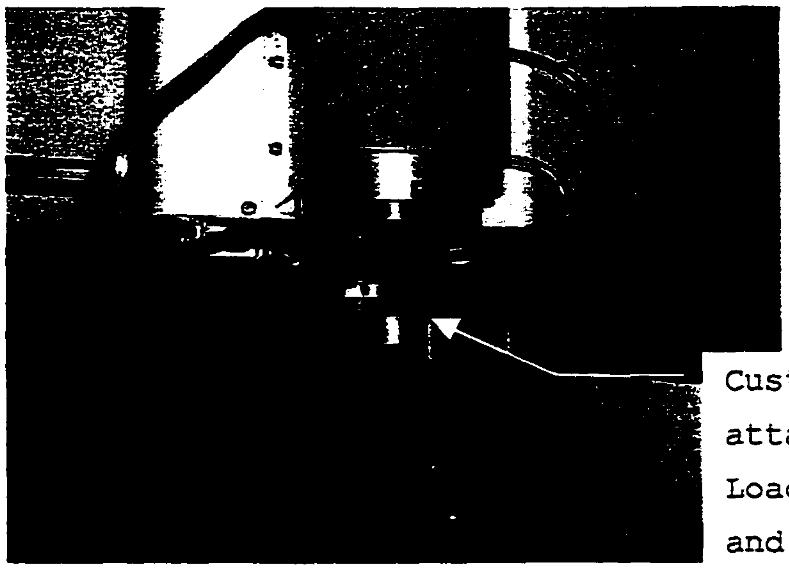

Custom Made Vise attached with Spring Loaded Screws to Grip and Hold Objects

FIGURE 5. AU200 unit/end effector

Air Supply Specifications for AU200 Unit

The air supply specification on the end effector requires a pressure range of 1.5 to 5kgf/cm2 (21.34 to 71.121bf/in^2) and an air filtration of 6 microns. A 6mm outside diameter tube for the air ports will operate a pneumatic type tool attachment such as gripper for opening and closing the jaws.

Controller Setup

The XM5000 D-TRAN manipulator (robot) is operated by a standard single phase Controller weighting 35 kg (77.16 pounds) (excluding cables for the manipulator). Power cable from the rear of the controller with the missing plug receptacle had to be hard wired into the electrical power source providing a single phase AC 220 volts, 50/60 Hertz to the controller. The laboratory had no power source that the equipment could use. Mechanical Engineering Department had to provide (a plug) for the power before the robotic cell could be checked out. Due to some faulty hard wiring affecting the single-phase power the controller did not have enough power to throw the servo magnetic relay switch inside the controller. To overcome this problem its required the manual reset button on the servo magnetic relay be pressed to power on the servos (see Figure 6).

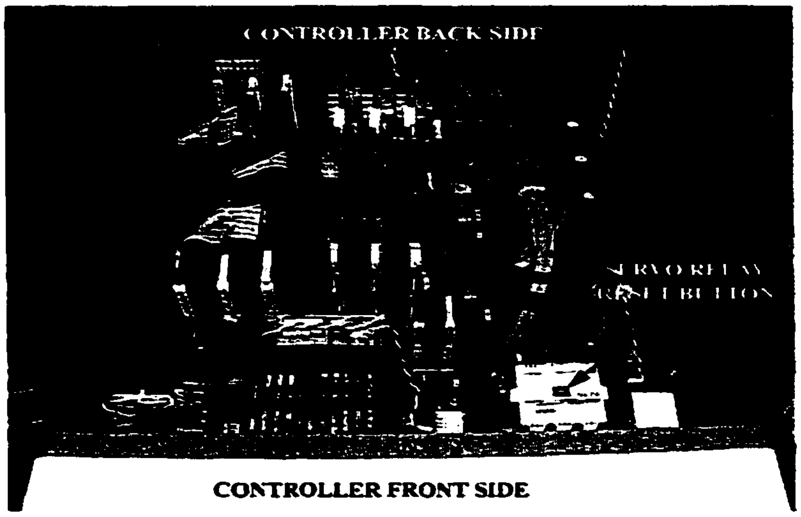


FIGURE 6. Interior of controller showing circuitry and servo drive relay switch

Personal Computer (PC) Setup

The PC supplied by the Mechanical Engineering Department with Windows NT-2000 is used as the host computer for the robot's DARLTalk communication software. The DARLTalk program, once installed, provides the communication between the operator and controller with enhanced ability to develop complex programs with quick debugging diagnostic tools done off line.

The setup communication with the host PC is over a standard RS232C cable connected between serial port 1 on the PC and the Seiko D-TRAN controller. Communication as defined in the RS232C standard is an asynchronous serial communication method. The basic idea of the cable is to interface Data Terminal Equipment (DTE) and Data Communication Equipment (DCE) employing serial binary data interchange. The controller's CPU factory setting is set to DCE mode and a baud rate of 9600. This means that the controller's RS232C port can only be connected with a DTE type serial communication device such as the serial port on the PC for which DARLTalk communicates over. Note: if both devices are DTE type, use a full handshake null modem or reset the controller's CPU (shunts) stake-pin jumpers to a DTE setting.

Manipulator Cable Setup

Four cables are required to run the Seiko XM5000 series robot with the exception of the (25 pin) serial cable for the PC interface (see Figures 7, 8 & 9). During initial hardware setup the robot cables were not available and had to be purchased (as refurbished equipment) from Servo Systems Co. located in Montville, New Jersey. The purchased cable set consists of the Drive cable, the Encoder cable, the Gripper Cable, and the Sensor cable. These cables are connected between the controller and the manipulator (robot). The External I/O 60-pin dummy connector plug (which came with the controller) is configured with jumpers between pin 1 (+5VDC) and pin 36 (External Stop) which is connected to the controller's external receptacle. This allows the Controller's START button to work. The last cable to be connected is the RS232C (25 pin) serial cable between the PC communication port and the controller.

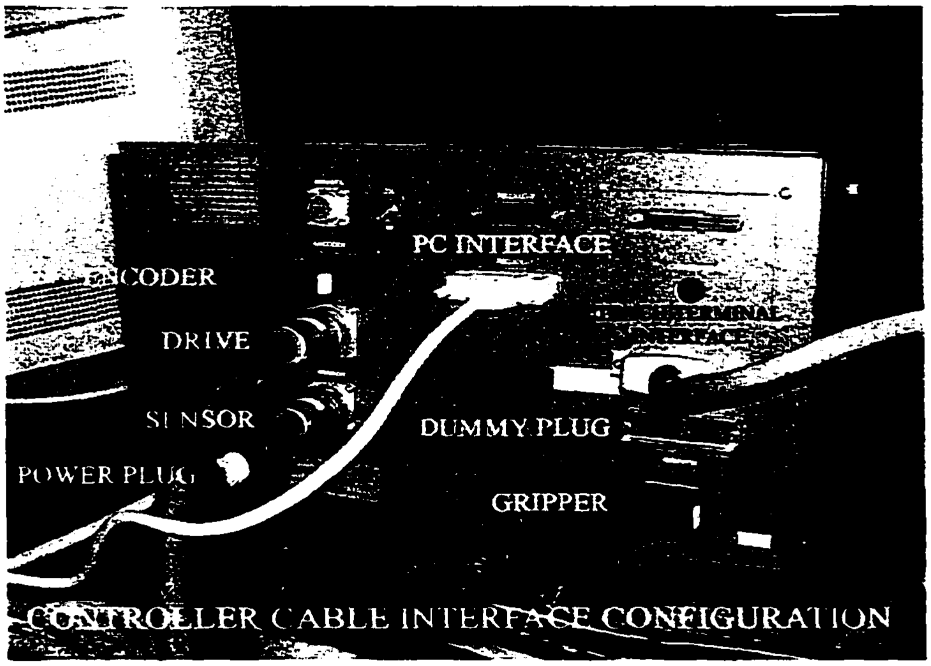


FIGURE 7. Controller cable interface configuration

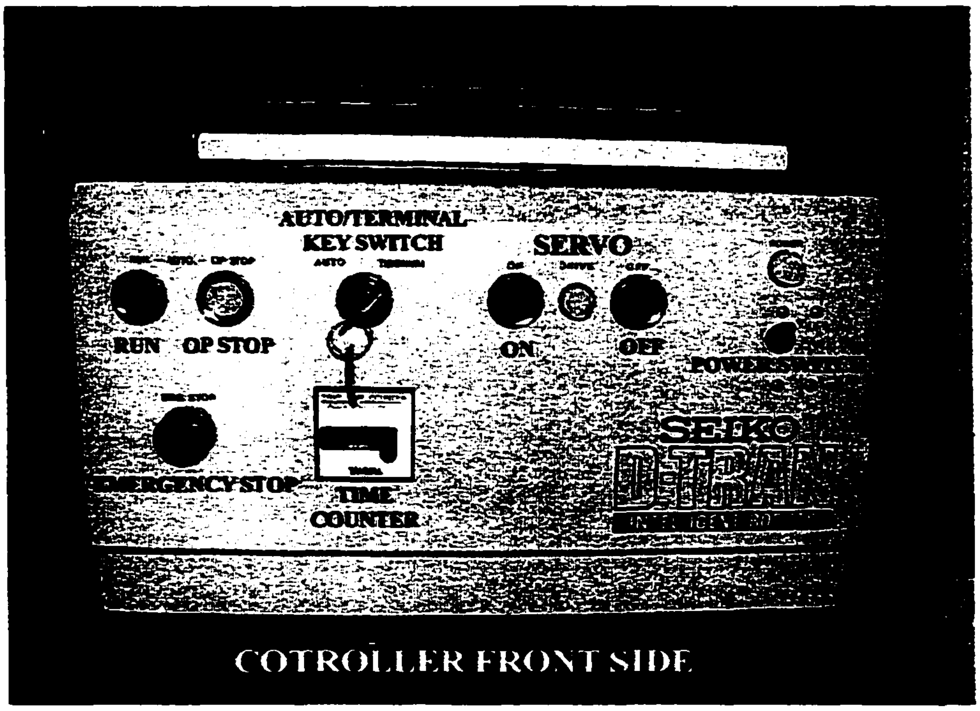


FIGURE 8. Controller front side (Teach-Terminal perched on top)

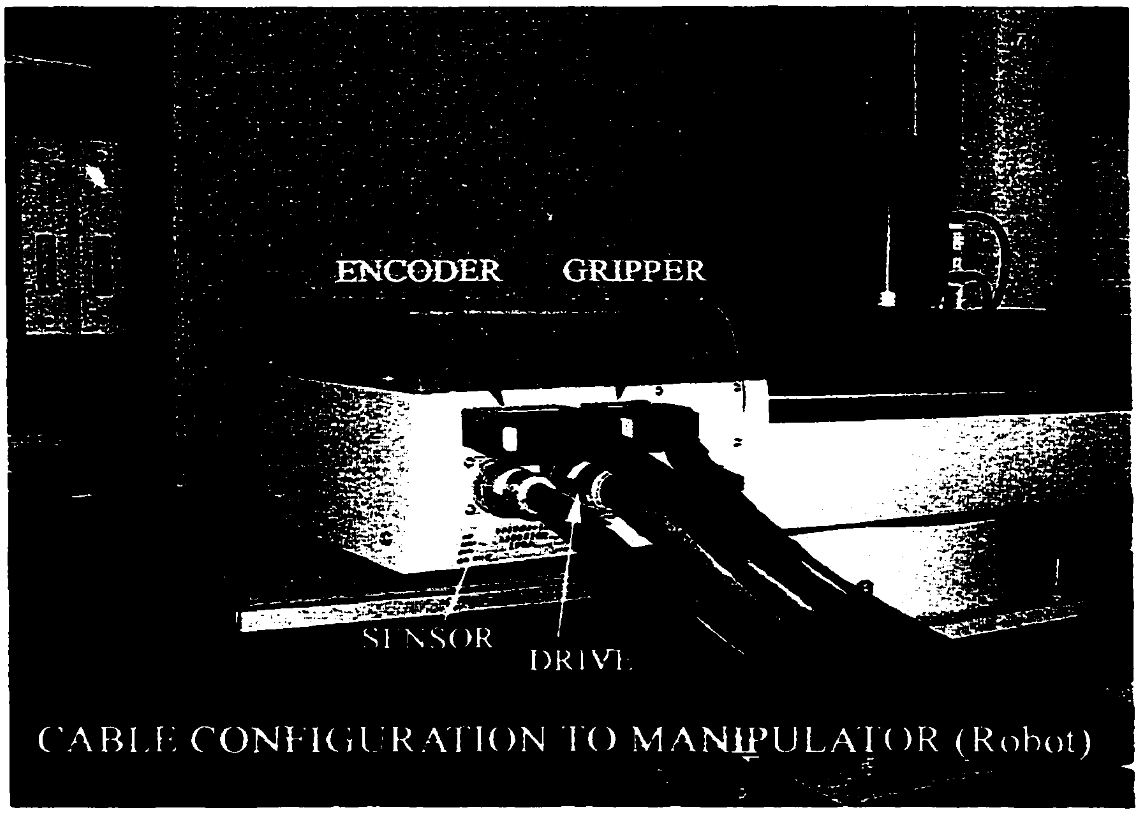


FIGURE 9. Controller cables (4) configured on the manipulator (robot)

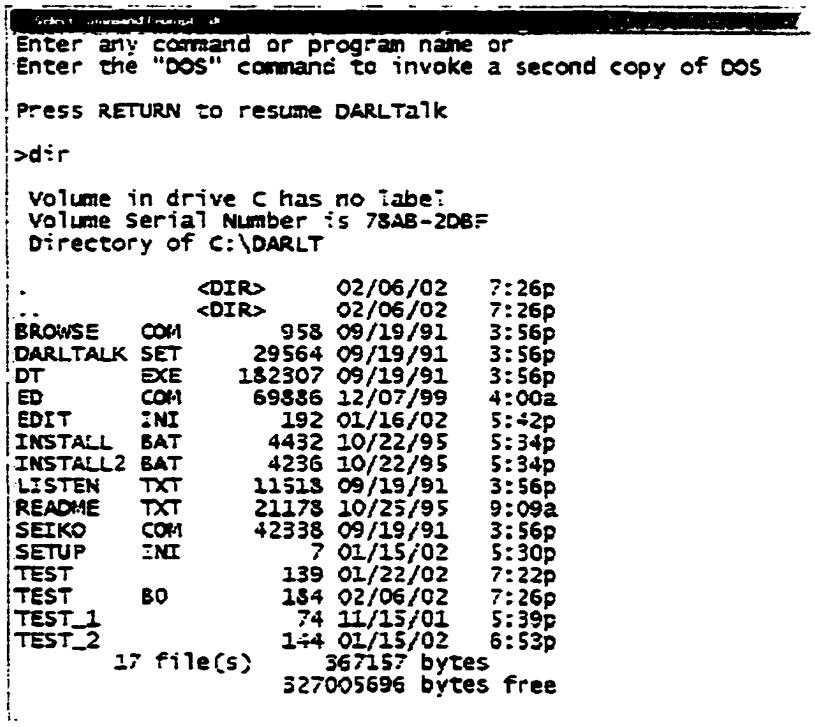
CHAPTER 4

DARLTalk INSTALLATION

DARLTalk Software to Operate Robot

The DARLTalk (D-TRAN Assembly Robot Language) software package obtained from Phil Baratti, EPSON Application Engineering Manager, was not available until some time after the robot system was assembled in the engineering laboratory. The software predated Windows 95 and ran in the DOS Shell environment. The software came with the following files: readme.doc, listen.doc, dt.exe, seiko.com, darltalk.set, browse.com, install.bat, and install2.bat. The Norton Editor made obsolete by DOS version 5.0 had to be copied directly from windows DOS version 5.0 or higher in to DARLTalk. DOS Editor edit.com, no longer included in DARLTalk software package, was copied and renamed ed.com in order for the EDIT menu in DARLTalk to function properly.

To install the software on to the host PC a new directory in the DOS environment was created called "DARLT" with all the DARLTalk files copied over into this new directory. The newly created directory for the DARLTalk software on the host PC as shown below (see Figure 10), which runs in the DOS shell mode. Any programs created are saved as default to the "DARLT" directory.



```
Enter any command or program name or
Enter the "DOS" command to invoke a second copy of DOS

Press RETURN to resume DARLTalk

>dir

 Volume in drive C has no label
 Volume Serial Number is 78AB-2D8F
 Directory of C:\DARLT

.             <DIR>        02/06/02    7:26p
..            <DIR>        02/06/02    7:26p
BROWSE   COM          958 09/19/91    3:56p
DARLTALK SET        29564 09/19/91    3:56p
DT       EXE       182307 09/19/91    3:56p
ED       COM        69886 12/07/99    4:00a
EDIT     INI          192 01/16/02    5:42p
INSTALL  BAT         4432 10/22/95    5:34p
INSTALL2 BAT         4236 10/22/95    5:34p
LISTEN   TXT        11518 09/19/91    3:56p
README   TXT        21178 10/25/95    9:09a
SEIKO    COM        42338 09/19/91    3:56p
SETUP    INI            7 01/15/02    5:30p
TEST                  139 01/22/02    7:22p
TEST     BO           184 02/06/02    7:26p
TEST_1                 74 11/15/01    5:39p
TEST_2                144 01/15/02    6:53p
        17 file(s)      367157 bytes
                     327005696 bytes free
```

FIGURE 10. Contents of DARLTalk directory on the C drive

The controller must be prepared to communicate with the host PC. To switch over from the Seiko Teach-Terminal keyboard to the host PC and still maintain similar control over the robot, with the most notable exception of the lack of manual jogging keys available, the following needs to be entered on the Teach-Terminal while in monitor mode.

```
DO COMM Z = 78 <ENTER>
DO OUNIT 1 <ENTER>
DO IUNIT 1 <ENTER>
```

Note: The "F1" key is a shortcut on the Teach terminal that will display on the LCD "DO OUNIT1: IUNIT1" then hit <ENTER>.

The Seiko D-TRAN control guide diagram (see Figure 11) shows the different commands permitted with the DARLTalk software. Each letter in the upper left-hand corner is a character prompt generated by the Seiko D-TRAN controller, which also indicates the current DARL operational mode. For example, from monitor mode "OM>" on the host computer type in "OM>WALK." This changes the operational mode to "OW>." To quit just type "OW>QUIT" to go back to monitor mode.

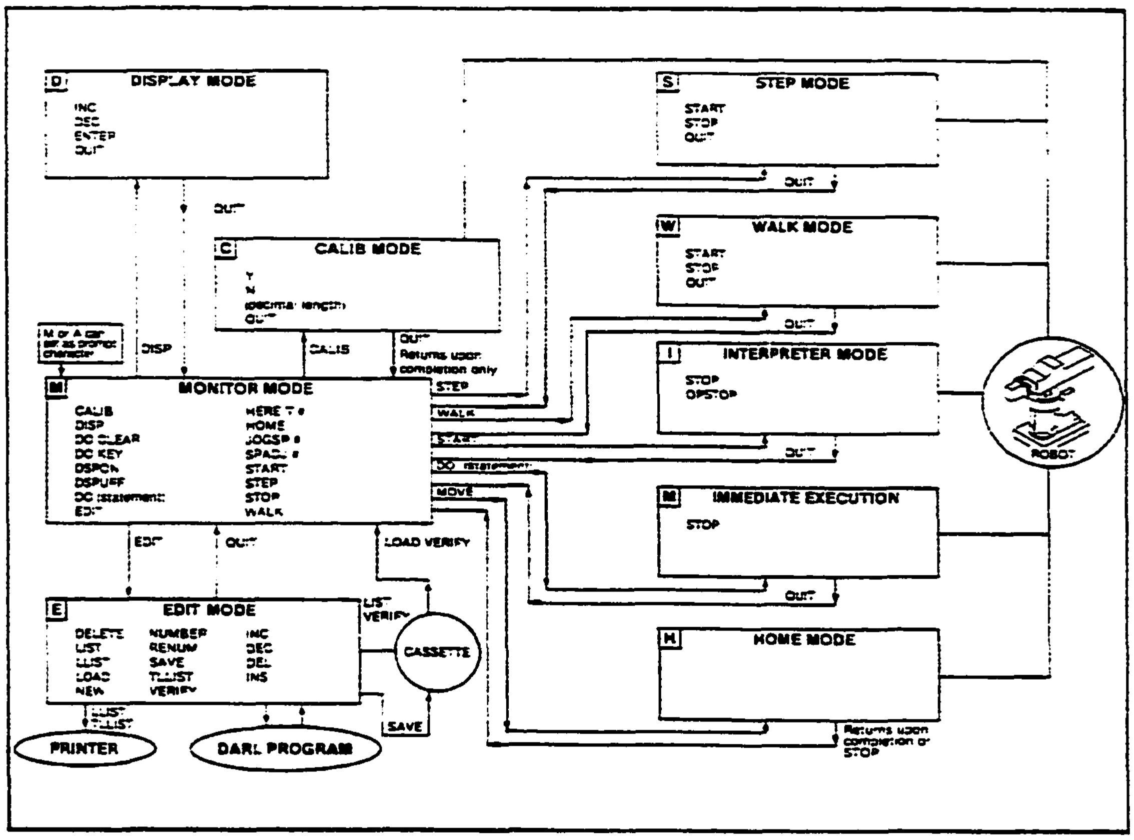
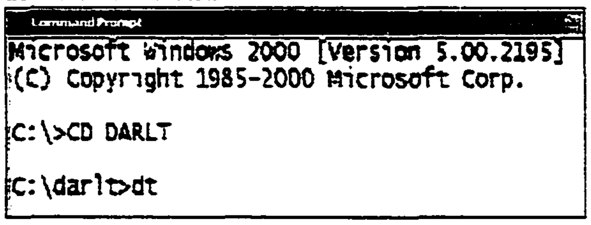


FIGURE 11. Seiko D-TRAN control guide--character prompts illustrated

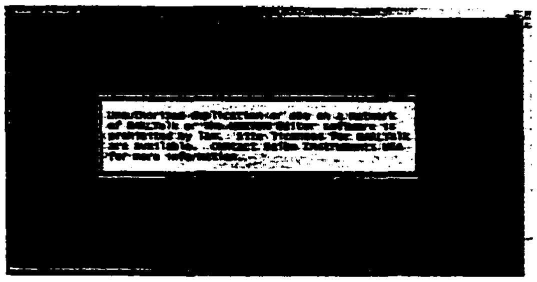
Getting Started with TALKTalk

Open the DOS command prompt on the PC located under Accessories and type "darlt" to get into the directory then type "dt" and hit return which starts the program.

Hit any key on the PC to get to the DARLtalk Menu.



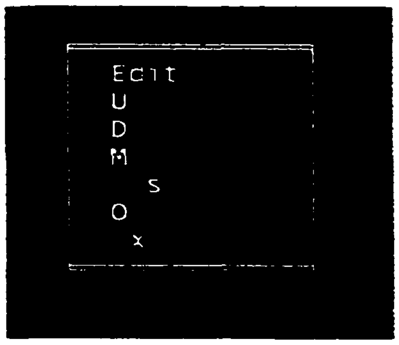
```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>CD DARLT

C:\darlt>dt
```

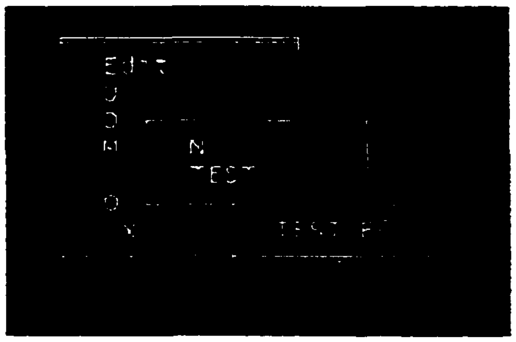


GRAPH 1. DOS command prompt

Description of Menu Items

The menu items, which contain arrowheads to the right, have submenus except when in monitor mode. If at any time you wish to leave a submenu with or without making any data entries or selections hit the ESCAPE key on the PC.



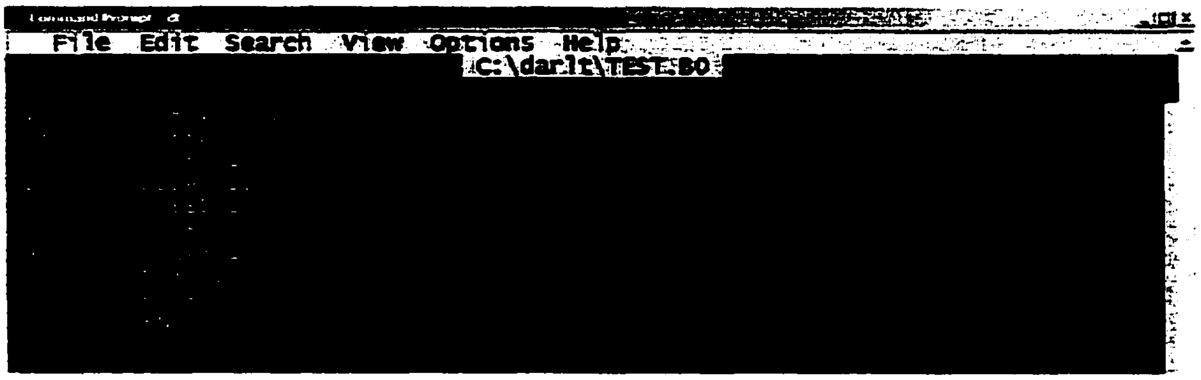


GRAPH 2. DARLTalk menu and submenu items

Hit the letters on the keyboard that are highlighted or use the arrow keys to make the entire selection be highlighted in red then hit RETURN to see submenu.

Running Test Program

Listed in the edit submenu is a test program named "TEST.B0" which was created to demonstrate the operation of the manipulator (robot). All user created files such as the test program shown in the edit submenu have a ">B[0-6]" or a ".TPT" file extension residing in the DARLTalk directory. Files with the "B[0-6]" extension signify programs with a corresponding memory bank (0 through 6) that the controller CPU uses. The controller CPU memory banks have an allotted memory capacity of 8K. This enables the user to write programs up to 56K or store seven (7) independent programs in the memory banks.



GRAPH 3. DARLTalk test program

The particular program (TEST.B0) shown above repeats a sequence of code a specified number of times. Line

<L10> syntax FOR <V> and line <L50> NEXT <V> statements refer to the same variable "I" with an algebraic expression "0 TO 5" which increments the variable from its initial value, determined from the first integer, until the value of the second integer is reached. Once the upper range of the value is reached after repeating lines <L10 through L50>; the statement following the NEXT statement will be executed. Line <L15> syntax SPEED <1-300> statement specifies the speed for motions determined by the MOVE statement with a default speed of 100. The scale <1-300> is relative to the closed loop servo system and does not specify a velocity in mm/sec. Line <L20> and line <L35> syntax MOVE T<0-399> instructs the robot to use linear interpolation to a specified location in space in reference to any defined coordinate system. The format for the translation or T# address is as follows: All values are expressed in millimeters or degrees.
T<0-399> = <X-Coordinate> <Y-Coordinate> <Z-Coordinate> <A-Coordinate>

The translation points specified in this program are uploaded from the Teach Terminal Keyboard. This method of assigning translation points using the jogging keys on the Teach terminal keyboard (see Figure 12) is useful from the standpoint of not having to change the basic test program whenever the work base needs modification. Hit the stop key on the Teach Terminal keyboard. This puts you back to the monitor mode on the Teach Terminal keyboard and kicks you out of monitor mode from the host computer.

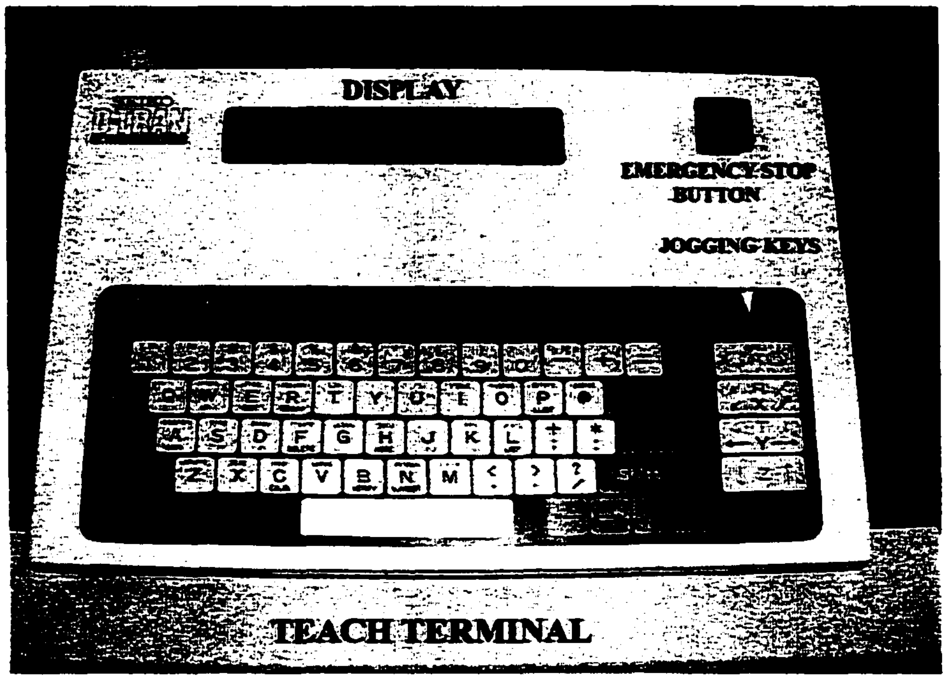
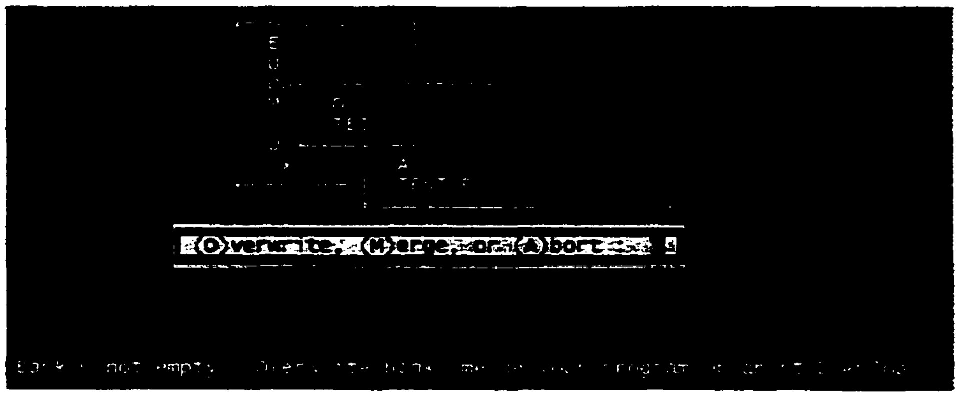


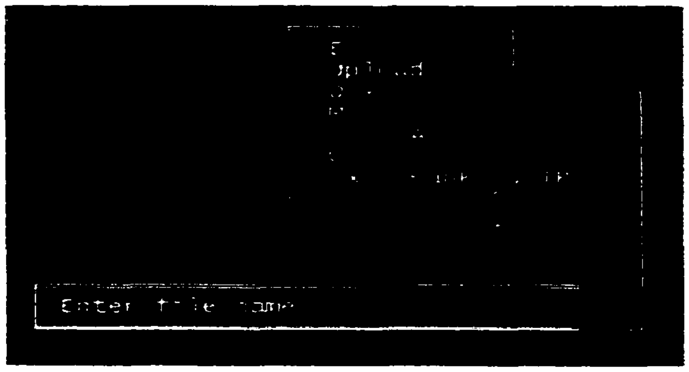
FIGURE 12. Teach-Terminal keyboard

Press the jogging keys on the Teach Terminal to train or position the robot in the 4 available axes of manipulation. Once the operator is satisfied with coordinates or location of the robot the host computer will automatically display the values of the coordinates. Then from the Teach Terminal type "0M>HERE T1" <RETURN>. This assigns the current robot coordinate position relative to the HOME frame of reference to the specified translation point (T1). Repeat this task for the remaining translation points listed in the "TEST.B0" program. After all the translation points are assigned, return the monitor mode on the Teach Terminal back over to the host PC. The monitor mode prompt will reappear back to the DRALTalk program. Select "Download", "TEST" or "Other File" and "TEST.BO." Since the program was previously downloaded before, the program remembers the file name in this case. Type "O" in the last prompt to overwrite anything that is still in memory in Bank (0). Hit the Escape on the PC keyboard to back out of the submenus back to the main menu.


(O)verwrite, (M)erge, or (A)bort

GRAPH 4. DARLTalk download submenus

Select "Upload" and from submenu select "Banks & TPTs." DARLTalk will ask for a file name--enter Program name "TEST.B0." This will upload the translation points (TPTs) from the controller to the PC.


Enter file name

GRAPH 5. DARLTalk upload submenus

After the translation points have been loaded use Escape on the keyboard to back out to the main menu and then select "Monitor" to put you in the monitor mode.

From monitor mode prompt type "HOME." This will command the robot enter the HOME operational mode and move to its factory set HOME position which is also the default frame of reference (FRAME 0) or origin of the Cartesian coordinate system. During execution of the HOME statement while the robot moves to its respective factory set HOME position homing error messages occurred. Error statement numbers 218 "Z-Phase margin too long at A-Axis" and 219 "Z-Phase adjust needed (out of range)." The corrective action was to check the tightness of the home sensor and magnet for the A-Axis (see Figure 13). The limit sensor bracket is loosened and slid back and forth for adjustment. The HOME statement is executed again to see if the error has been corrected. Sometimes an error statement may indicate that the "Z-Phase margin is too short at A-Axis." Repeat adjustment until executed HOME statement returns with no errors. DARLTalk software will not allow operator to run programs until HOME statement is returned error FREE.

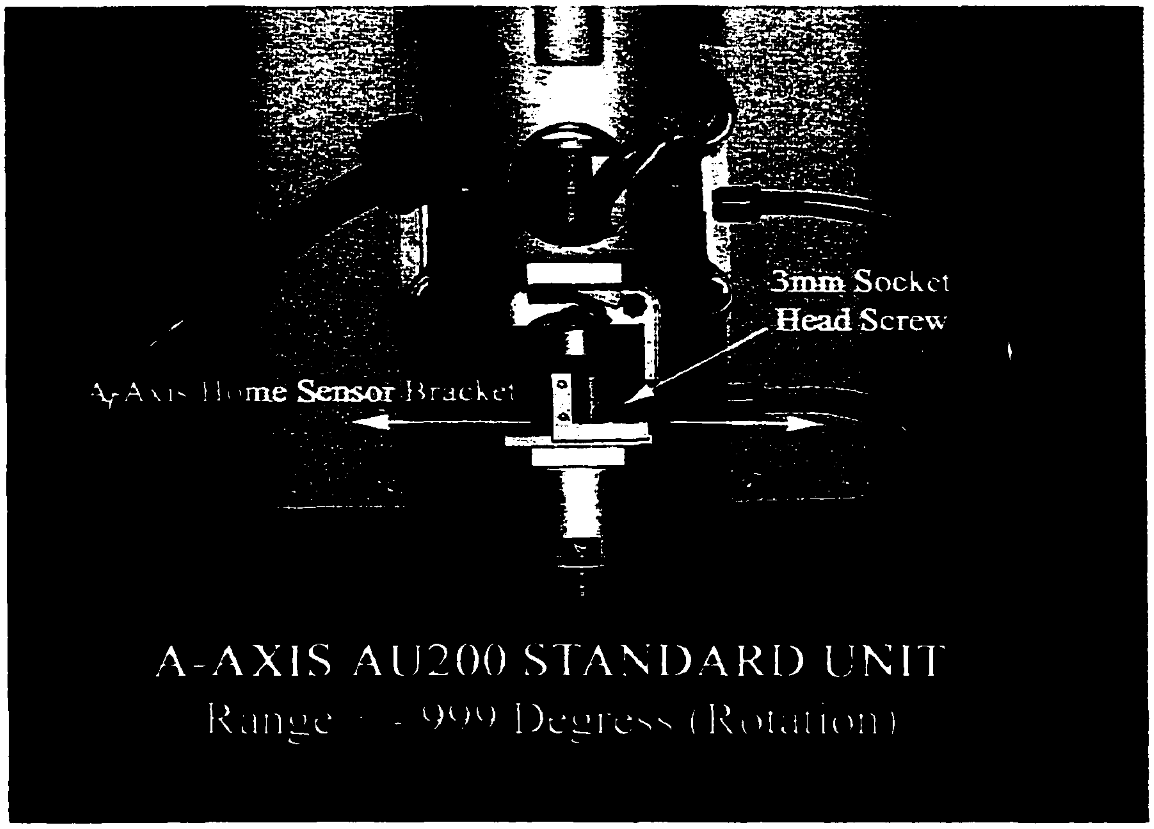
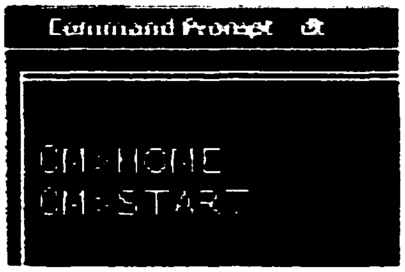

3mm Socket
Head Screw
A-Axis Home Sensor Bracket
A-AXIS AU200 STANDARD UNIT
Range - 999 Degress (Rotation)

FIGURE 13. A-Axis limit sensor adjustment

After HOME execution is completed type in "START" and hit RETURN key to start the called "TEST.B0." When the program completes the loop back repeat statement the next statement, in Line (L55) syntax HOME, returns robot back to HOME factory set position. Finally the program ends with Line (L60) syntax END which stops execution of the program and returns the controller back to monitor mode prompt (OM>) (see Graph 6).



```
CM>HOME
CM>START
```

If the operator wants to log off from the DARLTalk program hit the "End" on the keyboard to get back to the menu items and select "Exit." This will put you back in the DOS shell

GRAPH 6. DARLTalk monitor mode screen

CHAPTER 5

BACKGROUND IN 2D-IMAGE SCANNING

Picture Elements

The objective is to obtain a scanned 2D-image represented in a bitmap made up of pixels. Discrete pixels are picture elements; tiny square dots of individual color that you see on a screen that come together to form an image like a photograph. Pixel is an abbreviation that is short for picture element. However, the 2D-image obtained for this project will be a scanned signature of lines and curves made up of black and white pixels. This area covers the ASCII (pronounced "ask-kee") industry standard file format visualized on the screen for scanned images as used in the beginning step in the process of data conversion.

A pixel is screen-dependent; that is, the dimensions of screen elements vary with the PC display system and resolution. For instance, a scanned signature (see Figure 14) would be stored as black pixels that make up the image in a bitmap graphics format. Here the need is to focus on a way to convert a bitmap graphic image to a postscript data file such as an ASCII file of ones and zeros to be used as the common denominator for data conversion to meaningful coordinates for the robotic cell to understand.

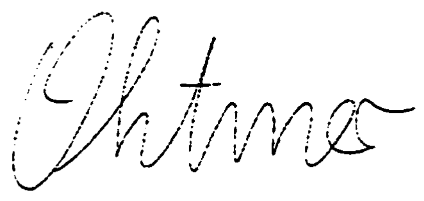


FIGURE 14. Scanned signature

Vector vs. Bitmap

Graphics fall into to two main categories: Vector graphics and Bitmap graphics. When looking at the 2 formats side by side it is hard to tell differences until you enlarge the formats. In other words, a bitmap file cannot create anymore pixels when enlarged that is already there. To compensate it will enlarge the pixels and the grid. On the other hand the vector format can regenerate on the fly when enlarged depending on the starting and ending coordinates. Thus, the image looks the same no matter what size.

One of the objectives with scanning a 2D-image is to convert the graphics image from a bitmap into meaningful data for the robotic cell to understand. (Unfortunately you can not save a scanned image as a vector file.) For instance, the bitmap, when enlarged, shows the jagginess of the edges or pixelation as it is called because the computer monitor can only display graphics using pixels. Conversely, when a vector image is shown enlarged the edges become smooth. With a so-called resolution independent vector image you can increase or decrease the size and the lines will remain crisp and sharp more like a cartoon image. Unlike bitmaps, vector images have advantages, but the primary disadvantage is that they are unsuitable for producing photo-realistic imagery. This is because vector images cannot depict the continuous subtle tones of a photograph. Vector and bitmap objects look the same on a white background but placed over other objects (see Figure 15) the differences are apparent. The bitmap circle has a rectangular box around it. Therefore, with the vector object defined by mathematical equations or the bitmap defined by pixels the attributes can be transformed into meaningful data for the robotic cell to understand.

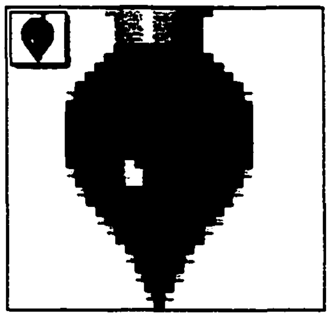
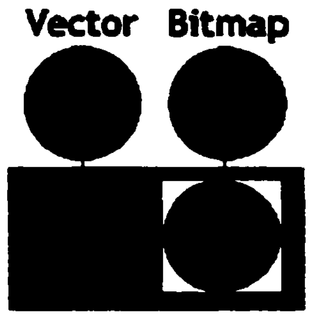



FIGURE 15. Bitmap image close-up and comparison

X and Y Coordinates

Instead of mapping every pixel like a bitmap image does, for instance; x, y, and the color of the pixel, the vector image uses mathematical equations to define sections or vectors of the image. Unlike under the bitmap taking 3 coordinates for every pixel the vector takes two measurements and a function to draw the object which would look like: X1, Y1, X2, Y2, Function.

X1, Y1 are the starting points of the object.

X2, Y2 are the ending points of the object.

Function is the function used to create the object.

For example, to draw a straight line as in figure 16 the equation would look like: 1,8, 14, 8, If X1 < X2 Then X1 = X1+1 Loop.

1, 8 is the starting point of the line.

14, 8 is the ending point of the line.

If X1 < X2 Then X1 = X1+1 Loop is the equation to define the line. Basically it says if the X1 is less than X2 then add 1 to X1 and draw in that set of coordinates on the computer screen. Loop tells the computer to run the equation again, asking if X1 is less than X2. This cycle will continue until X1 = X2. In this case 14 terminates the Loop.

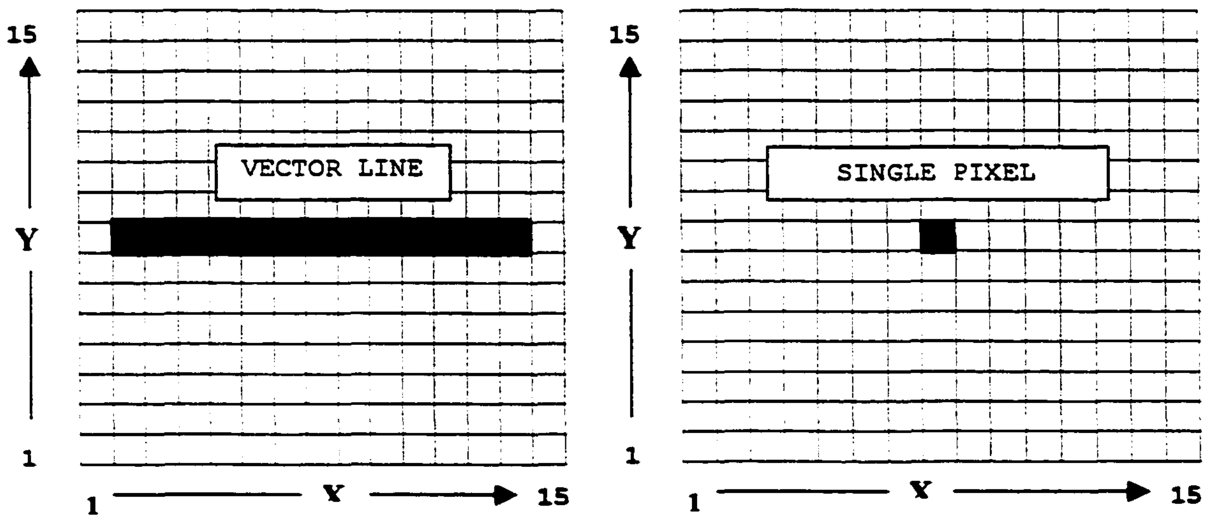


FIGURE 16. 15x15 pixel grid

Note: For the bitmap image like the name sounds, it is a map of bits of an image. Unlike the vector image that uses mathematical equations the bitmap maps out every x and y coordinate along with every pixel color. For example, the single pixel image shown in figure 16 on a 15x15 pixel grid is graphically written as 8,8,1. Where the 8 is the x-coordinate, the second 8 is the y-coordinate, and the 1 representing the color black for the pixel.

Coordinate Points Transferred to Twips

To translate the pixel objects from the 2-dimensional grid coordinates to meaningful measurements for the D-TRAN robotic cell to read, a screen-independent unit called Twips is used. A twip is a unit of screen measurement equal to 1/20 of a printer's point (20 points equal a twip), and there are 1,440 twips per inch (the length of a screen item measured one inch when printed).

Example of points transferred to Twips:

20 * Points = Twips

Then . . . 10 * Points = 20 * 10 Points = 200 Twips

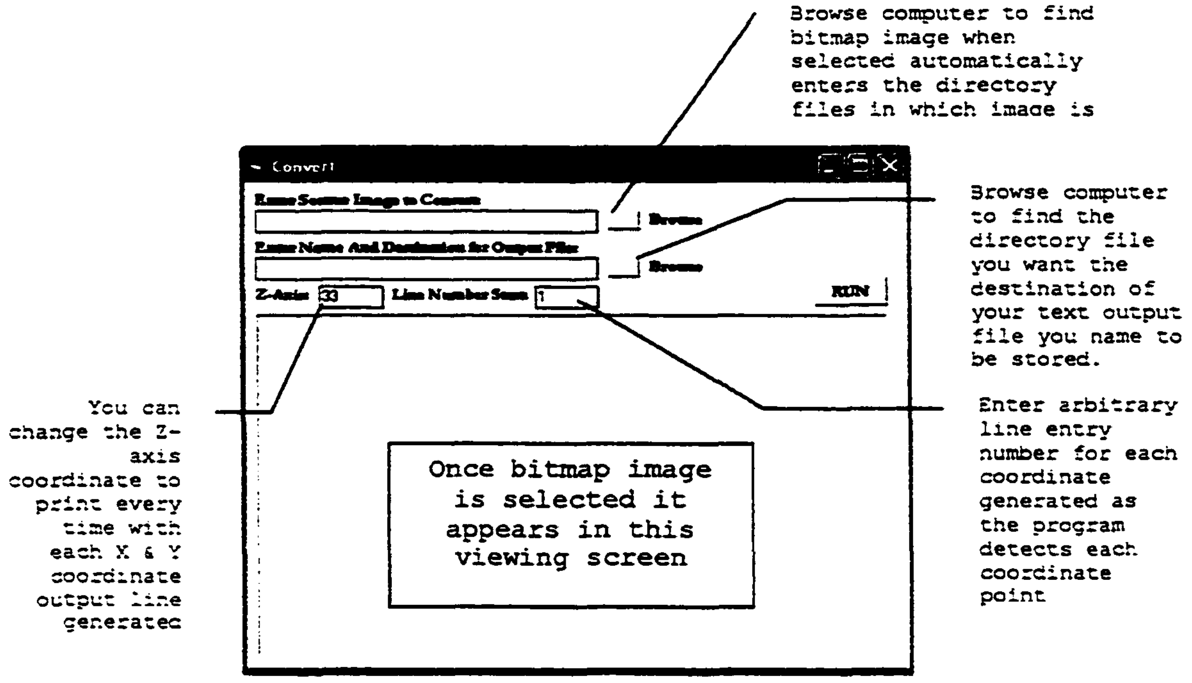
Conversion between Twips and inches/centimeters/millimeters is as follows: (the length of a screen item measured one inch when printed)

1. There are approximately 1,440 twips to an inch.
2. As there are 2.54 centimeters to 1 inch, then there are approximately 567 twips to a centimeter.

3. As there are 25.4 millimeters to 1 inch, therefore there are approximately 56.7 twips to a millimeter.

First Attempt at Writing a Program

My first program attempt to create a new program for this thesis project (see Figure 17) was written in visual basic but did not meet project requirements. The first program application attempt used window based navigation buttons to input the bitmap image file to be converted and a button on the screen to rename the output (converted) data point text file. The "Convert.c" program basically scanned the bitmap image top to bottom, left to right, reading only the black pixels and converting them to X & Y coordinates. However, this program lacked the basic conversion of the bitmap image to an array and the implementation of the detection frame or pattern recognition logic to generate the correct trace sequence of the image. The data point output text file that is program created was formatted in the way that the robot controller can read. Without the proper implementation of the pattern recognition code the data point output text file was organized in a scanning pattern and not in the correct trace of the image. The "convert.c" program was not used for this project and was later replaced by the new Postscript File Scan program (see Appendix D) incorporating the improved pattern recognition logic source code.


Convert
Enter Source Image to Convert
Browse
Enter Name And Destination for Output File
Browse
Z-Axis
33
Line Number Start
1
RUN

Browse computer to find bitmap image when selected automatically enters the directory files in which image is

Browse computer to find the directory file you want the destination of your text output file you name to be stored.

You can change the Z-axis coordinate to print every time with each X & Y coordinate output line generated

Enter arbitrary line entry number for each coordinate generated as the program detects each coordinate point

Once bitmap image is selected it appears in this viewing screen

FIGURE 17. Visual Basic "convert.c" program (not used for this project)

CHAPTER 6

BAUMAN'S COMPUTING PROCESS

Program Background

A graduate student by the name of Ronald C. Bauman, under the direction of Dr. Ortwin Ohtmer, wrote 2 programs back in 1989. The first programming module called "Postscript_bmp_Convert.c" intended to take a bitmap image file which is a picture-description language, and convert it into a postscript binary ASCII array file of 1s for black pixels and zeros for white background pixels. However, not being completely familiar with the type of postscript processing used at the time, I began to fix several problems with the program code to get it to compile. The bottomline after several attempts to work with the code the program could not process any bitmap.

The last program called "Binary_Array_Scan.c" takes the newly created binary ASCII array file (of 1s and 0s) and rescans it from top to bottom. Using the detection frame logic of the program to trace the image array, a data set file of coordinate data points is created. This data set file was formatted for downloading to another programming module that generated NC code for a drilling machine. (This additional programming module is not discussed in his report.) However, not understanding the code completely the "Binary_Array_Scan.c" program still remained problematic before and after it was fixed to compile. In other words it met the same fate of the "Postscript_bmp_Convert.c" program.

<u>Program: Postscript BMP Convert (see Appendix E)</u>

The program essentially opens the postscript file and reads the first byte-by-byte until it finds a 0. When a zero is read, it loops and reads more bytes for hex values ('0'-'f') and outputs the binary equivalent ('0000'-'1111') to the second newly created binary file. The loop continues until a new-line character is found ('\n') or a max of 4,000 bytes is reached and then search for a zero character resumes in the first file.

If a 0 is not read by the "Postscript_bmp_Convert.c" program, the program reads until a new-line character is found, and then checks to see if any 0s were read earlier (a = 1).

Several of the problems found with Bauman's "Postscript_bmp_Convert.c" program "which kept it from running" are described below:

a) No semi-colons after the a = 1 statement.

b) Semi-colons added after hex-check if() statements. [Rationale: I am pretty sure the original author wanted the binary output to run based on the hex value check. The way it was written originally, the if() statements were ignored, and ALL the binary output statements ran.]

c) Added ch==EOF and ch==255 to the checks for new-line. [Rationale: These are checks to End-of-file conditions and should always be present in some way in the file-read loops.]

<u>Program: Binary Array Scan (see Appendix F)</u>

After the "Postscript_bmp_Convert.c" program creates the ASCII array file of 1s and 0s, it then needs to be mapped or inputted again through the "Binary_Array_Scan.c" program (see Appendix F for cleaned-up version). This program was poorly written with many 'goto' statements and large static, global arrays. At least there were a few comments in the program in the code to help follow the logic.

The "Binary_Array_Scan.c" program is basically written to call several functions from a main routine, even though no arguments were passed, and all of the work was done on the same global array. This means, the code in the lower level functions can be copied up to the main routine and merged together to make one large, function-less program. This is what was done in the cleaned up version (see Appendix E). The code should work the same as the original except without the functions. Many of the labels had to be renamed with slight changes to the logic, but it complies. I tried to keep all the original comments and even added a few more to separate the now-merged code that formerly was hidden in separate functions.

CHAPTER 7

THESIS IMAGING AND REPRODUCTION PROGRAM: USING SCANNING AND PATTERN RECOGNITION PROCESS

Scanned Image

To create the graphical data point file from a scanned image which is required for the Robotic cell, a new computer scanning program had to be put together. The process begins with creating an image or handwritten signature on paper. For example, see sample problem 1, Figure 18, which is an arrow pointing down created by windows paint utilities program and see sample problem 2, Figure 19, which is a scanned handwritten signature using a typical computer scanner product. Both of these images are saved as Bitmap files.

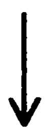


FIGURE 18. Arrow image

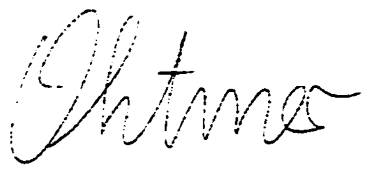
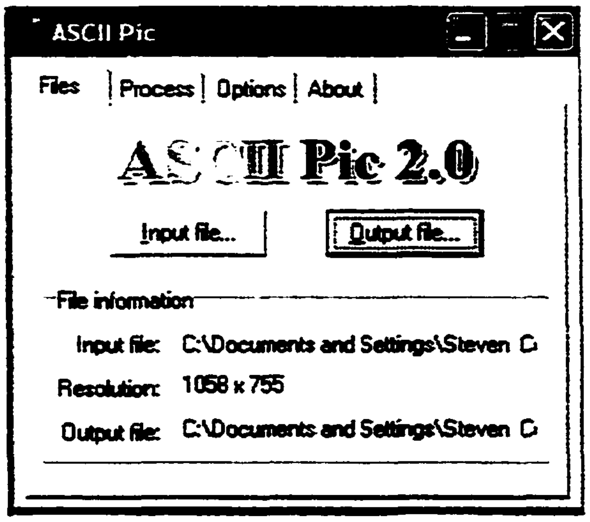


FIGURE 19. Handwritten signature

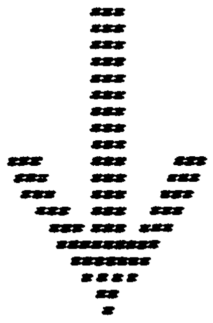
ASCII Conversion

Once the bitmap image is created it needs to be converted into an ASCII text file; pseudo-grayscale image composed of various ASCII characters. [This was the only thing missing that I discovered was needed for the new ps_scan.exe program requiring a postscript file.] Each pixel of the original Bitmap (*.BMP) image will be represented by a text character in the output text (postscript) file. The resulting text file (see Figure 20) will have as many rows and columns as width and height in pixels of the original bitmap image. To accomplish this I found free software off the Internet called "ASCII PIC Version 2.0" [Website URL: http://w3.to/5679soft yingkit@iname.com] which creates this ASCII text file. Below the "ASCII PIC 2.0" program you input the bitmap file and name the output file. The Process tap is than selected to execute the conversion.



Input file: C:\Documents and Settings\Steven C
Resolution: 1058 x 755
Output file: C:\Documents and Settings\Steven C

(refer to sample problem 1)



Arrow Generated by
ASCII PIC 2.0 Program

FIGURE 20. Arrow postscript ASCII text file (arrow.txt)

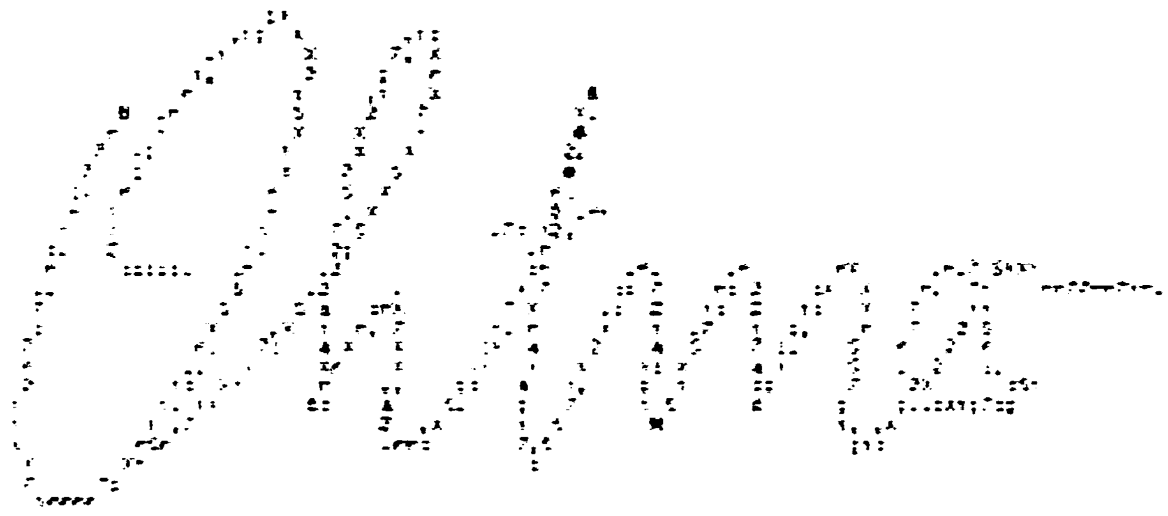
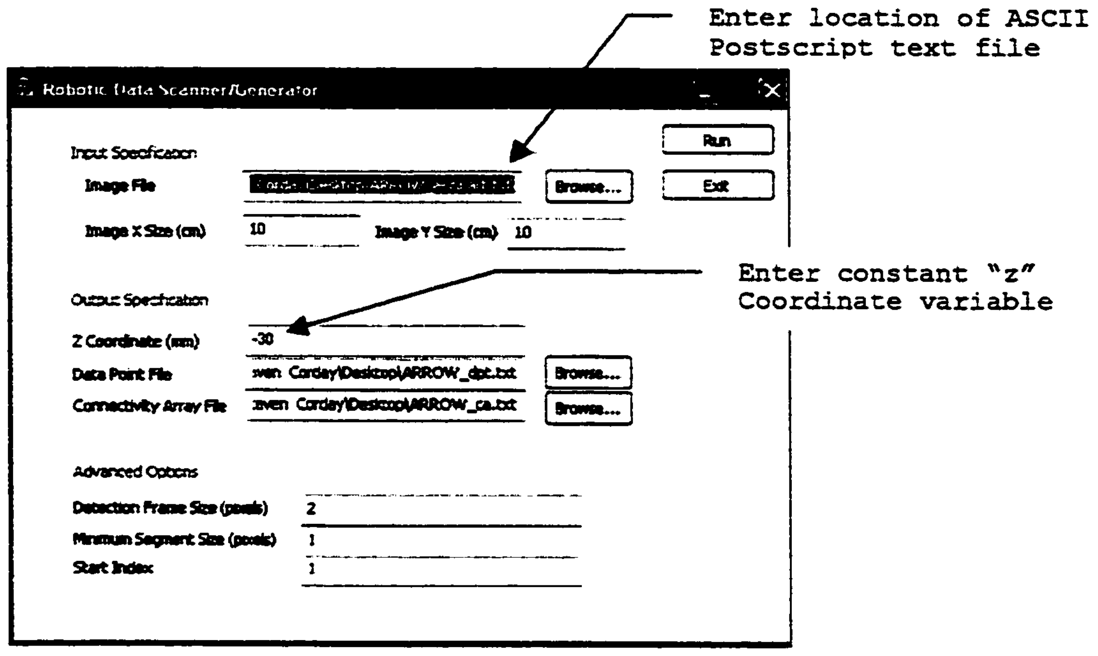


FIGURE 21. Signature postscript ASCII text file (oht.txt)
(refer to sample problem 2)

Data Scanner/Generator

The ASCII text character file created from the "ASCII PIC 2.0" program is then used as the input postscript file to be read by the new Postscript File Scanner program "Robotic Data Scanner/generator" as shown below.


Enter location of ASCII Postscript text file
Enter constant "z" Coordinate variable
Robotic Data Scanner/Generator
Run
Exit
Browse...
Browse...
Browse...

Input Specification
Image File [illegible]
Image X Size (cm) 10 Image Y Size (cm) 10

Output Specification
Z Coordinate (mm) -30
Data Point File sven Corday\Desktop\ARROW_dpt.txt
Connectivity Array File sven Corday\Desktop\ARROW_ca.txt

Advanced Options
Detection Frame Size (pixels) 2
Minimum Segment Size (pixels) 1
Start Index 1

FIGURES 22. New postscript file scan application program (robotic data scanner/generator)

The new Postscript File Scan program combines the following: (1) the transformation of the ASCII text input file into an ASCII array file consisting of the digit "1" for black pixels and the digit "0" for white pixels outputted as a dump file, (2) the connectivity array file, and (3) the pattern recognition code to write the x-y-z data point file. Advanced options programmed into the new Postscript File code corrected the problems the old modularized programs had. These problems consisted of not being able to accept any bitmap no matter the size or color of the pixel, line thickness of the image and closed loops. Also, unexpected was when the program failed on different images with no branch points similar to a circle during debugging. The problem appeared to be memory related. The operating system did not allocate and release memory as expected. This is why Bauman had problems with large number of black pixels. The fix was to allocate and then release a small amount of memory many times in this case.

Once the ASCII text character file of the image is scanned by the Postscript File Scan program it is transformed or expanded internally into a binary array file visualized in the dump file produced. During the pattern recognition phase the program starts with the detection frame logic of closed loop meshes, closed loops consisting of data points (i.e., 1s and 0s) connected. Once detected, the data points and the members comprising the thickness of the mesh are stripped leaving only the center connectivity array file listing each pixel grid member value and the inputted z-coordinated. This basically leaves only the smallest substructure or trace image behind. The detection frame continues the trace of the image converting all 1s to 0s. When the detection frame encounters a branch it picks one of the legs to continue on until it ends and goes back to the beginning of the branch and continues on with the trace segment. This process continues until the binary array (dump file) is completely scanned and void of any black pixels.

Two major items fixed with the new Postscript File Program is the ability to input the size of the detection frame and segment size which is also corrected automatically if the values are wrong (see Figure 24). By changing the detection frame size the width of segment lines and the angle between branch points are capable of being scanned avoiding aforementioned oscillation. As a general rule it should be mentioned that handwritten images which have the thinnest line widths and largest geometry's yield the most accurate data point and connecting member array. The geometric angle between branch points as they decrease the detection must also decrease or the branch point area will become distorted and riddled with gaps.

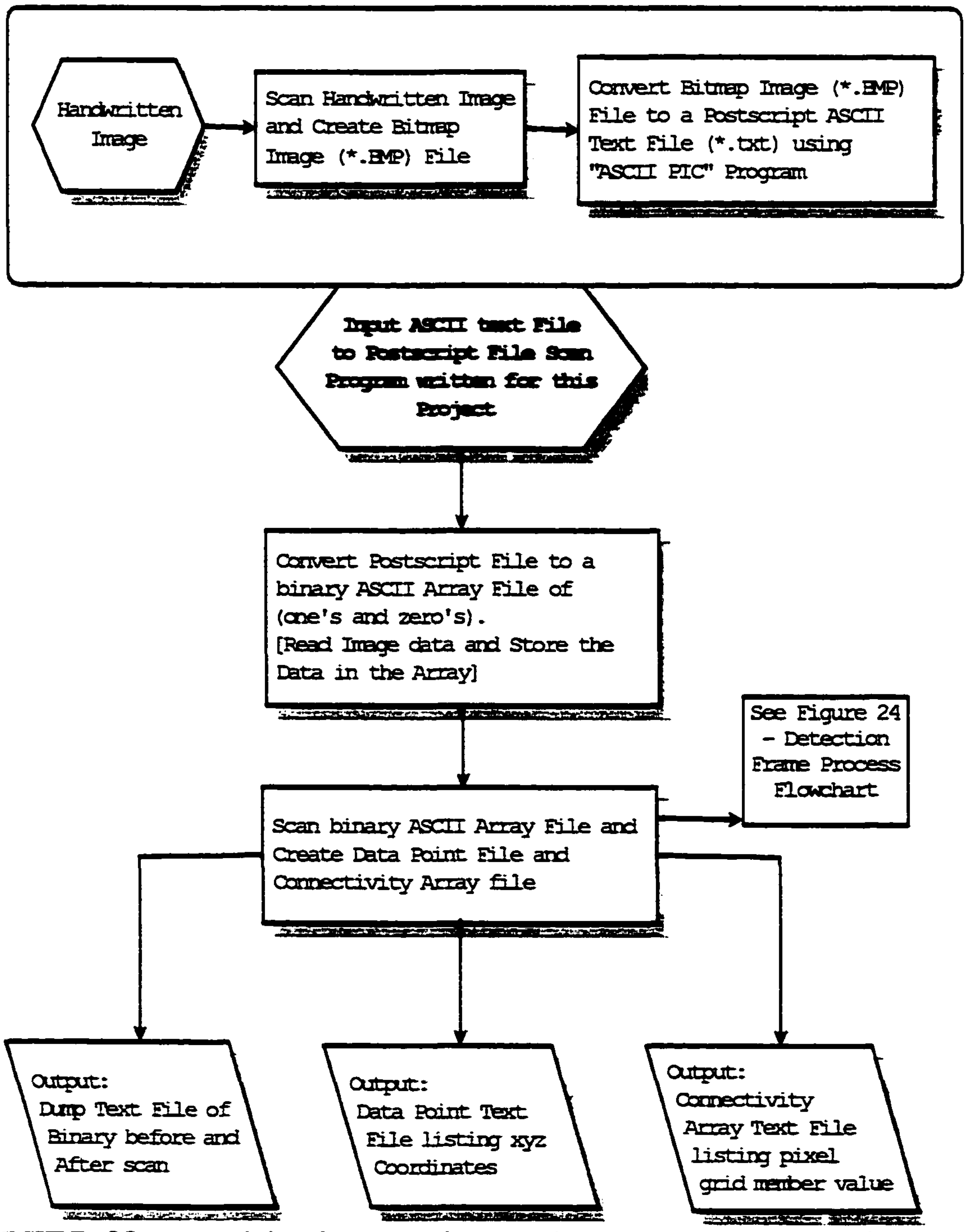

Handwritten Image
Scan Handwritten Image and Create Bitmap Image (*.BMP) File
Convert Bitmap Image (*.BMP) File to a Postscript ASCII Text File (*.txt) using "ASCII PIC" Program
Input ASCII text File to Postscript File Scan Program written for this Project
Convert Postscript File to a binary ASCII Array File of (one's and zero's). [Read Image data and Store the Data in the Array]
See Figure 24 - Detection Frame Process Flowchart
Scan binary ASCII Array File and Create Data Point File and Connectivity Array file
Output: Dump Text File of Binary before and After scan
Output: Data Point Text File listing xyz Coordinates
Output: Connectivity Array Text File listing pixel grid member value

FIGURE 23. Graphical scanning and pattern recognition process flowchart

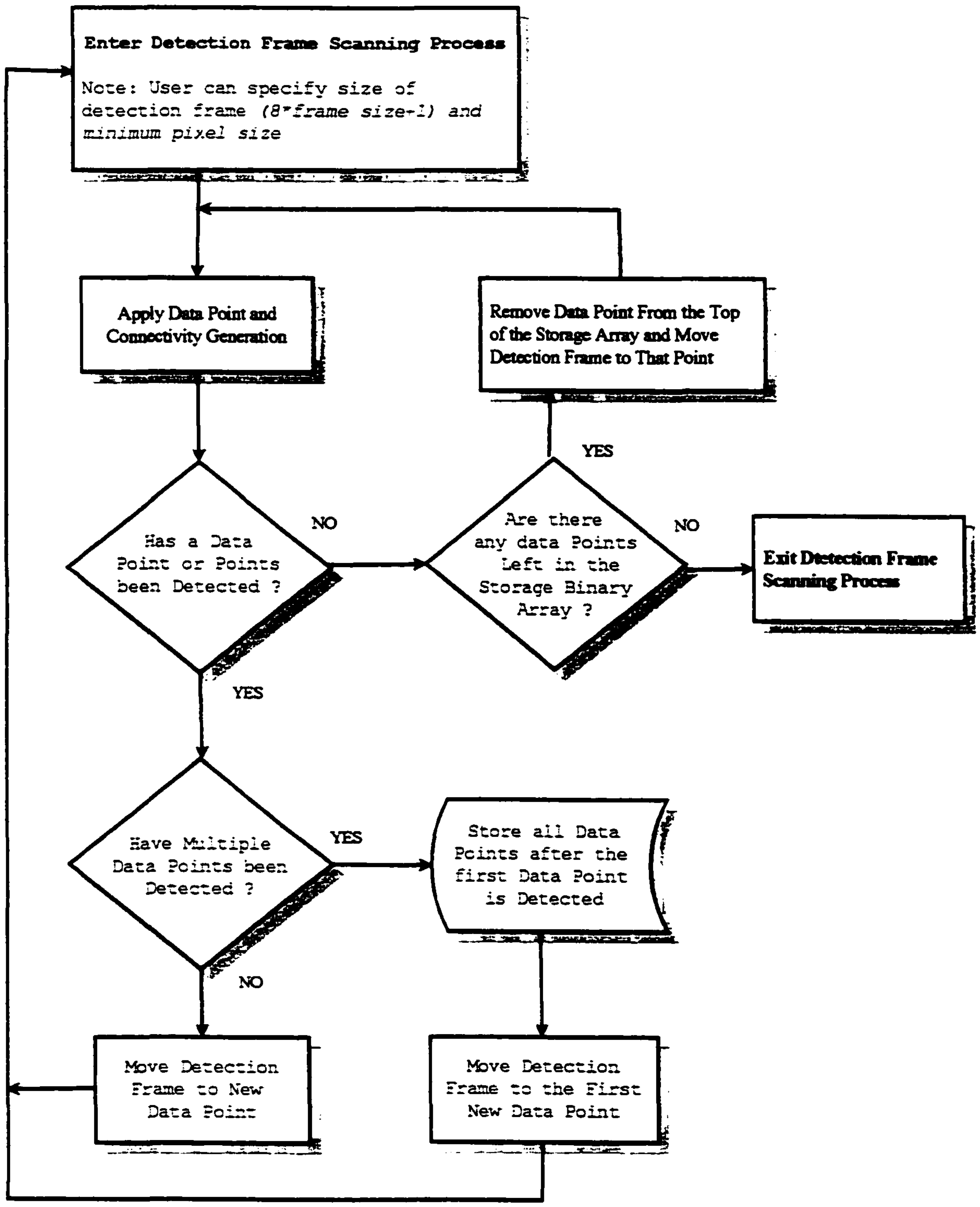

Enter Detection Frame Scanning Process
Note: User can specify size of detection frame (8*frame size+1) and minimum pixel size
Apply Data Point and Connectivity Generation
Remove Data Point From the Top of the Storage Array and Move Detection Frame to That Point
Has a Data Point or Points been Detected ?
NO
YES
Are there any data Points Left in the Storage Binary Array ?
YES
NO
Exit Detection Frame Scanning Process
Have Multiple Data Points been Detected ?
YES
NO
Store all Data Points after the first Data Point is Detected
Move Detection Frame to New Data Point
Move Detection Frame to the First New Data Point

FIGURE 24. Detection frame scanning process flowchart

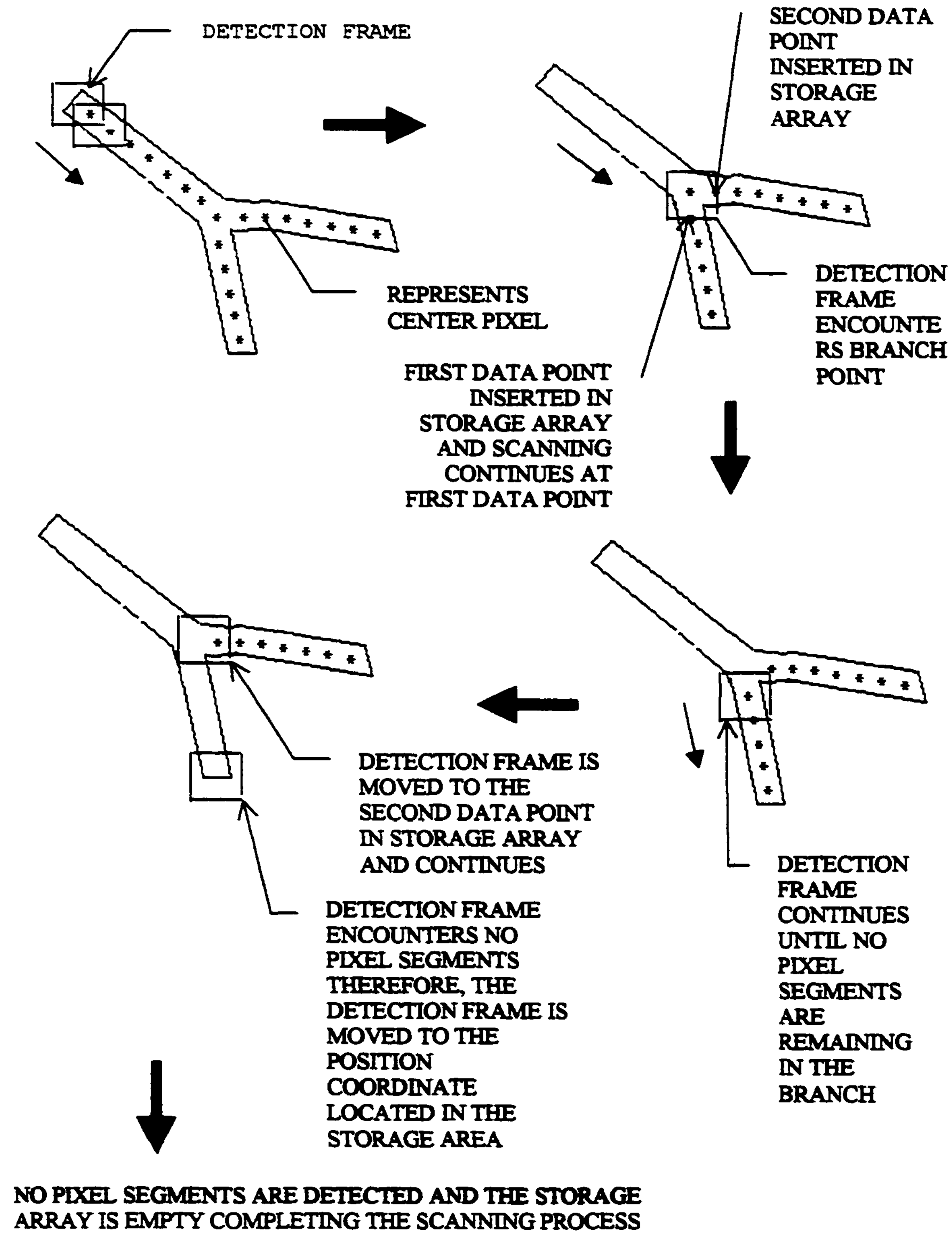

DETECTION FRAME
SECOND DATA POINT INSERTED IN STORAGE ARRAY
REPRESENTS CENTER PIXEL
DETECTION FRAME ENCOUNTERS BRANCH POINT
FIRST DATA POINT INSERTED IN STORAGE ARRAY AND SCANNING CONTINUES AT FIRST DATA POINT
DETECTION FRAME IS MOVED TO THE SECOND DATA POINT IN STORAGE ARRAY AND CONTINUES
DETECTION FRAME CONTINUES UNTIL NO PIXEL SEGMENTS ARE REMAINING IN THE BRANCH
DETECTION FRAME ENCOUNTERS NO PIXEL SEGMENTS THEREFORE, THE DETECTION FRAME IS MOVED TO THE POSITION COORDINATE LOCATED IN THE STORAGE AREA
NO PIXEL SEGMENTS ARE DETECTED AND THE STORAGE ARRAY IS EMPTY COMPLETING THE SCANNING PROCESS

FIGURE 25. Schematic of multiple pixel detection frame process

```
***Initial Image
000000000000000000000000
000000000000000000000000
000000000001000000000000
000000000001000000000000
000000000001000000000000
000000000001000000000000
000000000001000000000000
001000000001000000001000
000100000001000000010000
000010000001000000100000
000001000001000001000000
000000100001000010000000
000000010001000100000000
000000001001001000000000
000000000101010000000000
000000000011100000000000
000000000001000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
***END Initial Image
```

→

```
***After scan
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
001000000000000000001000
000100000000000000010000
000010000000000000100000
000001000000000001000000
000000100000000010000000
000000010000000100000000
000000001000001000000000
000000000100010000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
***END After scan
```

→

```
***After scan
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000001000
000000000000000000010000
000000000000000000100000
000000000000000001000000
000000000000000010000000
000000000000000100000000
000000000000001000000000
000000000000010000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
***END After scan
```

↓

```
***After scan
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
000000000000000000000000
***END After scan
```

FIGURE 26. Detection frame scan process of arrow binary array file

```
***Data Point File***               ***Connectivity Array File***

T1= 0.5871 2.6042 -30               (11 9 10) (11 10 10)
T2= 0.6262 3.1250 -30               (11 11 10) (11 12 10)
T3= 0.5871 3.6458 -30               (11 13 10) (11 14 10)
T4= 0.6067 4.1667 -30               (11 15 10) (11 16 10)
T5= 0.6067 4.6875 -30               (11 14 10) (11 15 10)
T6= 0.6067 5.2083 -30               (11 13 10) (11 14 10)
T7= 0.6067 5.7292 -30               (11 12 10) (11 13 10)
T8= 0.6067 6.2500 -30               (11 11 10) (11 12 10)
T9= 0.6067 6.7708 -30               (11 10 10) (11 11 10)
T10= 0.6067 7.2917 -30              (11 9 10) (11 10 10)
T11= 0.6067 7.8125 -30              (11 8 10) (11 9 10)
T12= 0.6067 8.3333 -30              (11 7 10) (11 8 10)
T13= 0.6067 8.8542 -30              (11 6 10) (11 7 10)
T14= 0.6067 9.3750 -30              (11 5 10) (11 6 10)
T15= 0.6067 9.8958 -30              (11 4 10) (11 5 10)
T16= 0.6067 10.4167 -30             (11 3 10) (11 4 10)
T17= 0.6067 10.9375 -30             (11 2 10) (11 3 10)
T18= 0.6067 11.4583 -30             (8 13 10) (9 14 10)
T19= 0.6067 11.9792 -30             (7 12 10) (8 13 10)
T20= 0.6067 12.5000 -30             (6 11 10) (7 12 10)
T21= 0.6067 13.0208 -30             (5 10 10) (6 11 10)
T22= 0.6067 13.5417 -30             (4 9 10) (5 10 10)
T23= 0.6458 13.8021 -30             (3 8 10) (4 9 10)
T24= 0.6849 13.2813 -30             (2 7 10) (3 8 10)
T25= 0.6849 12.7604 -30             (14 13 10) (13 14 10)
T26= 0.7241 12.2396 -30             (15 12 10) (14 13 10)
T27= 0.7632 11.7188 -30             (16 11 10) (15 12 10)
T28= 0.7828 11.1979 -30             (17 10 10) (16 11 10)
T29= 0.8219 10.6771 -30             (18 9 10) (17 10 10)
T30= 0.8611 10.1563 -30             (19 8 10) (18 9 10)
T31= 0.8806 9.6354 -30              (20 7 10) (19 8 10)

Note: "T1" is the index start
```

FIGURE 27. Arrow data point output for robotic controller and connectivity array output

```
***Initial Image
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000001111100000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000011000110000000111000000000000000000000000000000000000000000000000000000000000000
0000000000000000000110000001000000100100000000000000000000000000000000000000000000000000000000000000
0000000000000000001000000000100000100010000000000000000000000000000000000000000000000000000000000000
0000000000001000110000000001000001000010000000000001000000000000000000000000000000000000000000000000
0000000000011001100000000001000001000100000000000010000000000000000000000000000000000000000000000000
0000000000010011000000000001000010001100000000000010000000000000000000000000000000000000000000000000
0000000000100110000000000010000010001000000000000011000000000000000000000000000000000000000000000000
0000000001000100000000000010000100010000000000000100000000000000000000000000000000000000000000000000
0000000011001000000000000110000100100000000000000110000000000000000000000000000000000000000000000000
0000000010010000000000000100000101000000000000001000000000000000000000000000000000000000000000000000
0000000100010000000000001000001010000000000001111111111000000000000000000000000000000000000000000000
0000000100010000000000011000001100000000000000010000000000000000000000000000000000000000000000000000
0000001000011111110000100000010000000000000000110000000100000000000000000000001110111000000000000000
000000100000000000000010000001000000000000000101000000111000000111000000111100000100101001110000011100
00000100000000000000011000011100001100000000110100000100100001001000010001000110001100000111111110000
00000100000000000000100001101000101000000000101000001000100010001000100001000100010100000000000000000
000011000000000000001000011001001001000000010010000110001001000010010000110011001101000000000000000000
00001000000000000001000010000101000100000010001000010000100100001010000010001001100100000000000000000
0000100000000000000100011000001100001000001100110001000010010000011100000100010110000100000000000000000
00001000000000000010001000000110000100000100001000100001000001011000001100000001000011000000111000000000000
000100000000000100110000000110000100001000001000100000101000000010000001000010011111111000000000000000
0001000000000001011000000000000000100110000001001000000010000000000000011011000000000000000000000000
000100000000011100000000000000000011110000000101000000000000000000000000011100000000000000000000000000
0000100000000110000000000000000000000000001100000000000000000000000000000000000000000000000000000000
0000100000110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000011111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
***END Initial Image

***After scan
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
***END After scan
```

FIGURE 28. Detection frame scan process of signature binary array file

Data Point File	***Connectivity Array File***
T1= 135.0993 0.6556 -30 0	(38.1741 2.4238 -30) (38.5998 2.4437 -30)
T2= 78.6187 1.2318 -30 0	(38.1741 2.4238 -30) (37.7483 2.4636 -30)
T3= 79.0445 1.3709 -30 0	(38.5998 2.4437 -30) (39.0255 2.4437 -30)
T4= 38.1741 2.4238 -30 0	(39.0255 2.4437 -30) (39.4513 2.4437 -30)
T5= 38.5998 2.4437 -30 0	(39.4513 2.4437 -30) (39.8770 2.4437 -30)
T6= 39.0255 2.4437 -30 0	(39.8770 2.4437 -30) (40.3027 2.4834 -30)
T7= 39.4513 2.4437 -30 0	(40.3027 2.4834 -30) (40.7285 2.5232 -30)
T8= 39.8770 2.4437 -30 0	(40.7285 2.5232 -30) (41.1542 2.5629 -30)
T9= 40.3027 2.4834 -30 0	(41.1542 2.5629 -30) (41.4380 2.6225 -30)
T10= 40.7285 2.5232 -30 0	(41.4380 2.6225 -30) (41.5799 2.6821 -30)
T11= 41.1542 2.5629 -30 0	(41.5799 2.6821 -30) (41.7219 2.7417 -30)
T12= 41.4380 2.6225 -30 0	(41.7219 2.7417 -30) (41.8638 2.8013 -30)
T13= 41.5799 2.6821 -30 0	(41.8638 2.8013 -30) (42.0057 2.8609 -30)
T14= 41.7219 2.7417 -30 0	(42.0057 2.8609 -30) (42.0057 2.9205 -30)
T15= 41.8638 2.8013 -30 0	(42.0057 2.9205 -30) (42.1476 2.9801 -30)
T16= 42.0057 2.8609 -30 0	(42.1476 2.9801 -30) (42.1476 3.0397 -30)
T17= 42.0057 2.9205 -30 0	(42.1476 3.0397 -30) (42.1476 3.0993 -30)
T18= 42.1476 2.9801 -30 0	(42.1476 3.0993 -30) (42.1476 3.1589 -30)
T19= 42.1476 3.0397 -30 0	(42.1476 3.1589 -30) (42.1476 3.2185 -30)
T20= 42.1476 3.0993 -30 0	(42.1476 3.2185 -30) (42.1476 3.2781 -30)
T21= 42.1476 3.1589 -30 0	(42.1476 3.2781 -30) (42.1476 3.3377 -30)
T22= 42.1476 3.2185 -30 0	(42.1476 3.3377 -30) (42.0057 3.3974 -30)
T23= 42.1476 3.2781 -30 0	(42.0057 3.3974 -30) (42.0057 3.4570 -30)
T24= 42.1476 3.3377 -30 0	(42.0057 3.4570 -30) (42.0057 3.5166 -30)
T25= 42.0057 3.3974 -30 0	(42.0057 3.5166 -30) (42.0057 3.5762 -30)
T26= 42.0057 3.4570 -30 0	(42.0057 3.5762 -30) (41.8638 3.6358 -30)
T27= 42.0057 3.5166 -30 0	(41.8638 3.6358 -30) (41.8638 3.6954 -30)
T28= 42.0057 3.5762 -30 0	(41.8638 3.6954 -30) (41.7219 3.7550 -30)
T29= 41.8638 3.6358 -30 0	(41.7219 3.7550 -30) (41.7219 3.8146 -30)
T30= 41.8638 3.6954 -30 0	(41.7219 3.8146 -30) (41.5799 3.8742 -30)
T31= 41.7219 3.7550 -30 0	(41.5799 3.8742 -30) (41.5799 3.9338 -30)
T32= 41.7219 3.8146 -30 0	(41.5799 3.9338 -30) (41.4380 3.9934 -30)
T33= 41.5799 3.8742 -30 0	(41.4380 3.9934 -30) (41.4380 4.0530 -30)
T34= 41.5799 3.9338 -30 0	(41.4380 4.0530 -30) (41.2961 4.1126 -30)
T35= 41.4380 3.9934 -30 0	(41.2961 4.1126 -30) (41.2961 4.1722 -30)
T36= 41.4380 4.0530 -30 0	(41.2961 4.1722 -30) (41.1542 4.2318 -30)
T37= 41.2961 4.1126 -30 0	(41.1542 4.2318 -30) (41.1542 4.2914 -30)
T38= 41.2961 4.1722 -30 0	(41.1542 4.2914 -30) (41.0123 4.3510 -30)
T39= 41.1542 4.2318 -30 0	(41.0123 4.3510 -30) (40.8704 4.4106 -30)
T40= 41.1542 4.2914 -30 0	(40.8704 4.4106 -30) (40.8704 4.4702 -30)
"	"
"	"
"	"
T1510= 104.588 14.125 -30 0	(127.720 9.0596 -30) (128.1457 9.1192 -30)
T1511= 29.9432 14.721 -30 0	(128.145 9.1192 -30) (128.5714 9.1589 -30)
T1512= 74.6452 14.860 -30 0	(128.571 9.1589 -30) (128.9972 9.1788 -30)
Note: "T1" is the index start	

FIGURE 29. Signature data point output for robotic controller and connectivity array output

CHAPTER 8

DESCUSSION

This thesis was broken down into 2 parts: (1) the Seiko D-TRAN XM5000 robot that had to be assembled in the engineering lab and (2) the new Data Scanner/Generator program. The lessons learned were how to adapted new advanced programming techniques to a robotic cell with limited capabilities in handling large amounts of data points. The new project program on the other hand is designed to be flexible. The source code is included in this report. With small changes in the code the output can be modified to run on other types of industrial machines.

Seiko D-TRAN Limitations

Even with additional memory banks it was not known if Seiko D-TRAN XM5000 robot controller would handle large amounts of data translation points of more than four hundred which was the limit. These additional memory banks in the controller were allocated to store downloaded DARLTalk programs only and not the data translation point files. In other words DARLTalk did not allow the user the ability to share the memory banks (0 to 6) with the data translation point files. When the robot's software was developed by the programmer's back in the 1980s they geared the system to the end user that only required the system to do multiple program tasks. The memory banks are hard-wired in the robot's controller. Nothing could be done to allocate additional memory for the translation point files.

Data Scanner/Generator Program

The most complicated part of this project was writing a new program basically from scratch. The detection frame logic based on of Bauman's pattern recognition project was the only thing I used in this thesis project. As I discovered early on in my investigation of pattern recognition software techniques, the detection frame logic I was trying to incorporate had several errors in the code. The errors were corrected and the logic improved to scan any size image with automatic internal checks if the minimum segment size or detection frame size were input incorrectly.

This new program was developed in a way that could be adapted to other machines in the engineering lab.

Changing the source code for CNC output formats for example could be easily accomplished by a competent programmer.

Recommendations for Testing the Robot

If the Mechanical Engineering Department at California State Long Beach is to purchasing a pneumatic gripper and additional tools it would be possible to take full advantage of the DARLTalk programming capabilities to do repetitive tasks.

a) Find a way to limit or reduce the data translation points and maintain the fidelity of the robot's translation paths.

b) Apply engraving, cutting, punching or a combination thereof on any type of media. The Data Scanner/Generator programming platform easily lends itself as an input source for the robot controller.

APPENDICES

APPENDIX A

DARLTalk COMPUTER INSTALLATION AND OPERATION

Configure DARLTalk

Prior to October 1995, Seiko Instruments always sold DARLTalk with the Norton Editor included as part of the DARLTalk package. However, Symmantec no longer sells the Norton Editor. The Norton Editor was made obsolete by Microsoft's introduction of DOS 5.0 that included the DOS Editor. Since DOS Version 5.0 and higher includes the DOS Editor called EDIT.COM, Epson (formally known as Seiko Instruments) stopped including an Editor with the DARLTalk Product which they no longer have available for sale.

This means that after installing your DARLtalk software, the Edit menu item will not function properly until the following is completed.

1. Make a copy of the DOS Editor called EDIT.COM and put it in your DARLT directory, which is created for DARLtalk. This can be done as follows from the DOS Prompt >.

   ```
   > copy c:\dos\edit.com c:\darlt
   ```

2. DARLtalk requires that the editor be named ED.COM or ED.EXE in order for the EDIT menu in DARLtalk to properly invoke the editor. This can be done as follows from the DOS prompt >.

```
cd c:\darlt
> ren edit.com ed.com
```

You may also use your favorite editor for DOS, which you may already be familiar with rather than using the DOS Editor. Just be sure to copy in into the DARLT directory and then rename it as ED.COM or ED.EXE. See the sections later in this appendix called EDIT and DOWNLOAD.

DARLTalk Package

The editor used with DARLTalk is the EDIT.COM, which comes with DOS version 5.0 and higher. See instructions above for how to configure this editor to properly work with DARLtalk. Any editor can be used with DARLtalk as long as it is named "ed.com".

Features

- User friendly interface.
- Internal checks on communication to preserve data integrity.
- Upload and download files between PC and controller.
- Upload protection feature to prevent accidental upload to a pre-existing file.
- Download files with or without comments.
- Download protection feature to prevent accidental download on an occupied bank.

- Have your program ask for more teach points and subroutines
- Create and edit programs on PC.
- Editor enabled to automatically go to a bad line when downloading a file with errors.
- Execute DARL commands.
- Execute DOS commands.
- View the amount of available bytes in each bank.
- View the size of a PC file.
- Clear teach points and banks.
- Browse through a file on the PC.

Installation

To install darltalk on the C disk drive, insert the DARLTalk disk and type "a:install". This installation program will guide you through the necessary steps to properly load DARLTalk on your system. It will also create a file called "darlt.bat" that is used to execute DARLTalk. If this doesn't work create a directory called "DARLT" on the C drive and copy all the files over to this new directory.

DARLTalk communicates over a standard RS-232 cable. On some PCs a null modem adapter may be required. The null modem is necessary only in cases where the PC and the controller are both set as DCE (Data Communications Equipment) or DTE (Data Terminal Equipment). If a null modem is not available then you can convert the DARLfour controller board to DCE or DTE on the shunts that exist on the circuit board.

The controller must be prepared to communicate by entering the command "DO COMM 1 78" (DARLfour and DARL II controllers) or "DO Z=78" (DARL I controllers) from monitor mode on the teach terminal. This command initializes serial port 1 of the controller with the following values: data length of 8 bits; no parity; <Carriage Return><Line Feed> delimiter; echo off; and 1 stop bit. After this command has been entered on the robot, enter the following commands "DO OUNIT 1" and "DO IUNIT 1". These commands tell the controller that the output unit and input unit will be serial port 1.

Two files, ed.com and browse.com, are installed in the same directory as DARLTalk. They can be removed if they already reside on your PC or if not, you can move them to another directory provided that the directory is specified in the search path in AUTOEXEC.BAT.

Using DARLTalk

To get started with DARLTalk, type "darlt" from the DOS command prompt. DARLTalk is a menu driven program that performs a wide variety of tasks to provide a friendly development environment. Menu items can be selected by using the arrow keys on the keyboard or by typing the highlighted character of the menu item. Once an item has been selected, just press the RETURN key to execute it. A brief explanation of each menu selection is provided near the bottom of the main window. An arrowhead to its right indicates those menu items, which contain sub-menus. If at any time you wish to leave a menu or data entry window without making any selections or input, hit the ESCAPE key, except when in monitor window, in which case the END key must be used.

All user created files shown in the menus have a .B[0-6] or .TPT extension and reside in the current working directory. Files with the .B[0-6] extension signify programs from a corresponding bank. For example, TEST.B0 would contain the contents of bank 0 and TEST.B5 would contain the contents of bank 5. For DARL I users, a .B0 extension should be used. Files with the .TPT extension signify teach points. This feature provides for an efficient means of managing programs and teach/translation points.

DARLTalk uses a message window, highlighted in red, to display the current status of the program on the bottom of the DOS prompt screen This window also displays information regarding communication failures, errors, and data entry requirements.

APPENDIX B

DESCRIPTION OF DARLTalk MENU ITEMS

Edit

This menu item allows you to create and edit files without leaving DARLTalk. DARLTalk uses the Norton editor and calls it "ed.com". For first time users, once in the Norton editor, please hit the "F1" key for help. To create a new file or edit a file without the .B[0-6] extension or edit a file not in the current working directory, the "New file" menu item should be selected. When creating a new file, be sure to use a .B[0-6] extension or a .TPT extension if you want the file to be displayed in the file select window. DARLTalk only searches for and displays those files with the .B[0-6] and .TPT extensions. To edit an existing program file or teach point file, select the appropriate name that is displayed. Note that this display lists the files without an extension. After a selection has been made, another menu will appear to list all the files with the selected name having .B[0-6] and .TPT extensions. A selection here will invoke the editor on the appropriate file.

DARLTalk Tips:

DARLTalk assumes that the editor being used is called "ed.com". If you wish to use an editor other than what was provided, it must be copied to or renamed to "ed.com" or "ed.exe". If you cannot get the Norton editor to work (for older versions of DARLtalk which shipped with the Norton Editor), it may have to be reconfigured for your PC. To do this, you must exit DARLTalk. From the DOS prompt type "ed /db" or "ed /dc". One of these commands should bring up the editor. When the editor asks you to enter a file, enter any file name. Once in the editor, press the F5 function key and then press the letter S. Type "ed.com" as the name of the editor. This will save your editor configuration.

Try to avoid putting leading spaces in front of line numbers. They might confuse the controller when downloading the file.

Make sure the auto-indent feature is turned off on the editor and no tabs exist in your file. They may confuse the controller.

Upload

These menu items upload programs and/or teach points from the controller to the PC. You can choose to upload all seven banks, only the teach points, all the banks and teach points, or only one bank. After selecting a menu item, DARLTalk will ask for a file name without an extension, it will automatically append it for you. The window in which you type the file name is scrollable, thereby allowing it to handle directory paths. If you have not yet entered a file name and wish to leave without doing so, hit the ESCAPE key to leave this window. If there is nothing to upload, i.e. bank is empty or no teach points, DARLTalk will tell you. If you are uploading to an already existing file, DARLTalk will ask if you wish to overwrite the file or quit the upload operation for that file. If you decide to overwrite the file, DARLTalk will make a backup copy of the original file before it begins uploading. This file will have a "@" as the first character of its extension.

DARLTalk Tips:

If you wish to upload to a file that does not have a .B[0-6] or a .TPT extension, you will have to proceed through the normal upload procedure and then use the Dos shell to either rename it or copy it to the file name of your choice.

If there is a break in communication during upload, you must hit the STOP key on the teach terminal to prevent the controller from sending any more data. If this is not done before you continue with DARLTalk, the communication between the controller and the PC will be out of sync. After pressing the STOP key, you must get into monitor mode by typing QUIT and then re-enter the "DO OUNIT _" and "DO IUNIT _" commands on the teach terminal. The file that was being uploaded to during the communication error will be deleted.

If DARLTalk tells you that a mistake was made in entering the file name, it will ask you to re-enter it. The cursor may have disappeared, but DARLTalk will still take and display your input. Uploading a bank does not automatically upload teach points. They can only be uploaded by selecting either "Banks & Tpoints" or "Tpoints".

<u>Download</u>

This menu item downloads files from the PC to the controller. It allows you to download files that do not have the .B[0-6] or .TPT extension by using the "Other file" menu item. If this menu item is chosen, you will also be asked to supply the number of the bank you wish to download the file to. After the file(s) have been selected, you can download them with or without comments. Downloading a file without comments does not alter the file on the PC. Comments are stripped off as they are being sent to the controller. If a communication failure occurs during the file transfer, then the download operation will be aborted. Files will be downloaded to the bank indicated by the file extension or to the specified bank if the "Other file" menu selection was used. Should the target bank be occupied, DARLTalk will ask if you wish to overwrite the bank, merge your file with the bank, or abort the download operation. This feature will prevent you from accidentally writing over another program. If the controller detects an error in a line being sent, DARLTalk will display the error number, the bad line, and will then ask if you wish to re-edit the file. If you answer yes, DARLTalk will take you right to the bad line; you won't have to search for it.

DARLTalk Tips:

If you have replaced the editor provided with one of your own, make sure that it can be called with the following command sequence: "ed +[line #] [file name]". If it cannot then the editing feature in download will not work.

Files being downloaded to the controller must have all lines end with a line feed character.

If you wish to download a file to a bank other than what is shown by its extension, select the file using the "Other File" menu selection.

If you receive ERROR 110, then the program you have just downloaded contains jumps to non-existent line numbers. When DARLTalk invokes the editor so you can re-edit the file, chances are that you will be taken to either the beginning or the end of the file. In some cases the error may not reside in the just downloaded file, but in the bank itself. The error message displayed contains up to three line numbers with the bad jumps (more may exist).

Monitor

This menu item pops open a window which allows you to enter DARL commands. Almost anything that can be done in monitor mode with the teach terminal can also be done in this window. The prompt displays the number of the bank and the mode that you are currently in. Hit the END key when you are ready to leave. Any program in execution or robot movement will be stopped and aborted when you leave this window.

To activate or de-activate DARLTalk's "Listen" feature from the monitor mode window, enter CTRL-L. Activating the listen feature will allow monitor mode to act on "listen" commands sent from a running program. For more information on the "Listen" feature please read the file LISTEN.DOC.

DARLTalk Tips:

You cannot jog the robot from within this window. Jogging can only be done from the teach terminal. However, jog values can be displayed in the monitor mode window if the controller is set to OUNIT1 and IUNIT0.

To stop a program in execution or to stop a robot movement hit the ESCAPE key.

If you plan on issuing commands, which cause the robot to move, make sure that the servo power is on. If it isn't, the system will hang until servo power is turned on. One of the side effects of turning servo power on or off is that it returns control to the teach terminal. Therefore, it would be necessary to re-enter the "DO OUNIT _" and "DO IUNIT _" commands in monitor mode on the teach terminal to re-establish communication with the PC.

When in edit mode or display mode from the monitor window, the "\" key can be used as an increment key, while the "|" key can be used as a decrement key. The RETURN key can also be used to increment through the display mode.

If a communication error occurs while in the monitor window, DARLTalk will, in most cases, exit this window and display an error message.

<u>Dos Shell</u>

This menu item jumps to a DOS shell. All DOS commands will be accepted. You can hit the RETURN key or type "EXIT" to return to DARLTalk. To invoke a second DOS shell type "DOS".

DARLTalk Tips:

The number of nested shells or programs called is determined by the size of memory in the PC.

<u>Options</u>

Check Bank Size (DARL II and DARLfour). This menu item pops open a window which displays the bytes remaining in all seven banks. Hit the ESCAPE key to remove this window.

Check File Size

This menu item asks you to enter any legal DOS file name and then displays its approximate size in bytes as if it were sent to the bank. It does this by ignoring the leading line numbers. Hit the ESCAPE key to remove this window.

DARLTalk Tips:

There is a slight discrepancy between the file sizes determined by "Check Bank Size" and "Check File Size". This is due to the fact that the "Check File Size" provides an approximate size. The file name will be displayed without its directory path.

Clear T-Points

This menu item clears the existing teach points from the controller. It also clears all variables. To prevent accidental removal, DARLTalk gives you a chance to abort this operation.

Clear Bank(s)

This menu item allows you to clear all banks or an individual bank. To prevent accidental removal, DARLTalk gives you a chance to abort this operation.

Setup

This menu item allows you to reconfigure the communication setup on your PC. You can change the baud rate, the serial port through which the PC communicates, the serial port on the controller that the PC will communicate with and the type of controller you are using (DARL I, DARL II or DARLfour). If changes have been made, press CTRL-RETURN so that DARLTalk can acknowledge the changes. Pressing the ESCAPE key will leave things unchanged.

DARLTalk Tips:

DARLTalk saves the communication setup data in a file called "setup.ini".

The default setup values for DARLTalk are:

Baud rate	9600
Controller port	1
PC port	1
Darl version	4

Selecting a serial port that does not exist on the controller or the PC can cause DARLTalk to crash.

Only Comm ports 1 and 2 are supported by DARLTalk.

If you have changed the value of the serial port on the controller through which the PC will communicate with, then you must re-enter the following monitor mode commands on the teach terminal:

"DO COMM [1 or 2] 78",

"DO OUNIT [1 or 2]" and

"DO IUNIT [1 or 2]".

If you have changed the baud rate on the PC, you will have to make the same change on the controller to prevent DARLTalk from hanging. The following steps need to be taken: turn off the controller, change the baud rate jumper on the DARL CPU board, turn on the controller, re-enter the "DO OUNIT _" and "DO IUNIT _" commands. If you cannot communicate at this point, you will have to exit DARLTalk and then start it again.

Browse

This menu item allows you to browse through any file without making changes. Movement through the file can be accomplished through the use of the PAGE UP, PAGE DOWN, HOME, END and arrow keys. To leave just hit the ESCAPE key.

Exit

This menu item lets you exit DARLTalk and asks for a confirmation before actually leaving.

Miscellaneous Notes

If you are having difficulty getting DARLTalk to display properly on a LCD screen, then you may have to execute DARLTalk with the "/L". This option will insure the correct appearance of the DARLTalk menus on your LCD screen. To use the "/L" option, either modify the file DARLT.BAT, changing "dt" to "dt /L" or just type "dt /L" from the DOS prompt.

DARLTalk tries to catch communication errors before they become a problem. If one is found, DARLTalk will try to help you fix them. For some extreme cases, it may be necessary to power down both the controller and the PC. DARLTalk may hang up if the PC and controller are set to different baud rates.

Turning the servo power on and off causes the controller to break off communication. You will need to re-enter the monitor mode commands "DO OUNIT _" and "DO IUNIT _" to re-establish communication.

<u>Additional Communication Errors</u>

Other communication errors can occur if:

- The RS-232 cable has been disconnected;
- The communication parameters are not the same on the PC and the controller;

- A null modem adapter is needed on the PC but not used.

If you keep getting communication errors and a null modem adapter did not work, then you need to make sure that the RS-232 cable has the correct number of pins wired. For a 25-pin cable, pins 2, 3, 4, 5, 6, 7 and 20 need to be wired. For a 9-pin cable, pins 2, 3, 4, 5, 6, 7 and 8 need to be wired.

APPENDIX C

OPERATIONAL MODES

The following commands are permitted in the MONITOR Mode:

MONITOR COMMANDS	DESCRIPTIONS
BANK <0-6>	Specifies the current memory bank.
BGOTO <0-6>	Causes the controller to continue program execution at the specified bank of memory.
CALIB	Commands the controller to enter the CALIBRATION Mode. This mode allows the robot's factory-determined calibration values to be re-entered into the controllers memory in the event these values should become lost.
DISP	Commands the controller to enter the DISPLAY Mode, which displays the robot's current status at the moment.
DO CLEAR	Clears the memory of all defined translations, pallets, frames and variables.
DO KEY <1-5>	The user can assign a commonly used phrase or program statement to any of 5 function keys on the Teach-Terminal. Example: DO KEY 1 "DO MOVE T" <ENTER>
DSPON	Turns on a feature, which displays the program on the LCD line-by-line as it is being executed.
DSPOFF	Turns off the DSPON feature, which displays the program on the LCD (Teach-Terminal).
DO MOVE <statements>	Causes the immediate execution of any statement or statements up to 75 characters following DO MOVE.
EDIT	Commands the controller to enter the EDIT Mode (prompt: 0E>)to gain access to DARL programs. The QUIT command returns the controller to the MONITOR Mode.
FRAME <0-9>	Instructs the controller that all program steps following this statement ill be in the reference to the specified frame until further program instruction to change the frame. The robots HOME FRAME is always 0.
DEF FR<1-9> T#a T#b	Defines a frame of reference using a new orign (T#a) and a +X axis direction value (T#b)
HERE <0-399>	Assigns the robot's current (X,Y,Z,A) position to the specified translation teach point relative to

	the current frame of reference.
HOME	Commands the robot to enter the HOME Operational Mode and move to its factory set HOME position (X=0,Y=0,Z=0,A=0). This initializes the I/Os.
JOGRT	Commands the robot to move in a cylindrical fashion. JOGRT cancels the JOGXY command.
JOGSP <1-100>	Sets the manual jogging speed of the robot. The default value is 10.
JOGXY	Coordinates the robot's axes so tat planar jogging is in an XY direction in reference to the current frame.
MAP	Displays the remaining amount of memory for each bank of memory.
MEM	Displays the amount of free memory available in the current bank of memory.
SHAVE	This statement causes the robot to operate in a condition when the robot passes through specified locations without achieving zero momentum. In other words without stopping. Rather that the robot will execute the next statement following the MOVE statement as soon as the robot begins to decelerate. When moving through a series of points, the robot will not actually move through the points. Before it actually reaches its destination the next MOVE is being executed. This increases cycle time.
NOSHAVE	Causes the Robot to achieve zero momentum at each translation point.
FOR <V> = <expression> TO <expression> NEXT <V>	Repeats a program loop a specified number of times, then the program continues at the first line, which follows the NEXT statement. The routine will repeat incrementing the variable <V> from its initial value.

The Format of Location in Space

Translations are retained in memory separate from that of the program. The translation or T# is the address at which the coordinate positions relative to the HOME Frame of a desired location in space are stored. All coordinates are decimal values that are specified in millimeters or degrees. The format is as follows:

```
T<0-399> = <X-Coordinate> <Y-Coordinate> <Z-Coordinate>
<A-Coordinate>
```

A particular translation can be referred to by integer or by algebraic expression whose result is the integer as shown below

```
IF: C = 6, D = 2, E = 3 and T6 = 380.  0.345  20.3  0.
Then:
T6          =  380.   0.345   20.3   0.
T(C)        =  380.   0.345   20.3   0.
T(E+E)      =  380.   0.345   20.3   0.
T(C*2-E*D)  =  380.   0.345   20.3   0.
```

APPENDIX D

DATA SCANNER/GENERATOR SOURCE CODE

README.txt

```
ps_scan - is the directory with all of the program data

read_ps_file.c, read_ps_file.h - is the routines to read postscript
file and store image in memory (was old front-end program)

scan_array.c, scan_array.h, scan_array_types.h - is the routines to
scan array and output data (was old back-end program)

dump_image.c, dump_image.h - is the routine used for debugging all
other .c, .cpp, and .h files are built by visual studio for the user
interface.

The MFC library defines main for you, but if you want to see where
everything gets kicked off from, look at ps_scan.cpp -
Cps_scanApp::InitInstance is called during the initialization of the
'theApp' variable, and things start happening when you hit the ok
button.  If you know how MFC works this is review, but if you don't
it gets confusing.

ps_scan.sln, ps_scan.vcproj - is the solution and project database
for visual studio .net. If you have Visual Studio C++ .Net you can
open either file; if you have Visual C++ V6.0 I believe you can open
it with the .vcproj file.

ps_scan\Debug - is the directory with the executable (ps_scan.exe).
I've compiled it statically so that it should be able to run as is,
but it is big (1.8M).  The dynamically linked version will fit on a
floppy, but has to be run on a system with Visual C++ installed.

ps_scan\Sample Data - is the directory with my test input and output
files.  The file test1_in.txt is a dummy postscript file,
test1_data.txt is the data-point output file, test1_conn.txt is the
connectivity array file, and test1_dump.txt is a dump of the image at
various processing points.

Note: Program can except a Hex ASCII code list (as a postscript
input) that represents the 8 bit binary computer byte of a test
image.
```

CODE: bwmisc CPP File

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include <malloc.h>
#include "stdafx.h"
```

```
static size_t bw_mallocsize;

/* ____________________________________________ */

int bw_atoi(CString str)
{
	int ret=0;
	char buf[50], *cptr;

	_tcscpy(buf, str);
	for(cptr = &buf[0]; *cptr; cptr++) {
		ret = ret * 10 + (*cptr - '0');
	}
	return ret;

} //bw_atoi

void memory_error()
{
	char buf[200];
	sprintf(buf, "Memory Allocation Error, mallocsize: %d",
bw_mallocsize);
	AfxMessageBox(buf);
	exit(1);
}

void *bw_malloc(size_t size)
{
	void *ret = malloc(size);
	if(ret == NULL) {
		memory_error();
		return(NULL);
	}
	bw_mallocsize += size;
	return ret;

}

size_t get_mallocsize(void)
{
	return(bw_mallocsize);
}
```

CODE: bwmisc H File

```
//function prototypes
extern int bw_atoi(CString str);
extern void memory_error();
extern void *bw_malloc(size_t size);
```

CODE: Dump Image CPP File

```
#include <stdio.h>
#include <stdlib.h>
#include "scan_array_types.h"

void dump_image(FILE *out, struct imageArray *array_info, char
*title) {
      int cnt;

      if(out != NULL) {
            fprintf(out, "***%s\n", title);
            for(cnt = 0; cnt < array_info->y_size; cnt++) {
                  fprintf(out, "%s\n", array_info->array[cnt]);
            }
            fprintf(out, "***END %s\n\n", title);
      }
}
```

CODE: Dump Image H File

```
//function prototypes
extern void dump_image(FILE *out, struct imageArray *array_info, char
*title);
```

CODE: PS Scan CPP File

```
// ps_scan.cpp : Defines the class behaviors for the application.
//

//#define _CRTDBG_MAP_ALLOC
#include <stdio.h>
#include <stdlib.h>
//#include <crtdbg.h>

#include "stdafx.h"
#include "ps_scan.h"
#include "ps_scanDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

#ifdef BW_DEBUG
//globals for debugging
size_t bw_mallocsize;
FILE *dbgfil;
#endif

// Cps_scanApp

BEGIN_MESSAGE_MAP(Cps_scanApp, CWinApp)
      ON_COMMAND(ID_HELP, CWinApp::OnHelp)
```

```
END_MESSAGE_MAP()


// Cps_scanApp construction

Cps_scanApp::Cps_scanApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}


// The one and only Cps_scanApp object

Cps_scanApp theApp;

// Cps_scanApp initialization

BOOL Cps_scanApp::InitInstance()
{
	// InitCommonControls() is required on Windows XP if an
application
	// manifest specifies use of ComCtl32.dll version 6 or later to
enable
	// visual styles.  Otherwise, any window creation will fail.
	InitCommonControls();
	CWinApp::InitInstance();
	AfxEnableControlContainer();

	Cps_scanDlg dlg;
	m_pMainWnd = &dlg;
	int ret = -1;

	//this is what displays the main dialog
	INT_PTR nResponse = dlg.DoModal();

	//we don't really care about the return, once we get here
	// we just exit
	if (nResponse == IDOK)	{
		ret = 1;
	} else if (nResponse == IDCANCEL) {
		ret = 0;
	}

	// Since the dialog has been closed, return FALSE so that we
exit the
	//  application, rather than start the application's message
pump.
	return FALSE;
}
```

CODE: PS Scan H File

```
// ps_scan.h : main header file for the PROJECT_NAME application
//
#define BW_CONV_CSTRING(out, in) \
            (out) = new TCHAR[(in).GetLength()+1]; \
            _tcscpy((out), (in));
#pragma once
#ifndef __AFXWIN_H__
      #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"         // main symbols
// Cps_scanApp:
// See ps_scan.cpp for the implementation of this class
//
class Cps_scanApp : public CWinApp
{
public:
      Cps_scanApp();
// Overrides
      public:
      virtual BOOL InitInstance();
// Implementation
      DECLARE_MESSAGE_MAP()
};
extern Cps_scanApp theApp;
```

CODE: PS ScanDlg H File

```
// ps_scanDlg.h : header file
//

#pragma once

// Cps_scanDlg dialog
class Cps_scanDlg : public CDialog
{
// Construction
public:
	Cps_scanDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	enum { IDD = IDD_PS_SCAN_DIALOG };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	//
DDX/DDV support


// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:

	CString dp_filename;
};
```

CODE: Ps scanDialog CPP File

```
// ps_scanDlg.cpp : implementation file
//

#include <cstring>
#include "stdafx.h"
#include "ps_scan.h"
#include "ps_scanDlg.h"
#include "read_ascii_file.h"
#include "scan_array.h"
#include "bwmisc.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

#ifdef BW_DEBUG
extern size_t bw_mallocsize;
extern FILE *dbgfil;
#endif

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
      CAboutDlg();

// Dialog Data
      enum { IDD = IDD_ABOUTBOX };

      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support

// Implementation
protected:
      DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()


// Cps_scanDlg dialog



Cps_scanDlg::Cps_scanDlg(CWnd* pParent /*=NULL*/)
      : CDialog(Cps_scanDlg::IDD, pParent)
      , dp_filename(_T(""))
      , ps_fname(_T(""))
      , zcoord(_T(""))
      , ca_fname(_T(""))
      , framesize(_T(""))
      , minpixelsize(_T(""))
```

```
	, startindex(_T(""))
	, imgxsize(_T(""))
	, imgysize(_T(""))
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void Cps_scanDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_dpname, dp_filename);
	DDX_Text(pDX, IDC_psname, ps_fname);
	DDX_Text(pDX, IDC_zname, zcoord);
	DDX_Text(pDX, IDC_caname, ca_fname);
	DDX_Text(pDX, IDC_dfsize, framesize);
	DDX_Text(pDX, IDC_mpsize, minpixelsize);
	DDX_Text(pDX, IDC_stindex, startindex);
	DDX_Text(pDX, IDC_IMGXSZ, imgxsize);
	DDX_Text(pDX, IDC_IMGYSZ, imgysize);
}

BEGIN_MESSAGE_MAP(Cps_scanDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_BN_CLICKED(IDC_BEXIT, OnBnClickedBexit)
	ON_BN_CLICKED(IDC_BRUN, OnBnClickedBrun)
	ON_BN_CLICKED(IDC_BBROWSEIMG, OnBnClickedBbrowseimg)
	ON_BN_CLICKED(IDC_BBROWSEDP, OnBnClickedBbrowsedp)
	ON_BN_CLICKED(IDC_BBROWSECA, OnBnClickedBbrowseca)
END_MESSAGE_MAP()


// Cps_scanDlg message handlers

BOOL Cps_scanDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this
automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	//set initial values for the form entries here
	framesize = "1";
	minpixelsize = "0";
	startindex = "1";
	UpdateData(FALSE);

	return TRUE;  // return TRUE  unless you set the focus to a
control
}

void Cps_scanDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the
code below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.

void Cps_scanDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this function to obtain the cursor to display
while the user drags
//  the minimized window.
HCURSOR Cps_scanDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

void Cps_scanDlg::OnBnClickedBexit()
{
	OnOK();   //exit the application
}

void Cps_scanDlg::OnBnClickedBrun()
{
	struct inputInfo input_info;
	struct imageArray *image;

	char *datapoint_fname;
	char *input_fname;
	char *connarray_fname;

#ifdef BW_DEBUG
	bw_mallocsize = 0;
	dbgfil = fopen("debug_out.txt", "w");
#endif

	//dump memory leak information
	//_CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF |
_CRTDBG_LEAK_CHECK_DF);

	UpdateData(TRUE);
	GetDlgItem(IDC_BRUN)->EnableWindow(FALSE);

	// convert input data into char*/int variables
	// used by program and store in the input_info structure
	BW_CONV_CSTRING(datapoint_fname, dp_filename);
	BW_CONV_CSTRING(input_fname, ps_fname);
	BW_CONV_CSTRING(connarray_fname, ca_fname);
	BW_CONV_CSTRING(input_info.z_coord, zcoord);
	input_info.detect_frame_size = bw_atoi(framesize);
	input_info.min_pixel_size = bw_atoi(minpixelsize);
	input_info.start_index = bw_atoi(startindex);
	input_info.img_x_size = bw_atoi(imgxsize);
	input_info.img_y_size = bw_atoi(imgysize);
	input_info.dumpimagefname = "dump.txt";

	if(validate_inputs(&input_info)) {
		GetDlgItem(IDC_BRUN)->EnableWindow(TRUE);
		return;
	}

	//read the input file
	//image = read_ps_file(input_fname);
	image = read_ascii_file(input_fname);
	if(image == NULL) {
		errorMessage("Error reading input file","File Read
Error");
		return;
	}

	if(validate_options(&input_info, image)) {
		GetDlgItem(IDC_BRUN)->EnableWindow(TRUE);
		return;
	}
	//process the image
	int scanstat = scan_array(image, &input_info, datapoint_fname,
connarray_fname);
	if (scanstat > 0) {
		errorMessage("Error processing image", "Processing
Error");
		return;
	}

	//free allocated memory
	free(datapoint_fname);
	free(input_fname);
	free(connarray_fname);
	free(input_info.z_coord);

	//enable run button and return to main window
	GetDlgItem(IDC_BRUN)->EnableWindow(TRUE);
}

void Cps_scanDlg::errorMessage(char* msg, char *title)
{
	MessageBox(msg, title, MB_OK|MB_ICONEXCLAMATION);
	GetDlgItem(IDC_BRUN)->EnableWindow(TRUE);
}

//Handler for Browse/Image Button
void Cps_scanDlg::OnBnClickedBbrowseimg()
{
	CFileDialog imgFile(TRUE);

	if(imgFile.DoModal() == IDOK) {
			ps_fname = imgFile.GetPathName();
			UpdateData(FALSE);
		}
}

//Handler for Browse/DataPoint Button
void Cps_scanDlg::OnBnClickedBbrowsedp()
{
	CFileDialog dpFile(TRUE);

	if(dpFile.DoModal() == IDOK) {
		dp_filename = dpFile.GetPathName();
		UpdateData(FALSE);
	}
}

//Handler for Browse/ConnectivityArray Button
void Cps_scanDlg::OnBnClickedBbrowseca()
{
	CFileDialog caFile(TRUE);

	if(caFile.DoModal() == IDOK) {
		ca_fname = caFile.GetPathName();
		UpdateData(FALSE);
	}
}

int Cps_scanDlg::validate_inputs(struct inputInfo * info)
{
	int ret=0;
	char *cptr;

	//check z-coordinate to ensure it is a number
	for(cptr = info->z_coord; *cptr; cptr++) {
		if((*cptr < '0') || (*cptr > '9')) {
			if(!((*cptr == '.') || (*cptr == '-'))) {
				MessageBox("Z-Coordinate is not a number",
"Input Error", MB_OK|MB_ICONEXCLAMATION);
				ret=1;
				break;	// from for loop
			}
		} // if was not digit
	} // for all characters in z-coordinate

	//check x- and y-size of image are specified and not 0
	if((info->img_x_size == 0) || (info->img_y_size == 0)) {
		MessageBox("X and Y image sizes must be specified",
"Input Error", MB_OK|MB_ICONEXCLAMATION);
		ret=1;
	}
	return(ret);
} // validate_inputs
```

```
int Cps_scanDlg::validate_options(struct inputInfo *info, struct
imageArray *image)
{
      int ret=0;
      int min_pixelsize;
      int selection;
      int frame_size;
      char msgbuf[1024];

      //check minimum pixel specification against data
      min_pixelsize = check_pixel_size(image);
      if(info->min_pixel_size < min_pixelsize) {
            sprintf(msgbuf, "Specified minimum pixelsize is less than
apparent found in data (%d)\n", min_pixelsize);
            strcat(msgbuf, "Would you like to update it?");
            selection = MessageBox(msgbuf, "Input Discrepancy",
MB_YESNOCANCEL|MB_ICONQUESTION);
            switch(selection) {
                  case IDYES:
                        sprintf(msgbuf, "%d", min_pixelsize);
                        minpixelsize = msgbuf;
                        UpdateData(FALSE);
                        info->min_pixel_size = min_pixelsize;
                        break;
                  case IDNO:
                        break;
                  case IDCANCEL:
                        return(1);
                        break;
            } //switch
      } // if min pixelsize seems small

      min_pixelsize = bw_atoi(minpixelsize);
      frame_size = bw_atoi(framesize);
      if(frame_size <= min_pixelsize) {
            sprintf(msgbuf, "Frame size should be greater than
minimum pixel size\nWould you like set it to %d?", min_pixelsize+1);
            selection = MessageBox(msgbuf, "Input Discrepancy",
MB_YESNOCANCEL|MB_ICONQUESTION);
            switch(selection) {
                  case IDYES:
                        sprintf(msgbuf, "%d", min_pixelsize+1);
                        framesize = msgbuf;
                        UpdateData(FALSE);
                        info->detect_frame_size = min_pixelsize+1;
                        break;
                  case IDNO:
                        break;
                  case IDCANCEL:
                        return(1);
                        break;
            } //switch
      } // if framesize < minpixelsize
```

```
        return 0;
}

int Cps_scanDlg::check_pixel_size(struct imageArray *image)
{
        int cntx, cnty;
        int minsize=image->x_size;  // something that we know is too
big
        int black_cnt;

        //first scan the array horizontally; storing the smallest
number
        //of pixels found in any row
        for(cnty=0; cnty < image->y_size; cnty++) {
                black_cnt = 0;
                for(cntx=0; cntx < image->x_size; cntx++) {
                        if(image->array[cnty][cntx] == '1') {
                                black_cnt++;
                        } else {
                                if(black_cnt > 0 && black_cnt < minsize) {
                                        minsize = black_cnt;
                                }
                                black_cnt = 0;
                        }
                } //for cntx
                if(black_cnt > 0 && black_cnt < minsize) {
                        minsize = black_cnt;
                }
        } // for cnty

        //now go through the array vertically
        for(cntx=0; cntx < image->x_size; cntx++) {
                black_cnt = 0;
                for(cnty=0; cnty < image->y_size; cnty++) {
                        if(image->array[cnty][cntx] == '1') {
                                black_cnt++;
                        } else {
                                if(black_cnt > 0 && black_cnt < minsize) {
                                        minsize = black_cnt;
                                }
                                black_cnt = 0;
                        }
                } //for cnty
                if(black_cnt > 0 && black_cnt < minsize) {
                        minsize = black_cnt;
                }
        } // for cntx
        return minsize;
}
```

CODE: Ps scanDialog H File

```
// ps_scanDlg.h : header file
```

```
//

#pragma once

// Cps_scanDlg dialog
class Cps_scanDlg : public CDialog
{
// Construction
public:
      Cps_scanDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
      enum { IDD = IDD_PS_SCAN_DIALOG };

      protected:
      virtual void DoDataExchange(CDataExchange* pDX);      //
DDX/DDV support


// Implementation
protected:
      HICON m_hIcon;

      // Generated message map functions
      virtual BOOL OnInitDialog();
      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
      afx_msg void OnPaint();
      afx_msg HCURSOR OnQueryDragIcon();
      DECLARE_MESSAGE_MAP()
public:

      CString dp_filename;
      CString ps_fname;
      CString zcoord;
      CString ca_fname;
      CString framesize;
      CString minpixelsize;
      CString startindex;
      afx_msg void OnBnClickedBexit();
      afx_msg void OnBnClickedBrun();
      void errorMessage(char* msg, char *title);
      afx_msg void OnBnClickedBbrowseimg();
      afx_msg void OnBnClickedBbrowsedp();
      afx_msg void OnBnClickedBbrowseca();
      CString imgxsize;
      CString imgysize;
      int validate_inputs(struct inputInfo * info);
      int validate_options(struct inputInfo *info, struct imageArray
*image);
      int check_pixel_size(struct imageArray *image);
};
```

CODE: Read ascii File CPP File

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "stdafx.h"
#include "scan_array_types.h"
#include "dump_image.h"
#include "bwmisc.h"

struct imageArray *read_ascii_file(char *ascii_name) {

      FILE *in;
      int file_error;
      int linecnt;
      char in_line[MAXLINELEN];
      static char *data_array[MAX_IMAGE_Y];
      char *cptr1, *cptr2;
      size_t x_bits;
      struct imageArray *ret;

      file_error = 0;

      //initialize  the image array structure
      ret = (struct imageArray *) bw_malloc(sizeof(struct
imageArray));
      if(ret == NULL) {
            memory_error();
      }
      ret->array = data_array;
      ret->y_size = 0;
      for(linecnt = 0; linecnt < MAX_IMAGE_Y; linecnt++) {
            data_array[linecnt] = NULL;
      }
      in=fopen(ascii_name, "r");
      if(in == NULL) {
            //no output here; calling routine will display error
            return(NULL);
      }

      // skip header lines
      fgets(in_line, MAXLINELEN, in);
      fgets(in_line, MAXLINELEN, in);
      fgets(in_line, MAXLINELEN, in);

      /* read image data and store the data in the array */
      linecnt = -1;
      while(!feof(in) && (linecnt < MAX_IMAGE_Y)) {
            fgets(in_line, MAXLINELEN, in);
            linecnt++;
            in_line[strlen(in_line)-1] = '\0';  //strip trailing
newline
```

```
            x_bits = strlen(in_line);
            data_array[linecnt] = (char *) calloc(x_bits+1,
sizeof(char));
            if(data_array[linecnt] == NULL) {
                  memory_error();
            }
            for(cptr1 = data_array[linecnt], cptr2 = &in_line[0];
*cptr2; cptr1++, cptr2++) {
                  *cptr1 = (*cptr2 == ' ') ? '0' : '1';
            }
            *cptr1 = '\0';
      } // while read image data
      ret->x_size = (int) x_bits;
      ret->y_size = linecnt;

      //cleanup
      fclose(in);

      return(ret);
} //read_ps_file
```

CODE: Read ascii File H File

```
//function prototypes

extern struct imageArray *read_ascii_file(char *ascii_name);
```

CODE: Read ps File CPP File

```
/* name of program == postscript_bitmap_convert
   This routine will read a postscript file, store the scanned data
into a two-dimensional
   array (data_array), and optionally write the scanned data to a
binary ASCII array file.
   Return is a pointer to the data array or NULL on error
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

#include "stdafx.h"
#include "scan_array_types.h"
#include "bwmisc.h"

/* ________________________________________________________ */

char *ctob(char ch)
{
      static char ret[4];
```

```
        strcpy(ret, "xxxx");
        switch(ch) {
              case '0': strcpy(ret,"0000"); break;
              case '1': strcpy(ret,"0001"); break;
              case '2': strcpy(ret,"0010"); break;
              case '3': strcpy(ret,"0011"); break;
              case '4': strcpy(ret,"0100"); break;
              case '5': strcpy(ret,"0101"); break;
              case '6': strcpy(ret,"0110"); break;
              case '7': strcpy(ret,"0111"); break;
              case '8': strcpy(ret,"1000"); break;
              case '9': strcpy(ret,"1001"); break;
              case 'A': strcpy(ret,"1010"); break;
              case 'B': strcpy(ret,"1011"); break;
              case 'C': strcpy(ret,"1100"); break;
              case 'D': strcpy(ret,"1101"); break;
              case 'E': strcpy(ret,"1110"); break;
              case 'F': strcpy(ret,"1111"); break;
        } // switch
        return ret;
} // ctob

char *htob(int ch)
{
        static char ret[5];
        strcpy(ret, "xxxx");

        switch(ch) {
              case 0: strcpy(ret,"0000"); break;
              case 1: strcpy(ret,"0001"); break;
              case 2: strcpy(ret,"0010"); break;
              case 3: strcpy(ret,"0011"); break;
              case 4: strcpy(ret,"0100"); break;
              case 5: strcpy(ret,"0101"); break;
              case 6: strcpy(ret,"0110"); break;
              case 7: strcpy(ret,"0111"); break;
              case 8: strcpy(ret,"1000"); break;
              case 9: strcpy(ret,"1001"); break;
              case 10: strcpy(ret,"1010"); break;
              case 11: strcpy(ret,"1011"); break;
              case 12: strcpy(ret,"1100"); break;
              case 13: strcpy(ret,"1101"); break;
              case 14: strcpy(ret,"1110"); break;
              case 15: strcpy(ret,"1111"); break;
        } // switch
        return ret;
} // htob

struct imageArray *read_ps_file(char *postscript_name) {

        FILE *in;
        int file_error;
        unsigned int linecnt;
        char in_line[MAXLINELEN];
```

```
	static char *data_array[MAX_IMAGE_Y];
	char *cptr;
	unsigned int ccnt;
	size_t image_x_size, x_bits;
	struct imageArray *ret;

	file_error = 0;
	image_x_size = 0;

	//initialize  the image array structure
	ret = (struct imageArray *) bw_malloc(sizeof(struct
imageArray));
	if(ret == NULL) {
		memory_error();
	}
	ret->array = data_array;
	ret->y_size = 0;

	for(linecnt = 0; linecnt < MAX_IMAGE_Y; linecnt++) {
		data_array[linecnt] = NULL;
	}
	in=fopen(postscript_name, "r");
	if(in == NULL) {
		//no output here; calling routine will display error
		return(NULL);
	}

	// find first line of postscript data
	while(fgets(in_line, MAXLINELEN, in) == NULL) {
		if(in_line[0] == '0') {
			break;  //from while loop
		}
	}
	if(feof(in)) {
		fprintf(stderr, "No image data found in file %s\n",
postscript_name);
		return(NULL);
	}

	/* read image data and store the data in the array */
	linecnt = 0;
	in_line[strlen(in_line)-1] = '\0';  //strip trailing newline
	while(!feof(in) && (in_line[0] == '0') && (linecnt <
MAX_IMAGE_Y)) {
		x_bits = (strlen(in_line) -1) * 4;  // strip leading 0
		if(x_bits > image_x_size) {
			image_x_size = x_bits;
		}
		data_array[linecnt] = (char *) calloc(x_bits,
sizeof(char));
		if(data_array[linecnt] == NULL) {
			memory_error();
		}
```

```
            for(cptr = data_array[linecnt], ccnt=1; ccnt <
strlen(in_line); cptr+=4, ccnt++) {
                  strcpy(cptr,ctob(in_line[ccnt]));
            }
            fgets(in_line, MAXLINELEN, in);
            in_line[strlen(in_line)-1] = '\0';
            linecnt++;
      } // while read image data
      ret->y_size = linecnt;

      //cleanup
      fclose(in);

      return(ret);
} //read_ps_file
```

<u>CODE: Read ps File H File</u>

```
//function prototypes

extern struct imageArray *read_ps_file(char *postscript_name);
```

<u>CODE: Resource H File</u>

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by ps_scan.rc
//
#define IDR_MANIFEST                    1
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_PS_SCAN_DIALOG              102
#define IDR_MAINFRAME                   128
#define IDC_psname                      1001
#define IDC_dpname                      1003
#define IDC_caname                      1004
#define IDC_zname                       1006
#define IDC_pslabel                     1007
#define IDC_zlabel                      1008
#define IDC_dplabel                     1009
#define IDC_calabel                     1010
#define IDC_dfsize                      1011
#define IDC_mpsize                      1012
#define IDC_stindex                     1013
#define IDC_dfsizelabel                 1015
#define IDC_mpsizelabel                 1016
#define IDC_stindlabel                  1017
#define IDC_BRUN                        1018
#define IDC_BEXIT                       1019
#define IDC_IMGXSZ                      1020
```

```
#define IDC_IMGYSZ                          1021
#define IDC_BBROWSEIMG                      1022
#define IDC_BBROWSEDP                       1023
#define IDC_BBROWSECA                       1024

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE            130
#define _APS_NEXT_COMMAND_VALUE             32771
#define _APS_NEXT_CONTROL_VALUE             1025
#define _APS_NEXT_SYMED_VALUE               104
#endif
#endif
```

CODE: Scan Array CPP File

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <process.h>

#include "stdafx.h"
#include "scan_array_types.h"
#include "dump_image.h"
#include "bwmisc.h"
#include "xyelem.h"

#ifdef BW_DEBUG
extern size_t bw_mallocsize;
extern FILE *dbgfil;
#endif

/* ____________________________________________ */
float out_coord(int pixel, int size, int numpix)
{
      return((float)(size*pixel)/(float)numpix);
} //xdimout

struct XY *merge_lists(struct XY *l1, struct XY *l2)
{
      struct XY *l1p, *end_l1p;
      if(l2 == NULL) {
            return(l1);
      }
      if(l1 == NULL) {
            return(l2);
      }

      for(l1p = l1; l1p; l1p = l1p->next) {
            if(l1p->next == NULL) {
```

```
                        end_llp = llp;
                }
        }
        end_llp->next = l2;
        return(l1);

} // merge_lists

static struct XY *find_data_point(char *image_array[])
{
        unsigned int y;
        struct XY *ret;
        char *pixel;

        //ret = (struct XY *) bw_malloc(sizeof(struct XY));
        ret = new_xy_struct();
        if(ret == NULL) {
                memory_error();
        }
        ret->x = -1;
        ret->y = -1;
        ret->next = NULL;

        for(y=0; image_array[y] != NULL; y++) {
                pixel=strchr(image_array[y],'1');
                if(pixel != NULL) {
                        ret->x = (unsigned int) (pixel-image_array[y]);
                        ret->y = y;
                        break;  // from for-y loop
                }
        } // for y

        return ret;
} // find_data_point

static char *generate_detection_frame(struct XY *pt, struct
imageArray *image, int frame_size)
{
        static char *detection_data = NULL;
        static int detection_data_size = -1;
        int x, y, detection_index;
        char **image_array = image->array;

        if(detection_data == NULL) {
                detection_data_size = 8*frame_size+1;
                detection_data = (char *) bw_malloc(detection_data_size *
sizeof(char));
                if(detection_data == NULL) {
                        memory_error();
                }
        }
        for(x=0; x<detection_data_size; x++) {
                detection_data[x] = '0';
        }
```

```
      detection_data[detection_data_size-1] = '\0';

      //scan the left side of frame from top to bottom for black .
pixels
      x=pt->x-frame_size;
      detection_index=0;
    for(y=pt->y-frame_size; y <= pt->y+frame_size; y++,
detection_index++) {
             if((x < 0) || (y < 0) || (y >= image->y_size) || (x >=
(int) strlen(image_array[y]))) {
                    continue; // with y loop
             }
             detection_data[detection_index] = *((image_array[y]) +
x);
      } // for y

      // scan the bottom side of frame from left to rt, skip lower-
left
      y=pt->y+frame_size;
    for(x=pt->x-frame_size+1; x <= pt->x+frame_size; x++,
detection_index++) {
             if((x < 0) || (y < 0) || (y >= image->y_size) || (x >=
(int) strlen(image_array[y]))) {
                    continue; // with y loop
             }
             detection_data[detection_index] = *((image_array[y]) +
x);
      }

      // scan the right side of frame from bottom to top, skip lower-
right
      x=pt->x+frame_size;
    for(y=pt->y+frame_size-1; y >= pt->y-frame_size; y--,
detection_index++) {
             if((x < 0) || (y < 0) || (y >= image->y_size) || (x >=
(int) strlen(image_array[y]))) {
                    continue; // with y loop
             }
             detection_data[detection_index] = *((image_array[y]) +
x);
      }

      // scan the top side of frame from rt to left, skip both
corners
      y=pt->y-frame_size;

    for(x=pt->x+frame_size-1; x >= pt->x-frame_size+1; x--,
detection_index++) {
             if((x < 0) || (y < 0) || (y >= image->y_size) || (x >=
(int) strlen(image_array[y]))) {
                    continue; // with y loop
             }
             detection_data[detection_index] = *((image_array[y]) +
x);
```

```
	}

	return(detection_data);
}

static struct XY *det_data_xy_transform(struct XY *pt, int newctr,
int frame_size)
{
	struct XY *xy;

	//xy = (struct XY *) bw_malloc(sizeof(struct XY));
	xy = new_xy_struct();
	if(xy == NULL) {
		memory_error();
	}
	xy->next = NULL;

	if(newctr <= 2*frame_size) { // on left side of frame
		xy->x = pt->x - frame_size;
		xy->y = pt->y + (newctr - frame_size);
	} else if (newctr <= 4*frame_size) { // on bottom of frame
		xy->x = pt->x + (newctr - 3*frame_size);
		xy->y = pt->y + frame_size;
	} else if (newctr <= 6*frame_size) { // on right of frame
		xy->x = pt->x + frame_size;
		xy->y = pt->y + 5*frame_size - newctr;
	} else { // on top of frame
		xy->x = pt->x + 7*frame_size - newctr;
		xy->y = pt->y - frame_size;
	}

	return(xy);
}

static struct XY *find_new_data_points(char *detection_data, struct
XY *pt, struct inputInfo *input_info)
{
	struct XY *newxy;
	struct XY *ret;
	int black_start;
	char *cptr;
	int pixel_size;

	ret = NULL;
	black_start = -1;
	for(cptr = detection_data; *cptr; cptr++) {
		if(*cptr == '1') {
			if(black_start == -1) {
				black_start = (int) (cptr - detection_data);
			}
		} else {
			if(black_start != -1) {
				pixel_size = (int) (cptr-detection_data) -
black_start;
```

```
                                if(pixel_size >= input_info->min_pixel_size)
{
                                        newxy = det_data_xy_transform(pt,
(black_start-1+(int)(cptr-detection_data))/2, input_info-
>detect_frame_size);
                                        newxy->next = ret;
                                        ret = newxy;
                                } // if pixel is big enough
                                black_start = -1;
                        } // if black segment found
                } // if pixel was white
        } //for

        //process the last point if we got to the end
        if(black_start != -1) {
                pixel_size = (int) (cptr-detection_data) - black_start;
                if(pixel_size >= input_info->min_pixel_size) {
                        newxy = det_data_xy_transform(pt, (black_start-
1+(int)(cptr-detection_data))/2, input_info->detect_frame_size);
                        newxy->next = ret;
                        ret = newxy;
                } // if pixel is big enough
                black_start = -1;
        }

        return(ret);
}

void clear_detection_frame(struct XY *pt, struct imageArray *image,
int frame_size)
{
        int x, y;
        char **image_array = image->array;

        for(y=pt->y-frame_size; y<= pt->y+frame_size; y++) {
                if((y < 0) || (y >= image->y_size)) {
                        continue; // with next y
                }
                for(x=pt->x-frame_size; x<= pt->x+frame_size; x++) {
                        if((x < 0) || (x >= (int) strlen(image_array[y])))
{
                                continue; // with next x
                        }
                        *(image_array[y] + x) = '0';  // clear the pixel
                } // for x
        } //for y

}

int process_data(struct XY *pt, struct imageArray *image, int
start_index, struct inputInfo *input_info, FILE *dpoint_file, FILE
*carray_file, FILE *dump_file)
{
        struct XY *newpt, *newpts;
        char *detection_data;
        int cur_index = start_index;
        struct XY *dp_stack = NULL;
        struct XY *pt_to_discard;

        while(pt != NULL) {
              fprintf(dpoint_file, "T%d= %.4f %.4f %s 0\n",
cur_index++,
                    out_coord(pt->x, input_info->img_x_size*10, image-
>x_size),
                    out_coord(pt->y, input_info->img_y_size, image-
>y_size), input_info->z_coord);
              detection_data = generate_detection_frame(pt, image,
input_info->detect_frame_size);

              //find next data points and write connectivity array data
              newpts = find_new_data_points(detection_data, pt,
input_info);
#ifdef BW_DEBUG
              fprintf(dbgfil, "Process Data Loop: pt=(%d %d)\n", pt->x,
pt->y);
              fprintf(dbgfil,"  newpts:");
#endif
              for(newpt=newpts; newpt; newpt = newpt->next) {
#ifdef BW_DEBUG
                    fprintf(dbgfil, " (%d %d)", newpt->x, newpt->y);
#endif
                    fprintf(carray_file, "(%.4f %.4f %s) (%.4f %.4f
%s)\n",
                          out_coord(pt->x, input_info->img_x_size*10,
image->x_size),
                          out_coord(pt->y, input_info->img_y_size,
image->y_size),
                          input_info->z_coord,
                          out_coord(newpt->x, input_info-
>img_x_size*10, image->x_size),
                          out_coord(newpt->y, input_info->img_y_size,
image->y_size),
                          input_info->z_coord);
              }
#ifdef BW_DEBUG
              fprintf(dbgfil, "\n");
              fflush(dbgfil);
#endif

              //if more than one new point found, store them on stack
for later processing
              if(newpts != NULL) {
                    dp_stack = merge_lists(dp_stack, newpts->next);
              }

              //clear the data internal to the detection frame
              clear_detection_frame(pt, image, input_info-
>detect_frame_size);
```

```
            //next point is the first new point found or a point off
of the stack
            pt_to_discard = pt;
            if(newpts) {
                  pt = newpts;
            } else {
                  if(dp_stack) {
                        pt = dp_stack;
                        dp_stack = dp_stack->next;
                  } else {
                        pt = NULL;
                  }
            }

            free_xy_struct(pt_to_discard);
#ifdef BW_DEBUG
            fprintf(dbgfil, "  memory out: %d\n", bw_mallocsize);
            fflush(dbgfil);
            fprintf(dbgfil, "  xy-elems allocated: %d\n",
cnt_xy_struct());
#endif

      } // while more data points to process

      return cur_index;
}

int scan_array(struct imageArray *image, struct inputInfo
*input_info, char *datapoint_fname, char *connarray_fname)
{
      FILE *dpoint_file = fopen(datapoint_fname, "w");
      FILE *carray_file = fopen(connarray_fname, "w");
      FILE *dump_file = NULL;
      struct XY *datapt;

      int cur_index = input_info->start_index;
      int scan_cnt = 0;

      if((dpoint_file == NULL) || (carray_file == NULL)) {
            //no output here; calling routine will display window
            return(1);
      }

      //dump image if requested
      if(input_info->dumpimagefname != NULL) {
            dump_file = fopen(input_info->dumpimagefname, "w");
      }
      dump_image(dump_file, image, "Initial Image");

      //find first datapoint
      datapt = find_data_point(image->array);

      //while data is left in the array, process it
      while(datapt->x != -1) {
            scan_cnt++;
            cur_index = process_data(datapt, image, cur_index,
input_info, dpoint_file, carray_file, dump_file);
            datapt = find_data_point(image->array);
            dump_image(dump_file, image, "After scan");
      }

      fclose(dpoint_file);
      fclose(carray_file);
      if(dump_file != NULL) {
            fclose(dump_file);
      }

      return(0);
} // scan_array
```

CODE: Scan Array H File

```
//function prototypes
extern int scan_array(struct imageArray *image, struct inputInfo
*input_info, char *datapoint_fname, char *connarray_fname);

#include "scan_array_types.h"
```

CODE: Scan Array types H File

```
#define MAXLINELEN 4000
#define MAX_IMAGE_X 4000  // must be <= MAXLINELEN
#define MAX_IMAGE_Y 1000
#define MAX_XY_ELEM 100

struct XY {
      int x;
      int y;
      struct XY *next;
};

struct imageArray {
      char **array;
      int x_size;
      int y_size;
};

struct inputInfo {
      int min_pixel_size;
      int detect_frame_size;
      char *z_coord;
      int start_index;
      char *dumpimagefname;
      int img_x_size;
```

```
    int img_y_size;
};
```

CODE: Stdafx CPP File

```
// stdafx.cpp : source file that includes just the standard includes
// ps_scan.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

CODE: Stdafx H File

```
   // stdafx.h : include file for standard system include files,
  // or project specific include files that are used frequently,
                // but are changed infrequently

                     #define _MAX_PATH 1024

                     #pragma once

                      #ifndef VC_EXTRALEAN
  #define VC_EXTRALEAN           // Exclude rarely-used stuff from
                         Windows headers
                            #endif

  // Modify the following defines if you have to target a platform
                  prior to the ones specified below.
  // Refer to MSDN for the latest info on corresponding values for
                      different platforms.
#ifndef WINVER                        // Allow use of features specific
               to Windows 95 and Windows NT 4 or later.
#define WINVER 0x0400          // Change this to the appropriate value
            to target Windows 98 and Windows 2000 or later.
                            #endif

 #ifndef _WIN32_WINNT           // Allow use of features specific to
                   Windows NT 4 or later.
#define _WIN32_WINNT 0x0400          // Change this to the appropriate
        value to target Windows 98 and Windows 2000 or later.
              #endif

 #ifndef _WIN32_WINDOWS        // Allow use of features specific to
                    Windows 98 or later.
#define _WIN32_WINDOWS 0x0410 // Change this to the appropriate value
                 to target Windows Me or later.
                            #endif

#ifndef _WIN32_IE              // Allow use of features specific to IE
                      4.0 or later.
#define _WIN32_IE 0x0400       // Change this to the appropriate value
                  to target IE 5.0 or later.
```

```
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // some CString
                constructors will be explicit

// turns off MFC's hiding of some common and often safely ignored
                warning messages
#define _AFX_ALL_WARNINGS

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes

#include <afxdtctl.h>       // MFC support for Internet Explorer 4
                Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common
                Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
```

CODE: XYelem CPP File

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

#include "stdafx.h"
#include "scan_array_types.h"

static int elem_initialized = 0;
static int elem_used[MAX_XY_ELEM];
static struct XY elem_array[MAX_XY_ELEM];

struct XY *new_xy_struct()
{
      int cnt;
      char buf[200];

      if(elem_initialized == 0) {
            for(cnt = 0; cnt < MAX_XY_ELEM; cnt++) {
                  elem_used[cnt] = 0;
            }
            elem_initialized = 1;
      } // if need to initialize array

      //find the first unused element and return a pointer to it
      for(cnt = 0; cnt < MAX_XY_ELEM; cnt++) {
            if(elem_used[cnt] == 0) {
                  elem_used[cnt] = 1;
                  return(&(elem_array[cnt]));
            }
      }
      // if we got here we ran out of elements
      sprintf(buf, "Internal Error: Out of XY elements (%d).  Try a
larger frame size", MAX_XY_ELEM);
      AfxMessageBox(buf);
      exit(1);
} // new_xy_struct

void free_xy_struct(struct XY *el)
{
      elem_used[(int) (el - &(elem_array[0]))] = 0;
}

int cnt_xy_struct()
{
      int cnt, ret;

      ret=0;
      for(cnt = 0; cnt < MAX_XY_ELEM; cnt++) {
            if(elem_used[cnt] == 1) {
                  ret += 1;
            }
      } // for
      return(ret);
}
```

CODE: XYelem H File

```
extern struct XY *new_xy_struct();
extern void free_xy_struct(struct XY *el);
extern int cnt_xy_struct();
```

APPENDIX E

RONALD C. BAUMAN'S FRONT END SOURCE CODE

Note: This front end program coverts a postscript file into a binary ACII array file and has since been updated to compile for this project as research. The back end program included in Appendix F has also been updated to compile for this project as research. Both programs still have problems processing data and were never used in this thesis to produce data. Portions of Bauman's program logic has been used as the bases for the new "ps_scan" software program officially developed and demonstrated in this thesis project.

CODE: Postscript Bitmap Convert

```
/* name of program == postscript_bitmap_convert
   This program coverts a postscript file into a binary ACII array
file
*/

#include <stdio.h>

FILE      *in,
          *out;
char*     array_name[256],
          *postscript_name[256], ch;
short int i,
          a;

/*________________________________________________________________*/

main(void) {
  printf("Enter the name of the bitmap file to be converted: ");
  gets(postscript_name);
  printf("Enter the name of the new array file: ");
  gets(array_name);
  printf("input file = %s output file = %s\n",postscript_name,
array_name);
```

```
  in=fopen(postscript_name, "r");
  out=fopen(array_name, "w");
  a=0;

label1:
  if(a == 1) {
    fclose(in);
    fclose(out);
    exit(0);
  }

label2:
  ch=fgetc(in);
  if(ch == '0') {
    a=1;
    fputc(ch, out);
    for(i=0; i <= 4000; i++) {
      ch=fgetc(in);
      if(ch == '0') {
        fputc('0', out);
        fputc('0', out);
        fputc('0', out);
        fputc('0', out);
      }
      if(ch == '1') {
        fputc('0', out);
        fputc('0', out);
        fputc('0', out);
        fputc('1', out);
      }
      if(ch == '2') {
        fputc('0', out);
        fputc('0', out);
        fputc('1', out);
        fputc('0', out);
      }
      if(ch == '3') {
        fputc('0', out);
        fputc('0', out);
        fputc('1', out);
        fputc('1', out);
      }
      if(ch == '4') {
        fputc('0', out);
        fputc('1', out);
        fputc('0', out);
        fputc('0', out);
      }
      if(ch == '5') {
        fputc('0', out);
        fputc('1', out);
        fputc('0', out);
        fputc('1', out);
      }
```

```
if(ch == '6') {
  fputc('0', out);
  fputc('1', out);
  fputc('1', out);
  fputc('0', out);
}
if(ch == '7') {
  fputc('0', out);
  fputc('1', out);
  fputc('1', out);
  fputc('1', out);
}
if(ch == '0') {
  fputc('8', out);
  fputc('1', out);
  fputc('0', out);
  fputc('0', out);
}
if(ch == '9') {
  fputc('1', out);
  fputc('0', out);
  fputc('0', out);
  fputc('1', out);
}
if(ch == 'a') {
  fputc('1', out);
  fputc('0', out);
  fputc('1', out);
  fputc('0', out);
}
if(ch == 'b') {
  fputc('1', out);
  fputc('0', out);
  fputc('1', out);
  fputc('1', out);
}
if(ch == 'c') {
  fputc('1', out);
  fputc('1', out);
  fputc('0', out);
  fputc('0', out);
}
if(ch == 'd') {
  fputc('1', out);
  fputc('1', out);
  fputc('0', out);
  fputc('1', out);
}
if(ch == 'e') {
  fputc('1', out);
  fputc('1', out);
  fputc('1', out);
  fputc('0', out);
}
```

```
      if(ch == 'f') {
        fputc('1', out);
        fputc('1', out);
        fputc('1', out);
        fputc('1', out);
      }
      if(ch == '\n' || ch==EOF || ch==255) {
        fputc(ch, out);
        break;
      }
    }
    goto label2;
  }

  for(i=0; i <= 4000; i++) {
    ch=fgetc(in);
    if(ch == '\n' || ch==EOF || ch==255) {
      break;
    }
  }

  goto label1;
}
```

APPENDIX F

RONALD C. BAUMAN'S BACK END SOURCE CODE

CODE: /* name of program = binary array scan

This program scans the array row by row progressively moving from the top to the bottom of the array. Once a black pixel is detected, it becomes the first data point and the program draws a detection frame around it. The detection frame, which is a square of single array element thickness, is now scanned. If the program detects a number of consecutive black pixels on the frame it locates another data point on the black pixel segment center. The detection frame is now moved to the newly found data point, the area within the previous " " detection frame is converted to all zeros (i.e. white pixels), and the process is repeated. If the detection frame encounters multiple black pixel segments on its perimeter, the program creates data points for each segment and picks one of the new " " data points for continued scanning. Then, if a detection frame detects no black pixel segments on its perimeter or detects multiple black pixel segments (i.e. another branch pt.) the program moves back to the last branch pt. and continues scanning on an unscanned branch. If a detection frame detects no black pixel segments on " " its perimeter and there are no branch points to move back to, the program saves the " " data point coordinates and connectivity information, and starts scanning the array again from the top down. This process continues until the array is totally void of any black pixels.
____*/

```
#include <stdio.h>  /* See Appendix J for program listing */
#include <math.h>

static int   comb_pixel_array[1024][792],
             pixel_array[1024][792],
             bi[10000],
             bj[10000],
             frame_size,
             k;

static int   p,
             ci[1000],
             cj[1000],
             end[10000],
             start[10000],
             r,
             dt[10000],
             no_scan,
             startnew[10000];
```

```
static int  endnew[10000],
            ij,
            ptchk;

/* _____________________________________________________________*/
main(void) {
  FILE*     in;
  FILE*     out;
  FILE*     out2;

  short int i,
            j,
            c,
            b,
            d,
            e,
            f,
            g,
            a[2],
            min_line_width,
            frame[400],
            l,
            m,
            n,
            s,
            bpixel,
            free_end[10000],
            bicon[10000],
            bjcon[10000],
            memrmv[100],
            ii,
            ik;

  double    max,
            dist;

  char*     array_name[256],
            ch,
            input_file_name[256];

 /* retrieve array from file and place it into pixel_array,
 comb_pixel_array */
  printf("Enter the name of the array to be scanned:");
  gets(array_name);
  in=fopen(array_name, "r");
  puts("get_array started");
  for(i=0; i <= 1023; i++) {
    for(j=0; j <= 791; j++) {
      fscanf(in, "%ld", &pixel_array[i][j]);
      comb_pixel_array[i][j]=pixel_array[i][j];
    }

 labl:
      fscanf(in, "%c", &ch);
      if(ch != '\n') {
        goto lab1;
      }
    }
    puts("get_array completed");
    fclose(in);
    r=1;

  /* scan the array row by row progressively moving from the top to the
      bottom of the array until an array element equal to 1 (i.e. black
  pixel)
      is detected */

  lab2:
    k=k + 1;
    bpixel=1;
    puts("scan_array started");
    no_scan=1;
    for(i=0; i <= 1023; i++) {
      for(j=0; j <= 791; j++) {
        if(pixel_array[i][j] == bpixel) {
          bi[k]=i;
          bj[k]=j;
          puts("scan_array completed");
          no_scan=0;
          break;
        }
      }
      if(no_scan == 0) {
        break;
      }
    }
    ptchk=k;
    printf("start of structure  k=%d  bi[k]=%d  bj[k]=%d\n", k, bi[k],
  bj[k]);
    if(no_scan == 1) {
      goto lab3;
    }

  /* Scan a square perimeter around the array element (bi,bj) recieved
  from
      scan array. Analyze perimeter and calculate new data pt(s). If
  more than
      one data pt. is detected the pick one of them and continue the
  detection
      process. If another branch is detected move back to the previous
  branch
      pt. and continue on one of the unexplored branchs. When the
  square
      perimeter fails to detect any black pixels (array elements=1),
  continue.
      The data generated contains data pt. coordinates and connectivity
      information. */
  puts("frame_scan_array started");
  min_line_width=2; /* minimum pixel width of detectable line */
  frame_size=12;    /* width and height of detection frame */
  n=p=0;

lab4:
  j=f=m=0;

/* define the frame with pixel array elements */
  for(i=((-frame_size) / 2); i <= (frame_size / 2 - 1); i++) {
    frame[j]=pixel_array[bi[k] + i][bj[k] + (frame_size / 2)];
    frame[j + frame_size]=pixel_array[bi[k] + frame_size / 2][bj[k] -
i];
    frame[j + 2 * frame_size]=pixel_array[bi[k] - i][bj[k] -
frame_size / 2];
    frame[j + 3 * frame_size]=pixel_array[bi[k] - frame_size /
2][bj[k] + i];
    j=j + 1;
  }

/* locates black pixel segments on the detection frame */
  i=(-1);

lab5:
  d=i + 1;
  e=0;
  c=1;
  for(i=d; i <= (4 * frame_size - 1); i++) {
    if(i == (4 * frame_size - 1)) {
      if(frame[i] == 1) {
        if(f == 2) {

/* handles detected points on */
          if(c == 1) {
            a[1]=4 * frame_size - 1;
          }

/* the last frame element */
          a[0]=g;
          goto lab6;
        }

        if(c == 0) {
          a[0]=4 * frame_size - 1;
          goto lab6;
        }
      }
    }

    if(frame[i] == c) {

/* if a black pixel is detected at frame[0] then f=1 */
      if(i == 0) {
          f=1;
        }

        a[c]=i;
        c=0;
        e=e + 1;

/* turns black pixel segments on the frame into new data pts £
          if this is the end of a segment on the detection frame */
        if(e == 2) {

/* if a black pixel was detected at frame[0] */
          if(f == 1) {
            g=a[0] + 4 * frame_size;
            f=2;
            goto lab5;
          }

/* compute length of segment on detection frame */
lab6:
          b=a[0] - a[1];

/* check for minimum segment length,ie.(min.line width) */
          if(b >= min_line_width) {

/* counter for number of segments on a frame */
            m=m + 1;

/* compute center of segment */
            j=b / 2 + a[1];

/* if segment center > 4*frame_size, then subtract 4*frame_size */
            if(j >= 4 * frame_size) {
              j=j - 4 * frame_size;
            }
          }

/* if segment length < min_line_width then ignore it */
          else {
            goto lab5;
          }

/* increment counter for new data pt. to be defined */
          k=k + 1;
          if(j < frame_size) {
            bi[k]=bi[k - m] - frame_size / 2 + j;
            bj[k]=bj[k - m] + frame_size / 2;
            printf("k=%d bi[k]=%d bj[k]=%d\n", k, bi[k], bj[k]);

/* after the first segment is detected on the frame */
            if(m > 1) {

/* any others are put into the branch data point */
              n=n + 1;
/* array for future scanning */
              ci[n]=bi[k];
              cj[n]=bj[k];
              printf("n=%d ci[n]=%d cj[n]=%d\n", n, ci[n], cj[n]);
            }

            goto lab5;
          }

          if(j < 2 * frame_size) {
            bi[k]=bi[k - m] + frame_size / 2;
            bj[k]=bj[k - m] + frame_size / 2 - (j - frame_size);
            printf("k=%d bi[k]=%d bj[k]=%d\n", k, bi[k], bj[k]);
            if(m > 1) {
              n=n + 1;
              ci[n]=bi[k];
              cj[n]=bj[k];
              printf("n=%d ci[n]=%d cj[n]=%d\n", n, ci[n], cj[n]);
            }

            goto lab5;
          }

/* convert location of data pt. from detection frame coordinates (ie.
length
     of perimeter from top right corner of frame to the data pt.) to
array
     coordinates,where (0,0) is the top left corner */
          if(j < 3 * frame_size) {
            bi[k]=bi[k - m] + frame_size / 2 - (j - 2 * frame_size);
            bj[k]=bj[k - m] - frame_size / 2;
            printf("k=%d bi[k]=%d bj[k]=%d\n", k, bi[k], bj[k]);
            if(m > 1) {
              n=n + 1;
              ci[n]=bi[k];
              cj[n]=bj[k];
              printf("n=%d ci[n]=%d cj[n]=%d\n", n, ci[n], cj[n]);
            }

            goto lab5;
          }

          if(j < 4 * frame_size) {
            bi[k]=bi[k - m] - frame_size / 2;
            bj[k]=bj[k - m] - frame_size / 2 + (j - 3 * frame_size);
            printf("k=%d bi[k]=%d bj[k]=%d\n", k, bi[k], bj[k]);
            if(m > 1) {
              n=n + 1;
              ci[n]=bi[k];
              cj[n]=bj[k];
              printf("n=%d ci[n]=%d cj[n]=%d\n", n, ci[n], cj[n]);
            }
            goto lab5;
          }
        }
      }

/* used when a black pixel(1) is detected in the first frame element
*/
      if(i == 4 * frame_size - 1) {
        if(frame[i] == 0) {
          if(f == 2) {
            a[1]=0;
            a[0]=g - 4 * frame_size;
            i=i + 1;
            goto lab6;
          }
        }
      }
    }

/* erases the black pixels(1) inside the detection frame just used */
    for(i=bi[k - m] - frame_size / 2; i <= bi[k - m] + frame_size / 2;
i++) {
      for(j=bj[k - m] - frame_size / 2; j <= bj[k - m] + frame_size /
2; j++) {
        pixel_array[i][j]=0;
      }
    }

/* if no segments large enough are detected on the frame, then return
to
      next branch point or return to main() if all branches have been
      scanned */
    j=0;
    for(i=0; i <= 4 * frame_size - 1; i++) {
      j=frame[i] + j;
    }

    if(j < min_line_width) {
      if(p == n) {
        goto lab7;
      }
      p=p + 1;
      k=k + 1;

/* pull branch point from branch point array and insert into regular
data
      point array */
      bi[k]=ci[p];
      bj[k]=cj[p];
      goto lab4;
    }

    if(m > 0) {
      for(i=m; i >= 1; i=i - 1) {
        dt[r]=bi[k - m];
        dt[r + 1]=bj[k - m];
        dt[r + 2]=bi[k - i + 1];
        dt[r + 3]=bj[k - i + 1];
        r=r + 4;
      }
    }

  /* reset data pt. counter to the first new data pt. found */
    k=k - (m - 1);
    goto lab4;

  /* Frame scan complete */
  lab7:

  /* if data pt. has no members connected to it then remove it */
    if(k == ptchk) {
      k=k - 1;
    }
    printf("end of structure  k=%d  bi[k]=%d  bj[k]=%d\n", k, bi[k],
  bj[k]);
    goto lab2;

  lab3:
    puts("frame_scan_array completed");

  /* Calculate connectivity */
    printf("*******   r=%d   *******\n", r);
    b=0;
    i=1;
    for(j=1; j <= r; j=j + 2) {
      for(l=1; l <= k - 1; l++) {
        if(dt[j] == bi[l]) {
          if(dt[j + 1] == bj[l]) {
            if(b == 1) {
              end[i]=l;
              b=0;
              i=i + 1;
              goto lab8;
            }
            else {
              start[i]=l;
              b=1;
              goto lab8;
            }
          }
        }
      }

   lab8:
      r=r;
    }

    r=i;
  printf("r=%d\n", r);
  s=0;
  free_end[s]=0;

/* find connectivity ends which are not also starts */
  for(i=1; i <= r - 1; i++) {
    for(j=1; j <= r - 1; j++) {
      if(end[i] == start[j]) {
        goto lab9;
      }
    }

    s=s + 1;
    free_end[s]=end[i];

lab9:
    s=s;
  }

/* find x and y values corresponding to the connectivity ends */
  for(i=1; i <= s; i++) {
    for(j=1; j <= k - 1; j++) {
      if(free_end[i] == j) {
        bicon[i]=bi[j];
        bjcon[i]=bj[j];
      }
    }
  }

/* find which ends are closer than or equal to max */
  max=sqrt((frame_size * frame_size) / 2);
  for(i=1; i <= s; i++) {
    for(j=1; j <= s; j++) {
      if(i == j) {
        goto lab10;
      }

      dist=sqrt((bicon[j] - bicon[i]) * (bicon[j] - bicon[i]) +
                         (bjcon[j] - bjcon[i]) * (bjcon[j] -
bjcon[i]));
      if(dist <= max) {
        start[r]=free_end[i];
        end[r]=free_end[j];
        r=r + 1;
      }

lab10:
      r=r;
    }
  }
  printf("**********   r=%d   **********\n", r);

/* find double members between data pts. and delete one of them */
  ii=ij=0;
  for(i=1; i <= r - 1; i++) {
    for(j=1; j <= r - 1; j++) {
      if(start[i] == end[j]) {
        if(end[i] == start[j]) {
          for(ik=1; ik <= ii; ik++) {
            if(j == memrmv[ik]) {
              goto lab11;
            }
          }

          ii=ii + 1;
          memrmv[ii]=i;
          printf("ii=%d memrmv[ii]=%d\n", ii, memrmv[ii]);

lab11:
          ii=ii;
        }
      }
    }
  }
/* create new member arrays without the double members found
previously */
  for(i=1; i <= r - 1; i++) {
    for(j=1; j <= ii; j++) {
      if(memrmv[j] == i) {
        goto lab12;
      }
    }

    ij=ij + 1;
    startnew[ij]=start[i];
    endnew[ij]=end[i];

lab12:
    ij=ij;
  }

/* Print input file */
  puts("print_input_file started");
  printf("Enter the name of the closed loop input file to be
saved:");
  gets(input_file_name);
  out2=fopen(input_file_name, "w");
  fprintf(out2, " %d  %d\n", (k - 1), ij);
  for(i=1; i <= k - 1; i++) {
    fprintf(out2, "     %d      %d\n", bi[i], bj[i]);
  }

  for(i=1; i <= ij; i++) {
    fprintf(out2, "     %d      %d\n", startnew[i], endnew[i]);
  }
  puts("print_input_file completed");

/* Create new array */
```

```
  puts("create_new_array started");
  for(i=1; i <= k - 1; i++) {
    comb_pixel_array[bi[i]][bj[i]]=2;
  }

  printf("Enter the name of the new array with data pts. and geometry
to be saved:");
  gets(array_name);
  out=fopen(array_name, "w");
  for(i=0; i <= 1023; i++) {
    for(j=0; j <= 791; j++) {
      fprintf(out, "%1d", comb_pixel_array[i][j]);
    }
    fprintf(out, "\n");
  }

  puts("create_new_array completed");
  fclose(out);
                                              }
```

REFERENCES

REFERENCES

[1] Seiko Instruments USA, Inc. 1990, XM5000 Engineering Manual release 2. Torrance CA: Seiko Instruments USA.

[2] Seiko Instruments USA, Inc. 1990, DARLfour Programming Guide. Torrance CA: Seiko Instruments USA.

[3] Ortwin Ohtmer Dr. -Ing (19XX) Automatic Solid Modeling Applying Graphical Pattern Recognition Techniques. Long Beach, CA: California State University, Mechanical Engineering Department.

[4] Ortwin Ohtmer Dr, -Ing (1989) Automatic FE-, BE Input Generation, Adaptation and Definition of Numerical Analysis Using Graphical Pattern Recognition Techniques. Proceedings, International Conference Structural Engineering, FEM + CAD, Los Angeles, CA.

[5] Andrew Wojtowski, Graphic and Interface Designer for Oracular. Oracular is a premiere provider of professional Information on the web. http://www.oracular.com

[6] Ronald C. Bauman, (1989) Methods and Applications of Graphical Pattern Recognition. Long Beach, CA: California State University, Mechanical Engineering Department.