THREE DIMENSIONAL SURFACE DISPLAY DEVICE

By

JEREMY MAYER

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

ACKNOWLEDGMENTS

The author greatly appreciates the support of the Center for Intelligent Machines and Robotics at the University of Florida for providing a Research Assistantship. The author would also like to thank Dr. Carl Crane for the opportunity to work on a project that involved mechanics, an embedded controller, and computer programming. Additionally, the help and support from the other members of the committee, Dr. Joseph Duffy, Dr. Eric Schwartz and Dr. Michael Nechyba is greatly appreciated. Finally, the author is thankful to his friend and colleague Arfath Pasha. His expertise in computer programming made the Internet-based user interface possible.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

THREE DIMENSIONAL SURFACE DISPLAY DEVICE

By

Jeremy Mayer

December 2001

Chairman: Dr. Carl D. Crane III
Major Department: Mechanical Engineering

This dissertation presents the design of a device that displays three-dimensional (3-D) shapes. The device can also be used to take 3-D data from an existing 3-D object for input into a computer.

Past attempts at designing this type of a device can be categorized three ways: computer graphics imaging, swept volume, or light projection. Many of the past devices fall short either by the limitation of trying to display 3-D objects on two-dimensional (2-D) screens, or by the requirement that the user must wear special goggles or headsets.

In the device presented herein, an actual physical 3-D surface is created that can be seen and touched by multiple people and from multiple view directions. The device creates the 3-D surfaces from an array of axially parallel steel pins. When the individual pins are displaced under computer control to their proper offset positions, the heads of all the pins form the 3-D surface.

CHAPTER 1
INTRODUCTION AND BACKGROUND REVIEW

Scientists, engineers and inventors have designed and demonstrated devices that display three-dimensional surfaces. Three general categories for these devices are as follows:

- Computer graphics display devices.
- Devices that sweep light through a 3-D display volume.
- Devices that project light.

Currently there are no devices described in the literature or on the market that actually create a physical three-dimensional surface. Creating a true 3-D surface has some clear advantages over the other methods for 3-D surface display.

Three-Dimensional Display Using Computer Graphics

Computer display screens, although only two-dimensional, can simulate three-dimensional objects by creating perspective views similar to the perspective view of a cube shown in Figure 1-1. The sheet of paper only provides two dimensions, yet the cube appears to be displayed in three dimensions.

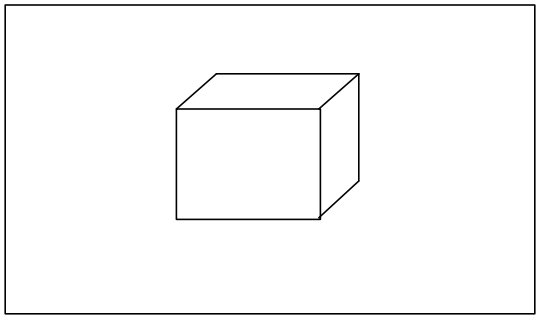


Figure 1-1. Perspective view of a cube

Computer graphics packages are continuously improving this process and improving the significant computations that are required for this type of image display. In addition to creating the perspective view, the software can remove hidden lines and surfaces; and add shading, lighting, and shadows in order to enhance the illusion of three dimensions (Wat91). Figure 1-2 is a computer graphics image created using 3D Studio Max® CAD package.

Figure 1-2. Computer image created using 3D Studio Max® CAD package

Virtual Reality (VR) equipment takes the display of 3-D objects using computer graphics one step farther. Most of the VR equipment is designed to be used with computer screens. The equipment incorporates binocular stereoscopic vision techniques to add 3-D depth perception to the objects. This means that two separate images are generated and viewed. The left eye views the left image and the right eye views the right

image. The two images differ slightly due to visual parallax based on the distance between the eyes. Objects appear to have depth.

The VR equipment that uses a standard computer monitor also requires special VR glasses. The glasses block the left and right eyes in sync with the monitor. In addition to blocking the vision of the respective eyes, the VR software can alter the display depending on the orientation of the viewer with respect to the screen. If the viewer moves to the left, for example, the monitor updates and displays the proper new perspective. This is achieved by a transmitter that is built into the VR glasses and a receiver that is placed above the monitor to determine the position of the viewer.

Head-mounted VR displays use the same technology, but have essentially put two computer screens in front of the viewer's eyes. The viewer is not required to look in the direction of the stationary computer screen. This technique provides a more 'interactive' experience.

Three-Dimensional Display Using Moving Arrays of LEDs

Two implementations using moving arrays of light emitting diodes are now discussed. Figure 1-3 shows an apparatus developed by Ken-ichi Kameyama and Koichi Ohtomi (Kam93). Figure 1-4 is an apparatus patented by Edward Berlin in 1979 (Ber79).

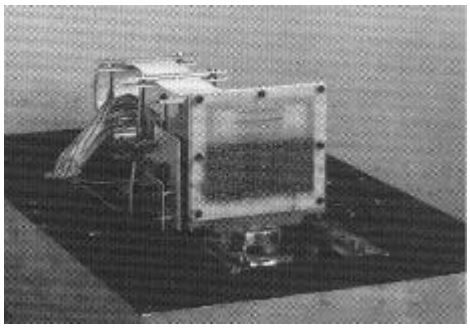


Figure 1-3. Kameyama-Ohtomi Apparatus

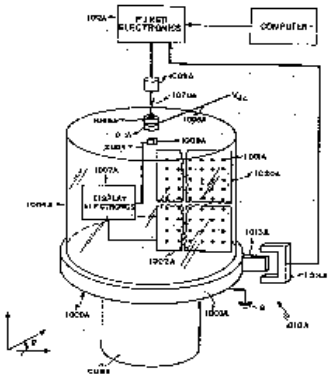


Figure 1-4. Berlin Apparatus

The underlying science behind this method for 3-D display is that the individual LEDs are illuminated at proper times while the entire array sweeps through the display volume. The human eye retains the LED illumination information at the respective places within the display volume, which over time appears to be a 3-D object. The Berlin Apparatus uses a rectangular array of LEDs that spins on its long axis and the Kameyama-Ohtomi Apparatus uses a rectangular array of LEDs that is reciprocated at high speeds through the display volume. Figure 1-5 shows 3-D display images from the Kameyama-Ohtomi Apparatus. The first image is a wire frame display of a drinking mug, and the second image is a surface representation of the mug.

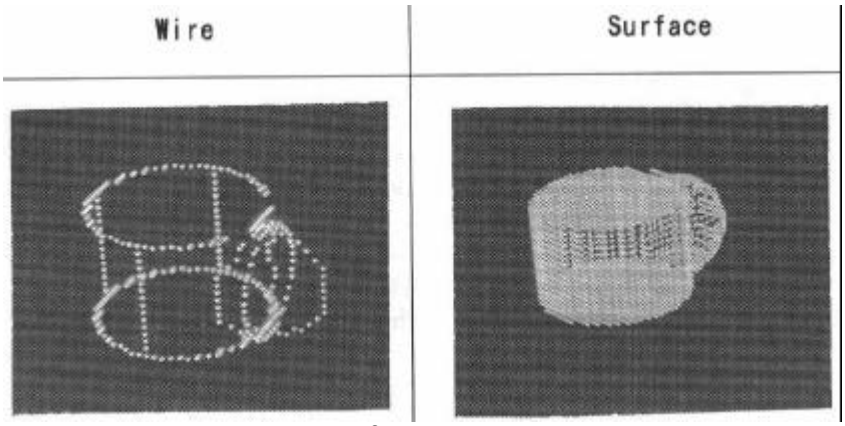


Figure 1-5. Display Images from the Kameyama-Ohtomi Apparatus

Three-Dimensional Display by Projecting Light

Several methods have been demonstrated that use some form of light projection for displaying objects in three dimensions. The mirror stereoscope was invented in 1833 by Sir Charles Wheatstone (Lip90). This viewing mechanism only allows the viewer's right eye to view the right perspective image and the left eye to view the left perspective image. This invention marked the discovery of binocular stereoscopic vision, which as stated earlier adds 3-D depth perception to the scene being viewed.

Binocular stereoscopic vision is also used in 3-D movie making. The two perspective images are projected onto the screen and then the viewers wear special polarizing glasses to separate the respective images for the left and right eyes.

In 1995, Ralf Jurczyk, a graduate student in Mechanical Engineering at the University of Florida also designed and built a device that displayed 3-D objects (Jur95). Jurczyk's device consisted of a screen that was mounted on a rotor. The screen would spin at high speed, while a computer and projection system projected an image onto the screen. An image of a three-dimensional object could be displayed in the spinning volume of the screen. A schematic diagram of Jurczyk's design is shown in Figure 1-6.

Probably the most commonly thought of method for true 3-D object display is 'holography.' Holography was predicted in 1947 by the British scientist, Dennis Gabor, but a real demonstration of the technique was not performed until after the laser was invented in the early 1960s (Fir72).

The creation of a hologram has traditionally involved a complex process involving laser light, optical equipment, and a holographic recording plate. Currently, research centers are working on computer-generated holograms (CGH). This method for 3-D display is not yet practical because of the amount of information that the CGH must

contain. A typical 300 mm by 300 mm holographic plate, for example, can contain over 10 terabytes of information.

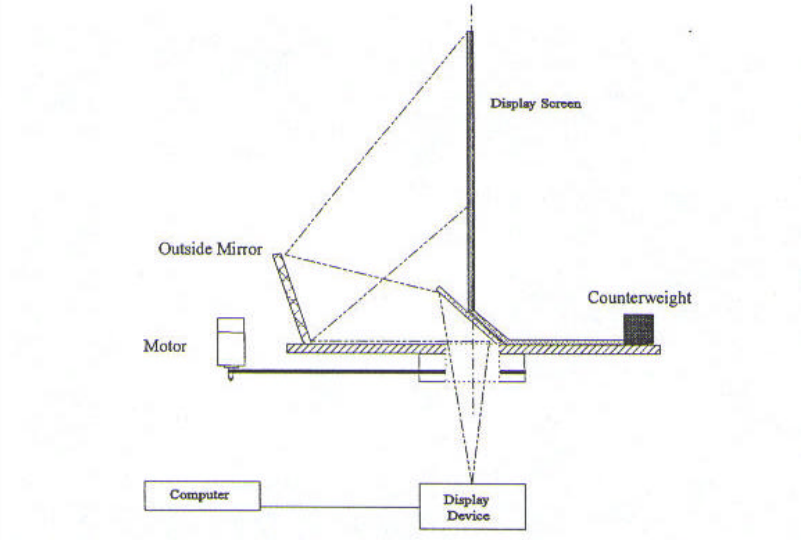


Figure 1-6. Schematic diagram of Jurczyk's design

CHAPTER 2
MECHANISM DESIGN

Introduction

The original objective that was decided upon during the brainstorming phase was to develop a device that would have the following characteristics:

- Presents a three-dimensional representation of the object being displayed;
- Does not require the user to wear any special glasses or head mounted tracking system;
- Allows multiple users to simultaneously view the same object. Such a device will greatly aid an operator (or team of operators) in tele-operation and tele-supervision operations.

In addition to the above characteristics, it was decided that the device should actually create physical three-dimensional surfaces. This requirement excluded any implementation that would involve a projection of light to somehow create the illusion of a 3-D object.

Some final requirements were to design a device that would not use any consumables and would have the ability to quickly reconfigure itself in order to display another object. These requirements would set the device apart from current rapid prototyping technologies which use a type of consumable material to construct the 3-D object. Also with rapid prototyping, the time to construct the object can be several hours.

The approach that was finally selected was inspired by the apparatus shown in Figure 2-1.

Figure 2-1. Axially parallel grid of steel pins

This grid has approximately 2000 axially parallel steel pins. By placing an object into the grid of steel pins (a hand for example), a projection of the 3-D object is formed by the heads of the pins on the other side.

The intent of this research was to design a mechanism that could control the individual pin displacements for this type of apparatus. A 3-D image on a computer screen could be sent to this apparatus for display. In addition to being able to display 3-D objects, the device was to have the ability to capture 3-D data from existing objects for digitization into a computer.

Design Description

Mechanism for Three-Dimensional Display

The first engineering challenge that presented itself during the initial brainstorming on how to control a grid of axially parallel steel pins was the proximity of neighboring pins. The original benchmark during the design phase was the grid apparatus shown in Figure 2-1 and this apparatus has the steel pins packed tightly at

approximately two millimeters between centers. Therefore, the challenge became how to design a mechanism that would control the pin offsets into this very limited space.

The solution decided upon was to use additional vertical space and spread the 2-millimeter center-to-center proximity to something more manageable by using push-pull cables. Push-pull cables are simply steel core cables surrounded by plastic sleeves. They can be used to transmit forces bi-directionally over large distances or around obstacles. Connecting these types of cables at one end to the grid of tightly packed steel pins and then flaring them out at the other end allows more design space for a pin offset control mechanism.

The description of the pin offset control mechanism design is more easily understood by first viewing the diagram shown in Figure 2-2. This diagram only shows three cables. Control of an apparatus like the one shown in Figure 2-1 would require nearly 2000 cables because there are that many pins that the cables would connect to at the top.

The sleeves of the push-pull cables are fitted into an upper plate and only the steel cores protrude straight down from the moving upper plate. Because there is friction between the core of the cables and the sleeve, if the upper plate translates down, the core of the cable will also translate down. During this motion of the upper plate moving down, if the end of the core hits a surface such as the upper surface of the part called a swivel, this will cause the core of the cable to move up with respect to its sleeve. When this happens, the steel pin that is connected to the other end of the core will translate up at the same but opposite speed as the upper plate.

Notice that the second solenoid in the diagram appears to be energized which has pulled the second swivel to the open position, while the first and the third solenoids are not energized. Therefore, the pin that is controlled by the second cable will stop moving

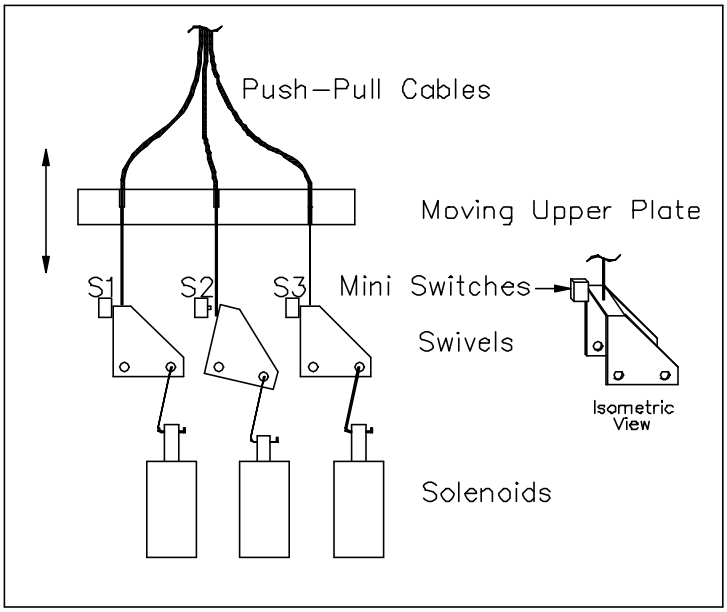


Figure 2-2. Pin offset control mechanism

while the first and third pins will still be moving up. As an example, when the upper plate moves down another ¼ of an inch, the first solenoid will get energized; when the upper plate moves down another ½ of an inch the third solenoid will get energized. The result from this sequence of actions would have the center pin in the lowest position, the first pin ¼ of an inch higher and the third pin ¼ of an inch higher than the first.

With this type of setup, the position of every steel pin can be controlled. The job of the controlling computer during a display sequence is to move the upper plate and to fire the individual solenoids at the proper time during the motion.

Mechanism for Three-Dimensional Object Digitization

This very same mechanism can be used to take three-dimensional data measurements from an existing object. Suppose that all of the solenoids are energized, putting all of the swivels into the open position. All of the miniature switches (labeled S1, S2 and S3 in Figure 2-2) can be read by the controlling computer and will now read as 'open.' Now, a 3-D object is free to be pressed into the grid of axially parallel steel pins attached to the other end of the cables because the top surfaces of the swivels are currently not creating an obstruction to the cable cores.

After the pins have been offset, the solenoids can be de-energized. Because the solenoids are spring-loaded to the closed position, the edge of the top surface of the swivels will now be resting against the cable core on any of the pins that have been translated.

Now the upper plate is controlled to start translating upward which also starts all of the core cables to move upward. When an end of a core cable finally passes the top edge of the respective swivel, the swivel finally will fully close on the miniature switch. The controlling computer will register that the switch has closed and by looking at the distance that the upper plate has traveled it can determine what the original displacement was for that pin. All of the pin offsets can be determined in this manner.

The Eight-Pin Prototype

An 8-pin prototype of the 3-D display mechanism design was developed to demonstrate the design concept. A system block diagram of the prototype is shown in Figure 2-3 and a picture of the prototype mechanism is shown in Figure 2-4. A Motorola MC68HC11 microprocessor was used to control the mechanism. The embedded control

software is discussed in Chapter 4. A graphical interface that runs in a Java applet on a PC and communicates using serial communications to the MC68HC11 microcontroller is discussed in Chapter 5.

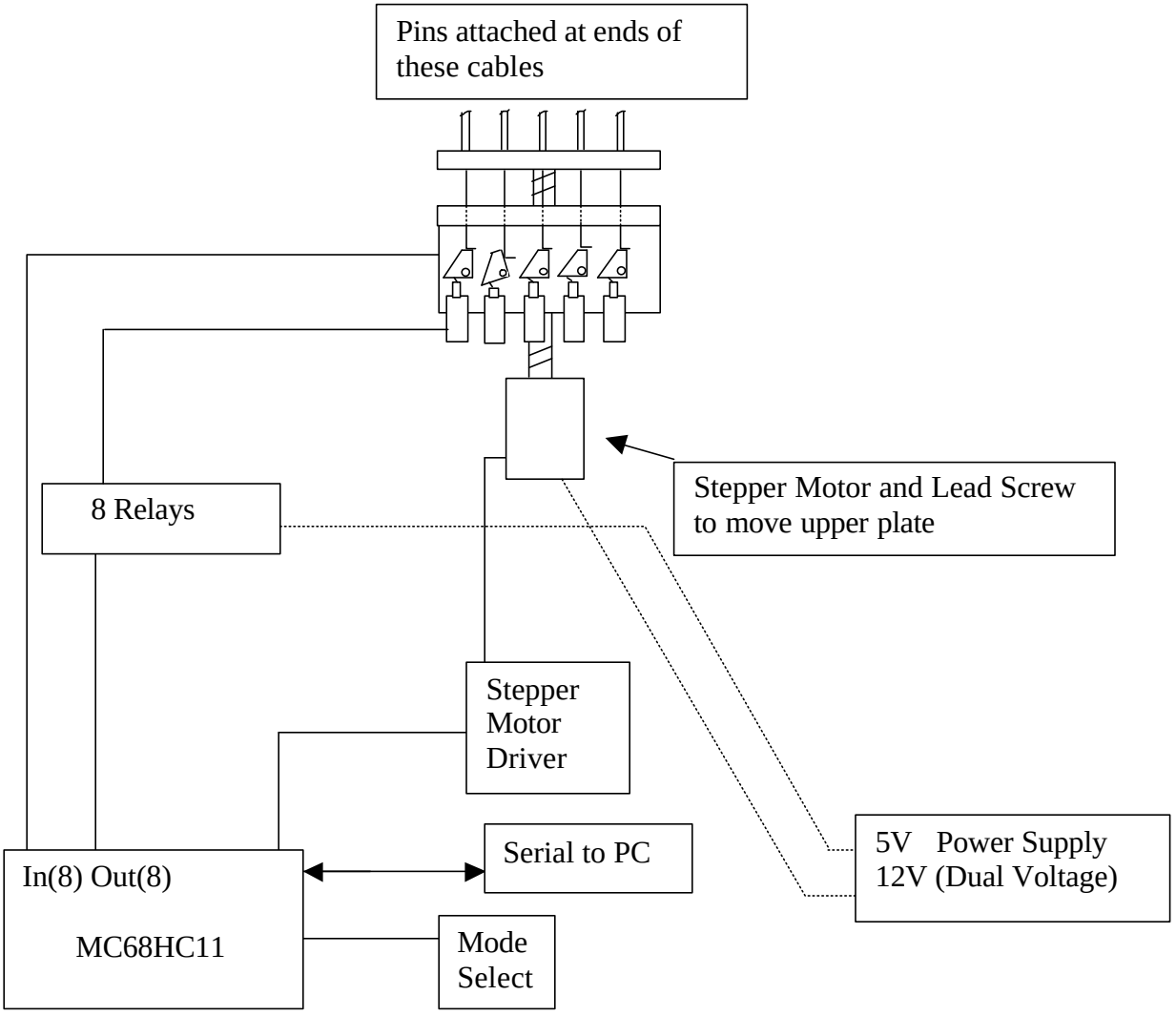


Figure 2-3. System block diagram

A stepper motor and lead screw were used to move the upper plate. Eight digital output signals were connected to relays to control the solenoid actuators. A mode select switch is used to toggle between 3-D object display mode and 3-D object digitize mode. Eight

miniature switches that sense the position of the swivel parts were connected to eight digital input ports.

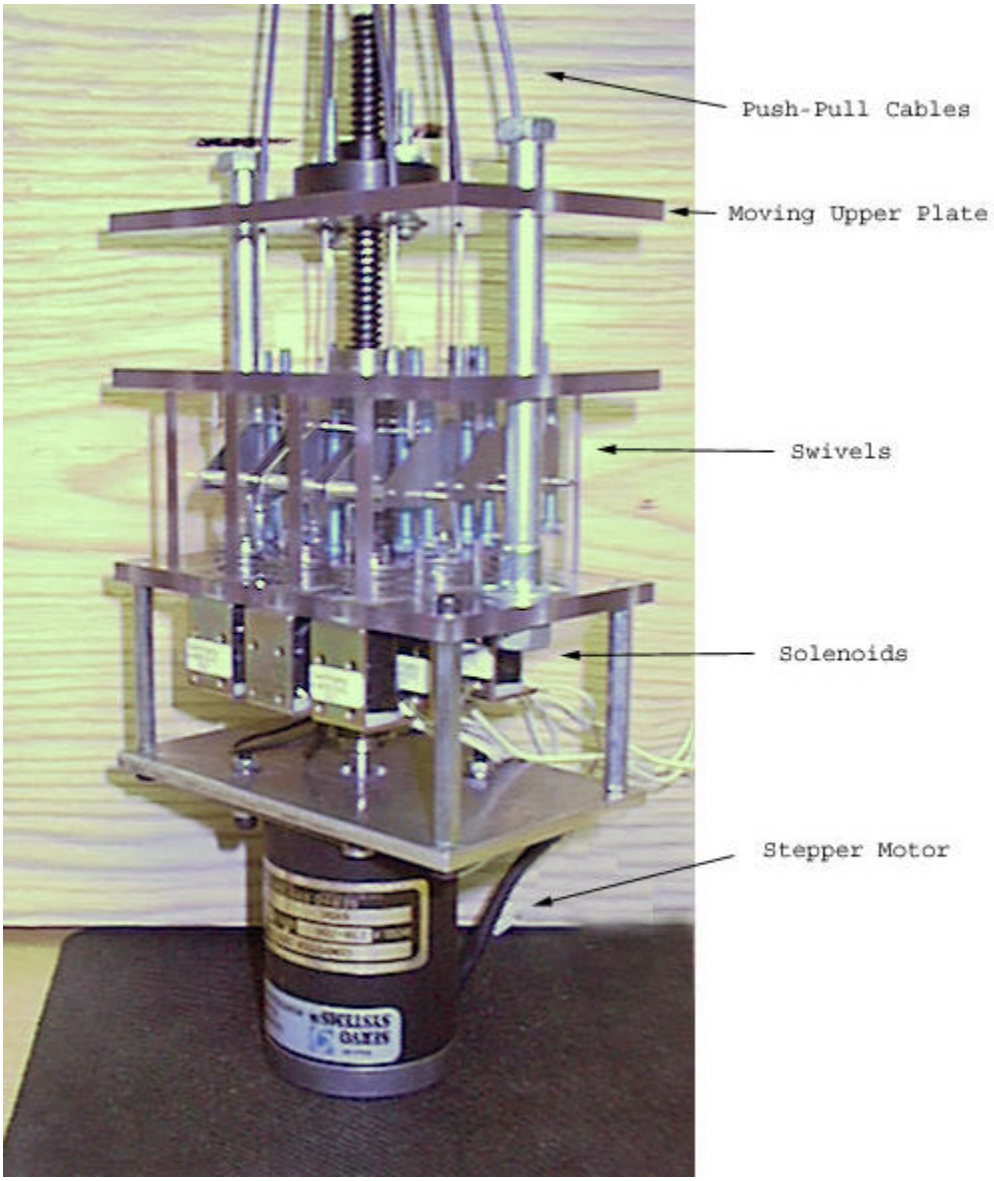


Figure 2-4. Eight-pin prototype mechanism

CHAPTER 3
RASTER SCAN

One of the difficulties inherent in the design for this type of three-dimensional display device is the overall quantity of actuators that need to be controlled individually. There were only 8 actuators for the first prototype that was developed. There were plenty of digital output pins on the controlling computer to handle this small quantity of actuators. When the mechanism requires control of thousands of actuators, however, a method commonly referred to in the design of digital systems as 'raster scanning' will be needed to significantly reduce the number of digital pins needed. It turns out that the number of digital pins needed is:

$$P = 2\sqrt{A} \qquad (3.1)$$

where P is the number of pins and A is the number of actuators. One thousand actuators could be controlled with just over 60 digital output pins, for example.

As an example, consider 16 actuators that will be controlled by 8 digital signals. The actuators need to be wired together into a grid pattern as shown in Figure 3-1. All of the positive leads are connected horizontally in rows and all of the negative leads are connected vertically in columns. At one end of each of the rows there is a P-channel MOSFET (metal oxide semiconductor field effect transistor) and at one end of each of the columns there is a N-channel MOSFET. In this configuration, with the controlling computer's digital ports connected to the gate of the MOSFET's, they can be used as fast high-current switches.

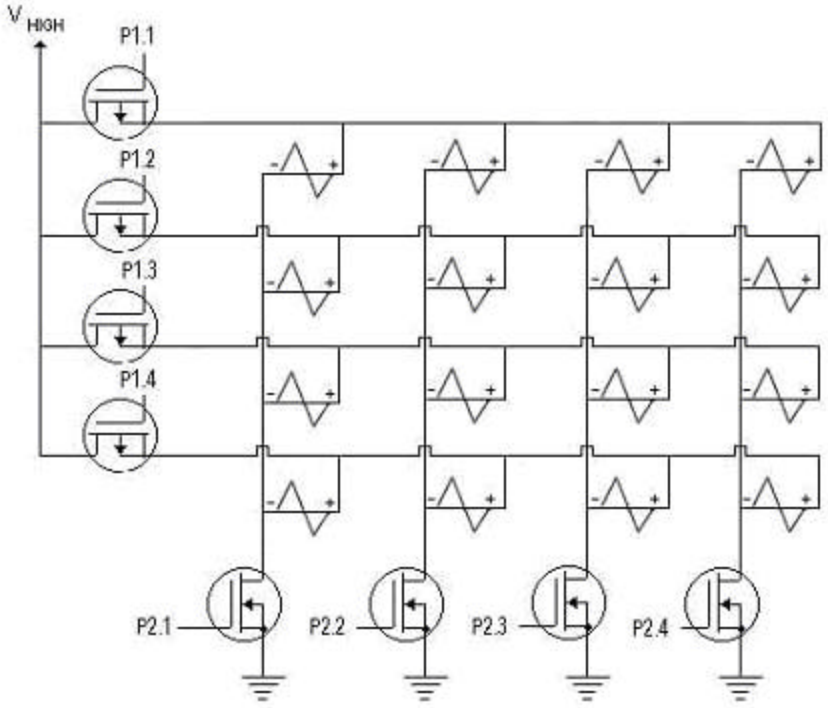


Figure 3-1. Four-by-four grid of shape memory alloy actuators and control circuitry

When a digital-1 is written to a port that is connected to the gate of an N-channel MOSFET, the 'switch' is on (digital-0 is off). Conversely, when a digital-0 is written to the gate of a P-channel MOSFET, the 'switch' is on (digital-1 is off).

Now it is only a matter of timing for the controlling computer to turn on any pattern of actuators in the grid. For the first time segment, the controlling computer handles the column-1 actuators, and for the second time segment, the controlling computer handles the column-2 actuators, and so on.

As an example pattern, consider turning on, for a few seconds, only the actuators that lie on the diagonal of the square grid (from the upper left corner to the lower right corner). During the first time segment a 0111 is written to the 4 bits labeled P1.1-P1.4 (Output Port 1). This effectively only connects the first column to ground. Also, a 1000 is written to the 4 bits labeled P2.1-P2.4 (Output Port 2). In this case, only the upper left

corner actuator sees a voltage. The pattern is continued using the procedure outlined in Table 3-1.

Table 3-1. Procedure for turning on the actuators that lie on the diagonal

Time Segment	Port 2	Port 1
1	1000	0111
2	0100	1011
3	0010	1101
4	0001	1110

Suppose we want to energize this pattern of actuators for 2 seconds. With a time segment of 10 milliseconds, the controlling computer would need to continue looping through these four steps 50 times. During the two seconds, each actuator sees 50 pulses, one 10 millisecond pulse every 40 milliseconds. That is, each actuator experiences a 25% duty cycle.

CHAPTER 4
EMBEDDED CONTOLLER SOFTWARE

A flow chart for the embedded controller software is shown in Figure 4-1. The program (see Appendix B) is written in assembly language for the MC68HC11 microcontroller.

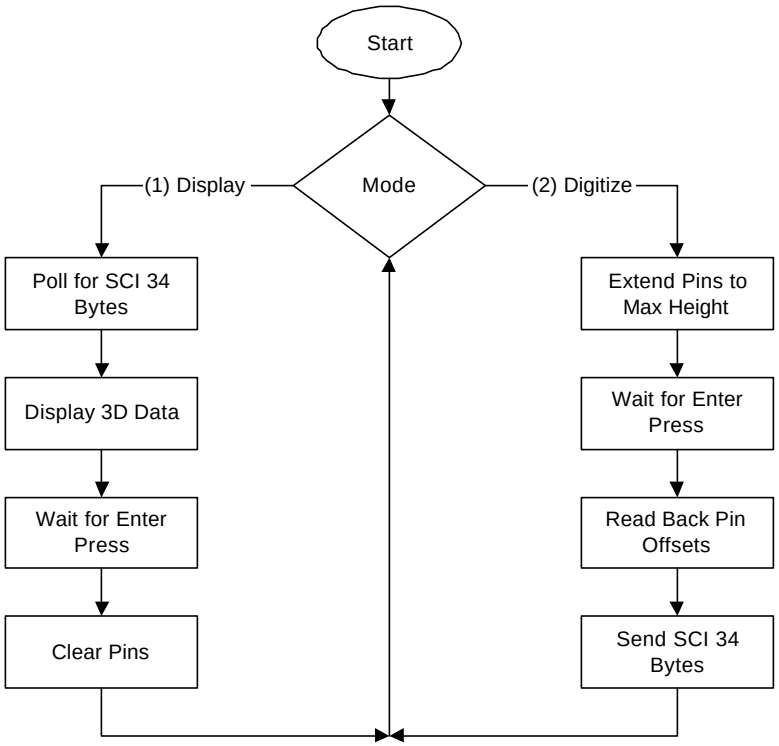


Figure 4-1. Flow chart for the embedded control program

When the program starts, the mode select switch is checked to determine whether to perform a 3-D object display or a 3-D object digitize. If 3-D object display is selected, the controller immediately will begin waiting to receive 34 bytes of serial data from the PC. The one inch of total pin travel was segmented into 34 discrete levels, so the 34

bytes represent the pattern of solenoids to energize at each level. The bytes are written to a table in the controller's RAM.

Next, the controller begins the display sequence. During the display sequence the upper plate travels 1 inch. This travel is divided into 34 separate stepper motor movements and at each interval a byte from the 34-byte table is sent to the port, which energizes the respective solenoids. The stepper motor controller box has two important inputs that need to be handled by the microprocessor in order to perform a stepper motor movement. One of them is labeled 'cw/ccw' and the other is labeled 'step in'. When the first input is given 5V and second is given a 20-50 kHz square wave input, the motor will move clockwise (cw) one count for every rising edge. Conversely, when the first input is given 0V and the second is given a 20-50 kHz square wave input the motor would move counter clockwise (ccw) one count for every rising edge. The controller uses an interrupt service routine to create the desired square wave for the display sequence.

Now the 3-D pin offsets have been displayed and the controller will wait for an 'enter' key press before it will clear the pins and loop back to the beginning. The clear sequence involves the same stepper motor movement only in the reverse sense.

If the mode select key was set for the 3-D digitize mode, first the controller will move the stepper motor 34 movements so that the upper plate will be in the full up position. Between the 33rd and 34th movement the controller will energize all solenoids. This will free the cable cores for movement.

Next the controller will wait for an 'enter' key press. During this wait, the pins will be offset by the user. After the enter key is pressed the controller will read back the pin offsets. This also involves a 34-segment stepper motor movement and at each

segment the miniature switches are checked for transitions. When transitions are detected, the respective segment index numbers are recorded for each of the pins. This information is stored in the 34-byte table.

Finally this new 34-byte table is sent serially to the PC and the program loops back to the beginning.

CHAPTER 5
THE GRAPHICAL USER INTERFACE

The schematic diagram in Figure 5-1 shows how the graphical user interface is part of the 3-D display system. The system consists of three components: the display device, a server computer, and the client applet. The design incorporates a client-server approach in order to allow users to communicate with the device over the Internet.

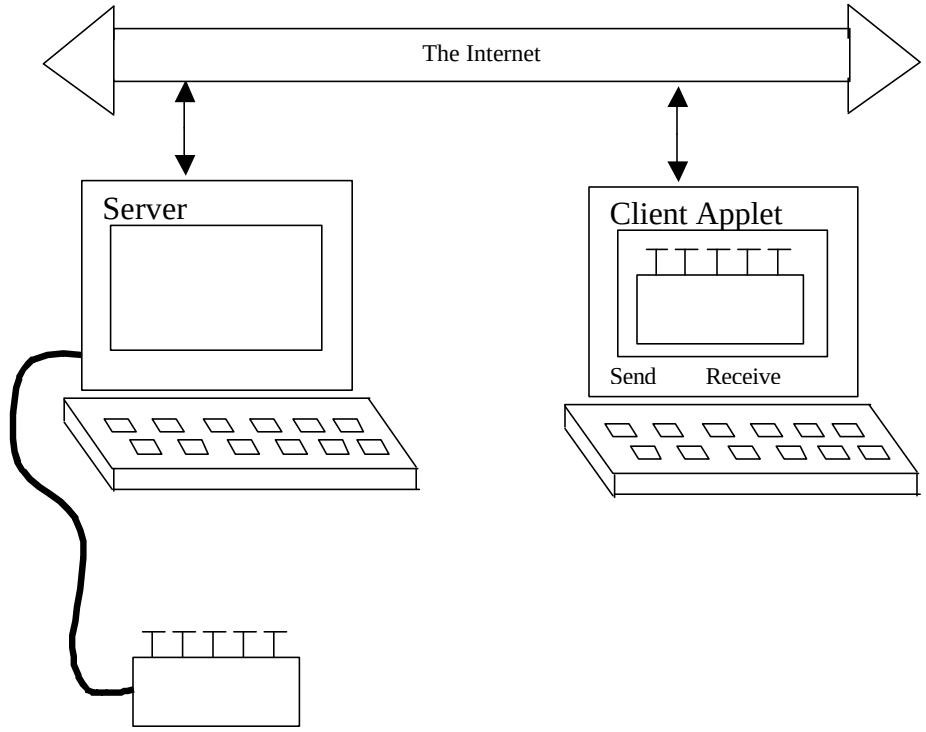


Figure 5-1. Graphical user interface as part of the 3-D display system

The display device is connected to the server computer via a serial port. The server computer is connected to the Internet and runs a server program at all times. The job of the server program is to wait for a client to connect to it, and when a client does establish a connection, the server program acts as in intermediary agent between the display device

and the client applet. The client applet consists of an interactive 3-D model of the display device developed in VRML and two Java buttons called send and receive. Figure 5-2 shows a picture of the VRML graphical model. Although there are only 8 movable pins, the model has an entire row of pins that mirror the position of the front movable pins in order to give a more 3-D representation. The applet is also programmed to connect with the server running on the server computer while it is being initialized. The type of connection used is socket based. As discussed in Chapter 4, there are two modes of operation (send and receive). During a send operation, the user running the client applet first will create a 3-D surface by clicking and dragging the individual pins in the VRML model and then the user presses the send button. The data is sent via the Internet to the server computer, and finally the data is sent serially to the display device by the server program.

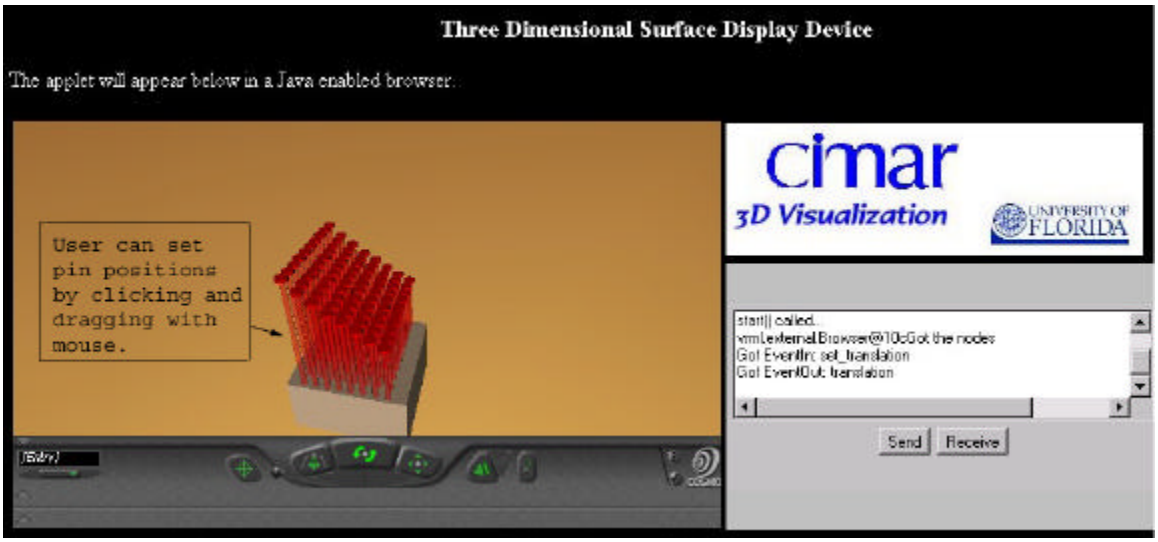


Figure 5-2. The VRML model

During a receive operation, the user at the server computer physically creates a 3-D surface using the 3-D display device and then presses the ‘enter’ button present on the

device. The data is read by the device, serially sent to the server computer, and sent via the Internet to the client applet by the server program. The client applet, on receiving the data, updates its 3-D model to match the surface created on the display device.

A listing of all of the software programs that were created is shown in Table 5-1. The dissertation by Jahoon Lee, a recent University of Florida doctoral graduate, entitled "Investigation of the Quality Indices of In-Parallel Platform Manipulators and Development of Web Based Analysis Tool", has a complete explanation about the use of VRML in Java Applets (Lee00). The client computer must have a Cosmo Player Plug-in 2.1.1 and a Java 1.2 enabled Internet browser.

The applet sends and receives data with the server using what is called a socket connection. This means that the Applet program knows the server's Internet Protocol (IP) address, which it uses to locate and make the connection to the server.

Table 5-1. List of software programs for the graphical user interface

Program File Name:	Description:	Page:
pins.pinsApplet.html	HTML file for the Java Applet	(B)50
PinsApplet.java	Java Applet that excepts the VRML file	(B)50
PinServer.java (main) PinServerFrame.java PinData.java SerialRead.java SerialWrite.java cimarImg.jpg	Java Applications to make PC a Server	(B)57 (B)59 (B)60 (B)62 (B)65 -
pin.wrl pinColumn.wrl pinMatrix.wrl	VRML files for the 3-D graphics	(B)66 (B)67 (B)68

Most browsers do not allow applets to perform network I/O without obtaining some form of authentication. Authentication can be made with the use of the Java *javakey* tool to create a signed applet. Since the 3-D display system is essentially a prototype model, a simpler approach was used to overcome this security issue. The browser

settings were altered to accept unsigned applets and the operating system was informed of the applet as being a safe applet with the help of the Java *policytool* tool. More information on the Java tools mentioned above can be obtained from Sun Microsystems documentation on Java tools (www.java.sun.com).

CHAPTER 6
SYSTEM TESTING

Figures 6-1, 6-2 and 6-3 are images of some sample shapes that were created using the Display Mode. The shapes were created using the graphical user interface and then sent to the display device. The image on the left was created on the computer screen and the image on the right is an actual picture of what the device created.

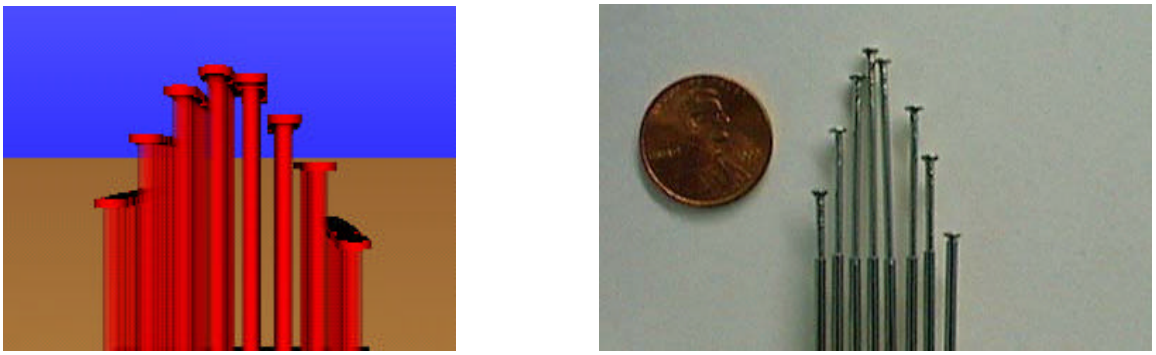


Figure 6-1. An inverted parabola shape (graphics to actual)

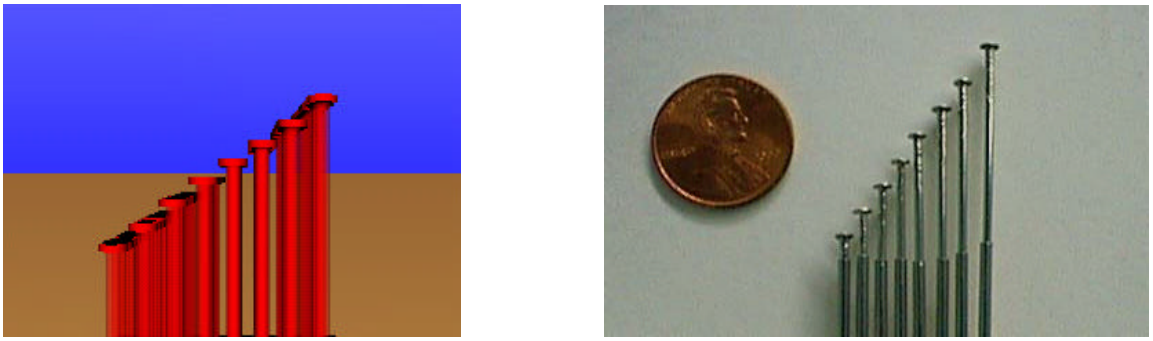


Figure 6-2. A linear shape (graphics to actual)

Figure 6-4 shows how the Digitize Mode is used. 3-D data from an existing object is read by the device, and then the data is sent to update the graphical model.

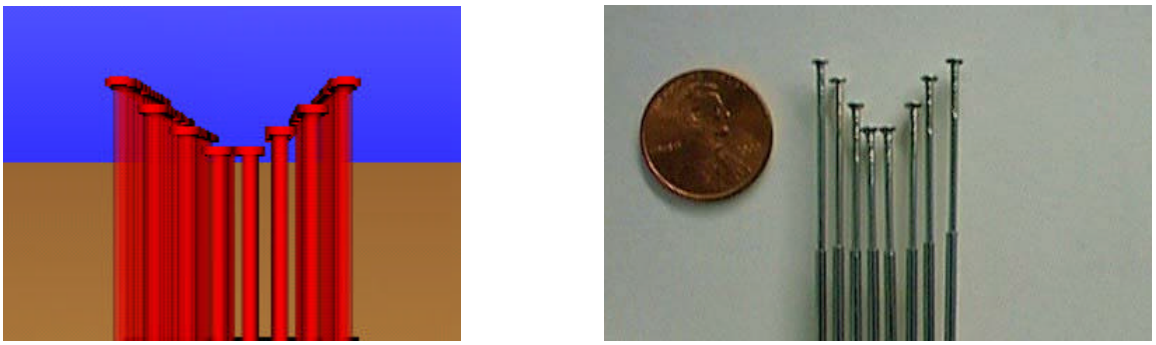


Figure 6-3. A linear V-shape (graphics to actual)

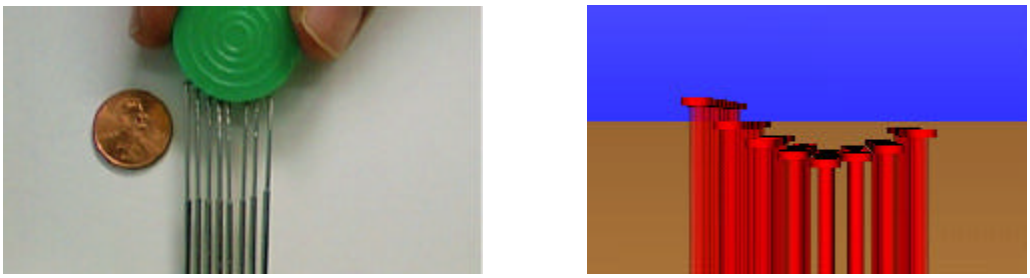


Figure 6-4. An object being digitized by the device (actual to graphics)

The device is designed to have a maximum pin travel of one inch. The computer algorithm cuts the one-inch travel into 34 discrete increments. This means that theoretically the pin offsets are controllable to about 0.03-inch (0.75 millimeters) increments. Also, there will be some accumulation of position error around these discrete increments. Table 6-1 shows the results of a test for position accuracy for each of the eight pins. Each pin was instructed to move to the middle offset position, and then a measurement of the actual offset was taken using digital calipers. The table shows the mean and standard deviation of the measured data from 6 tests of each pin.

Table 6-1. Results of pin position accuracy test

Pin #:	1	2	3	4	5	6	7	8
Mean (mm)	15.14	14.59	15.05	14.92	14.87	15.24	13.60	13.99
Standard Dev. (mm)	0.248	0.212	0.281	0.280	0.230	0.184	0.307	0.204

There are several components within the display device, which can contribute to this measured error. One known error is due to backlash in the push-pull cables. There is by design a small amount of clearance between the outside diameter of the core cable and the inside diameter of the plastic sleeve. This clearance causes an error in position from one end of the cable to the other. A second main contributor to error is due to the upper moving plate, which carries all of the push-pull cable sleeves. During the motion of this plate, the parallel relationship between it and the lower plate is not perfectly maintained.

CHAPTER 7
CONCLUSION AND FUTURE WORK

This thesis presents the design of an electro-mechanical device for displaying three-dimensional shapes and for digitally recording 3-D data from existing objects. The method for 3-D display is unique in that an actual surface is created which multiple people can view from multiple angles.

The development of an 8-pin prototype complete with graphical user interface is also included. The prototype demonstrates that this mechanism design is feasible. Some of the lessons that were learned during the prototype development and future work that would make the next generation for this type of a device very practical are:

- Obtain a new, low power, small actuator.
- Improve the pin offset sensor.
- Improve the algorithm that converts 3-D solid data into pin offset data.
- Add an image projection system.

The prototype mechanism uses eight 4.5-ohm open-frame solenoids for actuation. These actuators would not be practical in a design requiring hundreds of actuators because of their size and power consumption. Two possibilities for actuators are shape memory alloy (SMA) wire actuators and miniature rotational solenoids. In Appendix A there is a section on a SMA wire actuator design that was tested. The main benefit in using an SMA wire actuator is the small design space that is required. The best actuator in terms of power consumption is a miniature rotational solenoid. The prototype solenoid

requires more than 5 Watts of power, the SMA actuator requires 1.2 Watts, and the rotational solenoid pictured in Figure 7-1 requires only 0.3 Watts.

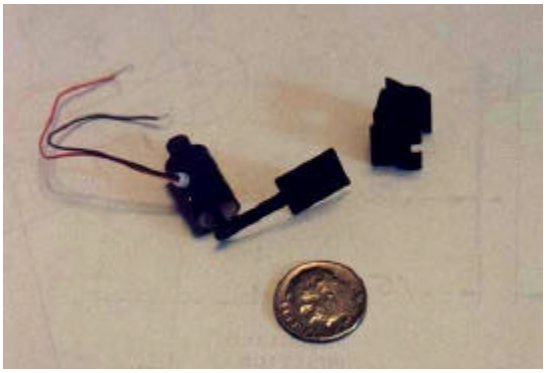
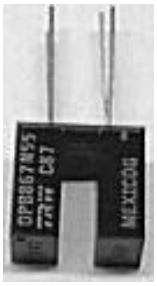


Figure 7-1. Miniature rotational solenoid

Manufacturer	Daco Instrument Co.
Nominal Voltage	4 V DC
Coil Resistance	24 Ohms
Part Number	SK-960

The design of the sensors labeled S1, S2 and S3 in Figure 2.2 for the prototype was simply a metal strip that hit two contacts to close a circuit. During testing, the sensors did not always close properly and this led to unreliable information. A better design would incorporate an optical sensor similar to the one shown in Figure 7-2.



Distributor	Newark Electronics
Supply Voltage	4.5-16 V DC
Rated Current	10 mAmps
Stock Number	52F6152

Figure 7-2. Optical sensor

One possible application for this type of a 3-D display device is as a visual aid during computer aided design (CAD). During the design of a 3-D solid part, the CAD user would be able to use the device to provide a true physical representation of the design. To make this happen, a sophisticated computer algorithm would need to be

developed that would convert the 3-D solid data into pin-offset information that could be used by the display device.

Finally, adding a projected image would make this method for 3-D object display much more realistic. If the heads of all of the steel pins were given a white finish, a texture image could be projected from above that would appear on all of the pins. A skin texture could be projected when a hand surface is being displayed by the pins, for example. Although the display device would only have 10 pins per inch, a 300 pixels per inch image could be projected from above and provide a much more realistic display.

APPENDIX A
SHAPE MEMORY ALLOY ACTUATOR DESIGN AND TEST APPARATUS

As mentioned in the conclusion section, one possible actuator as an alternative to a solenoid is a shape memory alloy (SMA) wire actuator. A test apparatus was built in order to test both SMA actuators and the raster scan technique that was discussed in Chapter 3.

Shape Memory Alloy Actuator Design

The book titled "Actuator Design Using Shape Memory Alloys" by Tom Waram (War93) has a detailed explanation of wire actuator design using SMA actuators. The following discussion uses this method to design an actuator that could be used for the 3D display device.

To actuate the swivel parts in the 3D display device prototype requires a pulling force of 200 grams (using a 25% factor of safety, 250grams) and a stroke of 4mm. The suggested wire actuator diameter can be found from equation A.1:

$$D = \sqrt{\frac{4F}{ps}} \qquad \text{(A.1)}$$

where D is the wire diameter, F is the load, and σ is the maximum stress. Using a maximum design stress of 140 MPa to ensure good cycle life calls for a 150 micron diameter wire actuator.

The wire actuator length L, will be calculated assuming a low temperature strain ε_l of 55% or 0.55. The high temperature strain calculated from equation A.2 is 0.002.

$$e = \frac{s}{E} \qquad \text{(A.2)}$$

where E is the Young's Modulas at high temperature.

The wire actuator length is then determined from equation A.3:

$$L = \text{stroke}/(\varepsilon_l - \varepsilon) \qquad \text{(A.3)}$$

This calls for a wire actuator length of 80mm.

Test Apparatus Design

The test apparatus shown in Figure A-1 was built in order to test both the SMA wire actuators and the raster scanning technique that was discussed in Chapter 3. The actuators are approximately 80mm in length. Only six digital pins are needed in order to control the entire 3x3 grid of actuators. A simple program (listed at the end of this section) was written in order to sequentially energize the various actuators. For the first time segment, two actuators are energized, and then a second set and so on until all of the actuators in the 3x3 grid have been energized.

Conclusions

The raster scanning technique works great although it will add a significant amount of additional programming. The SMA actuators are nice in the sense that they use such a small amount of space. One drawback is the fact that they need to be reset. After the power is removed from the actuators they cool down but it then takes additional

force in order to return them to the original state. Another problem is the responsiveness. Unlike the solenoids that fire relatively quickly, the SMA actuators take about a second to contract.

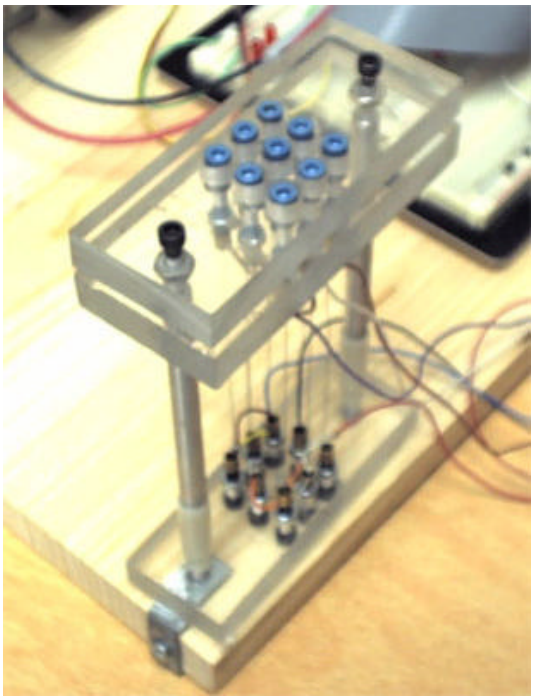


Figure A-1. Shape memory alloy test apparatus

Embedded Controller Code for the Test Apparatus Program

```
*** Jeremy Mayer
*** memory shape alloy rater scan program
*** 3/6/01
*******************************************************
*  Connections: 3 I/O ports from PortB bit 1 3 and 5
*               3 I/O ports from PortD bit 3 and 5
*******************************************************

** Equates:

BAUD   EQU $2B  ; Baud-rate control register to set the baud rate
SCCR1  EQU $2C  ; Serial Communication Control Register-1
SCCR2  EQU $2D  ; Serial Communication Control Register-2
SCSR   EQU $2E  ; Serial Communication Status Register
SCDR   EQU $2F  ; Serial Communication Data Register
EOS    EQU $04    ; User-defined End Of String (EOS) character
CR     EQU $0D    ; Carriage Return Character
LF     EQU $0A    ; Line Feed Character
```

```
ESC    EQU $1B

BASE   EQU $1000
TIC1   EQU $10
TIC2   EQU $12
TIC3   EQU $14
TI4/O5 EQU $1E
TOC2   EQU $18
TCTL1  EQU $20
TCTL2  EQU $21
TMSK1  EQU $22
TFLG1  EQU $23
TFLG2  EQU $25
TMSK2  EQU $24
TCNT   EQU $0E
OPTION EQU $39
ADCTL  EQU $30
ADR1   EQU $31
ADR2   EQU $32
ADR3   EQU $33
ADR4   EQU $34
PACTL  EQU $26

* masks
BIT7   EQU %10000000
BIT6   EQU %01000000
BIT5   EQU %00100000
BIT4   EQU %00010000
BIT10  EQU %00000011
BIT2   EQU %00000100
BIT1   EQU %00000010
BIT0   EQU %00000001
INV6   EQU %10111111
INV2   EQU %11111011

********** Port Equates:
PORTB  EQU $1004                ;use PortB-1357 pins 41 39 37 35
PORTD  EQU $1008                ;use PortD-3&5 pin 23&25
DDRD   EQU $09

INIT   EQU $FFA9
B_OUT  EQU $FFC1    ;location of JMP to .OUT2BS
B_O    EQU $FFB8    ;location of JMP to .OUTA

PE1    EQU %00000001;Value for ADCTL

       ORG $00EB
       JMP RTI_ISR

*** Main:

       ORG $00f0
       JMP MAIN

PROMPT FCB CR, LF
       FCC @Memory Shape Alloy Raster Scan Program Started@
       FCB CR, LF
       FCB EOS

TABLE  FCB $52
       FCB $E0
       FCB $E0
       FCB $70
```

```
       FCB $D0
       FCB $C2
       FCB $58
       FCB $D0
       FCB $C8
       FCB $D0
       FCB $B2
       FCB $B2
       FCB $00
       fcb $00
       fcb $00

CTR    RMB 1
CTR1   RMB 1
CTR2   RMB 1

ADDRS  RMB 1
ADDRS2 RMB 1

MAIN   LDS #$0041
*      BCLR TMSK2,X BIT10  ;t-cnt overflow FOR 32.768 MSEC


       JSR InitSCI
START  LDX #PROMPT
       JSR OutStr

       LDAA #0
       STAA CTR
       STAA CTR1
       STAA CTR2

       LDX #TABLE
       STX ADDRS

       LDX #BASE

       ldaa #$ff
       staa DDRD,X

       LDAA #%00000011
       STAA PACTL,X

       LDAA #BIT6
       STAA TMSK2,X

       CLI
       WAI

HERE   BRA HERE

*************************************************************
* ISR to raster scan proper patterns
*************************************************************
RTI_ISR      LDX #BASE
       LDAA #BIT6
       STAA TFLG2,X

       LDAA CTR            ;CTR for 3 seconds until next column
       CMPA #$5E
       BEQ SWICH
       BRA CONT
SWICH  LDAA #$00
```

```
        STAA CTR
        INC ADDRS2
        INC ADDRS2
        INC ADDRS2

        INC CTR2            ;CTR2 for 4 contexts
        LDAA CTR2
        CMPA #4
        BEQ QUIT

CONT    LDX ADDRS

        LDAA CTR1           ;CTR1 for 3 rows, will be 0, then 1, then 2
        CMPA #0
        BEQ GOTIT
        CMPA #1
        BNE GO1
        INX                 ;CTR1 is 1
        BRA GOTIT

GO1     INX                 ;when it gets here CTR1 must be 2
        INX
        LDAA #$FF
        STAA CTR1

GOTIT   LDAA 0,X
        STAA PORTB
        LSRA
        STAA PORTD

        BRA OVER
QUIT    LDAA #$80
        staa PORTB
        ldaa #$ff
        staa PORTD
        SWI

OVER    INC CTR
        INC CTR1

        RTI
```

- END OF PROGRAM

APPENDIX B
COMPUTER CODE

The following embedded assembly code was written for the Motorolla MC68HC11 Microcontroller. It controls the 3-D Display Device.

```
*********************************************************************
* Embedded Controller Software for 3D Display Device:
*********************************************************************
*
*Program has one path for Display-Mode and another for Digitize-Mode
*
* Solenoids controlled from Expansion Port OUTP2 at $4000
* Swivel Sensors read from Expansion Port at $2000
*
* Mode Switch - PortA bit 0
* Waiting LED - PortA bit 5
* Enter Key   - PortA bit 1
* Stepper Motor–PortA bit 4
*
*********************************************************************
** SYMBOL DEFINITIONS
************************************************
* 68HCII REGISTERS
************************************************
STACK  EQU    $8FFF
BASE   EQU    $1000  ;BEGINNING OF REGISTERS
PORTA  EQU    $00
TOC2   EQU    $18    ;OUTPUT COMPARE 2 REGISTER
TCTL1  EQU    $20    ;TIMER CONTROL REGISTER
TMSK1  EQU    $22    ;TIMER MASK1 REGISTER
TFLG1  EQU    $23    ;TIMER FLAG1 REGISTER
TMSK2  EQU    $24    ;TIMER MASK2 REGISTER
SCSR   EQU    $2E
SCDR   EQU    $2F
BAUD   EQU    $2B
SCCR1  EQU    $2C
SCCR2  EQU    $2D
EOS    EQU    $04    ; User-defined End Of String (EOS) character
OUTP2         EQU $4000
* MASKS
BIT7   EQU    %10000000
BIT6   EQU    %01000000
BIT10  EQU    %00000011
BIT1   EQU    %00000010
BIT0   EQU    %00000001
INV6   EQU    %10111111

************************************************
** DATA SECTION
************************************************
       ORG    $00DC
```

```
       JMP    OC2_ISR      ;OC2 INTERRUPT VECTOR (IN BUFFALO)

       ORG    $20
PWMLO  FDB    $300   ;LOW OUTPUT TIME
PWMHI  FDB    $300   ;HIGH OUTPUT TIME
CTR    RMB    2
CTR2   RMB    2
STOP   RMB    1
CTR3   FCB    $00
CTR4   RMB    1
BITCOM FCB    $00
LASTBYT       RMB    1
TEMP   RMB    1

       ORG    $8000  ;Data Blocks of 34 solenoid patterns

TABLE  RMB    34
       FCB    EOS
       FCB    EOS

************************************************
** MAIN PROGRAM
************************************************
       ORG    $9000

*-----------------------------------------------
* INITIALIZATION
*-----------------------------------------------
* INITIALIZE STACK & Registers
       LDS    #STACK
       LDX    #BASE

* Set the Pre-Scaler Frequency for TCNT (during first 64 cycles)
       BCLR   TMSK2,X BIT10 ;PR1:PRO=00 FOR 32.768 msec (default)

       LDAA #$FF           ;close all solenoids
       STAA OUTP2

       JSR InitSCI

* INITIALIZE OUTPUT COMPARE OC2
* (Bit7:6=OM2:OL2  Hi=11, Lo=10, Toggle=01, Disconnect=00)
       BSET   TCTL1,X BIT7  ;OM2:OL2=10
       BCLR   TCTL1,X BIT6  ;  FOR SET TO LOW

begin  ldaa #$00
       staa CTR3
       staa BITCOM
       staa LASTBYT

       JSR MODE            ;puts mode into accum. A
       ANDA #%00000001
       BEQ digit
       bra display
digit  jmp digitiz

display
************1. loop till recieve 34 bytes
************2. pins 4 display algorithm
************3. wait
************4. clear
       ldab #$00
       ldy #TABLE
```

```
byteme
       jsr InChar
       staa 0,Y
       incb
       cmpb #34            ;get the 34 bytes into table
       beq GOTIT
       iny
       bra byteme

GOTIT
       LDX #BASE
       LDAA PORTA,X        ; set plate to go down
       ANDA #%11101111
       STAA PORTA,X

*-------------------------------------------------
*2. Step through 34 increments
*-------------------------------------------------

       ldx #$8000

       LDAA #34
Pats
       LDAB 0,X                  ;loop for the 34 increments
       stab OUTP2
       JSR ADV
       INX
       DECA
       BNE Pats

*********** stepper motor now at down position

       LDAA #$FF
       STAA OUTP2

       JSR WAIT

       JSR CLEAR

       jmp begin


digitiz
************1. extend pins                    ;put digitize3D code here
************2. wait
************3. record
************4. send 34 bytes

       ldaa #$00           ;initialize data manipulation variables
       staa CTR3
       staa BITCOM
       staa LASTBYT

       LDX #BASE

       LDAA PORTA,X        ; set plate to go down
       ANDA #%11101111
       STAA PORTA,X

*-------------------------------------------------
* Get all Pins in upmost postion and open all solenoids
*-------------------------------------------------
```

```
HERE
       LDAA #0
getup
       LDAB 0,X                  ;move plate up to 32nd increment
       JSR ADV
       INCA
       CMPA #31
       BNE getup

       LDAB #00            ;index 32 (Open all Solenoids)
       stab OUTP2

       JSR ADV
       JSR ADV             ;34th ADV

*********** stepper motor now at up position

       LDAA #$FF           ;close all Solenoids
       STAA OUTP2

       JSR WAIT

       JSR RECORD

       JSR MANIP

       LDX #$8000          ;send out 34 bytes on Serial Port
       JSR OutStr

       jmp begin

************************************************
** SUBROUTINE MANIP - data manipulation
** convert the recorded bytes to a form acceptable for display
************************************************

MANIP  ldx #$8000

B0     ldaa 0,x
       staa TEMP

       ldab BITCOM
       andb #%00000001                 ;? is BITCOM bit0 == 0?
       bne SETNX0
       bra G0

SETNX0 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%00000001
       staa 0,x
       bra B1

G0
       ldaa LASTBYT
       anda #%00000001                 ;? is LASTBYT bit0 == 1? brnch if 0
       beq SETNX0

       ldaa 0,x
       anda #%00000001                 ;? is bit0 == 0? brnch if 1
       bne SETNX0

*      ;this is a valid solenoid throw spot
*      ;bit0 is already clear
```

```
       ldab BITCOM
       orb #%00000001
       stab BITCOM               ;set BITCOM bit0

*****************************************************************

B1
       ldab BITCOM
       andb #%00000010                  ;? is BITCOM bit1 == 0?
       bne SETNX1
       bra G1

SETNX1 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%00000010
       staa 0,x
       bra B2

G1
       ldaa LASTBYT
       anda #%00000010                  ;? is LASTBYT bit1 == 1?
       beq SETNX1

       ldaa 0,x
       anda #%00000010                  ;? is bit1 == 0?
       bne SETNX1

*      ;this is a valid solenoid throw spot
*      ;bit0 is already clear

       ldab BITCOM
       orb #%00000010
       stab BITCOM               ;set BITCOM bit1

*****************************************************************

B2
       ldab BITCOM
       andb #%00000100                  ;? is BITCOM bit2 == 0?
       bne SETNX2
       bra G2

SETNX2 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%00000100
       staa 0,x
       bra B3

G2

       ldaa LASTBYT
       anda #%00000100                  ;? is LASTBYT bit2 == 1?
       beq SETNX2

       ldaa 0,x
       anda #%00000100                  ;? is bit2 == 0?
       bne SETNX2

*      ;this is a valid solenoid throw spot
*      ;bit2 is already clear

       ldab BITCOM
       orb #%00000100
       stab BITCOM               ;set BITCOM bit2
```

```
*****************************************************************

B3
       ldab BITCOM
       andb #%00001000                  ;? is BITCOM bit3 == 0?
       bne SETNX3
       bra G3

SETNX3 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%00001000
       staa 0,x
       bra B4

G3

       ldaa LASTBYT
       anda #%00001000                  ;? is LASTBYT bit3 == 1?
       beq SETNX3

       ldaa 0,x
       anda #%00001000                  ;? is bit3 == 0?
       bne SETNX3

*      ;this is a valid solenoid throw spot
*      ;bit2 is already clear

       ldab BITCOM
       orb #%00001000
       stab BITCOM               ;set BITCOM bit3

*****************************************************************

B4
       ldab BITCOM
       andb #%00010000                  ;? is BITCOM bit4 == 0?
       bne SETNX4
       bra G4

SETNX4 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%00010000
       staa 0,x
       bra B5

G4

       ldaa LASTBYT
       anda #%00010000                  ;? is LASTBYT bit4 == 1?
       beq SETNX4

       ldaa 0,x
       anda #%00010000                  ;? is bit4 == 0?
       bne SETNX4

*      ;this is a valid solenoid throw spot
*      ;bit4 is already clear

       ldab BITCOM
       orb #%00010000
       stab BITCOM               ;set BITCOM bit4

*****************************************************************

B5
```

```
       ldab BITCOM
       andb #%00100000                  ;? is BITCOM bit5 == 0?
       bne SETNX5
       bra G5

SETNX5 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%00100000
       staa 0,x
       bra B6

G5

       ldaa LASTBYT
       anda #%00100000                  ;? is LASTBYT bit5 == 1?
       beq SETNX5

       ldaa 0,x
       anda #%00100000                  ;? is bit5 == 0?
       bne SETNX5

*      ;this is a valid solenoid throw spot
*      ;bit4 is already clear

       ldab BITCOM
       orb #%00100000
       stab BITCOM               ;set BITCOM bit5


*****************************************************************

B6
       ldab BITCOM
       andb #%01000000                  ;? is BITCOM bit6 == 0?
       bne SETNX6
       bra G6

SETNX6 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%01000000
       staa 0,x
       bra B7

G6

       ldaa LASTBYT
       anda #%01000000                  ;? is LASTBYT bit6 == 1?
       beq SETNX6

       ldaa 0,x
       anda #%01000000                  ;? is bit6 == 0?
       bne SETNX6

*      ;this is a valid solenoid throw spot
*      ;bit6 is already clear

       ldab BITCOM
       orb #%01000000
       stab BITCOM               ;set BITCOM bit6

*****************************************************************

B7
       ldab BITCOM
       andb #%10000000                  ;? is BITCOM bit7 == 0?
```

```
       bne SETNX7
       bra G7

SETNX7 ldaa 0,x                  ;this pin is complete keep storing 1's
       ora #%10000000
       staa 0,x
       bra B8

G7

       ldaa LASTBYT
       anda #%10000000                  ;? is LASTBYT bit7 == 1?
       beq SETNX7

       ldaa 0,x
       anda #%10000000                  ;? is bit7 == 0?
       bne SETNX7

*      ;this is a valid solenoid throw spot
*      ;bit6 is already clear

       ldab BITCOM
       orb #%10000000
       stab BITCOM               ;set BITCOM bit7


*****************************************************************

B8
       LDAA TEMP
       STAA LASTBYT        ;before inx store current byte as lastbyte
       INX
       INC CTR3
       LDAA CTR3
       CMPA #34
       BNE AGAIN           ;increment x and loop through all 34 time steps
       BRA DONE
AGAIN  JMP B0
DONE

       RTS

************************************************
** SUBROUTINE ADVance - move stepper down 255 counts
************************************************

ADV    PSHX

       LDX #BASE

       LDAA #$ff
       STAA CTR

       ldaa #0
       STAA STOP

* ENABLE OC2 INTERRUPT
       BSET TMSK1,X BIT6
* TURN ON INTERRUPT SYSTEM
       CLI

WAIT41 LDAB STOP
       CMPB #1
```

```
       BNE WAIT41

* TURN OFF INTERRUPT SYSTEM
       SEI

       PULX
       RTS

*************************************************************************
** SUBROUTINE Record - records the 8 pin offsets
*************************************************************************
RECORD
       LDX #BASE

       LDAA PORTA,X        ; set plate to go up
       ORA #%00010000
       STAA PORTA,X

       LDX #$8000

       LDAA #0
godwn
       jsr KILLT                ;move plate down 32 incr. (read)
       LDAB $2000
       STAB 0,X
       JSR ADV
       INX
       INCA
       CMPA #31
       BNE godwn

       JSR ADV

       JSR ADV                  ;home at 34 again

       JSR ADV                  ; advance 512 counts
       JSR ADV
       JSR ADV
       JSR ADV
       JSR ADV
       JSR ADV                  ;40th ADV

       LDX #BASE
       LDAA PORTA,X        ; set plate to go down
       ANDA #%11101111
       STAA PORTA,X

DELAY  DEC CTR                  ;kill 256 E-clocks before direction change
       BNE DELAY

       JSR ADV                  ; advance 512 counts
       JSR ADV
       JSR ADV
       JSR ADV
       JSR ADV
       JSR ADV                  ;total=34

       LDAA PORTA,X        ; set plate to go up
       ORA #%00010000
       STAA PORTA,X

DELY   DEC CTR                  ;kill 256 E-clocks before direction change
       BNE DELY
```

```
       RTS

************************************************
** SUBROUTINE WAIT - wait for toggle switch press
************************************************

WAIT   ldx #BASE

       LDAA PORTA,X        ;turn on waiting light
       ANDA #%11011111
       STAA PORTA,X

contin LDAA PORTA,X        ;poll untill continue switch is pushed
       ANDA #%00000010
       BNE contin

       LDAA PORTA,X        ;turn waiting light off
       ORA #%00100000
       STAA PORTA,X

       RTS

*********************************************************************
** SUBROUTINE CLEAR - moves stepper up 36 x 256 counts, down 512 counts
*********************************************************************
CLEAR
       LDX #BASE

       LDAA PORTA,X        ; set plate to go up
       ORA #%00010000
       STAA PORTA,X

       ldaa #40            ; call ADV 40 times, later move back down 6 ADV's
       staa CTR2
MAS    JSR ADV
       DEC CTR2
       BNE MAS

       LDAA PORTA,X        ; set plate to go down
       ANDA #%11101111
       STAA PORTA,X

DELY2  DEC CTR                   ;kill 256 E-clocks before direction change
       BNE DELY2

       JSR ADV                   ; advance 512 counts
       JSR ADV
       JSR ADV                   ;at zero here
       JSR ADV
       JSR ADV                   ;down one more
       JSR ADV

       LDAA PORTA,X        ; set plate to go up
       ORA #%00010000
       STAA PORTA,X

DELY3  DEC CTR                   ;kill 256 E-clocks before direction change
       BNE DELY3

       RTS

*********************************************************************
```

```
** SUBROUTINE MODE - decide mode (display 3d or digitize 3d)
*   Send back decision in accum. A
* toggle switch (PA0), continue switch (PA1), LED (PA5)
*************************************************************************
MODE   ldaa #$15
       staa CTR4

modlp  dec CTR4            ;blink 15 cycles
       beq mlpend

       LDX #BASE

       LDAA PORTA,X        ;turn waiting light off
       ORA #%00100000
       STAA PORTA,X

       LDAA PORTA,X        ;different flash speeds for 2 modes
       ANDA #%00000001
       BNE notdis

       ldaa #$00           ;kill time with enter button to escape
       ldab #$20
eat    psha
       LDAA PORTA,X        ;if enter is pushed get out
       ANDA #%00000010
       BEQ mlp
       pula
       deca
       bne eat
       decb
       bne eat

notdis ldaa #$00           ;kill time with enter button to escape
       ldab #$20
eat2   psha
       LDAA PORTA,X        ;if enter is pushed get out
       ANDA #%00000010
       BEQ mlp
       pula
       deca
       bne eat2
       decb
       bne eat2

       LDAA PORTA,X        ;turn on waiting light
       ANDA #%11011111
       STAA PORTA,X

       ldaa #$00           ;kill time with enter button to escape
       ldab #$20
eat3   psha
       LDAA PORTA,X        ;if enter is pushed get out
       ANDA #%00000010
       BEQ mlp
       pula
       deca
       bne eat3
       decb
       bne eat3


       bra modlp           ; loop back for $5 times
```

```
mlp    pula
mlpend LDAA PORTA,X        ;turn waiting light off
       ORA #%00100000
       STAA PORTA,X

       ldaa PORTA,X

       RTS

************************************************
** INTERRUPT SERVICE ROUTINE
************************************************
       ORG    $A000

* INTERRUPT FROM OC2?
OC2_ISR      LDX    #BASE
       BRCLR  TFLG1,X BIT6 RT_OC2 ;IGNORE ILLEGAL
*                                 ;   INTERRUPT
       DEC CTR
       BNE CONT

       INC STOP
* Disable OC2 interrupt
       BCLR TMSK1,X BIT6

CONT
*-----------------------------------------------
* SERVICE OUTPUT COMPARE 2
*-----------------------------------------------
* CLEAR OC2 FLAG
       BCLR   TFLG1,X INV6

* WAS LAST OUTPUT HIGH? (Bit7:6=OM2:OL2  Hi=11,Lo=10)
       BRSET  TCTL1,X BIT6 LASTHI ;BRANCH ON YES

* PROGRAM NEXT OUTPUT TO BE HIGH
*   SET OC2 OUTPUT ACTION
       BSET   TCTL1,X BIT6  ;OM2:OL2=11 (bit 7 already 1)

*   SET HIGH OUTPUT COMPARE TIME
       LDD    TOC2,X
       ADDD   PWMLO
       STD    TOC2,X
       BRA    RT_OC2

* PROGRAM NEXT OUTPUT TO BE LOW
*   SET OC2 OUTPUT ACTION
LASTHI BCLR   TCTL1,X BIT6  ;OM2:OL2=10

*   SET LOW OUTPUT COMPARE TIME
       LDD    TOC2,X
       ADDD   PWMHI
       STD    TOC2,X

* RETURN FROM INTERRUPT
RT_OC2 RTI


************************************************************************
** SUBROUTINE KILLT - wait
************************************************************************
KILLT  ldaa #$00
       ldab #$ff
```

```
burn2  deca
       bne burn2
       decb
       bne burn2
       RTS

*************************************************************************
*                       SUBROUTINE - InitSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*              sets up the SCI port for 1 start bit, 8 data bits and
*              1 stop bit.  It also enables the transmitter and receiver.
*              Effected registers are BAUD, SCCR1, and SCCR2.
* Input         : None.
* Output        : Initializes SCI.
* Destroys      : None.
* Calls         : None.
*************************************************************************
*
InitSCI PSHA                     ; Save contents of A register
       PSHX

       LDX #BASE
       LDAA #$30
       STAA BAUD,X               ; Set BAUD rate to 9600
       LDAA #$00                 ; Set SCI Mode to 1 start bit,
       STAA SCCR1,X              ;     8 data bits, and 1 stop bit.
       LDAA #%00001100                ; Enable SCI Transmitter
       STAA SCCR2,X              ;     and Receiver

       PULX
       PULA               ; Restore A register
       RTS                ; Return from subtoutine

*************************************************************************
*                SUBROUTINE  -  InChar
* Description: Receives the typed character into register A.
* Input         : None
* Output        : Register A = input from SCI
* Destroys      : Contents of Register A
* Calls         : None.
*************************************************************************
*
InChar

       LDX #BASE
       LDAA SCSR,X                ; Check status reg.
       ANDA #%00100000                 ;     (load it into A reg)
*                                       ; Check if receive buffer full
       BEQ InChar                      ; Wait until data present
       LDAA SCDR,X                  ; SCI data ==> A register

       RTS                             ; Return from subroutine

*************************************************************************
*                       SUBROUTINE -  OutChar
* Description: Outputs the character in register A to the screen after
*              checking if the Transmitter Data Register is Empty
* Input         : Data to be transmitted in register A.
* Output        : Transmit the data.
* Destroys      : None.
* Calls         : None.
*************************************************************************
*
```

```
OutChar  PSHB                    ; Save contents of B register
       ldy #BASE
Loop1   LDAB SCSR,Y              ; Check status reg (load it into B reg)

       ANDB #%1000000                  ; Check if transmit buffer is empty

       BEQ Loop1                 ; Wait until empty

       STAA SCDR,Y               ; A register ==> SCI data

       PULB                      ; Restore B register
       RTS                       ; Return from subtoutine
*
*********************************************************************
*                       SUBROUTINE -  OutStr
* Description: Outputs the string terminated by two EOS's. The starting
location
*              of the string is pointed by X register. Calls the OutChar
*              subroutine to display a character on the screen and
*              exit once EOS has been reached.
* Input         : Starting location of the string to be transmitted
*               : (passed in X register)
* Output        : Prints the string.
* Destroys      : Contents of X register.
* Calls         : OutChar.
*********************************************************************
*
OutStr:       PSHA
OutStr1:
       LDAA   0,X                ; Get a character (put in A register)

       CMPA   #EOS               ; Check if it's EOS
       BEQ    onemore                  ; check for two EOS's in a row
       BRA    notend
onemore       INX
       LDAA   0,X
       CMPA   #EOS
       BEQ    Done
       DEX
       LDAA 0,X

notend  BSR     OutChar                 ; Print the character by calling
OutChar
        INX
       BRA     OutStr1
Done:
       PULA
        RTS                      ; Return from subtoutine

*****************************************************************
*      END OF PROGRAM
*****************************************************************
```

The following HTML code, Java Applet, Java Applications, and VRML code compromise the graphical user interface for the 3-D Display Device:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
```

```
<TITLE>
HTML Test Page
</TITLE>
</HEAD>
<BODY bgcolor="#000000">
<p align="center"><font color="#FFFFFF"><b><font size="+1">Three Dimensional
Surface
  Display Device</font></b></font></p>
<p><font color="#FFFFFF">The applet will appear below in a Java enabled
browser.</font></p>
<table width="87%">
  <tr>
    <td rowspan="2" width="48%"><font size=1><embed src="pinMatrix.wrl"
align="baseline" border="0" width="538" height="309">
      </embed></font></td>
    <td width="52%" height="106"><a href="http://cimar.me.ufl.edu/CIMAR"><img
src="clientImg.jpg" width="317" height="100" border="0"></a></td>
  </tr>
  <tr>
    <td width="52%" height="204"><font size=1><applet
  codebase = "."
  code     = "pins.PinsApplet.class"
  name     = "TestApplet"
  width    = 317
  height   = 203
  hspace   = 0
  vspace   = 0
  align    = top
  mayscript
>
      </applet></font></td>
  </tr>
</table>
<p><BR>
</p>
<P ALIGN="CENTER">Pins<FONT SIZE=1> </font></P>
</BODY>
</HTML>
```

```
/* PinsApplet.java */
package pins;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.io.*;
import java.net.*;
import java.util.*;
import vrml.external.field.EventInSFVec3f;
import vrml.external.field.EventOutSFVec3f;
import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;


/**
 * PinsApplet uses java1.2 API. It contains the
 * following functionality.
 * - A graphical user interface to get user inputs
 * - Communicates with a VRML model running in an
 *   embedded VRML plugin.
```

```
 * - Communicates with the PinServer program.
 *
 * @author  Arfath Pasha & Jeremy Mayer
 * @version 1.0 (7/9/01)
 */
public class PinsApplet extends Applet
{
  // may be executed as a standalone program
  private boolean isStandalone = false;
  // toggle to debug mode
  private static boolean DEBUG = true;
  // stores an instance of the browser object
  private Browser browser = null;

  // VRML nodes for each column of pins in the pin matrix
  private Node col1 = null;
  private Node col2 = null;
  private Node col3 = null;
  private Node col4 = null;
  private Node col5 = null;
  private Node col6 = null;
  private Node col7 = null;
  private Node col8 = null;

  // Event nodes for the pin translations
  private EventInSFVec3f  newTranslation1 = null;
  private EventOutSFVec3f oldTranslation1 = null;
  private EventInSFVec3f  newTranslation2 = null;
  private EventOutSFVec3f oldTranslation2 = null;
  private EventInSFVec3f  newTranslation3 = null;
  private EventOutSFVec3f oldTranslation3 = null;
  private EventInSFVec3f  newTranslation4 = null;
  private EventOutSFVec3f oldTranslation4 = null;
  private EventInSFVec3f  newTranslation5 = null;
  private EventOutSFVec3f oldTranslation5 = null;
  private EventInSFVec3f  newTranslation6 = null;
  private EventOutSFVec3f oldTranslation6 = null;
  private EventInSFVec3f  newTranslation7 = null;
  private EventOutSFVec3f oldTranslation7 = null;
  private EventInSFVec3f  newTranslation8 = null;
  private EventOutSFVec3f oldTranslation8 = null;

  // text area
  private static TextArea textOutput = new TextArea();
  // button to send information to the PinServer
  private Button sendButton = new Button();
  // button to receive information from the PinServer
  private Button receiveButton = new Button();
  // pin heights received from the PinServer
  private Float fPinHts[] = new Float[8];
  private String sPinHts = new String();

  /* network variables */
  // socket based buffered input stream
  private BufferedReader in;
  // socket based output stream
  private PrintWriter out;
  // socket connection
  private Socket socket;


        /** converts pin heights from floats to String
         */
```

```
private void htsToString() {
  Integer iHt;
  sPinHts = new String();
  for(int ii=0; ii<8; ii++) {
      iHt = new Integer((int)((fPinHts[ii].floatValue()*33f)/2f));
      sPinHts = sPinHts.concat(iHt.toString());
      sPinHts = sPinHts.concat(" ");  // white space is the tokenizer
  }
}


 /** converts pin heights from String to floats
  */
private void htsToFloat() {
 StringTokenizer st = new StringTokenizer(sPinHts);
 int ctr = 0;
 Integer iHt;
   while (st.hasMoreTokens()) {
       iHt = new Integer(st.nextToken());
       fPinHts[ctr] = new Float(((float)iHt.intValue()*2f/33f));
       ctr++;
   }
}

 /** Constructor for the applet
  */
public PinsApplet() throws IOException {
  // connect to PinServer
  //IPADDRESS MUST BE CHANGED
  InetAddress addr = InetAddress.getByName("128.227.244.198");
  System.out.println("addr "+ addr);
  socket = new Socket(addr, 8080);
  textOutput.append("Socket Connection created:\n\t" + socket.toString() +
                                                                    "\n");
}

 /** Initialize the applet
  */
public void init()
{
  if(DEBUG)
           textOutput.append("init() called...\n");

  try
  {
    jbInit();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}

 /** Component initialization
  */
private void jbInit() throws Exception
{
  textOutput.setColumns(50);
  textOutput.setRows(5);
  this.setSize(new Dimension(400,300));
  sendButton.setLabel("Send");
  sendButton.addActionListener(new java.awt.event.ActionListener()
  {
```

```
    public void actionPerformed(ActionEvent e)
    {
      sendButton_actionPerformed(e);
    }
  });
  receiveButton.setLabel("Receive");
  receiveButton.addActionListener(new java.awt.event.ActionListener()
  {

    public void actionPerformed(ActionEvent e)
    {
      receiveButton_actionPerformed(e);
    }
  });
  this.add(textOutput, null);
  this.add(sendButton, null);
  this.add(receiveButton, null);
}


 /** Start the applet
  */
public void start()
{
  if(DEBUG)
          textOutput.append("start() called...\n");

  /* VRML */
  // get an instance of the browser
  browser = (Browser) vrml.external.Browser.getBrowser(this);
  if(DEBUG)
          textOutput.append(browser.toString());

  try
        {
              // get instances of the nodes
              col1 = browser.getNode("col1");
              col2 = browser.getNode("col2");
              col3 = browser.getNode("col3");
              col4 = browser.getNode("col4");
              col5 = browser.getNode("col5");
              col6 = browser.getNode("col6");
              col7 = browser.getNode("col7");
              col8 = browser.getNode("col8");

              textOutput.append("Got the nodes\n");

              //get the newTranslation EventIn
              newTranslation1 = (EventInSFVec3f)
                                        col1.getEventIn("set_translation");
              newTranslation2 = (EventInSFVec3f)
                                        col2.getEventIn("set_translation");
              newTranslation3 = (EventInSFVec3f)
                                        col3.getEventIn("set_translation");
              newTranslation4 = (EventInSFVec3f)
                                        col4.getEventIn("set_translation");
              newTranslation5 = (EventInSFVec3f)
                                        col5.getEventIn("set_translation");
              newTranslation6 = (EventInSFVec3f)
                                        col6.getEventIn("set_translation");
              newTranslation7 = (EventInSFVec3f)
                                        col7.getEventIn("set_translation");
```

```
                newTranslation8 = (EventInSFVec3f)
                                        col8.getEventIn("set_translation");
                textOutput.append("Got EventIn: set_translation\n");

                //get the oldTranslation EventOut
                oldTranslation1 = (EventOutSFVec3f)
                                        col1.getEventOut("translation");
                oldTranslation2 = (EventOutSFVec3f)
                                        col2.getEventOut("translation");
                oldTranslation3 = (EventOutSFVec3f)
                                        col3.getEventOut("translation");
                oldTranslation4 = (EventOutSFVec3f)
                                        col4.getEventOut("translation");
                oldTranslation5 = (EventOutSFVec3f)
                                        col5.getEventOut("translation");
                oldTranslation6 = (EventOutSFVec3f)
                                        col6.getEventOut("translation");
                oldTranslation7 = (EventOutSFVec3f)
                                        col7.getEventOut("translation");
                oldTranslation8 = (EventOutSFVec3f)
                                        col8.getEventOut("translation");
                textOutput.append("Got EventOut: translation\n");
        }
        catch (InvalidNodeException ne)
        {
                textOutput.append("Failed to get node:" + ne);
        }
        catch (InvalidEventInException ee)
        {
                textOutput.append("Failed to get EventIn:" + ee);
        }
        catch (InvalidEventOutException ee)
        {
                textOutput.append("Failed to get EventOut:" + ee);
        }

  /* network */
  try {
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    out = new PrintWriter(new BufferedWriter(
                    new OutputStreamWriter(socket.getOutputStream())), true);
  }
  catch(Exception ee) {
      textOutput.append("Error: unable to create network I/O in"+
                                                    "ClientApplet.start()\n");
  }
}

 /** Stop the applet
  */
public void stop()
{
  if(DEBUG)
       textOutput.append("stop() called...\n");

  try {
    out.print("END");
    socket.close();
  }
  catch(Exception ee) {
      System.out.println("Error: Failed to close socket in " +
                                                   "ClientApplet.stop() "+ ee);
  }
```

```
  textOutput.append("closing socket...");
}


 /** Destroy the applet
  */
public void destroy()
{
if(DEBUG)
  textOutput.append("destroy() called...\n");
}


 /** sends pin heights from VRML to server
  * when the send button is pressed.
  */
void sendButton_actionPerformed(ActionEvent e)
{
    // getting pin heights from VRML
    textOutput.append("Getting pin heights from display device...\n");
    float [] translations = new float[3];
    translations = oldTranslation1.getValue();
    fPinHts[0] = new Float(translations[1]);
    translations = oldTranslation2.getValue();
    fPinHts[1] = new Float(translations[1]);
    translations = oldTranslation3.getValue();
    fPinHts[2] = new Float(translations[1]);
    translations = oldTranslation4.getValue();
    fPinHts[3] = new Float(translations[1]);
    translations = oldTranslation5.getValue();
    fPinHts[4] = new Float(translations[1]);
    translations = oldTranslation6.getValue();
    fPinHts[5] = new Float(translations[1]);
    translations = oldTranslation7.getValue();
    fPinHts[6] = new Float(translations[1]);
    translations = oldTranslation8.getValue();
    fPinHts[7] = new Float(translations[1]);

    // sending pin heights to server
    textOutput.append("Sending pin heights to display device...\n");
    this.htsToString();
    out.println(sPinHts);
}


 /** recieve pin heights from server and display in VRML
  * when the recieve button is pressed.
  */
void receiveButton_actionPerformed(ActionEvent e)
{
  // request data
  textOutput.append("requesting pin heights from display device...\n");
  out.println("GETDATA");
  // recieving data from server
  textOutput.append("receiving pin heights from display device...\n");
  try {
    sPinHts = in.readLine();
  }
  catch( Exception ee) {
    System.out.println("Error: Failed to get data from server in" +
                                        " receiveButton_actionPerformed() "+ee);
  }
```

```
    // convert string to float
    this.htsToFloat();

    // recieved values being sent to vrml
    textOutput.append("Displaying heights in VRML...\n");
    float[] translations = new float[3];
    translations[1] = fPinHts[0].floatValue();
    newTranslation1.setValue(translations);
    translations[1] = fPinHts[1].floatValue();
    newTranslation2.setValue(translations);
    translations[1] = fPinHts[2].floatValue();
    newTranslation3.setValue(translations);
    translations[1] = fPinHts[3].floatValue();
    newTranslation4.setValue(translations);
    translations[1] = fPinHts[4].floatValue();
    newTranslation5.setValue(translations);
    translations[1] = fPinHts[5].floatValue();
    newTranslation6.setValue(translations);
    translations[1] = fPinHts[6].floatValue();
    newTranslation7.setValue(translations);
    translations[1] = fPinHts[7].floatValue();
    newTranslation8.setValue(translations);
  }


   /** Main method
    */
  public static void main(String[] args)
  {
    PinsApplet applet  = null;
    try {
      applet = new PinsApplet();
    }
    catch(Exception ee) {
      System.out.println("Failed to instantiate applet in main() "+ee);
    }

    applet.isStandalone = true;
    if(DEBUG)
      System.out.println("Applet initialized");

    Frame frame = new Frame();
    frame.setTitle("Applet Frame");
    frame.add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    if(DEBUG)
      System.out.println("Applet started");

    frame.setSize(400,320);
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) / 2,
                                      (d.height - frame.getSize().height) / 2);
    frame.setVisible(true);
  }
}
```

```
/* PinServer.java */
package pinserver;
```

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.comm.*;
import java.lang.*;

/**
 * This class must be executed to run the PinServer program.
 * Creates an instance of the PinServer program.
 * Establishes the socket connection and contains
 * procedures for communication with the applet.
 *
 * @author  Arfath Pasha & Jeremy Mayer
 * @version 1.0 (7/9/01)
 */
public class PinServer {
  boolean DEBUG = false;  // toggle for debugging
  boolean packFrame = false;
  // choose a port outside of the range 1-1024
  public static final int PORT = 8080;
  ServerSocket s;

   /** Construct the application
    */
  public PinServer() throws IOException {
    // create the frame
    PinServerFrame frame = new PinServerFrame();

    // create an instance of the data structure
    PinData data = new PinData();

    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame)
      frame.pack();
    else
      frame.validate();
    frame.setVisible(true);

    /* server implementation */
    try {
      while(true) {
        s = new ServerSocket(PORT);
        frame.textArea.append("\nStarted Server Socket: "+ s.toString() + "\n");
        Socket socket = s.accept();   // blocks until a connection occurs

        try {
          frame.textArea.append("Client Connection Accepted: " +
                                                    socket.toString()+ "\n");
          BufferedReader in = new BufferedReader(new InputStreamReader(
                                                   socket.getInputStream()));
          // output is automatically flushed by PrintWriter
          PrintWriter out = new PrintWriter(new BufferedWriter(
                       new OutputStreamWriter(socket.getOutputStream())), true);

          while(true) {
            /* get client message */
            String str = new String(in.readLine());

            /* service client message */
            if(str.length() != 0) {
              if(str.equals("GETDATA")) {       // send data to client
                frame.textArea.append("\nClient requested data\n");
```

```
                // get data from serial port and update PinData
                SerialRead read = new SerialRead();
                if (DEBUG)
                          System.out.println("Serial Read instantiated");

                 // updata data with input from serial port
                data.update(read.read());
                if(DEBUG)
                 System.out.println("data read");

                // send data to client
                frame.textArea.append("Display device sent " +
                                                    data.getStrData() + "\n");
                out.println(data.getStrData());
                frame.textArea.append("data sent to client\n");
              }
              else if(str.equals("END")) {
                break;  // close socket conenction
              }
              else {
                frame.textArea.append("\nClient sent data\n");
                frame.textArea.append(str + "\n");

                // updata PinData with client data

                data.update(str);
                // send PinData to serial port

                frame.textArea.append("Sending data to serial port\n");
                SerialWrite write = new SerialWrite(data.getByteData());
                frame.textArea.append("Data sent to serial port\n");
              }
            }
          }
        }
        catch(Exception ee) {
          System.out.println("Error:Server I/O problem[PinServer.PinServer()]\n"
                                                                          + ee);

        }
        frame.textArea.append("Client Connection Closed: "+ socket.toString()+
                                                                          "\n");
        socket.close();   //always close the socket
        s.close();
      }
    }
    finally {
      s.close();   // always close the socket
    }
  }

  //Main method
  public static void main(String[] args) {
    try {
      new PinServer();  // start the PinServer
    }
    catch(Exception ee) {
      System.out.println("Error: Could not start PinServer [PinServer.main()]\n"
                                                                          + ee);
    }
  }
}
```

```
/* PinServerFrame.java */
package pinserver;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


/**
 * PinServerFrame builds the Graphical user
 * interface for the PinServer
 *
 * @author  Arfath Pasha & Jeremy Mayer
 * @version 1.0 (7/9/01)
 */
public class PinServerFrame extends JFrame {
  JScrollPane scrollPane = null;
  JTextArea textArea = null;
  JLabel imgLabel = null;


   /** Construct the frame
    */
  public PinServerFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try  {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }


   /** Component initialization
    */
  private void jbInit() throws Exception  {

        textArea = new JTextArea(10,50);
    textArea.setText("Server Log:");

    scrollPane = new JScrollPane(textArea);

    imgLabel = new JLabel();
    imgLabel.setIcon( new ImageIcon("pinserver/cimarImg.jpg") );
    imgLabel.setVisible(true);

    this.getContentPane().setLayout(new BorderLayout());
    this.setSize(new Dimension(500, 350));
    this.setTitle("Pin Server");
    this.getContentPane().add(scrollPane, BorderLayout.NORTH);
    this.getContentPane().add(imgLabel, BorderLayout.SOUTH);
  }


   /** Overridden so we can exit on System Close
    */
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if(e.getID() == WindowEvent.WINDOW_CLOSING) {
```

```
      System.exit(0);
    }
  }
}
```

```
/* PinData.java */
package pinserver;

import java.util.*;
import java.lang.*;


/**
 * Data structure for maintaining and manipulating
 * the pin data.
 *
 * @author  Arfath Pasha & Jeremy Mayer
 * @version 1.0 (7/9/01)
 */
public class PinData {
  // pin heights
  Integer [] data = null;


   /** constructor. Initializes the pin data structure
    */
  public PinData() {
    data = new Integer[8];
    for(int ii=0; ii< 8; ii++)
     data[ii] = new Integer(-1);
  }


   /** updates the pin data with a string as input
    * @param strData input data in String format
    */
  public void update(String strData) {
   StringTokenizer st = new StringTokenizer(strData);
   int ctr = 0;
     while (st.hasMoreTokens()) {
         data[ctr] = new Integer(st.nextToken());
         ctr++;
     }
  }


   /** updates pin data with a byte array as input
    * @param byteData input data as an array of bytes
    */
  public void update(byte [] byteData) {
    int decr = 33;

    for(int jj=0; jj<34; jj++)
    {
       if ( (byteData[jj] & (byte)0x01) == 0) data[7] = new Integer(decr);
       if ( (byteData[jj] & (byte)0x02) == 0) data[6] = new Integer(decr);
       if ( (byteData[jj] & (byte)0x04) == 0) data[5] = new Integer(decr);
       if ( (byteData[jj] & (byte)0x08) == 0) data[4] = new Integer(decr);
       if ( (byteData[jj] & (byte)0x10) == 0) data[3] = new Integer(decr);
       if ( (byteData[jj] & (byte)0x20) == 0) data[2] = new Integer(decr);
       if ( (byteData[jj] & (byte)0x40) == 0) data[1] = new Integer(decr);
```

```
     if ( (byteData[jj] & (byte)0x80) == 0) data[0] = new Integer(decr);

     decr--;
  }
}


 /** returns the pin data in Integer form
  * @return array of Integers containing pin heights
  */
public Integer [] getIntData() {
    return data;
}


 /** returns the pin data in String format
  * @return a String containing pin heights
  */
public String getStrData() {
  String strData = new String();
  for(int ii=0; ii<8; ii++) {
      strData = strData.concat(data[ii].toString());
      strData = strData.concat(" ");  // white space is the tokenizer
  }
  return strData;
}


 /** returns the pin data in byte form
  * @return array of bytes containing pin heights
  */
public byte [] getByteData() {
  byte[] offPattern = new byte[34];

  // fill byte array with all $FF's
  for (int ii=0; ii<34; ii++) offPattern[ii] = ((byte)0xFF);

  for(int ii=0; ii<34; ii++)
  {
     if (data[0].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xfe));
     if (data[1].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xfd));
     if (data[2].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xfb));
     if (data[3].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xf7));
     if (data[4].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xef));
     if (data[5].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xdf));
     if (data[6].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0xbf));
     if (data[7].intValue()==ii) offPattern[ii] = (byte) (offPattern[ii] &
                                                          ((byte)0x7f));
  }

  return offPattern;
}

  /** Test method
   */
public static void main(String[] args) {
```

```
    String str = new String("1 5 3 9 5 6 7 8 ");
    String retStr = new String();
    Integer [] retInt = new Integer[8];

    PinData data = new PinData();
   // data.toInt(str);
    retStr = data.getStrData();
    retInt = data.getIntData();


    System.out.println("retStr = " + retStr + ".." );
    for(int ii=0; ii<8; ii++) {
       if(retInt[ii] == null)
       System.out.println("retInt is null");
      else
         System.out.println(retInt[ii].toString() );
    }
  }
}
```

```
/* SerialRead.java */
package pinserver;

import java.io.*;
import java.util.*;
import javax.comm.*;
import java.lang.*;


/**
 * SerialRead listens to the serial port for
 * inputs. Once an input is received, an eventhandler
 * is executed and the data is read from the serial port
 * into a data buffer.
 *
 * @author  Arfath Pasha & Jeremy Mayer
 * @version 1.0 (7/9/01)
 */
public class SerialRead implements SerialPortEventListener  {
  static CommPortIdentifier portId;  // port identifier
  static Enumeration portList;       // an enumeration of all the ports
  InputStream inputStream;  // input stream to receive data from the port
  SerialPort serialPort;    // an instance of the serial port
  boolean portFound;        // boolean to check if the roght port is found
  boolean dataRead;         // boolean to check if all the data has been read
  boolean DEBUG = false; // toggle for debugging
  byte[] readBuffer;     // data buffer
  int bytesRead;         // number of bytes read


   /** creates an instance of the serial port
    */
  public SerialRead() {
       // initialize variables
       bytesRead = 0;
    readBuffer = new byte[34];
    portFound = false;
    dataRead = false;

    if(DEBUG) System.out.println("creating an instance of SerialRead...");
```

```
// get an enumeration of the ports
portList = CommPortIdentifier.getPortIdentifiers();

if(DEBUG) System.out.println("portList: " + portList);

while (portList.hasMoreElements()) {
  portId = (CommPortIdentifier) portList.nextElement();

  if(DEBUG) System.out.println("portId: "+portId);

  if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
    if(DEBUG) System.out.println("got serial port");

    if (portId.getName().equals("COM1")) {
      if(DEBUG) System.out.println("created an instance of SerialRead!!");
          portFound = true;
      break;
    }
  }
}

if(DEBUG) {
  System.out.println("got the serial port for reading");
  System.out.println("perparing to read from the serial port");
}

// open the serial port
try {
  serialPort = (SerialPort) portId.open("pinserver", 2000);
} catch (PortInUseException ee) {
  System.out.println("Error: port in use: " + ee);
}

if(DEBUG) System.out.println("serialPort opened");

// create an input stream
try {
  inputStream = serialPort.getInputStream();
} catch (IOException ee) {
  System.out.println("Error: unable to create an input stream: " + ee);
}

    if(DEBUG) System.out.println("input stream created");

// add an event handler for the port
try {
  serialPort.addEventListener(this);
} catch (TooManyListenersException ee) {
  System.out.println("Error:unable to add event handler to the serialport:"
                                                                      +ee);
}

    if(DEBUG) System.out.println("event listener added");

// set port parameters
serialPort.notifyOnDataAvailable(true);
try {
  serialPort.setSerialPortParams(9600,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);
} catch (UnsupportedCommOperationException ee) {
  System.out.println("Error: unable to set port parameters: " + ee);
```

```
  }

  if(DEBUG) System.out.println("params set");
}


 /** listens for input from the serial port and returns an array of
  * bytes when all the input is read. (34 bytes of data)
  * @return an array of bytes received from the serial port
  */
public byte [] read() {
  if(portFound == false) {
    System.out.println("Error: port not found. read failed");
    return readBuffer;
  }

  while(!dataRead); //System.out.println("waiting for data");

  if(DEBUG) System.out.println("input stream closed");

  try {
     inputStream.close();
  } catch(IOException ee) {
     System.out.println("IOException cought in SerialRead.read() "+ee);
  }
  // remove the event handler
  serialPort.removeEventListener();
  // close the serial port
  serialPort.close();

  if(DEBUG) System.out.println("read() returning");
  return readBuffer;
}


 /** Event handler for the serial port. Gets data from the port
  * and stores it in a buffer as data is being received
  */
public void serialEvent(SerialPortEvent event) {
  if(DEBUG) System.out.println("event listener called...");

  switch(event.getEventType()) {
      case SerialPortEvent.BI:
      case SerialPortEvent.OE:
      case SerialPortEvent.FE:
      case SerialPortEvent.PE:
      case SerialPortEvent.CD:
      case SerialPortEvent.CTS:
      case SerialPortEvent.DSR:
      case SerialPortEvent.RI:
      case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
          break;
      case SerialPortEvent.DATA_AVAILABLE:
            byte[] tempBuffer;  // used for flushing port buffer
        try {
            // amount of data received
             int avl = inputStream.available();

             while (avl > 0) {
                    tempBuffer = new byte[34];
                    if(DEBUG) System.out.println("available = " + avl);
                    int numBytes = inputStream.read(tempBuffer);
```

```
                    if(DEBUG) System.out.println("numBytes= "+ numBytes);

                         for(int ii=0; ii<numBytes; ii++) {
                            readBuffer[bytesRead+ii] = tempBuffer[ii];

                            if(DEBUG) System.out.print(
                                new Byte(tempBuffer[ii]).toString()+" ");
                         }
                    bytesRead += numBytes;
                    avl = inputStream.available();
               }

               if(DEBUG) System.out.println("\ndata read into buffer: ");
                         for (int jj=0; jj<34; jj++) {
                            if(DEBUG) System.out.print(
                                   new Byte(readBuffer[jj]).toString()+" ");
                         }

           if(bytesRead == 34) {
            dataRead = true;
            if(DEBUG) System.out.println("bytesRead is 34");
           }
           } catch (IOException e) {}
         break;
    }  // end switch
  }
}
```

```
/* SerialWrite.java */
package pinserver;

import java.io.*;
import java.net.*;
import java.util.*;
import javax.comm.*;
import java.lang.*;


/**
 * SerialWrite writes data to the serial port.
 *
 * @author  Arfath Pasha & Jeremy Mayer
 * @version 1.0 (7/9/01)
 */
public class SerialWrite {
    static Enumeration portList = null;  // enumeration of the ports
    static CommPortIdentifier portId;    // port identifier
    static SerialPort serialPort;        // an instance of the serial port
    static OutputStream outputStream;    // ouput stream
    boolean DEBUG = false;    // toggle for debugging

   /** creates an instance of the serial port and writes the
    * given data to the port
    * @param byteStream array of bytes to be written to the port
    */
  public SerialWrite(byte [] byteStream) {
    // get an enumeration of the ports
    portList = CommPortIdentifier.getPortIdentifiers();

    if(portList == null)
```

```
      System.out.println("did not create an Enumeration of the ports");
    else
       if(DEBUG) System.out.println("created an Enumeration of the ports");

    while (portList.hasMoreElements()) {

      if(DEBUG)  System.out.println("checking port for serial port");

      portId = (CommPortIdentifier) portList.nextElement();

      if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {

        if(DEBUG) System.out.println("got the serial port");

        if (portId.getName().equals("COM1")) {
          if(DEBUG) System.out.println("got COM1");
          try {
            serialPort = (SerialPort)
            portId.open("SimpleWriteApp", 2000);
            if(DEBUG) System.out.println("port opened");
          } catch (PortInUseException e) {
            System.out.println("Error: Port is in use: "+e);
          }

          // create an output stream
          try {
            outputStream = serialPort.getOutputStream();
            if(DEBUG) System.out.println("created outp0ut stream");
          } catch (IOException e) {
            System.out.println("Error: unable to create ouput stream: " +e);}
          try {
            serialPort.setSerialPortParams(9600,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);
            if(DEBUG) System.out.println("Set port params");
          } catch (UnsupportedCommOperationException e) {
            System.out.println("Error: unable to set port params");}

          // write data to the serial port
          try {
                          for(int ii = 0; ii<34; ii++)
             outputStream.write(byteStream[ii]);
            if(DEBUG) System.out.println("data written to serial port!!");
            outputStream.close();
          } catch (IOException e) {
           System.out.println("Error: write failed: "+e);}
           serialPort.close();
        }
      }
    }
    if(DEBUG) System.out.println("exiting SerialWrite");
  }
}
```

```
#VRML V2.0 utf8

#A Single pin

Transform {
  children [
```

```
    Transform {
      children Shape {
  appearance Appearance {
    material Material {
      diffuseColor   1 0 0
      transparency   0.5
    }

  }

  geometry  Cylinder {
    radius  0.1
    height  4
  }

      }

      translation 0 1 0
    }
    Transform {
      children Shape {
  appearance Appearance {
    material Material {
      diffuseColor   1 0 0
    }

  }

  geometry  Cylinder {
    radius  0.2
    height  0.1
  }

      }

      translation 0 3 0
      rotation 0 0 1  0
    }
  ]
}
```

```
#VRML V2.0 utf8

#A column of pins

Transform {
  children [
    Inline {
      url "pin.wrl"
    }
    Transform {
      children Inline {
       url "pin.wrl"
      }

      translation 0 0 -0.4
    }
    Transform {
      children Inline {
       url "pin.wrl"
      }
```

```
      translation 0 0 -0.8
    }
    Transform {
      children Inline {
       url "pin.wrl"
      }

      translation 0 0 -1.2
    }
    Transform {
      children Inline {
      url "pin.wrl"
       }

      translation 0 0 -1.6
    }
    Transform {
      children Inline {
      url "pin.wrl"
      }

      translation 0 0 -2
    }
    Transform {
      children Inline {
      url "pin.wrl"
      }

      translation 0 0 -2.4
    }
    Transform {
      children Inline {
      url "pin.wrl"
      }

      translation 0 0 -2.8
    }
    Transform {
      children Inline {
      url "pin.wrl"
      }

      translation 0 0 -3.2
    }
    Transform {
      children Inline {
      url "pin.wrl"
      }

      translation 0 0 -3.6
    }
  ]
}
```

```
#VRML V2.0 utf8

#A Pin Matrix

Transform {
  children [
    Transform {
      children [
```

```
DEF sensor1 PlaneSensor {
  enabled TRUE
  maxPosition 0 2
}
DEF col1 Transform {
  children Inline {
    url "pinColumn.wrl"
  }

  translation 0 0 0
}
     ]
     translation -1.8 0 0
   }
   Transform {
     children [
DEF sensor2 PlaneSensor {
  maxPosition 0 2
}
DEF col2 Transform {
  children Inline {
    url "pinColumn.wrl"
  }

  translation 0 0 0
}
     ]
     translation -1.4 0 0
   }
   Transform {
     children [
DEF sensor3 PlaneSensor {
  maxPosition 0 2
}
DEF col3 Transform {
  children Inline {
    url "pinColumn.wrl"
  }

  translation 0 0 0
}
     ]
     translation -1 0 0
   }
   Transform {
     children [
DEF sensor4 PlaneSensor {
  maxPosition 0 2
}
}
DEF col4 Transform {
  children Inline {
    url "pinColumn.wrl"
  }

  translation 0 0 0
}
     ]
     translation -0.6 0 0
   }
   Transform {
     children [
DEF sensor5 PlaneSensor {
  maxPosition 0 2
```

```
  }
  DEF col5 Transform {
    children Inline {
      url "pinColumn.wrl"
    }

    translation 0 0 0
  }
       ]
       translation -0.2 0 0
     }
     Transform {
       children [
  DEF sensor6 PlaneSensor {
    maxPosition 0 2
  }
  DEF col6 Transform {
    children Inline {
      url "pinColumn.wrl"
    }

    translation 0 0 0
  }
       ]
       translation 0.2 0 0
     }
     Transform {
       children [
  DEF sensor7 PlaneSensor {
    maxPosition 0 2
  }
  DEF col7 Transform {
    children Inline {
      url "pinColumn.wrl"
    }

    translation 0 0 0
  }
       ]
       translation 0.6 0 0
     }
     Transform {
       children [
  DEF sensor8 PlaneSensor {
    maxPosition 0 2
  }
  DEF col8 Transform {
    children Inline {
      url "pinColumn.wrl"
    }

    translation 0 0 0
  }
       ]
       translation 1 0 0
     }
   ]
   translation 0 -4 0
}
Transform {
  children Shape {
    appearance Appearance {
      material Material {
```

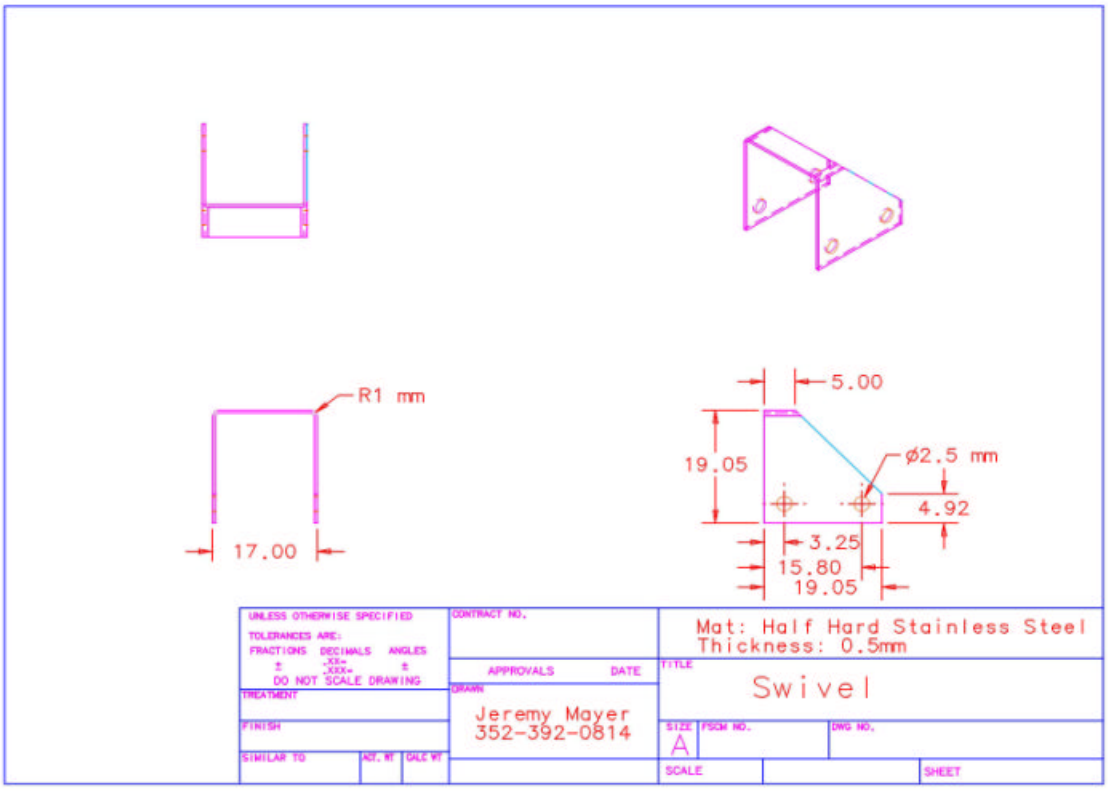
```
  ambientIntensity   0.23913
  diffuseColor  0.89 0.76 0.62
  specularColor 0.53 0.53 0.53
  emissiveColor 0 0 0
  shininess 0.93
  transparency  0
      }

    }

    geometry  Box {
      size 4 2 4
    }

  }

  translation -0.4 -4 -1.8
}
ROUTE sensor1.translation_changed TO col1.set_translation
ROUTE sensor2.translation_changed TO col2.set_translation
ROUTE sensor3.translation_changed TO col3.set_translation
ROUTE sensor4.translation_changed TO col4.set_translation
ROUTE sensor5.translation_changed TO col5.set_translation
ROUTE sensor6.translation_changed TO col6.set_translation
ROUTE sensor7.translation_changed TO col7.set_translation
ROUTE sensor8.translation_changed TO col8.set_translation
```

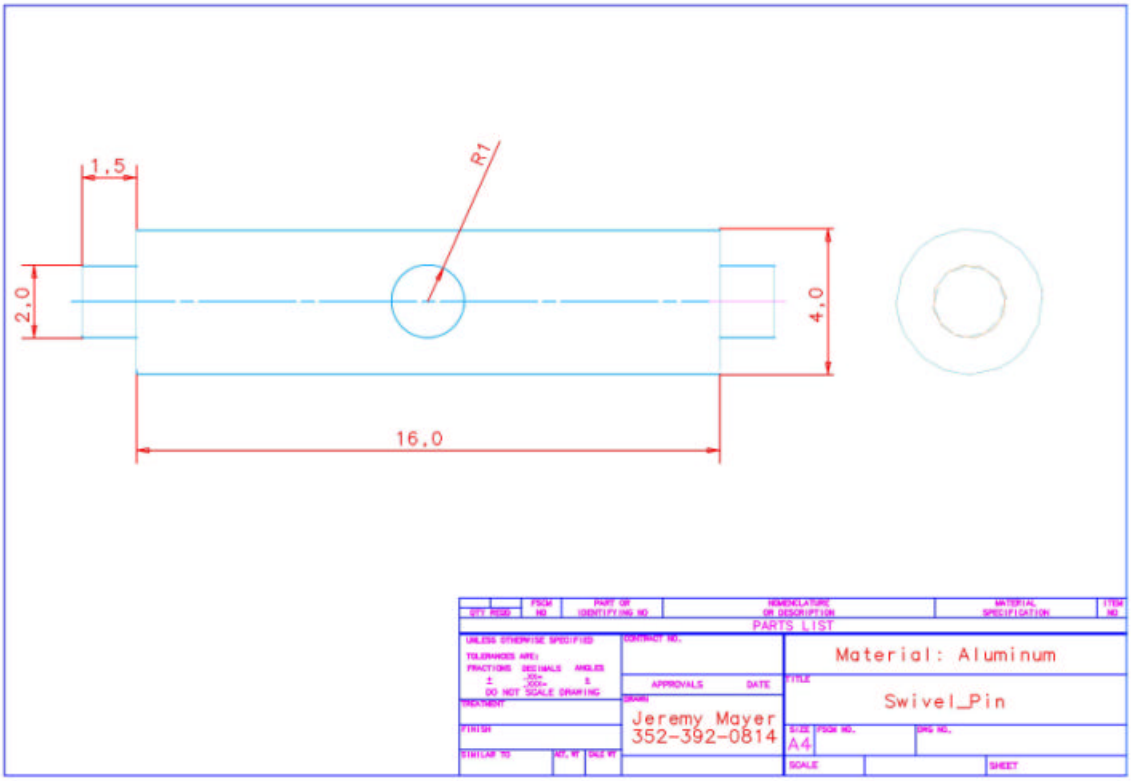
APPENDIX C
PART INFORMATION AND DRAWINGS

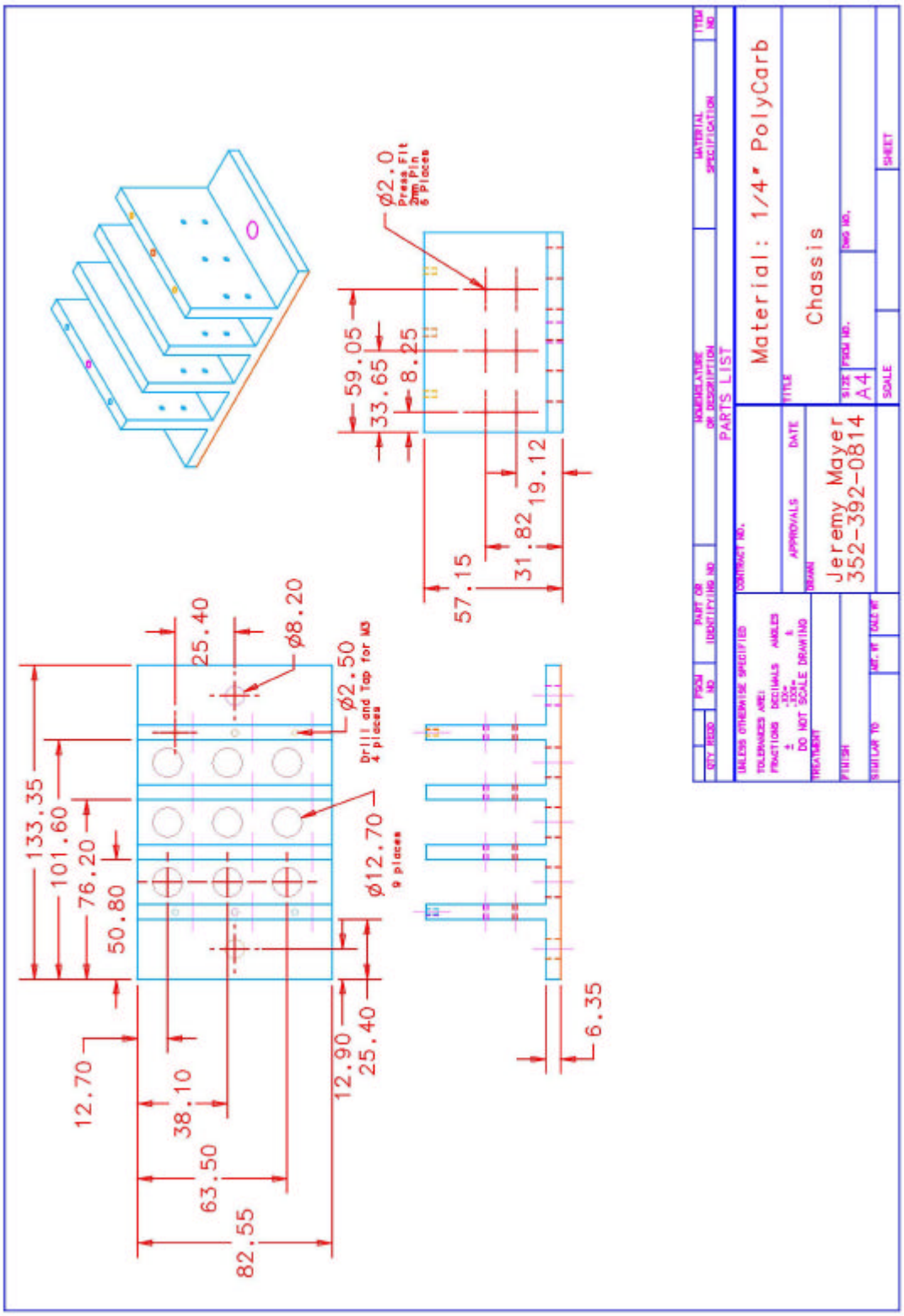
The following list contains information on the prototype parts that were ordered:

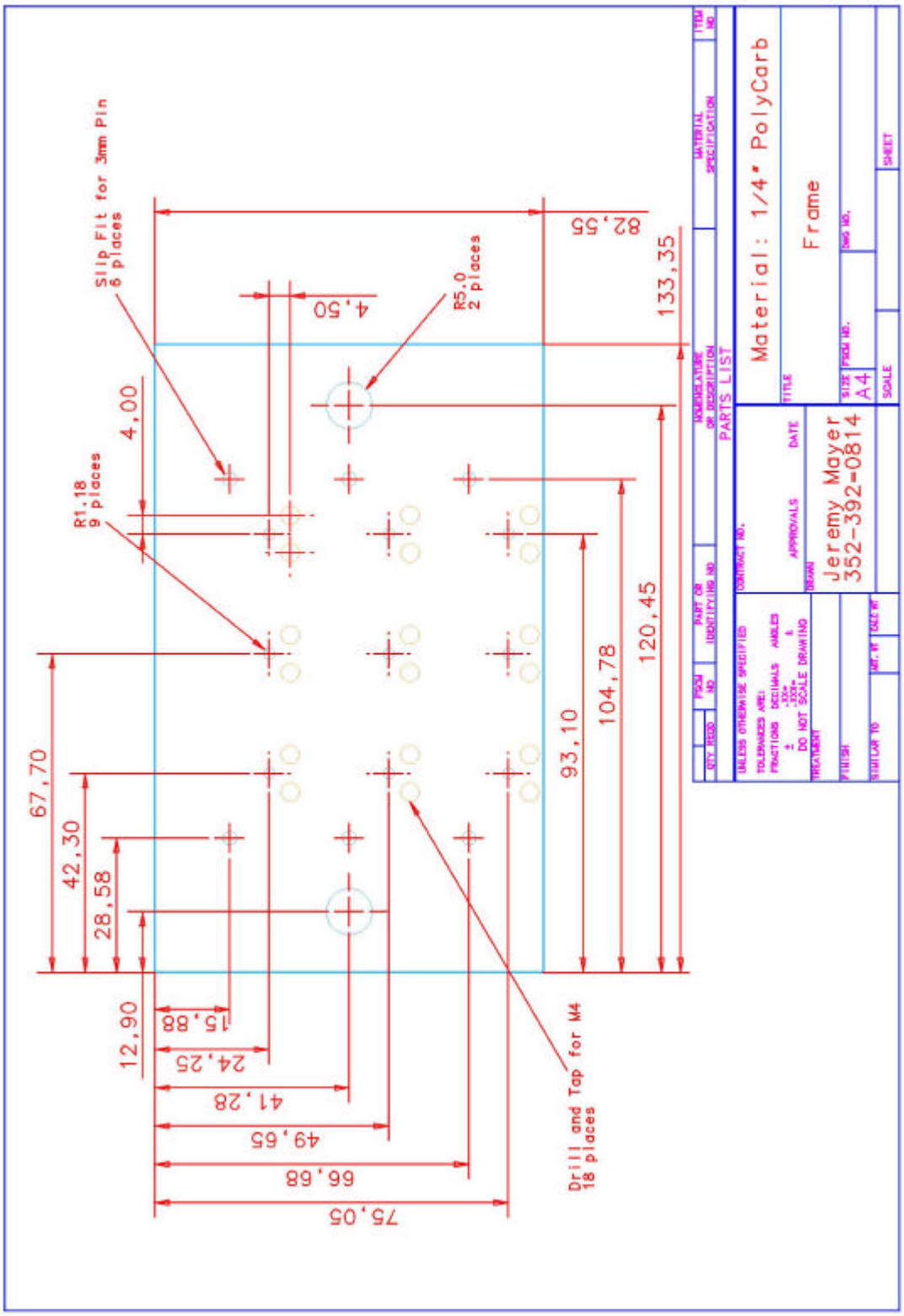
Part Description/Part Number:	**Supplier/Contact:**
Push-Pull Cables – C090N	Sava Ind.- (973) 835-0882
4.5-ohm Open Frame Solenoid – 4-9000-010-0021	Oak Grigsby – (800) 625-8333
10 turns-inch 3/8-inch Lead Screw - 6350K13	McMaster Carr – (404) 346-7000
10 turns-inch Precision ACME Nut - 95075A227	McMaster Carr – (404) 346-7000
Extra Thin Wall Tubing, 18 gauge – 5560K427	McMaster Carr – (404) 346-7000
Metric Blue M4 Screw, 50mm length – 91303A069	McMaster Carr – (404) 346-7000
Metric Blue M3 Screw, 6mm length – 91303A006	McMaster Carr – (404) 346-7000
Metric 2mm 18-8 Steel Dowel Pin – 91303A010	McMaster Carr – (404) 346-7000
Metric 3mm 18-8 Steel Dowel Pin – 91303A060	McMaster Carr – (404) 346-7000
Miniature Rotational Solenoid – 5462	Daco Ins. Co. – (203) 874-2515
6 Volt DC Stepper Motor – 23H-700	Servo Systems – (800) 922-1103
Stepper Motor Controller – CMD-50 98150510	Servo Systems –(800) 902-1103

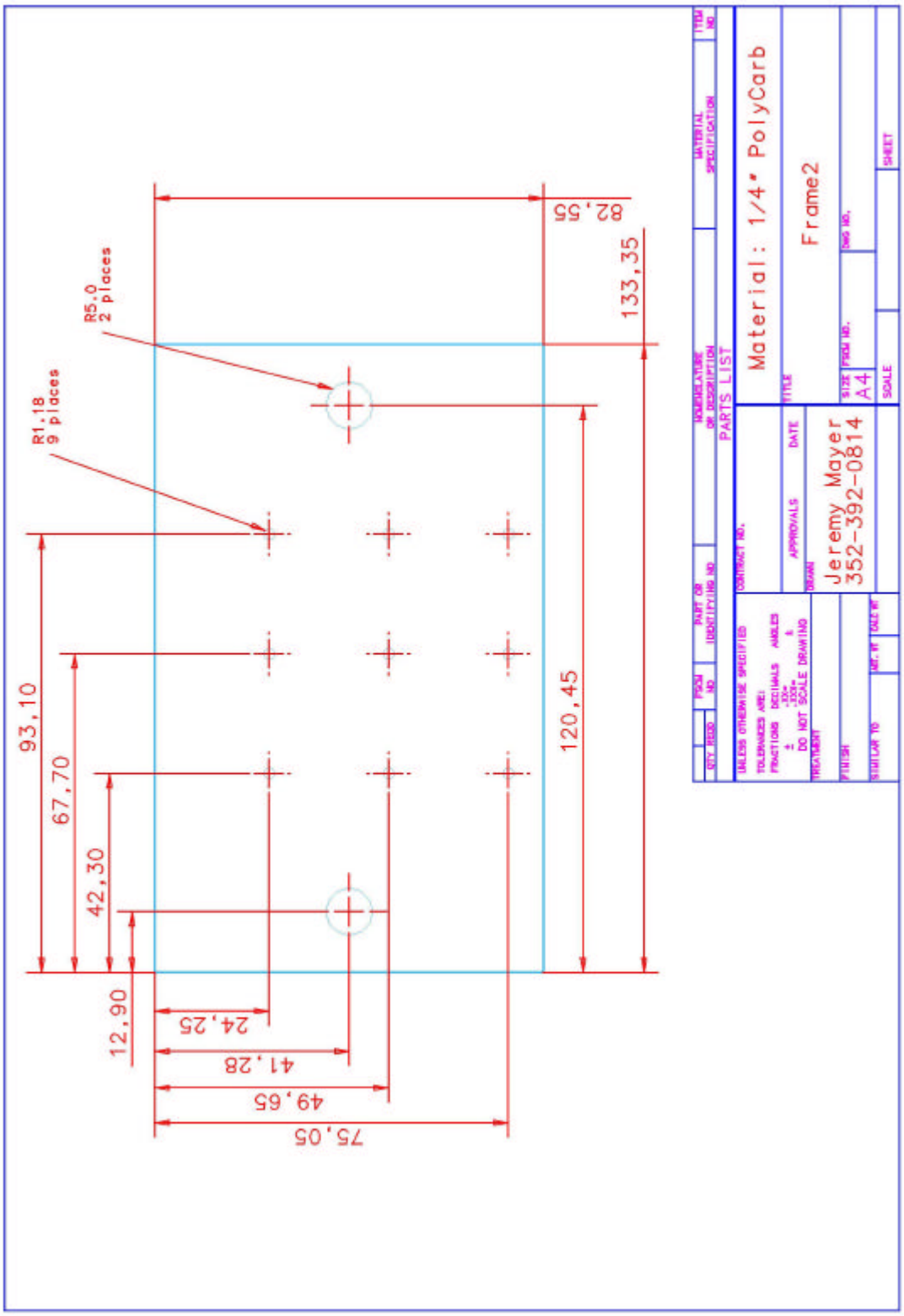
The following drawings were sent to Yuhas Tooling & Machining, Inc. in Milpitas, CA [(408) 934-9197] for part fabrication:


R1 mm
17.00
5.00
19.05
ø2.5 mm
4.92
3.25
15.80
19.05
UNLESS OTHERWISE SPECIFIED
TOLERANCES ARE:
FRACTIONS DECIMALS ANGLES
± .XX= ±
.XXX=
DO NOT SCALE DRAWING
CONTRACT NO.
APPROVALS DATE
TREATMENT
FINISH
DRAWN
Jeremy Mayer
352-392-0814
SIMILAR TO
MAT. WT
DALE WT
Mat: Half Hard Stainless Steel
Thickness: 0.5mm
TITLE
Swivel
SIZE
A
FSCM NO.
DWG NO.
SCALE
SHEET


1,5
R1
2,0
4,0
16,0
QTY REQD
FSCM NO
PART OR IDENTIFYING NO.
NOMENCLATURE OR DESCRIPTION
MATERIAL SPECIFICATION
ITEM NO.
PARTS LIST
UNLESS OTHERWISE SPECIFIED
TOLERANCES ARE:
FRACTIONS DECIMALS ANGLES
± .XX= ±
.XXX=
DO NOT SCALE DRAWING
CONTRACT NO.
APPROVALS DATE
TREATMENT
FINISH
DRAWN
Jeremy Mayer
352-392-0814
SIMILAR TO
MAT. WT
DALE WT
Material: Aluminum
TITLE
Swivel_Pin
SIZE
A4
FSCM NO.
DWG NO.
SCALE
SHEET


Material: 1/4" PolyCarb
Chassis
Jeremy Mayer
352-392-0814
A4
PARTS LIST
Ø2.0
Press Fit
2mm Pin
6 Places
59.05
33.65
8.25
57.15
31.82
19.12
133.35
101.60
76.20
50.80
25.40
Ø8.20
Ø2.50
Drill and Tap for M3
4 places
Ø12.70
6 places
12.70
38.10
63.50
82.55
12.90
25.40
6.35


Material: 1/4" PolyCarb
Frame
Jeremy Mayer
352-392-0814
A4
PARTS LIST
Slip Fit for 3mm Pin
6 places
R1.18
9 places
R5.0
2 places
Drill and Tap for M4
18 places
133,35
120,45
104,78
93,10
82,55
67,70
42,30
28,58
12,90
4,50
4,00
15,88
24,25
41,28
49,65
66,68
75,05


R5.0
2 places
R1.18
9 places
82,55
133,35
93,10
67,70
42,30
12,90
24,25
41,28
49,65
75,05
120,45
PARTS LIST
Material: 1/4" PolyCarb
Frame2
Jeremy Mayer
352-392-0814
A4

LIST OF REFERENCES

Ber79 Berlin, E.P., Three-Dimensional Display. U.S. Patent No. 4,160,973. July 10, 1979.

Fir72 Firth, I.M., Holography and Computer Generated Holograms. London, England. Mills and Boon Limited, 1972.

Jur95 Jurczyk, Ralf, *Design of an Autostereoscopic Display Unit*. Doctoral dissertation, University of Florida, 1995.

Kam93 Kameyama, Ken-Ichi, "A Direct 3-D Shape Modeling System." *1993 IEEE Annual Virtual Reality International Symposium*. V.93ch3336-5, 1993, pp. 519-524.

Lee00 Lee, Jahoon, Investigation of the Quality Indices of In-Parallel Platform Manipulators and Development of Web Based Analysis Tool. Doctoral dissertation, University of Florida, 2000.

Lip90 Lipton, Lenny, "Stereo vision on your workstation." *Mechanical Engineering*, V. 112, March 1990, pp. 36-39.

Mag72 Maguire, E.T., Computer-Controlled Three Dimensional Display. U.S. Patent No. 3,636,551. Jan 18, 1972.

Mor90 Morton, R.R.A., Three-Dimensional Display System. U.S. Patent No. 4,922,336. May 1, 1990.

O'B87 O'Brien, T., Gimballed Three-Dimensional Display System. U.S. Patent No. 4,636,081. Jan 27, 1987.

Oko76 Okoshi, Takanori, *Three-Dimensional Imaging Techniques*. Academic Press, New York, 1976.

Oko80 Okoshi, T., "Three-Dimensional Displays." *Proceedings of the IEEE*. V.68 No. 5, May 1980, pp. 548-564.

Sol91 Solomon, D.J., Three-Dimensional Volumetric Display System. U.S. Patent No. 4,983,031. January 8, 1991.

Wat91 Watt, Alan, "Fundamentals of Three-Dimensional Computer Graphics." *Computers and Geosciences*. V.17 No. 4, 1991.

War93 Waram, Tom, *Actuator Design Using Shape Memory Alloys*. T.C. Waram, Hamilton, Ontario Canada, 1993.

Yer00 Yerelan, S., Emery, H., The 8051 Cookbook for Assembly and C. Gainesville, Florida. Rigel Press, 2000.

BIOGRAPHICAL SKETCH

Jeremy Mayer was born in Cleveland, Ohio on May 1, 1971. His family moved to Ann Arbor, Michigan in August of the same year. After growing up in Ann Arbor, he attended Michigan State University where he received his Bachelor of Science degree in Mechanical Engineering. After graduating in 1994, he took a job with Hewlett-Packard Company in San Diego, California. He spent four years working in San Diego and one year working at the HP Barcelona Spain Division. In 1999 he decided to return to academics in order to pursue a Master of Science degree in Mechanical Engineering with a minor in Electrical and Computer Engineering at the University of Florida.