UNIVERSITY OF Cincinnati

Final Project Report – Mini 2-Axis CNC Plotter

Presented by:
Aswin Balasubramaniam

Course Name/Number: Advanced Microsystems (EEECE 6038C)
Class section: 001
Due Date: April 27th, 2018

For instructor use

Students Last Name		
Report		
Conclusion		
Total		

Table of Contents

Introduction ……... 3
Objective ……... 3
Solution & Method ……. 3
Resources …….. 4
Schedule: Gantt Chart……..…….. 4
Hardware Implementation ……...……...…… 5
Software Implementation …….. 7
Preliminary Results: Midterm Demo ……..…….. 12
Discussion on Preliminary Results ……..……. 13
Final Results: Final Demo……...….... 13
Discussions on Final Results ……. 14
Program ……. 15
Test Cases …….. 19
Conclusion ……..…… 20
References ……. 21
Notes ……. 22

Introduction:

Printing technology has been around since the year 220AD, and the earliest known form of printing was the woodblock printing. Since then, this field has been developing to further include movable parts and to finally develop the first printing press invented by Gutenberg in the 15th century. With the advent of silicon and microprocessors, there has been a shift from mechanical printer to electrical printers. The development in the field of electronics and embedded systems made way to the production of various kinds types of printing devices whose functions are only restricted by a person's imagination.

This project aimed to recreate a printing technique called CNC plotting that was quite popular in the early 19th century. During the initial phase of the project, the idea was to create a CNC plotter that drew image or texts on a 2D surface by interpreting the G-Code of the image or a text. A servo would be included that mimics the lifting of the pen to avoid drawing unnecessary lines. Towards the end of the semester, the project could successfully draw on a 2D surface and interpret G-Code files but could not integrate a servo to lift the pen, which would have avoided drawing unnecessary lines. The various sources of information that were referenced for this project are provided under the reference section of this report. This report provides information about the various details of the project and the outcome that was achieved towards the end of the semester.

Objective:

The objective of this project is to design and build a mini 2-axis CNC plotter that would help a user to automate plotting of neat hand-written texts or drawings using the Mikromedia PIC24 development board.

Solution & Method:

The mini 2-axis CNC plotter for the final project was built using parts found in old CD drivers, and the Mikromedia PIC24 development board was used as the controlling and processing unit. The base and stand required to make the CNC plotter were made from the case of the CD drives. The stepper motors found inside the CD dive were used to move the pen around on the X-Y axis. During the initial phase of the project, a small servo motor was used to move the pen up and down. The implementation of the servo had to be scrapped towards the end of the semester due to complications faced with the hardware, which will be discussed later in the lab report.

The objective of this project was to create a system that produces neat hand-written notes or drawings using the Mikromedia development board as the interface. The Mikromedia PIC24EP development board has a built-in TFT LCD touchscreen and a micro SD card reader that is utilized in this project to create an interactive GUI. A user can store the G-Code of a drawing on the SD card which the program reads and displays on the LCD screen. The user can then select the drawing and the plotter would then draw then the image or text by interpreting their G-Code. Using the graphical user interface designed for this project a user can control the plotting

machine, select the required GCode of a drawing, send the plotter to the home position, run a few test cases, and view the real-time position of the plotter.

Interpreting the G-Code to plot the image or text is the backbone of this project. A G-Code has many name variants but is the common name for the most widely used numerical control programming language. It is widely used in computer-aided manufacturing to control automated machine tools. The G-Code follows a set of standards to convert text or images drawn using vector graphics software like Inkscape, used for this project. The coordinate information present in the G-Code file is used to step motors in various directions to plot the desired message.

Resources:

The following are the resources utilized to complete the project.

- Software:
  - MPLABx IDE
  - XC16 Design Tools
  - Inkscape Software (or any other software that can convert drawings to G-Code)
- Hardware:
  - Stepper Motors (x2): Salvaged from old CD drives
  - Easy Driver Stepper Motor Drivers by Schmalz Haus (x2): Purchased from SparkFun
  - Mikromedia PIC24 Development Board (x1): Borrowed from the lab. This board has the HR4 8514S G5/3 version of the IL9342 LCD driver.
  - Keithley DC Power Supply: Limited to 9V Voltage and 1.5 Ampere Current
  - Pickit Debugger (x1)
  - Various mounts and screws: Salvaged from old CD drives

Schedule – Gantt Chart:

For the initial phase of the project leading to the mid-term demo, the Gantt chart shown in Figure 1 was created. Due to the complications faced during the implementation of hardware, the deadline for troubleshooting the device with version 1 of the code had to be pushed further down the month, which lead to delaying the deadline for version 2 of the code.

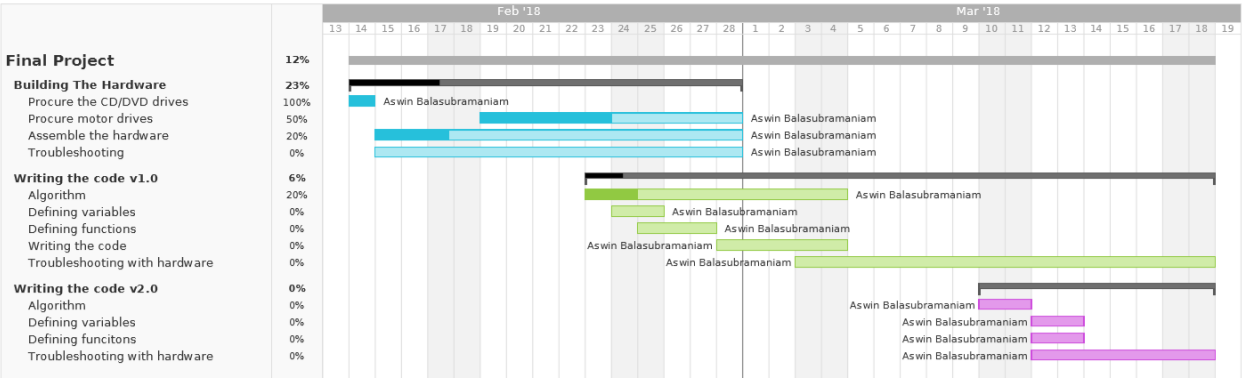

Figure 1: Gantt Chart Created During the Preliminary Phase of the Project (Version 1)

After the initial hardware and software troubleshooting was completed modifications were made to the Gantt chart to reflect realistic deadlines as shown in Figure 2.

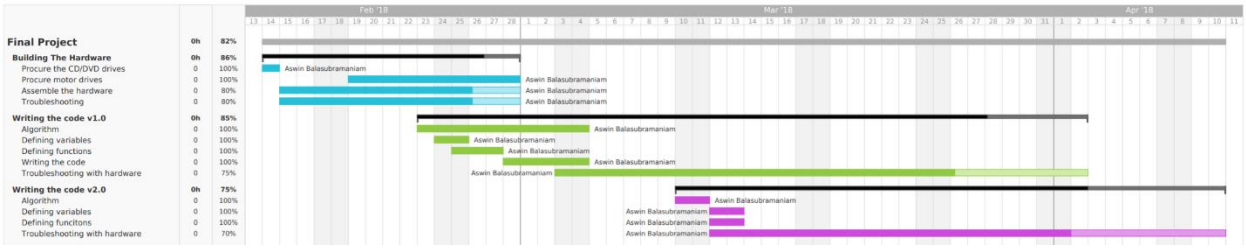

Figure 2: Updated Gantt Chart Created for the Midterm Demo of the Project (Version 2)

Despite sticking to the updated Gantt chart the live working prototype of the project could not be presented during the final demo week due to complications faced with the hardware on the previous day of the presentation, which will be further discussed under the discussion section of the prototype.

Hardware Implementation:

The schematic seen in Figure 3 shows the top-level hardware system design. The schematic shows all the hardware resources that were used to build this project.

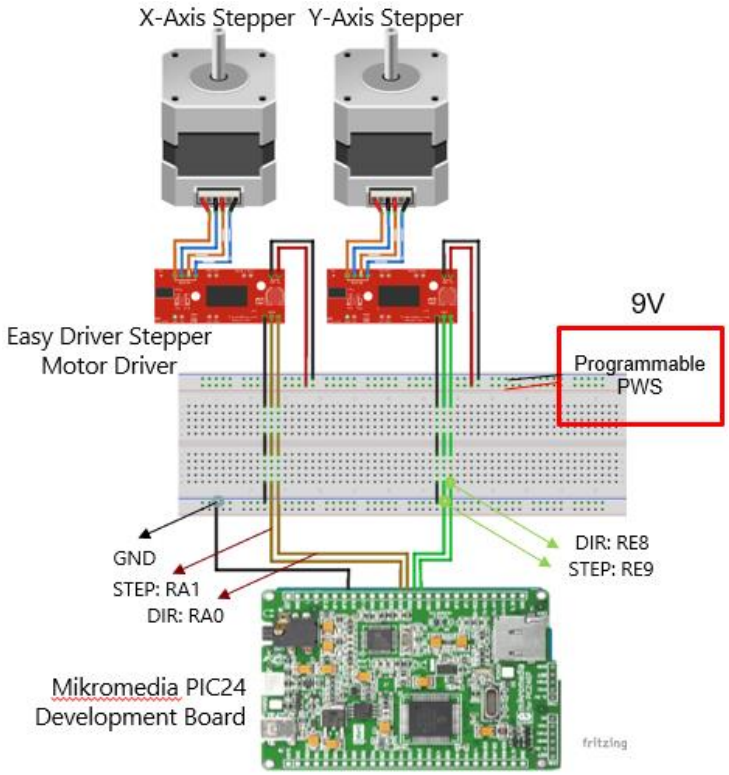

Figure 3: Top Level Hardware System Design

The stepper motors were procured from old CD drives for free. These motors are bipolar DC stepper motors that draw a maximum of 500mAmp of current each with a constant load, that is approximately 250mA/phase. These values were recorded when the motors were initially tested during the troubleshooting phase.

The easy driver stepper motor drivers were purchased on the Sparkfun website for $15 each. This device was designed by an individual called Schmal Zhaus. The developer's website was referenced to understand the working of this device. The motor driver was the key component required to complete the project. The Allegro 3967 driver chip is the main component on the EasyDrivers that drives the bipolar stepper motor. It has a true H-bridge design internally and sends current both way through each of the two coils. The EasyDriver can drive up to about 750mA/phase of a bipolar stepper motor. It defaults to 8 step micro stepping mode. The Allegro A3967 driver chip is a chopper micro-stepping driver, that has a variable max current from 150mA/phase to 750mA/phase. It can handle a maximum motor drive voltage of 30V and has an onboard 5V regulator. Figure 4 shows a labeled EasyDriver board. The following pins are used in this project:

- GND: There are three GND pins on the Easy Driver that are all connected together inside the board. The negative side of the power supply is connected to this pin.
- M+: This is the power input for the EasyDriver and is connected to the positive end of the power supply. The board can handle 6V to 30V of potential difference, and 2A (or more) of current.
- A and B: (four pins) These are the four output pins that are connected to the motors. A and B are the two coils of the motor and swapping the two wires reverses the direction of the motor. To get the connection proper multiple trials had to be made to determine which two wires are part of one motor.
- STEP: This is an input pin, where the signal is sent by the Mikromedia board. This has to be a 0V to 5V digital signal. Each rising edge of this signal will cause one step of motion.
- DIR (Direction): This is an input pin where the signal is sent by the Mikromedia board. This has to be a 0V to 5V digital signal. The level of this signal is sampled at each rising edge of STEP to determine which direction to take the step.

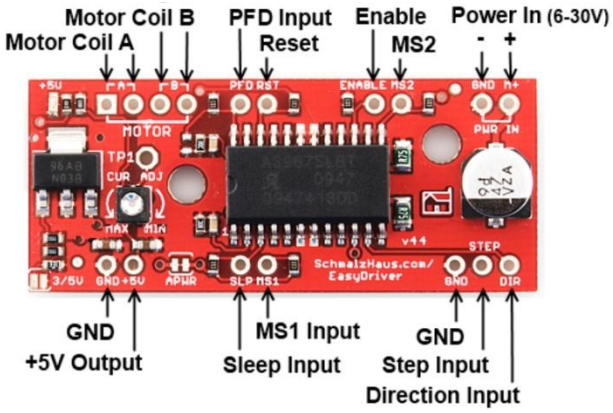

Figure 4: EasyDriver Motor Driver Board Pin Out Diagram

The procured components were then connected as shown in the schematic given in Figure 3. The Keithley Programmable Power Supply, available in the labs, was used to power up the entire setup.

The PIC24EP development board was used to generate the digital PWM signal to step the motor and set the direction signal. The digital I/O pins were used for this. Pins RA1 and RA0 were assigned to the x-axis motor, and pins RE8 and RE9 were assigned to the Y-axis motor. A digital PWM signal of 3.3V with 500Hz frequency and 50% duty cycle was programmed to step the motors. The RA1 and RE9 pins were used as the step output pins for both the motors, RA1 for x-axis motor and RE9 for the Y-axis motor. The RA0 and RA8 pins were used as the direction output pins for both the motors, RA0 for x-axis motor and RA8 for y-axis motor.

The PIC24EP board has an assigned set of pins that generate PWM signals. These were not used, as the pins could not be set up correctly on MPLAB X to generate the necessary current to drive the signal sent to the EasyDriver.

Software Implementation:

The program for the final project was written using the MPLAB X IDE. The XC16 compiler syntax was used to write the code. Figure 5 shows the top-level program logic flow.

First, the user will need to have the image or text ready to be plotted.

Second, the user will have to use InkScape, or a similar software to convert the text or image to a G-Code.

Third, the user would select the G-Code on the GUI, and the program interprets the G-Code to extract the coordinates for plotting.

Finally, the user will have their plotted-out text or image on the paper.

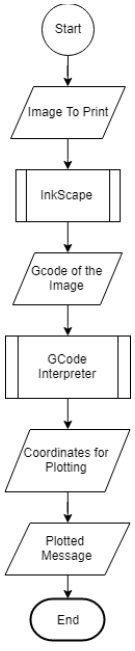

Figure 5: Top Level Program Logic Flow

GUI Logic Flow

Figure 6, shows the flow diagram of the GUI designed for this lab.

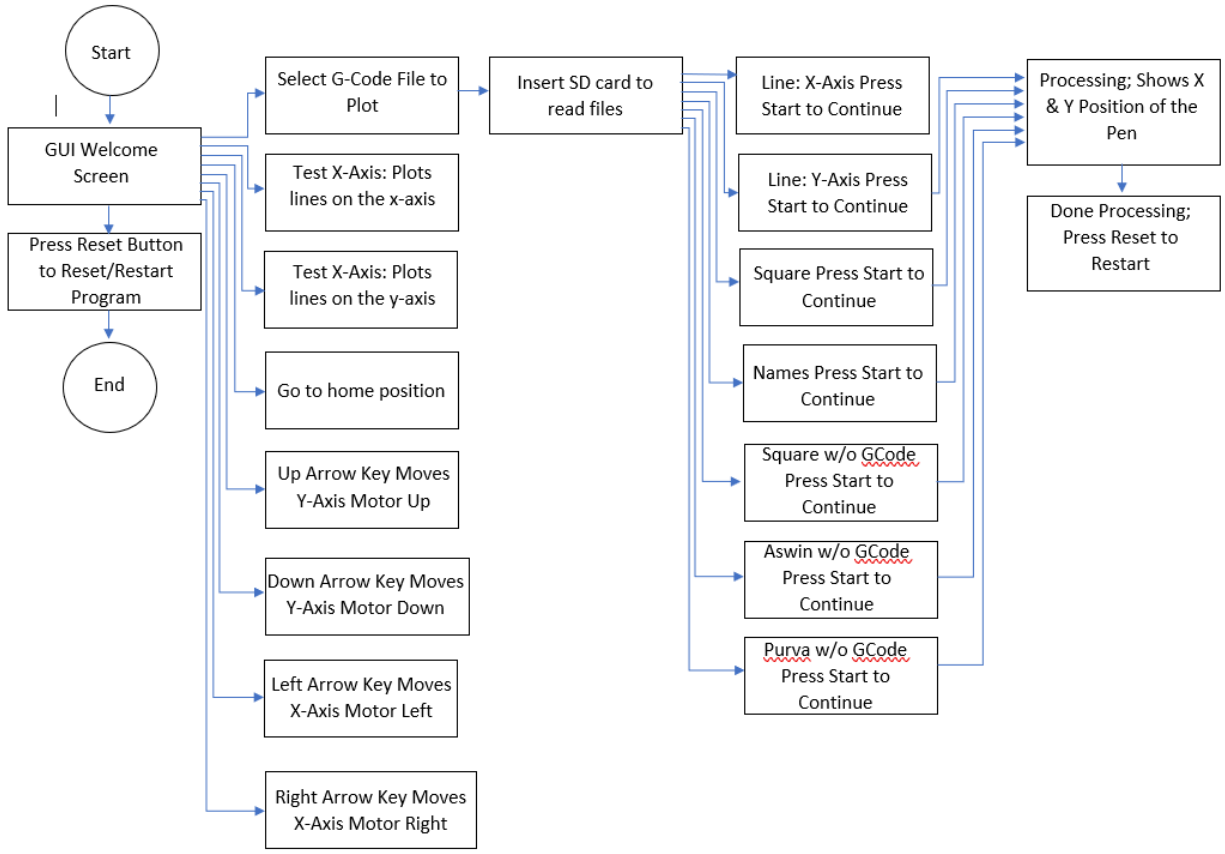

Figure 6: GUI Program Logic Flow Diagram

The first screen of the GUI the user sees is the welcome screen with all the options available to the user. Figure 7 shows the screenshot of the GUI's welcome screen. The user has the option to select the G-Code from the SD card inside the Mikromedia board or run sample test case of drawing lines on the X-axis and the Y-axis. The user can also interact with the system using the arrow keys. The left and right arrow key control the motor on the x-axis, and the up and down arrow key controls the motor on the y-axis. Finally, the user has the option to go back to the home position with a push of a button.

When using the system take care to switch on the power supply to activate the motors, before interacting with the system using the arrow keys. As the program does not take into consideration if the motor is alive pressing the arrow keys updates the value of the global variable Xpos and Ypos. This helps the program to check for end limits when the arrow keys are pressed that and also helps the code to know where the pen is when a user runs a G-code. The welcome screen also shows a user the real-time x and y position of the pen, as shown in Figure 8.

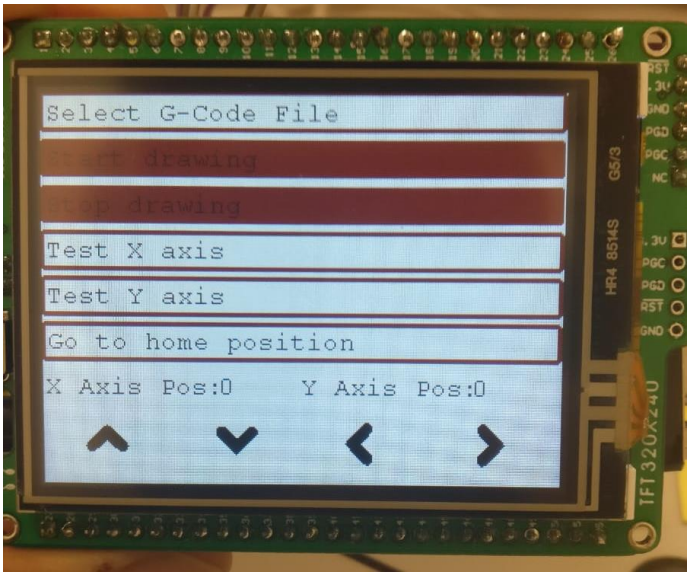

Figure 7: GUI Welcome Screen

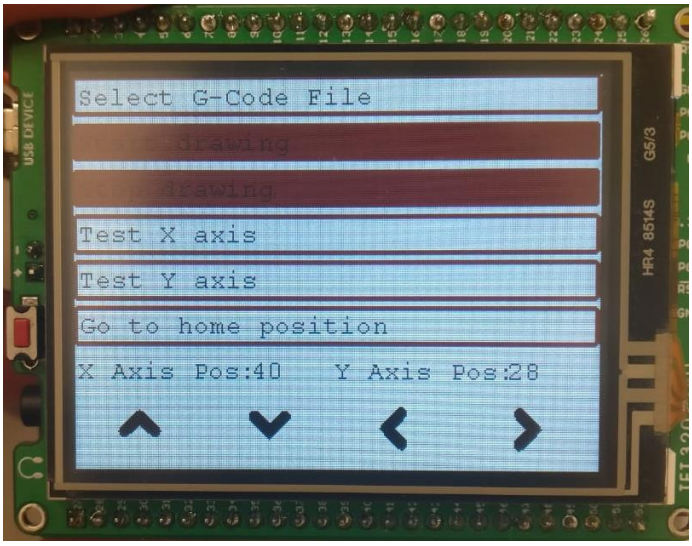

Figure 8: GUI Welcome Screen Showing X and Y Positions

Once the user selects the button "Select the G-Code File" a new page shows up with the list of .txt files read from the SD card. Figure 9, shows the list of .txt files available to the user. If the SD card is not inserted the program shows a prompt for the user to insert an SD card. The user always has the option to hit the reset button to restart the program. Once the user selects the desired file to be plotted the user will be shown a new welcome screen with a start and stop button. The stop button is still inactive in the program, but when the user presses the start button the plotting starts. During plotting the user sees the processing message on the LCD screen with the real-time X & Y position of the pen. The video attached to this lab report walks through all the above-mentioned functions of the GUI.

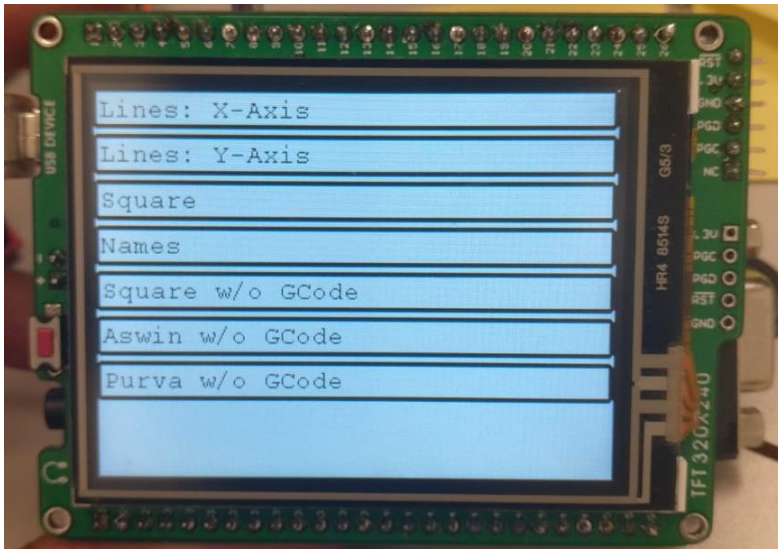

Figure 9: List of Available G-Code in .txt format

Interpreting the G-Code:

One of the objectives of this project is to interpret the G-Code of an image or text and plot the corresponding image or text using the coordinates from the G-Code. The G-Code follows a standard format which helps to decipher it easily using C programming. The list given below shows the various syntaxes used in a G-Code, that can be interpreted easily. Figure 10 shows a snippet of a G-Code created for this project.

G-Code syntax:

M300 S30: Z-axis moves up

M300 S50: Z-axis moves down

M114: Reports Position of Motors

M18: Stops all motors

G1: Indicates starting X and Y coordinates

G4: Indicates wait time (in ms)

G92: Indicates absolute start position

```
G21 (metric ftw)
G90 (absolute mode)
G92 X0.00 Y0.00 Z0.00 (you are here)

M300 S30 (pen down)
G4 P150 (wait 150ms)
M300 S50 (pen up)
G4 P150 (wait 150ms)
M18 (disengage drives)
M01 (Was registration test successful?)
M17 (engage drives if YES, and continue)

(Polyline consisting of 32 segments.)
G1 X30.06 Y8.86 F3500.00
M300 S30.00 (pen down)
G4 P150 (wait 150ms)
G1 X30.76 Y8.86 F3500.00
G1 X30.83 Y8.81 F3500.00
G1 X30.46 Y8.78 F3500.00
G1 X30.06 Y8.86 F3500.00
G1 X30.06 Y8.86 F3500.00
M300 S50.00 (pen up)
```

Figure 10: Snippet of G-Code Created for this Project

Figure 11 shows the logic flow diagram created to interpret the G-Codes created for this project. The algorithm created uses the standard syntax of G-Code as a reference.

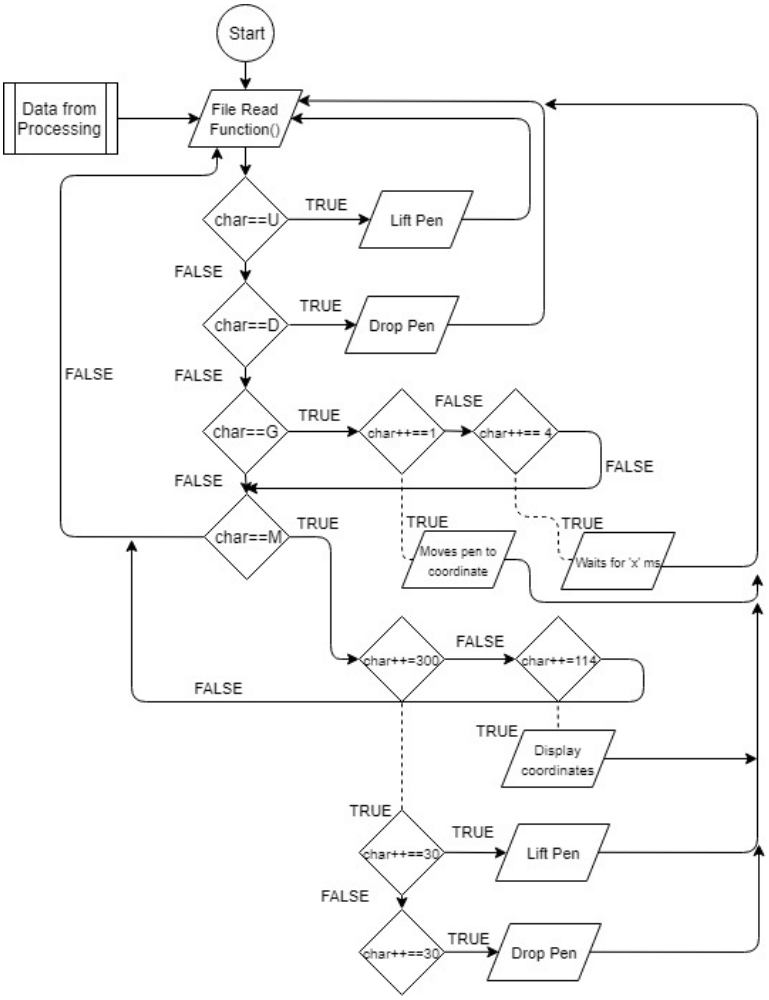

Figure 11: G-Code Interpreter Flow Diagram

Preliminary Results – Midterm Demo:

The first prototype for the midterm demo was created using foam boards as the base and as the support structures. The first prototype broke due to poor form and strength of the foam board. Figure 12 shows the broken first prototype. The foam base cracked when a high-frequency step signal was supplied to the motors. The momentum generated by the speed of motors had enough force to crack the foam base.

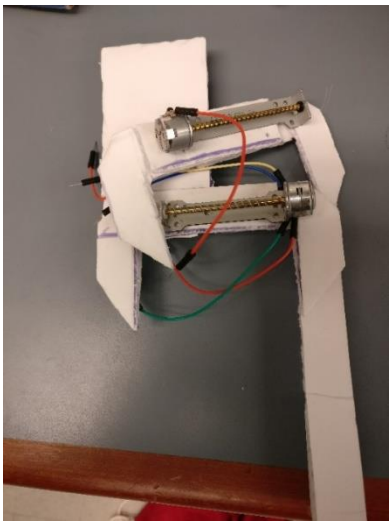

Figure 12: Prototype 1 Created for the Final Project

The lessons learned from building the first prototype was carried on to build the second prototype. The second prototype was built using the CD drive case/frame. The case/frame was made out of aluminum which is sturdier than foam. When the motors were tested on this frame the step signal was set to a frequency of 500Hz to avoid any other mishaps, and this is also the default frequency set in the program. This prototype incorporated the use of a servo to pull the pen up and push it down. Figure 13 shows the images of the second prototype. The construction of the arm to support the servo was very poor and which led to the plotter drawing curvy lines as opposed to straight lines. This prototype did not last long as the motor driver driving the motor drawing lines along the x-axis broke.

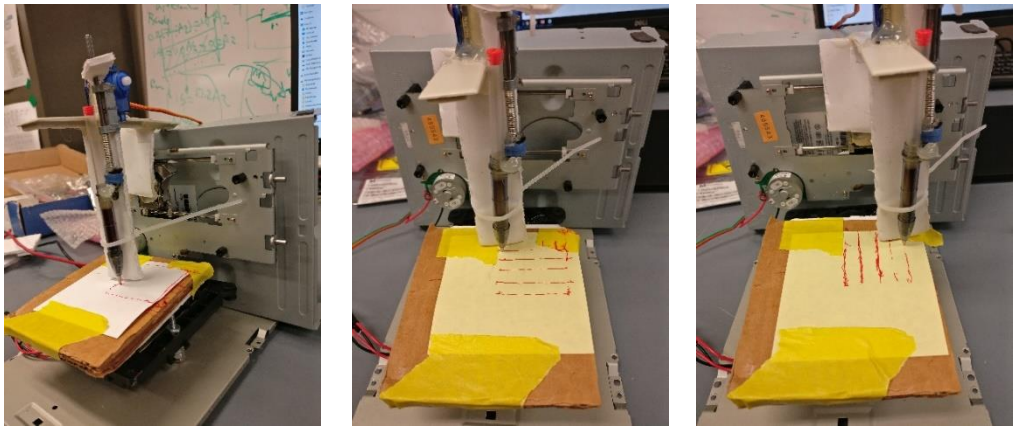

Figure 13: Prototype 2 Created for the Final Project

Discussions on Preliminary Results:

The failure of the first prototype can be owed to the poor judgment of using foam boards as a base and support structure. The foam could not handle the motors speed when it was stepped using the signal of 1kHz frequency.

The second prototype was also constructed poorly. This could be seen in the curvy lines that were created when the program was run, and the failure of the motor driver driving the motors on the x-axis. The reason for failure was hypothesized to be due to the load created by the arm supporting the servo motor. When the power supply was monitored, the current drawn by both the motors combined went up to 1.5 Amperes for a constant period. The draw of high current could have lead to the breaking of the Allegro A3967 driver chip. This hypothesis could not be confirmed, without destroying another EasyDriver board.

Final Results – Final Demo:

The third prototype was the final prototype that could be designed given the schedule for this project. This was the best of the three prototypes designed over the course of the semester. Extra caution was taken to not repeat the mistakes made on the earlier versions of the plotters. Figure 14 shows the entire system that includes the GUI and the plotter in separate images.

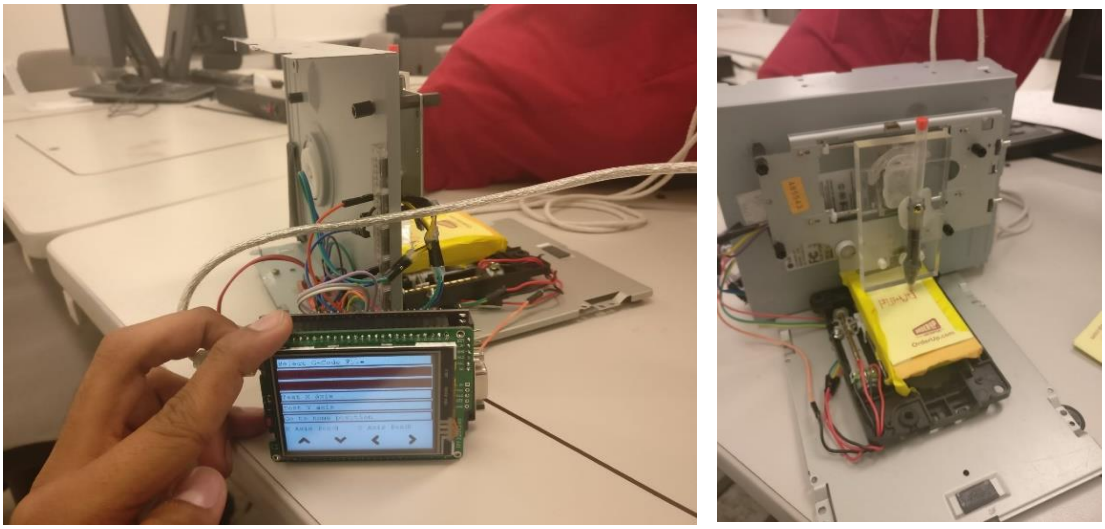

Figure 13: Prototype 2 Created for the Final Project

This version of the prototype did not include the servo that would have lifted the pen up or pushed the pen down. This was done to not repeat the mistake of applying extra load on the motor which could have broken another EasyDriver. The lack of this function created texts and images that had unnecessary lines either between letters or between parallel lines. Figure 14 shows the various texts that were plotted on the plotter and a square shape. Due to the lack of the function to lift the pen, one can see the plotter draws unnecessary lines between letters.

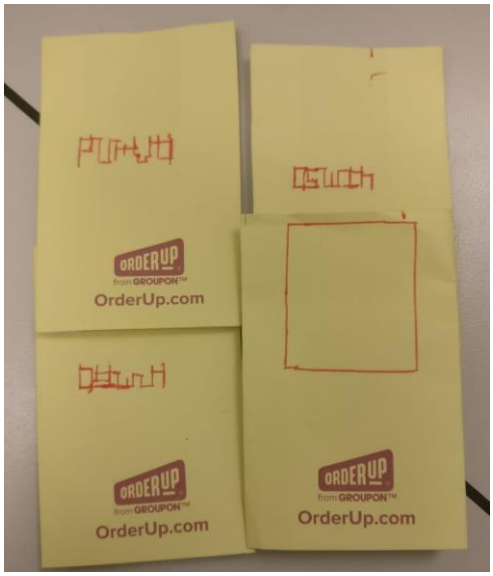

Figure 14: Sample Texts and Shape Created by the Plotter

A video of the plotter drawing parallel lines along the x-axis showing the entire system in action is attached to this lab report. In the vide one will be able to see the unnecessary lines drawn between the parallel lines, creating a drawing of zigzag lines along the x-axis, rather than parallel lines. Unfortunately, both the motor drivers of this prototype also broke before the final live demo, due to reasons still unknown.

Discussions on Final Results:

As mentioned above, extra care was taken to prevent the repetition of the mistakes conducted before. To check if the boards were actually broken the developer of the board has provided users of the EasyDriver few steps to troubleshoot the hardware to check if the board is damaged. The steps are as follows:

- With everything disconnected from the ED, measure the resistance from each of the four motor output pins to GND and M+. All 8 of these measurements should be over 1MOhm.
- Again, with everything disconnected, measure the resistance between the four motor output pins themselves. And again, all measurements should be over 1MOhm.
- Again, with everything disconnected, measure the resistance from STEP and DIR to GND and +5V pins. Again, all of these should read greater than 1MOhm.
- Now connect just GND and M+. Do not connect anything else to the ED. The LED should come on and stay on. Measure the voltage at +5V to GND. It should be right around 5V.
- Now measure the voltage at each of the four motor output pins to GND. Two of them should read at about the same voltage that is at M+, and the other two should be almost zero or approximately 0.018V.

The steps were followed to conclude that the boards were indeed broken. This was realized when the motor output voltage did not read 12 V, but rather smaller values in the range of 2V. The

reason for this is still unknown. During the run when the board stopped functioning the current drawn by both the motors were less than 1 Ampere. The Keithley power supply used to power the boards are of high quality and has given high-quality outputs every time and can be ruled out as a possible cause of the error. The motors that were used for this prototype has been the same since the first prototype. This rules out all the possible causes of errors that have been faced so far. The only other possible cause of the error is the shorting of both the boards, at the same time. But this hypothesis could not be confirmed due to the lack of information.

During the final demo presentation, the reviewers and I discussed the possible sources of errors that could break the board. One of the potential error that is not mentioned above is the possibility of another source drawing current unknown to the user.

The small drawback with the GCode was that every line small or big was considered to be a square when it was converted to the GCode, causing the system to take a long time to finish the plotting. To avoid this for the final demo manual coordinates were programmed and sample files w/o GCode were created and saved on the SD card. These files can be seen in Figure 9, the list of files available on the SD card.

Program:

Majority of the supporting code used for this lab was taken from the MLA resource folder, downloaded from Microchip's website. The program was written in the C language. The images of the arrows shown on the welcome screen were converted to a source and header file using the graphics resource converter application and the names of the images were referenced to display them on the screen. The main source code was written in parts to test the various resources of the Mikromedia board that was used. The first part that was tested was the touchscreen, and the code for this was written referencing the GTSU textbook by Lucio Di Jasio. The second part was opening and reading files on an SD card, and this was also written referencing the GTSU textbook. The third and final part was the G-Code interpreter, and this was written by myself referencing the syntax of G-Code. The code written for the final project is attached to the lab report. The main source code is well commented and would help a reader to easily follow the structure of the code.

Header Files: The header files for all the source files used for this lab were not directly added to the project. The folders in which they are present were referenced using the project properties. If referenced correctly the IDE does not display any errors.

From MLA resource files:

1. **HardwareProfile.h**
2. **GraphicsConfig.h**
3. **TouchScreenResistive.h**

These are the header files that were directly added to the project. There are more header files referenced within codes written by Microchip. Apart from these files the "embeddeShield.h" file created for the final project is shown below.

```c
#include <p24EP512GU810.h>
#include "embeddedShield.h"

void Init(void)
{
    RCONbits.SWDTEN=0;                  // Disable Watch Dog Timer
                                        // Configure Oscillator to operate the device at 40Mhz
                                        // Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
                                        // Fosc= 8M*40/(2*2)=80Mhz for 8M input clock
    PLLFBD=38;                          // M=40
    CLKDIVbits.PLLPOST=0;               // N1=2
    CLKDIVbits.PLLPRE=0;                // N2=2
    OSCTUN=0;                           // Tune FRC oscillator, if FRC is used
                                        // Clock switching to incorporate PLL
    __builtin_write_OSCCONH(0x03);      // Initiate Clock Switch to Primary
                                        // Oscillator with PLL (NOSC=0b011)
    __builtin_write_OSCCONL(0x01);      // Start clock switching
    while (OSCCONbits.COSC != 0b011); // Wait for Clock switch to occur
    while(OSCCONbits.LOCK!=1) {};       // Wait for PLL to lock
}


void initLEDs(void){
    // LED Specific Pins
    _TRISF0 = 0x0;          // set LED pins to outputs
    _TRISF1 = 0x0;
    _TRISF2 = 0x0;
    _TRISF4 = 0x0;
    _TRISF5 = 0x0;
    _TRISD8 = 0x0;
    _TRISD9 = 0x0;
    _TRISD10 = 0x0;
    _TRISD11 = 0x0;
    _TRISD12 = 0x0;

    LED0 = 0;               // default LED's to off
    LED1 = 0;
    LED2 = 0;
    LED3 = 0;
    LED4 = 0;
    LED5 = 0;
    LED6 = 0;
    LED7 = 0;
    LED8 = 0;
    LED9 = 0;
}

void setLEDBAR(unsigned short pattern){
    if ((pattern & 0x001) == 0x001)
        LED0 = 1; else LED0 = 0;
    if ((pattern & 0x002) == 0x002)
        LED1 = 1; else LED1 = 0;
    if ((pattern & 0x004) == 0x004)
        LED2 = 1; else LED2 = 0;
    if ((pattern & 0x008) == 0x008)
        LED3 = 1; else LED3 = 0;
    if ((pattern & 0x010) == 0x010)
        LED4 = 1; else LED4 = 0;
    if ((pattern & 0x020) == 0x020)
        LED5 = 1; else LED5 = 0;
    if ((pattern & 0x040) == 0x040)
        LED6 = 1; else LED6 = 0;
    if ((pattern & 0x080) == 0x080)
        LED7 = 1; else LED7 = 0;
    if ((pattern & 0x100) == 0x100)
        LED8 = 1; else LED8 = 0;
    if ((pattern & 0x200) == 0x200)
        LED9 = 1; else LED9 = 0;
}
```

```
void initSwitches(void){
    // Switch Specific Pins
    ANSELD = 0x00;
    SW1 = 0;
    SW2 = 0;
    _TRISD7 = 0x1;                      // set switch pin to inputs
    _TRISD13 = 0x1;
}

// initialize the ADC, select Analog input pins
void initADC( int amask)
{
    ANSELA = amask;    // select analog input pins
    AD1CON1 = 0x0000;   // auto convert after end of sampling
    AD1CSSL = 0;        // no scanning required
    AD1CON3 = 0x1F3F;   // max sample time = 31Tad, Tad = 2 x Tcy
    AD1CON2 = 0;        // use MUXA, AVss and AVdd are used as Vref+/-
    AD1CON1bits.ADON = 1; // turn on the ADC

    _TRISB9 = 1;        // make RB9 an input

    // Explorer 16 Development Board Errata (work around 2)
    // RB15 should always be a digital output
    _LATB15 = 0;
    _TRISB15 = 0;
} // InitADC

// sample/convert one analog input
int readADC( int ch)
{
    AD1CHS0  = ch;              // select analog input channel
    AD1CON1bits.SAMP = 1;       // start sampling, automatic conversion will follow
    Delayns(80);
    AD1CON1bits.SAMP = 0;
    while (!AD1CON1bits.DONE);   // wait to complete the conversion
    AD1CON1bits.DONE = 0;
    return ADC1BUF0;            // read the conversion result
} // readADC

void Delay( int t)              //0.263ms Delay function
{
    T1CON = 0x8000;                     // Enable tmr1, Tcy, 1:1
    while (t>0)                         // Wait for t (msec)
    {
        TMR1 = 0;
        while ( TMR1 < (FCY/3800));   // wait 0.263ms
        t=t-1;
    }
}

void button_start( void)        //Aids in receiving prompts from user
{
    int last =0;
    int VALUE =0;
    int current;
    while (1)
    {
        current = SW1;
        if (last!= current)
        {
            Delay(5);
            current = SW1;
        }
        if (last == 0 && current == 1)
        {
            VALUE = VALUE +1;
        }
        last = current;
        if (VALUE%2 == 0)
```

```
139             if (VALUE%2 == 0)
140             {
141                 return 0;
142             }
143         }
144     }
145     void error_msg( void)           //Function to display error message
146     {
147         LCDClear();
148         OutTextXY(3,15,"ERROR! Finger was lifted off");
149         OutTextXY(3,30,"before 3 seconds!");
150         OutTextXY(3,45,"Press Reset to start again!");
151         while(1);
152     }
153
154     void Delayns( int t)            //lns Delay function
155     {
156         T3CON = 0x8000;                 // Enable tmr1, Tcy, 1:1
157         while (t>0)                     // Wait for t (nsec)
158         {
159             TMR3 = 0;
160             while ( TMR3 < 1);   // wait lns
161             t=t-1;
162         }
163     }
```

Figure 15: Header File "embeddedShield.h" Used for the Final Project

Source Files

From MLA resource files:

1. **IL9341.c**
2. **LCDTerminal.c**
3. **LCDTerminalFont.c**
4. **Primitive.c**
5. **TimeDelay.c**
6. **TouchScreen.c**
7. **TouchGrid.c**
8. **uMedia.c**
9. **LCDmenu.c**
10. **FSIO.c**
11. **SD-SPI.c**
12. **TimeDelay.c**
13. **TouchScreenResistive.c**

The TouchScreen.c source file generalizes the access to the touchscreen interface providing generic initialization, calibration, non-volatile storage, and graphics object message generation.

The TouchScreenResistive.c source file includes the initialization function and the detecting position function.

Edits were made to the GraphicsConfig.h file to include the touchscreen lines. These lines are already present in the header file and may be commented. To include the lines uncomment them. The instructions given in the textbook GSTU was followed to successfully complete this.

Edits were made to HardwareProfile.h file as well. Certain ADC configuration bits in the header file were initialized for the PIC24F board. This was later edited to match the syntax used for the PIC24EP board.

The uMedia.c provides the uMBinit() function to initialize pins and ports.

The IL9341.c file provides the necessary lines of code for the display controller driver.

The Primitive.c file initializes the primitive layer of the graphics library.

The TimeDelay.c file has a few basic timing functions used in the driver.

The FSIO.c file has the necessary lines of code for the FAT file system support library.

The SD-SPI.c file has the necessary lines of code for the low-level SPI interface layer of the file system.

The Arrow.c and Arrow.h resource file created using the graphics resource converter consists of the arrow keys seen on the GUI.

The main source file "main.c" written for the final project consists of 1268 lines of code. To avoid creating a large report the screenshots of the main source code is not shown here. The code written for the lab is attached to the folder with this report.

Test Cases:

The test cases shown in the table in Figure 16 were conducted to test the functionality of the entire system. Since two working prototypes were created the information for both the prototypes are presented in the table. The table shows the test case, the description of the test case and the update of the testing.

Test Case	Description	Update
Stepper Motor	Test the working of stepper motor sending a high and low signal	P2: Done P3: Done
Motor Drives	Test the working of stepper motor with drivers, by sending a sample PWM signals	P2: Done P3: Done
Straight Line: X-axis	Draw straight lines along x-axis	P2: Done P3: Done
Straight Line: Y-axis	Draw straight lines along y-axis	P2: Done P3: Done
Square	Draw a square	P2: Done P3: Done

Figure 16: Table of Test Cases Run for the Final Project

Even though both the prototypes stopped working due to the misfortunate incident of the motor drivers breaking, the prototypes passed the test cases when it was functional.

Conclusion:

The main aim of the final project was to use the knowledge gained from the Advanced Microsystem Design course and apply it to build a system that uses the Mikromedia development board. The LCD touchscreen present on the development board, the SD card reader, and the PIC24EP microcontroller were used to build an interactive mini 2-axis CNC plotter. The main aim was to use the plotter to create neat hand-written texts and drawings using the Mikromedia development board and present the live demonstration during the finals week. This was not completely met due to the misfortunate incident of the motor drivers breaking before the final demo presentation. The reason for the motor driver's failure is still to be determined but the tests suggested by the developer of the board confirms the damage of the board. Prior to final demo presentation, the system was successful in implementing the hardware and software together.

The mini 2-axis CNC plotter system consisted of 2 stepper motors controlled by 2 EasyDriver motor driver break out the board. The necessary signal for to the motor driver was sent using the Mikromedia PIC24 development board. A GUI was also designed using the LCD touchscreen on the development board which allowed the user to interact with the system and select the necessary file to be plotted. The images of arrow keys on the GUI were used as buttons to control the plotter's x & y-axis. The layout of the GUI designed is shown in Figures 7 to 9. The main source code that controls the motors and the design of the GUI was written in C. The various header files and source files used to access the various capabilities of the Mikromedia development board was downloaded from the MLA resource files.

There are three videos attached to the lab report. The first one shoes the initial progress with prototype 2, the second video shows the recording of a live demo of the system functioning, and the third video walks through the GUI pages, and its functionality.

The project required extensive hardware troubleshooting due to the frequent breaking of the motor driver boards. Due to time spent in troubleshooting, all the intended objectives could not be achieved. Towards the end of the semester, the project still did not have the function of lifting and pushing the pen up and down.

If given 10 more days and a new order of motor drivers I believe I would be able to pinpoint the issue that broke the motor driver and complete the project to meet the objectives set at the beginning of the project. But aside from the misfortunate events that transpired, working on this project gave me the opportunity to work with the PIC24EP development board in depth and learn the various methods of troubleshooting. The lessons learned from this project will be carried on with me in the future to successfully build projects.

References:

- How to make mini CNC 2D plotter using scrap DVD drives: http://www.instructables.com/id/How-to-Make-Mini-CNC-2D-Plotter-Using-Scrap-DVD-Dr/
- Hack old CD roms into a CNC machine: http://www.tinkernut.com/portfolio/hack-old-cd-roms-into-a-cnc-machine/
- GRBL Git Hub Repository: https://github.com/grbl/grbl
- Stack Overflow: http://stackoverflow.com
- Processing User Manual: https://processing.org
- Pilot's Logbook, blog by Lucio Di Jasio; www.flyingpic24.com
- Easy Driver website: http://www.schmalzhaus.com/EasyDriver/
- Graphics, Touch, Sound, and USB User Interface Design for Embedded Applications, by Lucio Di Jasio
- Programming 16-bit PIC Microcontrollers in C, by Lucio Di Jasio
- Microchip 16-bit microcontrollers PIC24EP512GU810 Datasheet
- PIC24E Family Reference Manual
- MPLABx User Manual
- Pickit 3 User Manual
- Mikromedia Board User Manual
- Mikromedia Board Schematics
- Mikromedia Board Pinout
- ILI9341 LCD Datasheet
- Allegro A3967 driver chip datasheet
- Embedded Shield User Documentation V1.2, by Joe Lovelace

Notes:

With this lab report the following items are attached:

1. The program files used for the final project
2. The design review presentation submitted during the initial stage of the project
3. The midterm demo presentation
4. The final demo presentation
5. The video of prototype 2 testing
6. The video of prototype 3 in action
7. The video of the GUI walkthrough