University of British Columbia

Final Report

Name: Dinel Anthony

Student ID: 20189775

PHYS 319: XY Plotting Robot

Abstract

The objective of this project was to make use of the techniques learned during the lab, specifically Pulse-Width Modulation to build and code an X-Y Plotting Robot. The robot makes use of the A4988 Stepper Motor Driver to control the step size, power, and direction of two SM15DD railed stepper motors. Additionally, the robot makes use of an SG90 servo motor to control the height of the pen. In order to draw an image, the user inputs a series of steps in either X or Y by simulating a PWM using a delay, while also raising and lowering the pen as needed. These commands would be entered into an interrupt command, causing the robot to execute the drawing upon the press of either of the onboard buttons.

Introduction

The main motivation for this project stems from a previous course, APSC 160, in which we were first introduced to C Programming and Arduino control. That following summer I had bought an Arduino and one of my first projects was supposed to be the same X-Y plotting robot. However due to my lack of knowledge with circuitry as a whole (including essential skills such as PWM, interrupts, and troubleshooting), I had to put the project away. During Lecture 4, we discussed how to use the PWM and its applications, specifically with controlling motors. This immediately reminded me of the project which I had yet to complete. Further, this newfound theoretical knowledge allowed me to understand how to precisely control the motors without using obscure masks and built-in functions. However, simply controlling two stepper motors and a servo motor seemed rather simple and so I wanted to take the project a step further. My hope was to be able to have the user draw any of a variety of pre-designed images at their command, without having to re-flash the program each time. Inspired by previous work using interrupts, I incorporated the on-board buttons which would each generate their own pre-determined image. These components together resulted in an X-Y plotting robot which was capable of generating two images: one of the words "PHYS 319", and one of a house.

Theory

<u>SG90 Servo Motor:</u>

The SG90 servo motor is a motor which can rotate approximately 180° given a certain PWM signal. As long as the motor is powered, the arm will be locked in place and can withstand a torque of about $2.5\frac{kg}{cm}$. Inside of the servo motor is a DC motor coupled to a potentiometer. Every 20ms the servo motor checks for an input pulse, the width of which determines the rotation angle of the servo. It then holds this angle for the remainder of the 20ms until the next check. The input signal is sent to the input amplifier which creates a potential difference across the potentiometer, the feedback mechanism of which serves as an input for the DC motor. This causes the DC motor to turn until the potential difference across the potentiometer is 0, resulting in the motor powering off and holding its position (”What is a servo motor?”, 2015). From the data sheet, we see that a duty cycle (width of the input pulse) of 1.5ms corresponds to the middle of the servo, a 2ms duty cycle corresponds to a 90° rotation clockwise, and a 1ms duty cycle corresponds to a 90° rotation counter-clockwise (Robojax, n.d.). From this it can be inferred that any range of of duty cycle between 1ms and 2ms will result in some intermediate angle.

SM15DD Stepper Motor

The SM15DD Stepper Motor is a 5V, 18° per step railed bipolar stepper motor, most commonly found in DVD-ROMs. These types of stepper motors have two windings, labelled A and B, coiled around the motor. To drive the motor, alternating pulses must be sent to each winding such that one cycle is considered a ”step.” The precision of the motors is dependent on two factors: the diameter of the rail, as well as the configured microstepping of the motor. The larger the diameter of the rail, the further the object attached to the rail will travel in a single step. Microstepping is a means of combatting this. Through the use of a motor driver, the order and strength of the pulses sent to the coils can be varied to achieve various precision levels. The amount of microstepping levels is dependent on the motor driver used (Stenebo, 2020). A basic diagram of a bipolar stepper motor can be found in the appendix.

A4988 Stepper Motor Driver

The A4988 Stepper Motor Driver is used to easily regulate and control the signals sent to drive the connected stepper motor. One driver is typically needed for a single steppper motor. The driver consists of 16 pins: 2 to power the driver, 2 to power the motor, 4 to control the motor, 4 to control the microstepping, 2 to control the reset of the driver, 1 to control the steps of the motor, and 1 to control the direction of the motor. This motor driver in particular is capable of 5 microstepping modes: Full-step, Half-step, Quarter-step, Eighth-step, and Sixteenth-step. These are controlled by setting various combinations of the microstepping pins high or low. Included on the board is a potentiometer which can be adjusted to change the output current to the motor. When a signal is sent to the motor driver, the chip will automatically regulate the power to the coils and will drive them as instructed depending on the readings at the "STEP" pin and the "DIR" pin ("A4988", 2014).

Apparatus

To construct the project, first the servo motor was configured, then the stepper motors were configured, and finally the X-Y plotting robot was constructed. As such, each individual component will be discussed before the discussion of the complete apparatus.

SG90 Servo Motor:

The SG90 has three pins: ground, PWM, and VCC, labelled brown, orange, and red respectively. VCC was connected to the +5V line from the breadboard, the PWM pin was connected to P1.2 on the MSP-430, and ground was connected to the common ground on the breadboard. The PWM was enabled on P1.2 so as to be able to control the servo motor properly. Below, is the circuit diagram of the connections:

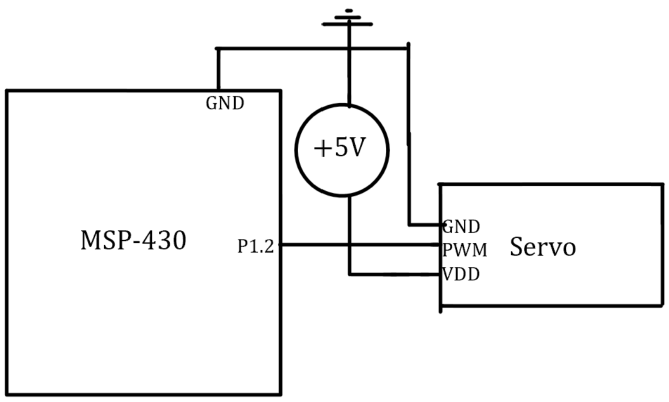


Fig. 1: Circuit Diagram of SG90 Servo Motor Connections

```
      TA0CTL = TASSEL_2 + MC_1 + TAIE +ID_0;
      TA0CCTL1 = OUTMOD_7;
      TA0CCR0 = 20000; // Period is 2ms
      P1DIR |= BIT2; // P1.2 as output
      P1SEL |= BIT2;  // Selection for timer setting
      TA0CCR1 = 1400; // Should set Servo motor down
      TA0CCTL1 = OUTMOD_7;  //CCR1 selection reset-set
      TA0CTL = TASSEL_2|MC_1;
```

Fig. 2: Script for Controlling Servo Motor

The script to control the servo motor was relatively simple. The timer and period for the PWM was setup according to the data sheet. I used the basic 1MHz clock since there was no need for quick communication for this project. Additionally, the period was set to 20ms. I created two functions which would rotate the servo motor up or down as needed. This allowed me to easily control whether the robot was actively drawing in its movements so that lines would not overlap unless necessary, allowing for clean drawings. As mentioned earlier, the servo was controlled through P1.2. I found experimentally that the best angle for holding the pen up was achieved when the duty cycle was set to 0.9ms. Similarly, the best angle for holding the pen down was achieve when the duty cycle was set to 1.4ms.

SM15DD Stepper Motor and A4988 Stepper Motor Driver:

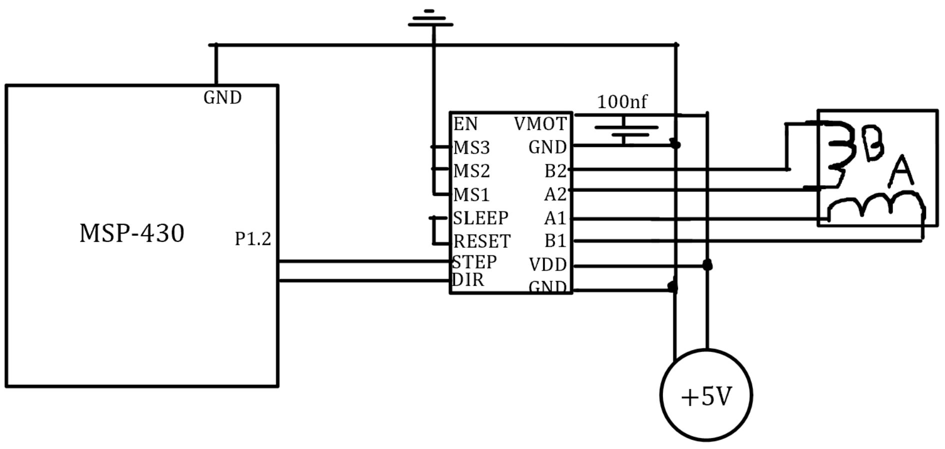


Fig. 3: Circuit Diagram of Stepper Motor Connections

Connections to the A4988 were relatively straightforward once the data sheet was read. For the stepper motor controlling motion in the x-direction, P6.3 was connected to "DIR," while P6.2 was connected to "STEP." Further, all of the microstepping pins were grounded, which served as setting them all to "LOW." This is because according to the data sheet, this resulted in "Full-step" mode, which produced the greatest amount of torque at the cost of some precision. Since the motors are relatively weak, and heat up quickly when overpowered, I opted for this method as a small decrease in precision was acceptable for the purposes of the project. "SLP" and "RST" pins were jumped together as this prevented the motor driver from entering sleep mode. Notably, the "EN," or enable pin was left unconnected as it defaults to "HIGH" which means that the driver is able to accept signals. +5V were fed to both "VMOT" and "VDD" as the SM15DD are 5V motors, and the driver also accepts 5V. The "GND" pins were connected to the same common ground as all of the microstep (MS) pins. Once the pairs of wires corresponding to each coil were determined, they were connected such that A1 and B1 were from the same coil, as were A2 and B2. This same process was repeated for the stepper motor controlling motion in the y-direction, however this time P6.0 was connected to ”STEP” and P6.1 was connected to ”DIR.”

```
while(1)
{
P6OUT ^= BIT2; // Swap P6.2 b/w HIGH and LOW
__delay_cycles(1100); // Wait 1.1ms
} // Simulate a 1.1ms pulse
```

Fig. 4: Script for Controlling Stepper Motor

Once the connections were secured, the next thing to do was to implement the code for controlling the motors. This was also extremely simple as all that was needed was to dictate which direction to move, and how many steps to take in that direction. For the x-motor which moved towards and away from the user, a direction of 0 corresponded to moving the platform towards the user, while a direction of 1 corresponded to moving the platform away from the user. Similarly, for the y-motor which moved parallel to the user, a direction of 0 corresponded to moving to the right, while a direction of 1 corresponded to moving to the left. The rail has a length corresponding to 600 steps, thus making the total available drawing area, 600 steps by 600 steps, assuming Full-step microstepping. To move the motors, I simulated a PWM pulse by setting the corresponding ”STEP” pin high, delaying for 1.1ms, and then setting the ”STEP” pin low again. One cycle of this is considered a ”step.” To complete multiple steps, the command was repeated inside of a for-loop. Once this was established it is simply a matter of understanding where the motor will be after completion of a loop, to be able to have control over the entire drawing area.

X-Y Plotting Robot

```
WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
//// Set up Timer for Servo ////
TA0CTL = TASSEL_2 + MC_1 + TAIE +ID_0;
TA0CCTL1 = OUTMOD_7;
TA0CCR0 = 20000;
TA0CCR1 = 900; // Servo start in UP position
//// Servo Motor on P1.2 ////
P1DIR |= BIT2;
P1SEL |= BIT2;  //selection for timer setting
//// Stepper Motors Setup ////
P6DIR |= 0b00001111; //YDIR on 6.1, YSTEP on 6.0, XDIR on 6.3, XSTEP on 6.2
P6OUT = 0; // 0 is forward for X, right for Y
//// Interrupt Setup ////
// Set P1.1 and 2.1 as pullup
P1OUT |= BIT1;
P2OUT |= BIT1;
// Enable Pullup resistor on P1.1 and P2.1
P1REN |= BIT1;
P2REN |= BIT1;
// Enable interrupts on P1.1 and P2.1
P1IE |= BIT1;
P2IE |= BIT1;
// Trigger interrupt on Falling Edge for P1.1 and P2.1
P1IES |= BIT1;
P2IES |= BIT1;
// Turn on interrupts and go in to LPM
_BIS_SR(LPM4b_bits + GIE);
```

Fig. 5: Pin Initialization for X-Y Plotting Robot

In putting the previous two components together, some connections were optimized to minimize cross-over of wires, and additional components were added to improve performance. The complete circuit diagram is available below:

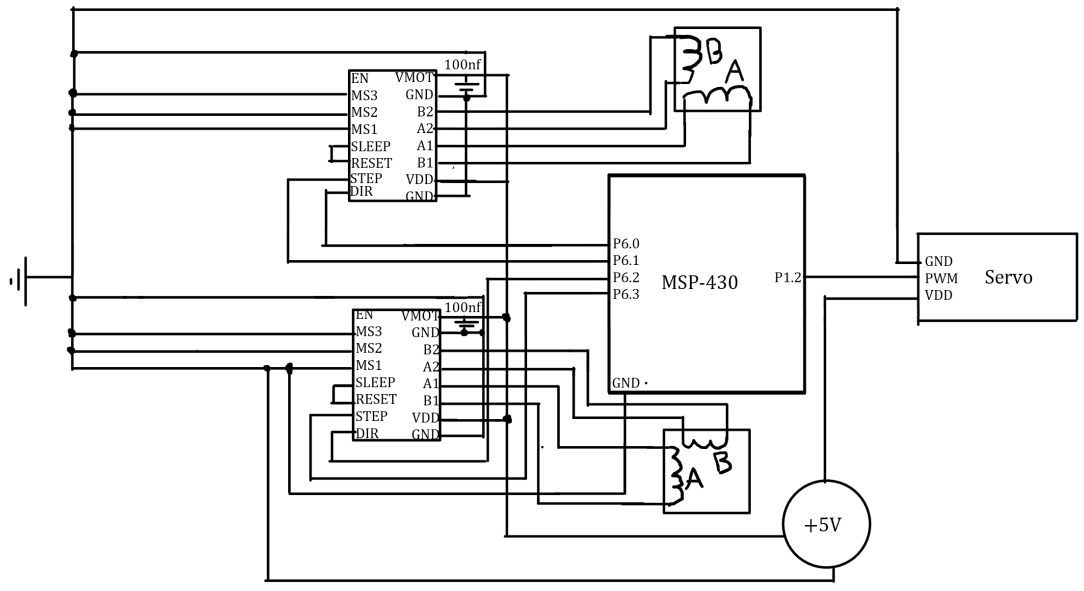


Fig. 6: Circuit Diagram for X-Y Plotting Robot

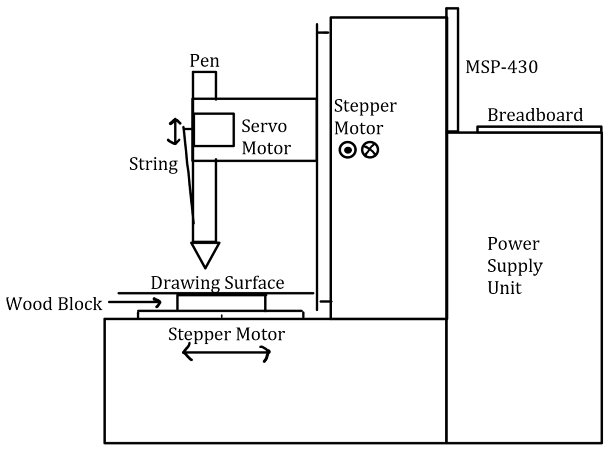


Fig. 7: Schematic of X-Y Plotting Robot

One of the changes made to the circuit when constructing the final product was that a 100nf capacitor was added in parallel to the motor power inputs of the drivers. Additionally, all ground connections were connected to the same common ground, this includes the MSP-430. Below is part of the function that I created to move the motor, along with what a call of the function would look like. To power the system, a PSU was modified to gain easy access to the +5V line, for a larger current draw. As can be seen in Fig. 7, a hole was drilled through the body of the pen, and a string was attached to the ink cartridge, which a spring forcing the pen down. The other end of the string was then attached to the servo motor such that rotating the servo motor upwards, compressed the spring, and allowed the pen to lift off of the page. Finally, a wooden block was placed on top of the motor that moved towards/away from the user so that a solid foundation could be provided for the drawing surface. Pictures of the apparatus can be found in the appendix.

```
if(dir==RIGHT && ((P6OUT >> XDIR) & 0))
//// If pen is supposed to move right and direction is right ////
{
    for(i=0;i<steps;i++)
    //// Move right by steps amount of steps ////
    {
        P6OUT ^= YSTEP;
        __delay_cycles(1100);
    }
}

else if(dir==RIGHT && ((P6OUT >> XDIR) & 1))
//// If pen is supposed to move right and direction is not right ////
{
    P6OUT ^= YDIR; // Change motor direction
    for(i=0;i<steps;i++)
    //// Move right by steps amount of steps ////
    {
        P6OUT ^= YSTEP;
        __delay_cycles(1100);
    }
}
```

Fig. 8: Function to Move Stepper Right

```
penMove(UP, 200);
penMove(LEFT, 100);
penMove(DOWN, 100);
penMove(RIGHT, 100);
```

Fig. 9: Script to Call the Movement Function

As can be seen in Fig. 8, each if-statement considers two things: which direction the motor is supposed to move and whether the motor is already set to move in that direction. The first condition (”dir==RIGHT”) is an argument that is passed into the function on function call. The second condition (”(P6OUT >> XDIR) & LEFT)”) shifts to the bit corresponding to the motor direction and checks whether the current direction is the intended direction. If the current direction is different from the intended direction, then the code jumps to the second if-statement which will flip the direction of the motor, before moving it. Inside the if-statement is a for-loop which will step the motor in the intended direction as many times as is passed into the loop. This block is repeated for each of the 4 possible directions. Fig. 9 shows that by creating the function described by Fig. 8, the actual commands become much more readable to the user.

Another notable addition was that the on-board buttons were introduced as interrupts to control which image was drawn. This was done by initializing interrupts on both pins as was done in Lab 4. When the button attached to P1.1 is pressed, the program to draw a house is executed. When the button attached to P2.1 is pressed, the program to write the words ”PHYS 319” is executed. Part of the code requires some attention:

```
__delay_cycles(500000);
penUp();
__delay_cycles(500000);

__delay_cycles(500000);
penDown();
__delay_cycles(500000);
```

Fig. 10: Script to Move Pen Up/Down

In Fig. 10 we see that there are delays of 500ms before and after changing the direction of the pen. This is because the servo motor rotates much slower than the stepper motors and as a result the pen needs time to move before the program can continue drawing.

```
//// Left Side of Roof ////
for(i=0;i<300,i++)
{
    penMove(RIGHT, 1);

    if(i%2 == 0)
    {
        penMove(UP, 1);
    }
}
//// Right Side of Roof ////
for(i=0;i<300,i++)
{
    penMove(RIGHT, 1);

    if(i%2 == 0)
    {
        penMove(DOWN, 1);
    }
}
```

Fig. 11: Script for Drawing the Roof of the House

Additionally, the code for drawing the roof of the house in Fig. 11 contains a for loop that increments both motors in steps of 1. The y-motor is incremented every other step so as to generate a slope, as is needed for a roof. To do this, I only incremented the y-motor on every even numbered step which was done using the ”modulo” operator, %.

The full code for each drawing can be found in the appendix, along with a video of ”PHYS 319” being printed. Note that before the initial run, the motor must be placed at the origin.

Results

After completing the entire setup, including both electrical components and writing the script, I am happy to say that the system runs as I had envisioned. The robot is first powered on by the power supply unit, then one of the two buttons is pressed to tell the robot which graphic to draw, and finally the motors move as they are supposed to, to generate the desired graphic. There were quite a few complications to resolve along the way, but the final product works exactly as intended. At first, I was concerned with the precision of the motors since the motors were set to Full-step, however as can be seen in the video in the appendix, this did not end up being an issue. The maximum drawing area was determined to be 600 steps by 600 steps as previously mentioned which translated to about a 50mm by 50mm drawing area. From this, I was able to calculate that 1 step was approximately 0.08mm. such a small step size is the reason I was able to avoid microstepping since such a small scale is not noticeable to the human eye.

In testing the motors, I had a very difficult time getting the motors to run. The first issue was that there was no data sheet for this motor available on the internet.

After a few days of searching, I was able to source one that had been translated from a Russian website, so the reliability of the data sheet was questionable. Further, every source regarding connecting the motors to the motor driver gave a different answer on how to connect the coils to the pins. Some said that the coils should be connected to the same letter (A or B), whereas others said that they should be separated. I eventually found that the latter was the correct option, however not before I had burned a total of 4 motor drivers and 1 MSP-430. This was because before I had found the sketchy data sheet, or found out how to connect the coils, the motors were powering, but not stepping. As a result, I considered this to be telling me that the motors were in fact 12V motors and not 5V motors. Therefore, I plugged in a 15V line for testing purposes, and was able to get the motors to finally move although with a lot of friction, and the motors themselves were heating up quite a bit. This led to 2 motor drivers burning out, which generated, a short circuit, causing the MSP-430 to be overloaded, and burn out as well. Later once the wiring had been fixed, and the X-Y Robot was put together, instead of a ground line, I fed a 12V line to the ground pin which cost me 2 more motor drivers.

Further, the supplied USB-power for the breadboard supplied an insufficient amount of current for powering the stepper motors. When measuring, the maximum current output of the supply was 0.4A, while the motors were drawing approximately 0.8A. To rectify this, as mentioned earlier, I modified a PC PSU to access the 5V line which offered up to 2A of current. This, coupled with the built in potentiometer of the motor drivers allowed me to control the current output to the motors for optimal driving speeds, while minimizing the heating of the motors. The motors did tend to heat up if they were asked to draw more than 3 images back-to-back and this is mainly due to current draw. Since the motors were carrying a larger load than their designed purpose, I had to increase their current draw so that the motors could provide more torque to turn the rails. As a result, the motors would perform beyond their optimal range, leading to heating. To calibrate this, I would drive the motor continuously and adjust the driver's potentiometer until the motor provided just enough torque to handle the weight of the load. This was repeated for both motors, and while still led to heating, was the most optimal solution.

There was also an issue with the pen not having enough time to lift or put down itself before the motors continued drawing. To fix this, I simply added a long enough delay between motions of the pen. This allowed me to ensure that the pen was either fully in contact with, or fully out of contact with the drawing surface before the stepper motors moved.

Finally, although not necessarily an issue, programming the drawings did get quite tedious at times. Since I was manually inputting how to draw each line, I had to first plan out how to allocate the available space as efficiently as possible as well as the most optimal paths to move the pen such that the drawing path was natural to follow. Another challenge I encountered was the method of drawing non-rectangular shapes. This puzzled me as I wanted to be able to produce as much of a complete product as possible, however with the way that the motors were being controlled, this proved to be rather difficult. In the end, I was able to draw lines at various angles by changing how many steps one motor took relative to another, which mimics the function of a slope which is rise over run. Similarly, the draw a curve, I would have to manually calculate a sample of points along the desired curve, and connect the spacing between the points using small steps along X and Y. The number of points would improve the resolution of the curve, as would implementing microstepping. Unfortunately, I did not have time to implement curves for this project.

In summary, the robot performed just as expected. Many of the issues encountered could have easily been solved if a data sheet for the motor were easily accessible, however that was not the case. This project was much more mechanically-focused and this is reflected in the simplicity of the code. Further, the goal of the project was to build a robot that was able to draw images, and so there is no quantitative data nor physical theory that requires comparison. The result of running the "PHYS 319" script can be found in the appendix.

Discussion

As a whole, I am extremely pleased with the overall accuracy, build quality, and implementation of the final product. Over multiple trials none of the motor drivers nor motors have burned out or caused issues, so I am confident that I have calibrated everything well. Adding the buttons to run the individual programs was a great addition since this made troubleshooting and overall user interaction much easier. Having said, this, there are a couple of improvements that if implemented, could take the project to the next level.

First, larger rails on the motors would be nice. The robot is fitted with a much larger drawing pad than the motors allow access to, and so motors with larger rails would expand the amount of drawings possible, or even allow multiple drawings to be made on the same page. The current drawing area is too small to do anything overly intricate, or big. This could be done by exchanging the current SM15DD motors, with NEMA-17 motors and then using an attachment to add a rail to them. These motors have much more torque and do not heat up as easily, which would allow for me to drive much larger rails, leading to bigger images. The NEMA-17's also have the added bonus of having a much smaller step angle. The step angle of the SM15DD stepper motor is 18° per step, where as the NEMA-17 stepper motor has a step angle of 1.8° per step: a whole 10-times smaller. This smaller step angle would have the added benefit of more intricate drawings as I would have much more control.

One other annoyance which I discussed during the results was the problem of drawing non-rectangular shapes. There exists a program called "GCTRL" which is able to take vectorized graphics and transform them into GRBL instructions which motors can easily interpret and implement. This effectively means that through this program, virtually any image can be drawn. Using another software called "Inkscape," users could upload lineart images, vectorize them, and then upload them into GCTRL. GCTRL would then convert the vectors into a map of coordinates and decide the optimal path to each point, and send those instructions to the motors. Unfortunately, this program communicates with the motors by converting the commands into Arduino code and then running the generated code through the Arduino IDE, which does not work for the purposes of this project. There does exist a program called ”Energia” which will convert Arduino code to something that can be run on the MSP-430, however such a translation does not exist for GCTRL which means that I am left with implementing the drawings on my own. In the future, I may try to implement my project in Arduino to get this functionality working.

Finally, I would like to make my code more modular. Since even a simple drawing such as a house can take many lines of code, as can be seen in the appendix, I would like to spend some time designing a system that only draws basic shapes, and then using those basic shapes I could combine them in any order I wanted to generate drawings. For instance I would have a function to draw a rectangle, whose arguments would be the length and width, as well as the coordinates of one of the vertices, and calling that function would then draw such a rectangle. This could be repeated for other shapes such as any n-polygon, and potentially even a circle. In fact, the same could be applied to creating a function that generates the letters of the alphabet, and the argument would be the size of the character. This would simplify the code extremely, and make the code more intuitive for a reader.

Conclusion

In conclusion, I have managed to create a functioning X-Y plotting robot, which has two preloaded drawings. I am pleased with the final result of my project, though there is definitely room for some improvement in the future. I feel that it was very worthwhile to construct for a number of reasons; the first of which being that it was simply rewarding being able to draw anything that I could imagine. Seeing my mental image of a drawing be translated right in front of me brought me a lot more joy than I thought it would. I was also able to play around and write short messages. Second, I learned a great deal whilst completing this project; using a motor driver for the first time, and controlling the motors through a customized PWM taught me a lot about the mechanics behind the everyday stepper motor. I also had to invest a considerable amount of time into learning the theory behind stepper motors and how to wire them since I did not have access to a data sheet for a considerable amount of time. Not only did this project help me discover the benefits and limitations of various microcontrollers, I now feel considerably more comfortable in designing new systems, and feel that the range of my ability when working with microprocessors and external components has improved drastically. I am very happy to have taken on this project, and will definitely try to reimplement my proposed improvements using an Arduino over the summer, as I would like to see how far I can take this project. In fact, should I replace the pen with either a drill head or laser, I could essentially transform the plotting robot into a CNC or laser engraver. Without the knowledge I was given in this class such aspirations would not have been possible.

References

Kotlicki, A. *Lab Manual.* (2023). Retrieved from https://phas.ubc.ca/ kotlicki/Physics_319/Lab_Manual_2023_AK.pdf

A4988 DMOS Microstepping Driver with Translator And Overcurrent Protection. Pololu. (2014). Retrieved from https://www.pololu.com/file/0J450/A4988.pdf

Apoorve. (2015, August 1). *What is a servo motor? - understanding the basics of servo motor working. Servo Motor Basics, Working Principle & Theory.* Retrieved from https://circuitdigest.com/article/servo-motor-working-and-basics

Robojax. (n.d.). *9 G Micro Servo - RoboJax. Robojax Servo SG90 Datasheet.* Retrieved from https://robojax.com/learn/arduino/robojax-servo-sg90_datasheet.pdf

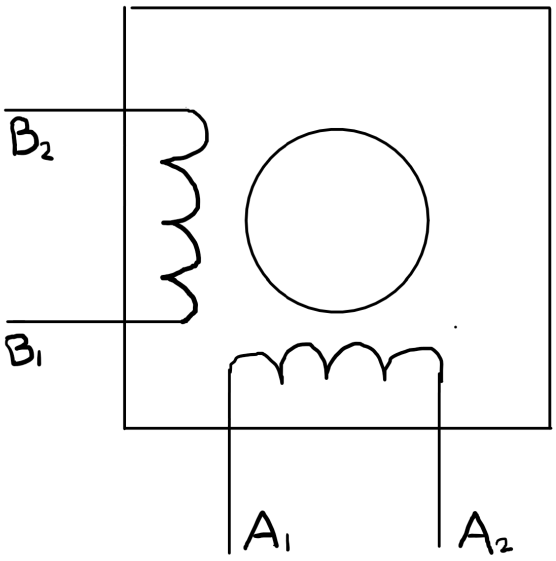
Stenebo, P. (2020, June 25). *Bipolar Stepper Motor with Driver Board.* Bends. Retrieved from https://bends.se/?page=notebook%2Felectronics%2Fstepper-2&lang=en&cid=0

Appendix

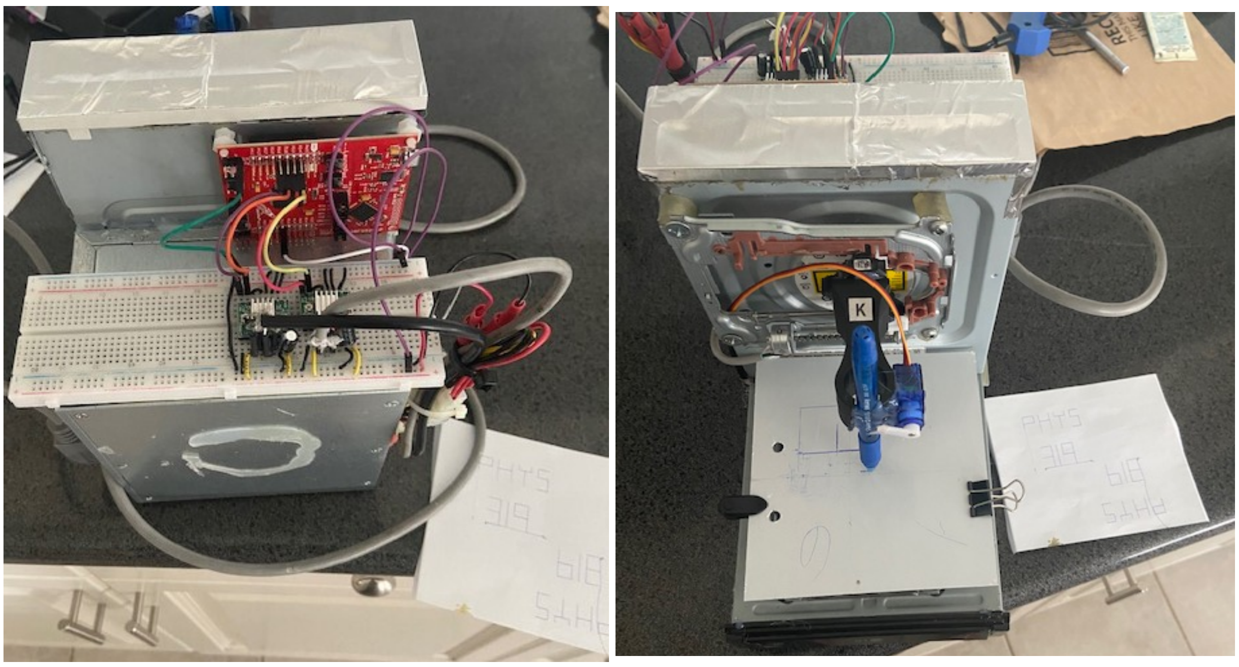
Printing "PHYS 319"

X-Y Plotting Robot: "PHYS 319"

Stepper Motor Basic Diagram



Images of the Completed Plotter



Entire Script

```
#include <msp430.h>
//// Necessary Definitions ////
#define UP 0
#define DOWN 1
#define LEFT 2
#define RIGHT 3
#define XDIR BIT3    // Based on Motor Driver Connection
#define YDIR BIT1    // Based on Motor Driver Connection
#define XSTEP BIT2   // Based on Motor Driver Connection
#define YSTEP BIT0   // Based on Motor Driver Connection
void penDown();
void penUp();
void penMove(int dir, int steps);
int i=0;             // Incrementer for steps
void main(void)
{
     WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    //// Set up Timer for Servo ////
    TA0CTL = TASSEL_2 + MC_1 + TAIE +ID_0;
    TA0CCTL1 = OUTMOD_7;
    TA0CCR0 = 20000;  // Period is 2ms
    TA0CCR1 = 900; // Servo start in UP position
    //// Servo Motor on P1.2 ////
    P1DIR |= BIT2;
    P1SEL |= BIT2;  //selection for timer setting
    //// Stepper Motors Setup ////
    P6DIR |= 0b00001111; //YDIR on 6.1, YSTEP on 6.0, XDIR on 6.3, XSTEP on 6.2
    P6OUT = 0; // 0 is forward for X, right for Y
    //// Interrupt Setup ////
    // Set P1.1 and 2.1 as pullup
    P1OUT |= BIT1;
    P2OUT |= BIT1;
    // Enable Pullup resistor on P1.1 and P2.1
    P1REN |= BIT1;
    P2REN |= BIT1;
    // Enable interrupts on P1.1 and P2.1
    P1IE |= BIT1;
    P2IE |= BIT1;
    // Trigger interrupt on Falling Edge for P1.1 and P2.1
    P1IES |= BIT1;
    P2IES |= BIT1;
    // Turn on interrupts and go in to LPM
    _BIS_SR(LPM4b_bits + GIE);
}

void __attribute__((interrupt(PORT1_VECTOR))) PORT1_ISR(void)
{
    //// Draw a House ////

    //// Initial Positioning ////
    penMove(UP, 400);
    penMove(RIGHT, 100);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
    //// Main Square of House ////
    penMove(DOWN, 400);
    penMove(RIGHT, 400);
    penMove(UP, 400);
    penMove(LEFT, 500);
    //// Left Side of Roof ////
    for(i=0;i<300,i++)
    {
        penMove(RIGHT, 1);
        if(i%2 == 0)
        {
            penMove(UP, 1);
        }
    }
    //// Right Side of Roof ////
    for(i=0;i<300,i++)
    {
        penMove(RIGHT, 1);
        if(i%2 == 0)
        {
            penMove(DOWN, 1);
        }
    }
    penMove(LEFT, 100);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
```

```
//// Door ////
penMove(LEFT, 150);
penMove(DOWN, 400);
__delay_cycles(500000);
penDown();
__delay_cycles(500000);
penMove(UP, 250);
penMove(LEFT, 100);
penMove(DOWN, 250);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
//// Window ////
penMove(LEFT, 50);
penMove(UP, 300);
__delay_cycles(500000);
PenDown();
__delay_cycles(500000);
penMove(UP, 76);
penMove(LEFT, 76);
penMove(DOWN, 76);
penMove(RIGHT, 76);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
penMove(LEFT, 38);
__delay_cycles(500000);
PenDown();
__delay_cycles(500000);
penMove(UP, 76);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
penMove(DOWN, 38);
penMove(LEFT, 38);
__delay_cycles(500000);
PenDown();
__delay_cycles(500000);
penMove(RIGHT, 76);
//// Go back to origin ////
penMove(LEFT, 276);
penMove(DOWN, 338);
```

```
    P1IFG &= ~BIT1; // Clear interrupt flag
}

void __attribute__((interrupt(PORT2_VECTOR))) PORT2_ISR(void)
{
    //// Draw "PHYS 319" ////

    //// Initial Positioning ////
    penMove(UP, 400);
    ///////// P ///////////
    __delay_cycles(500000);
    penDown();
    __delay_cycles(750000);
    penMove(UP,200);
    penMove(RIGHT, 100);
    penMove(DOWN, 100);
    penMove(LEFT, 100);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
    ////////// H //////////
    penMove(RIGHT, 150);
    penMove(UP, 100);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
    penMove(DOWN, 200);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
    penMove(UP, 100);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
    penMove(RIGHT, 100);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
    penMove(UP, 100);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
```

```
penMove(DOWN, 200);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
//////// Y ///////////
penMove(RIGHT, 50);
penMove(UP, 200);
__delay_cycles(500000);
penDown();
__delay_cycles(500000);
penMove(DOWN, 75);
penMove(RIGHT, 100);
penMove(UP, 75);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
penMove(DOWN, 75);
penMove(LEFT, 50);
__delay_cycles(500000);
penDown();
__delay_cycles(500000);
penMove(DOWN, 125);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
//////// S ///////////
penMove(RIGHT, 100);
__delay_cycles(500000);
penDown();
__delay_cycles(500000);
penMove(RIGHT, 100);
penMove(UP, 100);
penMove(LEFT, 100);
penMove(UP, 100);
penMove(RIGHT, 100);
__delay_cycles(500000);
penUp();
__delay_cycles(500000);
////////// 9 ///////////
penMove(DOWN, 550);
penMove(LEFT, 150);
__delay_cycles(500000);
```

```
    penDown();
    __delay_cycles(500000);
    penMove(UP, 200);
    penMove(LEFT, 100);
    penMove(DOWN, 100);
    penMove(RIGHT, 100);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
    /////// 1 ///////
    penMove(LEFT, 150)
    penMove(DOWN, 100);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
    penMove(UP, 200);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
    //////// 3 /////////
    penMove(RIGHT, 150);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
    penMove(RIGHT, 100);
    penMove(DOWN, 100);
    penMove(LEFT, 100);
    __delay_cycles(500000);
    penUp();
    __delay_cycles(500000);
    penMove(RIGHT, 100);
    __delay_cycles(500000);
    penDown();
    __delay_cycles(500000);
    penMove(DOWN, 100);
    penMove(LEFT, 100);
    penUp();
    //// Return to origin ////
    penMove(LEFT, 100);
    penMove(DOWN, 50);
    P2IFG &= ~BIT1; // Clear interrupt flag
}
```

```
void penUp()
{
    //// Turns the servo motor to set the pen up ////
    TA0CCR1 = 900;
    TA0CCTL1 = OUTMOD_7;  //CCR1 selection reset-set
    TA0CTL = TASSEL_2|MC_1;
}

void penDown()
{
    //// Turns the servo motor to set the pen down ////
    TA0CCR1 = 1400;
    TA0CCTL1 = OUTMOD_7;  //CCR1 selection reset-set
    TA0CTL = TASSEL_2|MC_1;
}

void penMove(int dir, int steps)
{
    //// Moves pen in given direction for given amount of steps ////

    if(dir==UP && ((P6OUT >> XDIR) & 0))
    //// If pen is supposed to move up and direction is up ////
    {
        for(i=0;i<steps;i++)
        //// Move up by steps amount of steps ////
        {
            P6OUT ^= XSTEP;
            __delay_cycles(1100);
        }
    }
    else if(dir==UP && ((P6OUT >> XDIR) & 1))
    //// If pen is supposed to move up and direction is not up ////
    {
        P6OUT ^= XDIR; // Change motor direction
        for(i=0;i<steps;i++)
        //// Move up by steps amount of steps ////
        {
            P6OUT ^= XSTEP;
            __delay_cycles(1100);
        }
    }
```

```
if(dir==DOWN && ((P6OUT >> XDIR) & 1))
//// If pen is supposed to move down and direction is down ////
{
    for(i=0;i<steps;i++)
    //// Move down by steps amount of steps ////
    {
        P6OUT ^= XSTEP;
        __delay_cycles(1100);
    }
}
else if(dir==DOWN && ((P6OUT >> XDIR) & 0))
//// If pen is supposed to move down and direction is not down ////
{
    P6OUT ^= XDIR; //Change motor direction
    for(i=0;i<steps;i++)
    //// Move down by steps amount of steps
    {
        P6OUT ^= XSTEP;
        __delay_cycles(1100);
    }
}
if(dir==RIGHT && ((P6OUT >> XDIR) & 0))
//// If pen is supposed to move right and direction is right ////
{
    for(i=0;i<steps;i++)
    //// Move right by steps amount of steps ////
    {
        P6OUT ^= YSTEP;
        __delay_cycles(1100);
    }
}
else if(dir==RIGHT && ((P6OUT >> XDIR) & 1))
//// If pen is supposed to move right and direction is not right ////
{
    P6OUT ^= YDIR; // Change motor direction
    for(i=0;i<steps;i++)
    //// Move right by steps amount of steps ////
    {
        P6OUT ^= YSTEP;
        __delay_cycles(1100);
    }
}
```

```
    if(dir==LEFT && ((P6OUT >> XDIR) & 1))
    //// If pen is supposed to move left and direction is left ////
    {
        for(i=0;i<steps;i++)
        //// Move left by steps amount of steps
        {
            P6OUT ^= YSTEP;
            __delay_cycles(1100);
        }
    }
    else if(dir==LEFT && ((P6OUT >> XDIR) & 0))
    //// If pen is supposed to move left and direction is not left ////
    {
        P6OUT ^= YDIR; // Change motor direction
        for(i=0;i<steps;i++)
        //// Move left by steps amount of steps ////
        {
            P6OUT ^= YSTEP;
            __delay_cycles(1100);
        }
    }
}
```