Drawing Robot

Michael Sekatchev, 58850397

April 11, 2022

PHYS 319 Final Project Report

Abstract

The report describes the creation of a programmable drawing robot. Controlled in C using the Launchpad MSP430 microprocessor, the robot follows hard-coded driving and drawing instructions to draw a pre-programmed shape on the surface below it using a Sharpie pen. It accomplishes this via a two-wheel drive system of stepper motors, as well as a servo motor for raising and lowering the Sharpie off the surface. The robot's constituent hardware and its integration in a circuit is described in detail, as well as the code used to run it. A successful result of the robot writing my name is shown, and future work and improvements are discussed, mainly with regards to the software (G-Code implementation) and writing accuracy improvements.

Contents

1 Introduction

The project was inspired by the booming 3D printing technology, which converts simple G-Code [13] instructions for creating a part in layers, into movement of their extruders. The intention originally was to use this movement principle in two dimensions to create a 2D plotter, which would move two sets of axis using stepper motors and control the lifting and lowering of a pen (identical to the enabling of a 3D printer's extruder).

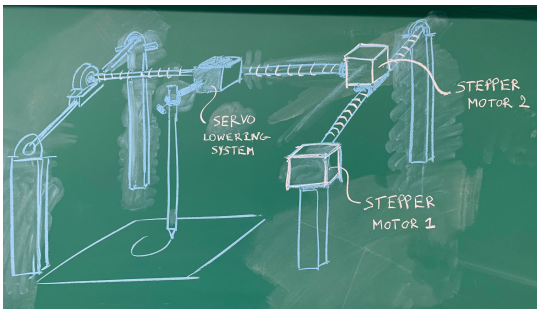

Figure 1. Sketch of the original idea: creating a 2D plotter using a system of stepper motors and a servo motor.

It was quickly understood that such a project would be more mechanically involving than electronics based (creating from scratch a system for moving a pen along two axis), and also not novel or interesting given the many commercial plotting systems available. Additionally a much greater number of parts needed to be purchased to accomplish this, upwards of $100 CAD. It was therefore decided to continue pursuing a project involving drawing, but instead of attempting to construct a plotter, to build a drawing robot, an arguably more exciting creation. The idea would be to use similar principles to a traditional 2D plotter or CNC machine (similar hard-coded instructions and sensors) to attempt to create a drawing robot. This new idea would simplify the hardware component of the project and driving down its costs while also rendering the coding part of the project more challenging.

2 Theory

In order to understand this project, a basic background on stepper motors and servo motors is necessary. This section will give an introduction to both types of motors, and how they are used in this project with the MSP430.

2.1 NEMA 17 Stepper Motors and A4988 Stepper Motor Drivers

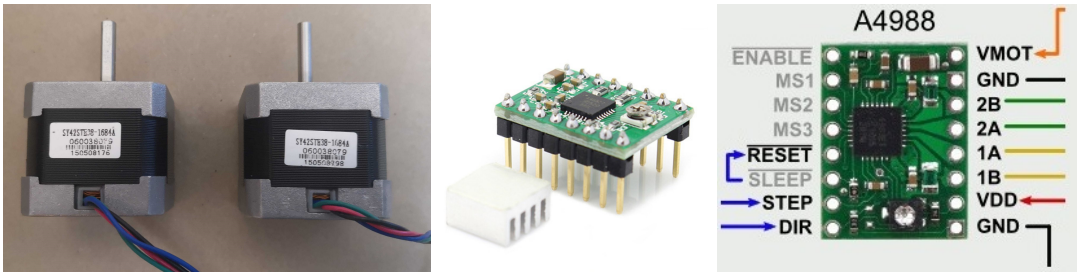

Figure 2. Left to right: image of the NEMA 17 stepper motors and the A4988 stepper motor driver (with attachable heatsink), schematic diagram for A4988.

The robot's platform sits on three wheels, two of which are driven by NEMA 17 stepper motors interfacing with the MSP430 through their A4988 stepper motor drivers. A stepper motor is a type of motor whose rotation is divided in "steps". Using short pulses (1ms for this project), the stepper motor can be instructed to step in the specified direction (clockwise or counter-clockwise). This stepping mechanism allows us to very precisely control the rotation angle of the motor's axle, making them a perfect choice for this robot, as well as other related projects requiring high accuracy rotational movement (3D printers, CNC machines, etc.). The direction and size of the step are specified by their associated stepper motor drivers.

Standard NEMA 17 motors were chosen for this project. The "17" refers to the size of their faceplate, 1.7 in $\times$ 1.7 in [10]. These types of stepper motor are 2-phase, meaning they are powered with two sets of coils (5-phase steppers are common too). They have a 1.8° step angle, corresponding to 200 steps/revolution [12]. NEMA 17 motors can take a wide range of voltages, between 8-35 V, which they receive through their motor drivers. Each phase of the motor requires a positive and a negative connection, resulting in 4 connections per motor. I used the 42HS40-1704 model NEMA 17 stepper motors, which take up to 1.7 A per phase [2]. The phases correspond to the connection colors as shown in the diagram below:

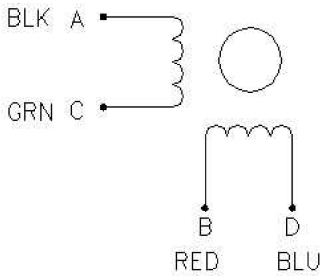

Figure 3. Schematic diagram for the 42HS40 model NEMA 17 stepper motor driver [2]. The phases are connected in pairs black-green, red-blue.

Pulses are sent to the coils in the correct configuration to advance the motor by a step, or to hold it in place. The control of these pulses is done through a stepper motor driver. For this project, the A4988 stepper driver for NEMA 17 motors [8] was used. This driver uses standard 3.3 V logic, supplied to its VDD pin (see figure 2), and receives power for the motor drivers through the VMOT pin, which it distributes to the motor through 4 pins (2 per phase as discussed above): 1A, 1B, 2A, and 2B. The two main inputs to the A4988 stepper motor are the STEP and DIR pins, which in this project are both connected to a pin on the MSP430. A pulse, or pulsed signal (PWM) is sent to the STEP pin to step the stepper motor in the direction specified by the DIR pin (either HIGH or LOW, which corresponds to either clockwise or counterclockwise). The SLEEP pin simply needs to recieve HIGH to turn the driver on. Finally, the MS1, MS2, and MS3 pins can be used to specify a fraction of a step (called a microstep) for the motor to step by, with the default being a standard full step. For this project, the smallest $1/16^{th}$ microstep resolution was selected, to allow for more fine control of the stepper motor (and hence Sharpie) position. This was done by supplying a HIGH signal to all 3 pins. A table of pins and corresponding microstep resolutions can be found in [9].

2.2 SG90 9G Servo motor

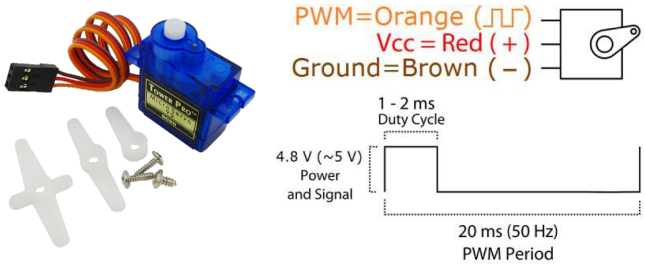

Figure 4. Left: image of the SG90 9G servo motor used in this project for controlling the raising and lowering of the Sharpie. The second horn from the left was used (the bar-type servo horn). Right: schematic diagram of the three inputs for the SG90, and a simple plot of the type of PWM signal the motor operates on [3].

The drawing robot makes use of a servo motor to control the raising and lowering mechanism of the Sharpie pen. They are made up of a motor and a position sensor, which provides the motor with constant feedback about the angle of the motor shaft, so the motor can adjust [4]. In this project, the standard SG90 9G servo motor was used (see image in figure 4). The 90 in the name refers to the range of motion of the stepper motor, restricted to $\pm 90°$ ($180°$ total), and the 9G refers to its weight, only 9 grams. This stepper motor can provide 2.5 kg-cm of torque, and uses an operating voltage of 4.8 V (nominally $\sim$5 V) [3]. The angle of stepper motor is set by controlling the duty cycle of the pulsed signal it receives through its orange PWM pin (see schematic in figure 4). In this project, the servo motor was given a signal at a constant period of 20 ms. The pulse width was set to either 1.5 ms for the lowered Sharpie position (nominally $0°$) and 0.75 ms for the raised Sharpie position (approximately $-45°$).

3 Apparatus

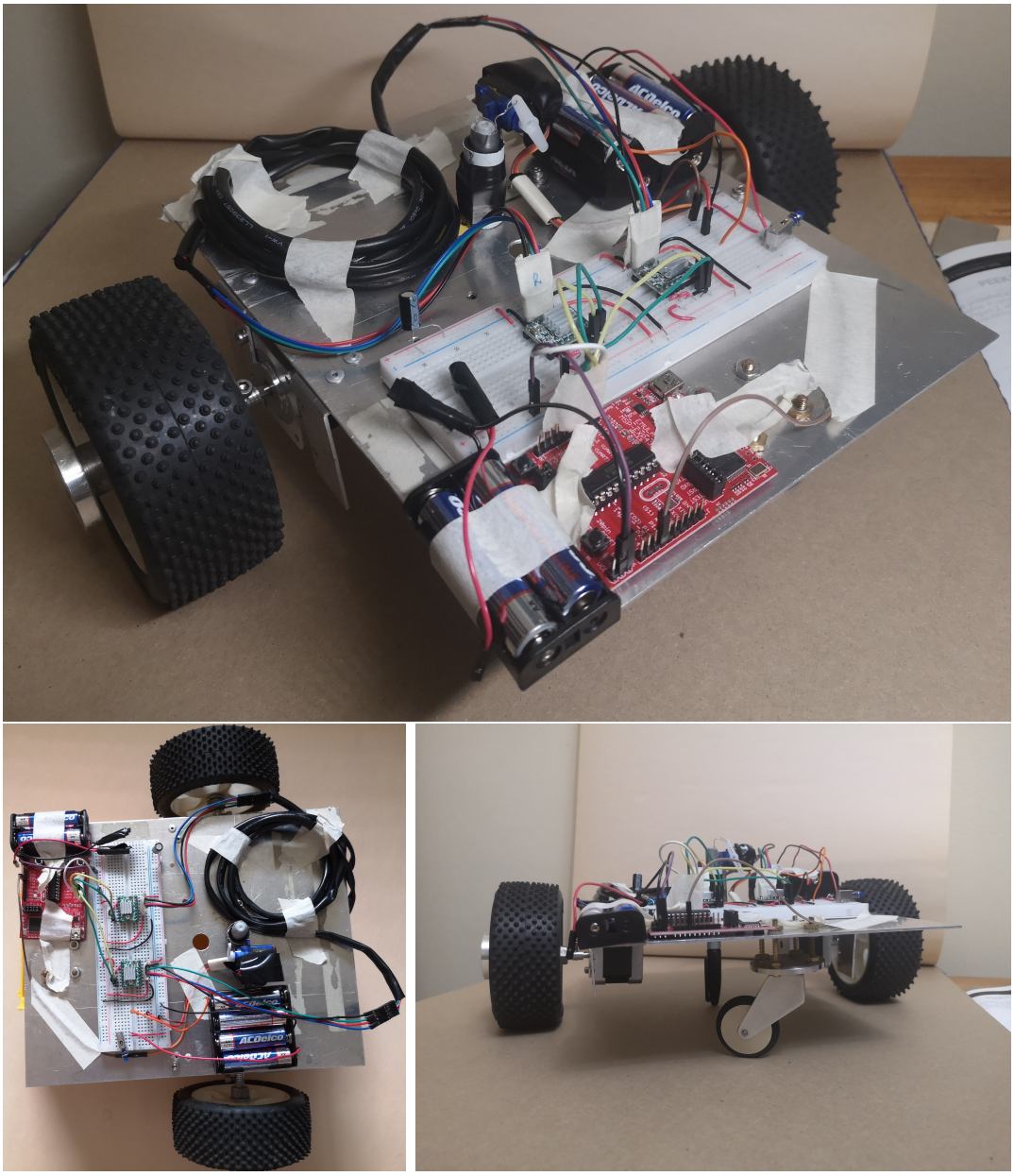

Figure 5. Images of the entire completed drawing robot.

This section will provide a complete overview of all aspects of the drawing robot, divided into hardware, circuitry, and software parts. The final result is shown in figure 5 above.

3.1 Hardware

3.1.1 Platform, stepper motors, and wheels

The base is a flat 20 cm×25 cm sheet of aluminium, on which two bent aluminium sheet brackets are riveted and used as mounting points for the two 42HS40-1704 NEMA 17

stepper motors, shown in figure 6 below:

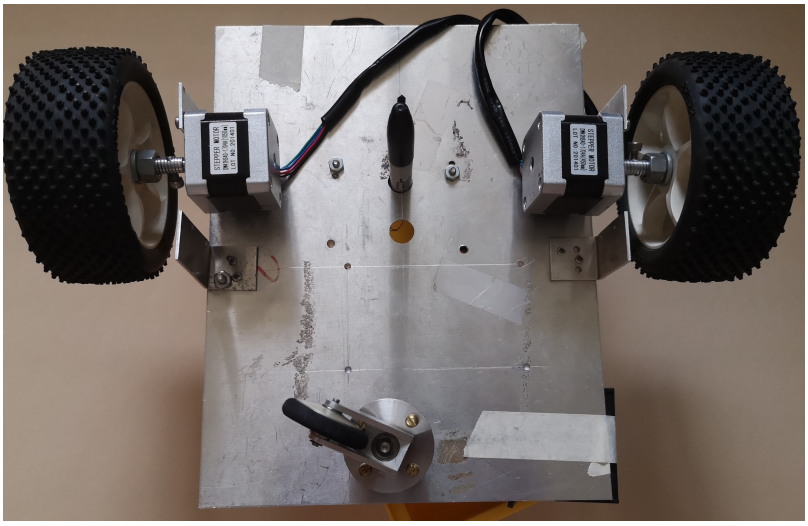

Figure 6. Image of the bottom of the robot. Stepper motor mounts are shown, as well as a hole for the Sharpie pen, and a dummy castor wheel on the bottom.

This platform was used for other projects, which explains the other hole, mounting brackets, and extra tape shown in figure 6. The platform initially used a set of DC motors paired with the wheels shown. The mounts of the wheels had to be machined to fit the 5 mm shaft on the NEMA 17 stepper motors [6], and new holes had to be machined in the aluminium mounting brackets to fit the stepper motor mounting holes. Also included on the platform was a dummy castor wheel, which simply acted as a third point of contact of the robot with the ground.

3.1.2 Sharpie servo raising/lowering system

The servo Sharpie raising-lowering system had to be added to the platform from scratch. The position of the Sharpie pen had to be exactly in the centre of the axle of the two wheels, to significantly simplify the coding part of the project: This mounting position ensures that the pen stays fixed in place while the robot turns on its axis, i.e. placing the pen here would mean that the wheels could simply be turned in opposite directions to accomplish a turn, and the pen would maintain its position on the drawing surface.

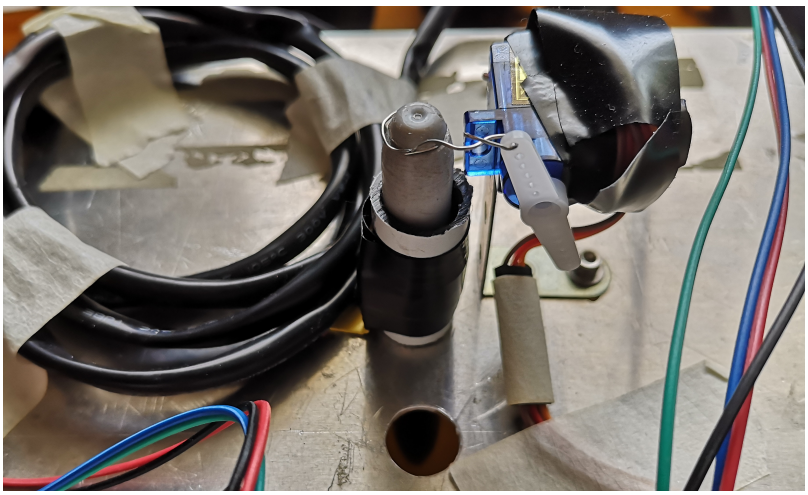

Figure 7. Close-up image of the Sharpie raising/lowering system.

The pen is restricted to move up and down through a 1/2 in PVC pipe, which is secured to the robot's aluminium base with a bent mending plate. A grove is cut at the top of the sharpie pen along its circumference, and is used to attached a metal wire, tied to a hole on the horn of the servo motor. The idea is to have the pen loosely attached to the servo to effectively convert the rotation of the servo into a linear movement of the pen. The pen is therefore pushed onto the page simply by its weight, rather than the force of the servo motor. An SG90 9G servo motor is used, as described in the Theory section. The servo motor itself is taped to another bent mending plate attached to the base sheet.

3.2 Circuitry

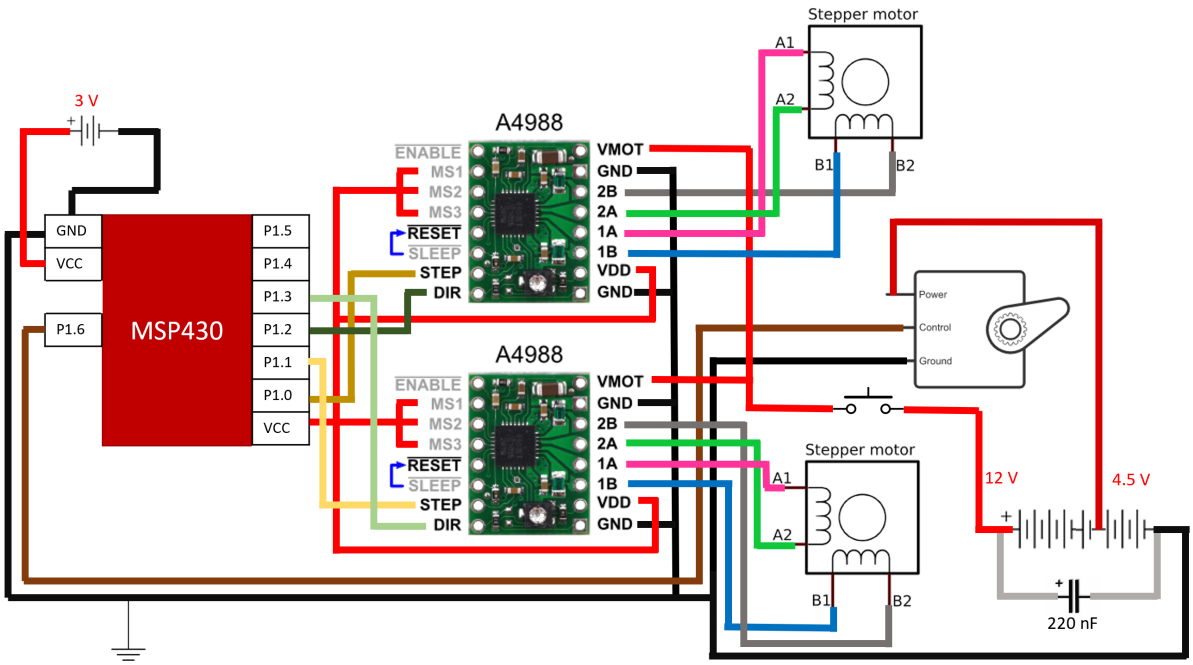

Figure 8. Complete circuit diagram for the drawing robot. Shows the MSP430, all three power sources, both stepper motors (with their drivers), the servo motor for the pen deployment system, and all necessary connections between these.

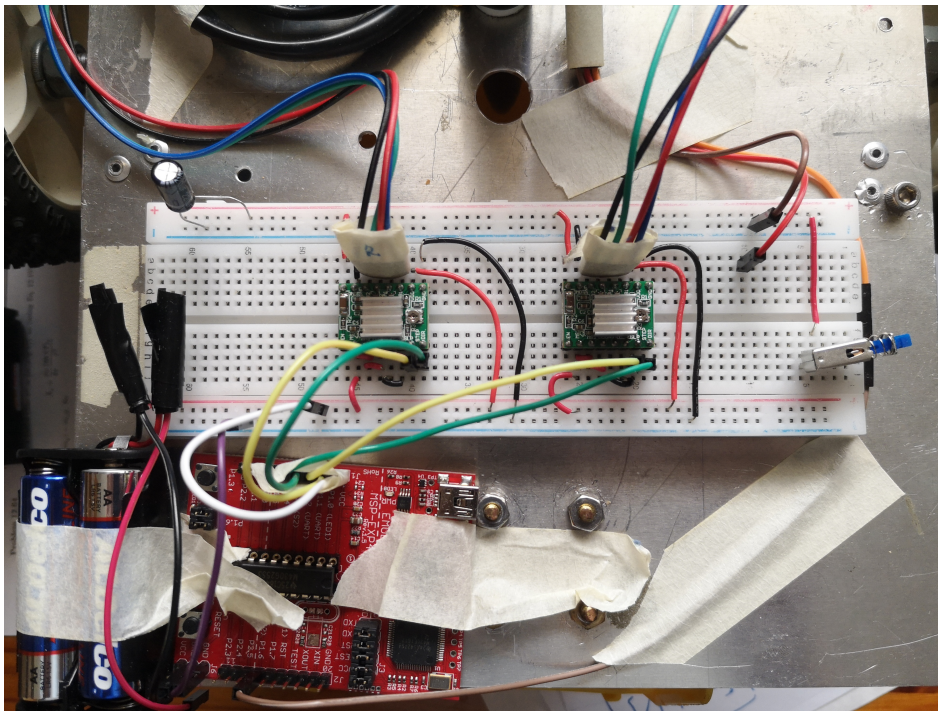

Figure 9. Closeup image of the drawing robot's breadboard and MSP430. The 3 V battery pack powering the MSP430's logic is also visible in the bottom left.

Figure 8 above shows the complete circuit diagram for this project. This section breaks down the explanation of the circuit into 3 components: power, stepper motors, and servo motor.

3.2.1 Power

As discussed in the Theory section, the SG90 9G servo motor requires a 5 V DC voltage, while the 42HS40-1704 NEMA 17 steppers are powered by 8-35 V DC. Additionally, the MSP430 and the A4988 stepper motor drivers operate on a 3.3 V DC logic level, meaning the project requires three separate DC voltages in total. In first tests a 15 V DC stationary power supply was used for the stepper motors, and an external 5 V UBC power supply was used for the servo motor. The MSP430 was supplied 3.3 V from a standard USB

power adapter. This introduced complications once the robot evolved to a drivable state, as this system of 3+ wires became easily tangled.

The solution was to introduce a system of battery packs, seen in the final product in figures 6 and 9. Using standard AA batteries connected in parallel was the simplest way of achieving the desired voltages, as multiples of 1.5 V could easily be selected.

The MSP430 uses a standard x2 AA battery pack shown on the bottom left of figure 9, and on the top left of the circuit diagram in figure 8, providing $1.5 \times 2 = 3$ V for the microprocessor's logic. This voltage is supplied to each of the A4988 stepper motor drivers' VDD input pins as shown in the circuit diagram. Though this is not the standard 3.3 V, it is within the recommended operating voltage of 1.8-3.6 V [1].

A standard x8 AA battery pack is used to supply $1.5 \times 8 = 12$ V to the stepper motors through the A4988 drivers. The battery pack can be seen on the bottom right of the circuit diagram in figure 8, and is also shown below in figure 10:

Figure 10. Images of the x8 AA battery pack. The left image shows the top layer of batteries, while the right image shows the soldering of wires to the ground and +12 V leads on the pack. The orange wire visible in both images is the +4.5 V branch used by the servo motor.

Wires were soldered to the battery pack to extract the full 12 V DC from it. This left the issue of obtaining the 5 V needed for the servo motor. to obtain 5 V, a voltage divider circuit could be built from the 12 V battery pack, or alternatively, the voltage could simply be extracted from 3 batteries in the same pack, providing $1.5 \times 3 = 4.5$ V to the

servo. The latter was selected: though compared to a voltage divider circuit, this method would mean that three batteries of the eight in the pack would discharge faster, it was an arguably simpler solution given the time constraints. To accomplish this, a wire was soldered to a lead on the battery pack that connects to the 3rd battery from the ground. Note that it is important to ensure that both voltages share the same ground (and hence the same ground wire was used) to avoid shorting the battery pack by connecting different grounds together (the plastic pack melts rather quickly...). A 220 nF capacitor is added between the 12 V and ground lead, which is recommended to stabilize the voltage delivery to the stepper motors [9]. Finally, a pushbutton switch is added to the 12 V lead to turn the stepper motors on and off, to allow for diagnosing the pen raising/lower mechanism separately while saving battery power.

3.2.2 Stepper motors

The minimal wiring diagram found on the Pololu Robotics and Electronics website [9] was used as the basis for controlling the two NEMA17 stepper motors with A4988 stepper motor drivers. The circuit from the website was extended to include two motors and drivers. As shown in figure 8, each driver is supplied with a 3 V logic voltage to VDD and a 12 V voltage to VMOT. As explained in the Theory section, they motors are driven in the direction specified by the DIR pin by pulses sent to the STEP pin. The left stepper motor's driver's STEP and DIR pins are connected to MSP430 pins P1.0 and P1.2 respectively, while the right's driver's pins are connected to P1.1 and P1.3 respectively, as shown in the circuit diagram. A4988 pins 1A, 1B, 2A, and 2B are connected to the appropriate pins on the two phases of each motor. The reset and sleep pins are hard wired together to simply keep the motors on at all times. Finally, the MS1, MS2, and MS3 pins controlling the microstep resolution are also hardwired together and connected to the 3 V logic, in order to increase the accuracy of the robot by changing the amount each motor steps of $1/16^{\text{th}}$ of a step (see the Theory section for more details).

3.2.3 Servo motor

The Sharpie raising/lowering mechanism's servo is connected to its 4.5 V power source, drawn from the 12 V battery pack as explained in section 3.2.1 (with a second lead connecting the motor to ground). The brown wire shown in figure 8 connects the servo's control pin to the MSP430's P1.6 pin, from which a PWM is used to send a pulsed 20 ms period 3 V signal to the motor, with the pulse width controlling the angle of its horn.

3.3 Software

Now that the hardware and circuit was in place, the robot could in theory be programmed to draw any two dimensional doodle. This section focuses on describing a systematic program structure composed of five functions that can be used to easily and legibly program the robot to draw any desired dooble. The main idea was to create a framework for controlling the direction and duration of the movement of each stepper motor by sending short pulses to the drivers, while also controlling the raising and lowering of the pen.

Please refer to the appended code (section 8.1) which uses the functions I describe in this section to direct the robot to write my name in Russian. It involves a composition of five functions, whose summary is given in figure 11 below:

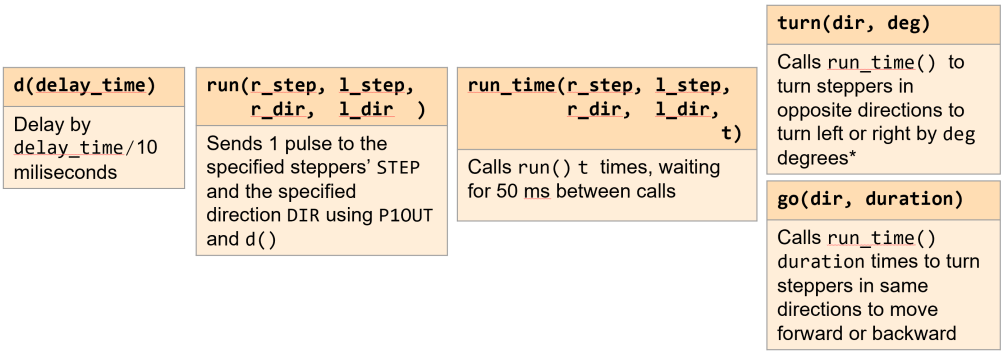

Figure 11. Summary of the five functions used to create instructions for the drawing robot.

3.3.1 Stepper motor control

The main function controlling the stepper motors is `run()`. Instead of using a PWM signal, which could be implemented here in the future, the `run()` function simple sends a

pulse using `P1OUT`: First, the correct set of pins for `P1OUT` is built based on the inputs to the `run()` function. More specifically, `on` and `off` bytes are first initialized as `0b00000000`. Then, each if statement adds numbers to these bytes based on whether or not a particular stepper needs to move, and which direction it needs to move in. Once the `on` and `off` bytes are built, they are sent out by the MSP430 using `d(10); P1OUT = on; d(10); P1OUT = off;`, i.e. there is a 1 ms pulse sent to both of the motors' STEP and DIR pins simultaneously.

The `run_time()` function simply extends the functionality of the `run()` function to send multiple pulses to the motors. Technically, this would be sufficient for beginning to program simple movement of the robot. However, two more programs were created to improve readability: `turn()` and `go()` functions call on `run_time()` using the appropriate stepper motor directions to either move the robot forward/back or left/right (in place) for a particular amount of time.

Notice that the input to the `turn()` function takes a `deg` variable, a number related to the number of degrees the robot should rotate on its axis by. This variable had to be calibrated to match the desired number of degrees with the number of microsteps the robot should take to accomplish said rotation. The calibration was done by instructing the robot to rotate by 90°, and fine tuning the number of microsteps until this rotation was exactly a right angle. The following relationship was established:

$$S = \frac{2100}{90}\,\theta$$

Where S is the number of microsteps the robot should make, and θ is the desired turn angle in degrees.

A few standard angles were hard coded as constants in the code and were sufficient for writing my name. For instance, `#define DEG45 1050` is a constant that can be used in the code for specifying a 45° rotation.

3.3.2 Servo motor control

The MSP430's PWM function was used to control the servo motor responsible for raising and lowering of the Sharpie pen onto the drawing surface. The code for accomplishing

this was adopted from a github repository [11]. The code was simplified to use one servo instead of two, and to use the same internal clock that we used in class. As discussed in the Theory section, the servo is sent a PWM signal with a period of 20 ms by setting `CCR1`, and the pulse width is modified by changing `CCR1` to control the angle the servo makes. Two constants, `UP` and `DOWN` were set to 750 and 1500 respectively, corresponding to the pulse width for an angle that raises and lowers the Sharpie pen onto the drawing surface. 1500, i.e. a pulse width of 1.5 ms, corresponds to an angle of 0° of the servo's shaft [11], while 750 (0.75 ms) corresponds to about −45°, or a clockwise rotation from the 0° position. With this setup in place, raising and lowering the pen was as simple as setting `CCR1` to either `UP` or `DOWN`.

3.3.3 Sample code for drawing the letter M

```
CCR1 = DOWN;
go(FWD,1200);
turn(LEFT,DEG15);
CCR1 = DOWN;
go(BAC,900);
turn(RIGHT,DEG30);
go(FWD,900);
turn(LEFT,DEG15);
go(BAC,1200);
CCR1=UP;
```

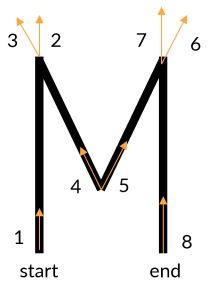

Figure 12. code and associated diagram illustrating how to code the robot to draw the letter M. The diagram on the right shows the position and orientation of the robot at each step in the drawing process.

Once the system described in the above sections was established, writing code for controlling the robot's movement became an incredibly intuitive and legible task. The above is a simple example of code that was used for drawing the letter M.

4 Results

Using the framework for creating drawing instructions discussed in section 3.3, I was able to write code for the robot to draw my (nick)name in Russian (using the Cyrillic alphabet): МИШИА.

I recorded two trials of the robot drawing my name. These can be found on youtube: Video 1, Video 2. The first video uses a USB power supply instead of the 3 V battery pack to power the MSP430's logic. The reason for doing a trial run with a 3.3 V USB power supply is described in the Discussion section. Images of the final results for each run are shown in figure 13 below:

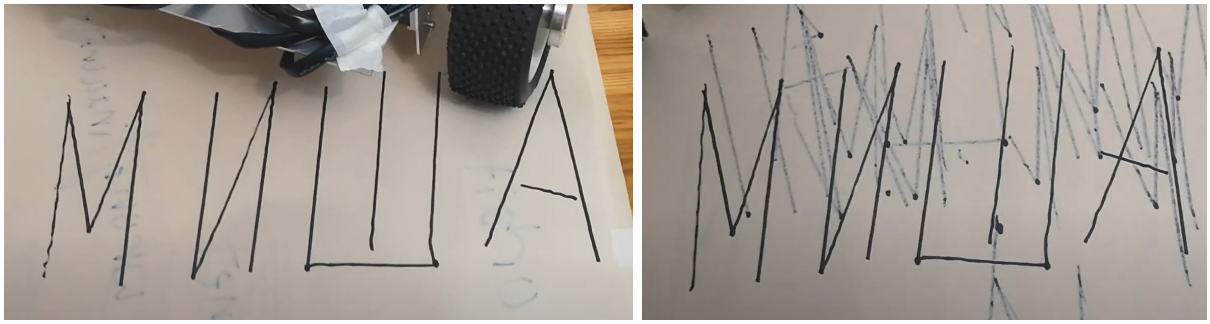

Figure 13. Images of drawings of my name in Russian created by the robot. Left: result of Video 1 (using USB power supply). Right: result of Video 2 (using x2 AA battery pack).

5 Discussion

Here we discuss the results presented in section 4, and address improvements and future directions for the project.

The project printed my name with a surprising degree of accuracy and reproducibility. A large source of error in the quality of the writing was the castor dummy wheel used at the front of the robot (shown most clearly in figure 5). The swiveling movement of this wheel shifted the robot slightly and prevented it from tracing exactly reproducible lines. While this could potentially be solved with complicated reverse kinematics, the simplest solution is to use a ball transfer unit; an example is shown in figure 14 below:

Figure 14. Image of a ball transfer unit that could replace the castor dummy wheel and serve as the simplest third contact point of the robot with the ground. [7]

This would remove the extra swivelling motion introduced by the dummy wheel.

Another observed issue could be the power supplies. Though a 3 V logic level should in theory suffice the MSP430 and the A4988 stepper motor drivers, it was clear from the results comparing drawings using the battery pack with a USB power supply (figure 13) that the voltage was not optimal for the stepper motor drivers' operation. The rougher and more slanted edges in the second image in figure 13 could be explained by the stepper motors skipping a few microsteps in their rotational motion. This issue becomes worse the more the battery pack discharges - one of the wheels would occasionally get jammed and fail to start moving, resulting in a complete failure of the drawing. These issues could be solved by using a single power source and selecting exactly 3.3 V from it using a voltage divider.

Improvements to the code could also be made to make it more universal. The current framework does not allow for the movement of the two stepper motors at different speeds, something that could be accomplished with the use of a PWM signal for each stepper. This would allow for drawing curves more easily. The step size of the servo could also be adjusted instead of simply hardwired at a $1/16^{\text{th}}$ microstep resolution. This could allow the robot to speed up when precision is not required, for instance for drawing straight lines.

Eventually, a program could be written for converting any image or G-Code into instructions for the drawing robot. I have started to work in that direction in Python (see figure 15 below). A suggestion made by one of the TAs (Bereket) in lab was the Turtle library [14], which would allow me to simulate the drawing that the robot would create.

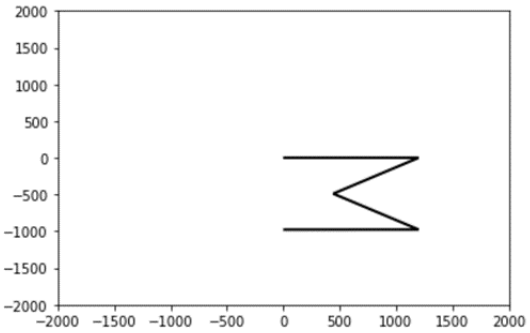

Figure 15. Plot of the letter M created in Python using the matplotlib.path library [5]. It uses a similar code structure to my robot, meaning any drawings could be tested here before being implemented into C for the MSP430.

6 Conclusions

I have always taken a great interest in building my own CNC type drawing machine (and even my own 3D printer), but never quite had all the proper resources to get started. So it has been incredible to finally see my idea come to fruition in the best way possible - in the form of a driving robot. Watching the robot flawlessly print out my name for the first time was very satisfying. Throughout this project, I learned about stepper motors and servo motors and how to correctly drive them with the MSP430. I got to use the Hantek's Oscilloscope function for verifying the signal that I was sending to the steppers and the servo. I learned about how to obtain different voltages from the same power source. I got a good opportunity to work with basic hand-tools, solder, and the drill press. In trying to take this project further than typing my name, I learned about reverse kinematics and various Python libraries for robot path control. Overall, the project was a very rewarding experience. I am looking forward to continuing to work on my drawing robot over this upcoming summer.

Finally, I would like to acknowledge some of the course TAs (Adam, Bereket and Yukiya), the technician (Sing Chow) and prof. Kotlicki for providing me with the guidance, experience, tools, and resources needed to accomplish this project. I would likely not have finished on time without them.

7 References

[1] A. Kotlicki, PHYS 319 site, "MSP430G2553 Data sheet (slas735)"

[2] Electronicos Calidas, "NEMA 17 42HS40 datasheet"

[3] Imperial College London, "SG90 9G servo motor datasheet"

[4] Kollomorgen, "Servo Motor Description"

[5] Matplotlib documentation, "Matplotlib.path"

[6] Mosaic Industries, "Stepper motor specifications"

[7] Omnitrack, "omnidirectional wheels and ball transfer units"

[8] Pololu Robotics and Electronics, "A4988 stepper motor driver datasheet"

[9] Pololu Robotics and Electronics, "A4988 stepper motor driver description"

[10] Robotdigg, "NEMA 17 42HS40 engineering drawing", ROBOTDIGG Equip Makers

[11] S. Wendler, "MSP430 PWM Servo Sample"

[12] StepperOnline, "NEMA 17 Stepper motor"

[13] Wikipedia contributors, "G-Code", Wikipedia

[14] Python 3.10.4 documentation, "Turtle"

8 Appendix

8.1 Code for writing my russian nickname МИШИА

```
#include "msp430.h"

// Definitions of a few constants to improve readability
#define R_F 1
#define L_F 1
#define L_B 0
#define R_B 0
#define DOWN 1500
#define UP 750
#define LEFT 1
#define RIGHT 0
#define FWD 1
#define BAC 0
#define DEG90 2100
#define DEG45 1050
#define DEG30 700
#define DEG15 350

// Purpose: delay the program by delay_time/10 ms
// Input: delay_time, an integer specifying how long to delay for
void d(int delay_time){
    int i=0;
    for (i=0; i<delay_time; i++){
        _delay_cycles(100);
    }
}
```

```
// Purpose: Sends 1 pulse to the specified steppers' STEP
//     in the specified direction DIR using P1OUT and d()
// Input:  r_step - 1 or 0: step right stepper?
//         l_step - 1 or 0: step left stepper?
//         r_dir - 1 or 0: right stepper direction
//         l_dir - 1 or 0: left stepper direction
void run(int r_step, int l_step, int r_dir, int l_dir){
    int on = 0b00000000;
    // Assemble the command we want to send to P1OUT
    // using conditions based off the 4 inputs.
    int off = 0b00000000;
    if(r_step){
        on+= 0b00000001;
    }
    if(l_step){
        on+= 0b00000010;
    }
    if(r_dir!=R_F){
        on+= 0b00000100;
        off+= 0b00000100;
    }
    if(l_dir==L_F){
        on+= 0b00001000;
        off+= 0b00001000;
    }
    // Send the pulse:
    d(10);
    P1OUT = on;
    d(10);
    P1OUT = off;
}

// Purpose: call run() t times, waiting for 50 ms between calls
// Input: same as run() above and t, an integer for the number of times to call
void run_time(int r_step, int l_step, int r_dir, int l_dir, int t){
    int count = 0;
    for(count = 0; count<t; count++){
        run(r_step,l_step,r_dir,l_dir);}
    d(500);
}

// Purpose: Calls run_time to turn steppers in opposite directions
//     to turn left or right by deg degrees*
// Input:  dir - specify direction to turn in, LEFT or RIGHT
//         deg - specify how much to turn by, in predefined angles above
//             (90DEG or 30DEG for example)
void turn(int dir, int deg){
    if(dir == LEFT){
        run_time(1,1,R_F,L_B,deg);}
    else{
        run_time(1,1,R_B,L_F,deg);
```

```
    }
}


// Purpose: Calls run_time duration times to turn steppers
//     in same directions to move forward or backward
// Input:  dir - specify direction to move in, FWD or BAC
//         duration - specify the number of steps to take in that direction
void go(int dir, int duration){
    if(dir == FWD){
        run_time(1,1,R_F,L_F,duration);
    }
    else{
        run_time(1,1,R_B,L_B,duration);
    }
}

void main(void) {
// P1.0 to R_STEP
// P1.1 to L_STEP
// P1.2 to R_DIR
// P1.3 to L_DIR
WDTCTL = WDTPW + WDTHOLD; //Stop WDT
P1DIR = 0b01001111;       //Set servo and steppers' output pins to 1
P1SEL |= BIT6;            // Add PWM functionality to P1.5 (for servo motor)
P1OUT = 0b00000000;       //Set all pins to OFF at the start

// Setting up for Servo:
// if SMCLK is about 1MHz (or 1000000Hz),
// and 1000ms are the equivalent of 1 Hz,
// then, by setting CCR0 to 20000 (1000000 / 1000 * 20)
// we get a period of 20ms
CCR0 = 20000-1;                          // PWM Period TA0.1

// setting 1500 is 1.5ms is 0deg. servo pos
CCR1 = 1500;                             // CCR1 PWM duty cycle

CCTL1 = OUTMOD_7;                        // CCR1 reset/set
TACTL  = TASSEL_2 + MC_1;                // SMCLK, up mode

d(50000); // Time to place down the robot and safely move away


// Letter M                              turn(LEFT,DEG15);
CCR1 = DOWN;                             go(BAC,1200);
go(FWD,1200);                            CCR1=UP;
CCR1 = UP;
turn(LEFT,DEG15);                        // Moving to the next letter
CCR1 = DOWN;                             turn(RIGHT,DEG90);
go(BAC,900);                             go(FWD,300);
turn(RIGHT,DEG30);                       turn(LEFT,DEG90);
go(FWD,900);
```

```
// Letter (backwards N)
// (cerilic for "i" sound)
go(FWD, 1200);
CCR1 = DOWN;
go(BAC, 1200);
turn(RIGHT, DEG15);
go(FWD, 1242);
turn(LEFT, DEG15);
go(BAC, 1200);
CCR1 = UP;

// Moving to the next letter
turn(RIGHT,DEG90);
go(FWD,300);
turn(LEFT,DEG90);

// Letter "sh" in cerilic
go(FWD, 1200);
CCR1 = DOWN;
go(BAC, 1200);
turn(RIGHT, DEG90);
go(FWD, 322*2);
turn(LEFT, DEG90);
go(FWD, 1200);
CCR1 = UP;
turn(RIGHT, DEG15);
go(BAC, 1242);
turn(LEFT, DEG15);
CCR1 = DOWN;
go(FWD, 1200);
CCR1 = UP;

// Moving to the next letter
turn(LEFT, DEG15);
go(BAC, 1242);
turn(RIGHT, DEG90 + DEG15);
go(FWD, 300);
turn(LEFT, DEG90);

// Letter "A"
turn(RIGHT, DEG15);
CCR1 = DOWN;
go(FWD, 1242);
turn(LEFT, DEG30);
go(BAC, 1242);
CCR1 = UP;
go(FWD, 414);
turn(LEFT, DEG90-DEG15);
CCR1 = DOWN;
go(FWD, 428);
CCR1 = UP;
turn(RIGHT, DEG30);
go(FWD, 2500);
turn(RIGHT, DEG90-DEG30);
}
```