Lab 6:CNC Machine Report

By: Jason Russo (jr826), Eldor Bekpulatov (eb654)

Video Link
https://youtu.be/Z8gxHZzyqMM

Introduction
For our final project we built a small scale CNC Machine. Instead of milling however, we decided to attach a pen to draw on a piece of paper placed on the bed of the machine. Our machine makes use of two bi-polar stepper motors each with DRV8825 drivers to move the pen along the X and Y axes. We also attached a positional servo to add the third dimension control, pens position, in our case a simple binary, up or down. The FRDM K-64 microcontroller is used to operate the motors and control the system in order to draw the desired image.

In the end, we created a system that uses two stepper motors to control the X and Y movements of the pen and a servo to raise and lower the pen. We provide an input with the FRDM K64 board in the form of motor movements. We were able to properly control our motors using the mbed libraries and C++ code, which is described below.

Hardware Description
Our initial goal was to build something that can draw on 8.5”x11” paper. After considering the costs and the manual labor needed to build such a machine, we decided to scale down our physical dimensions.

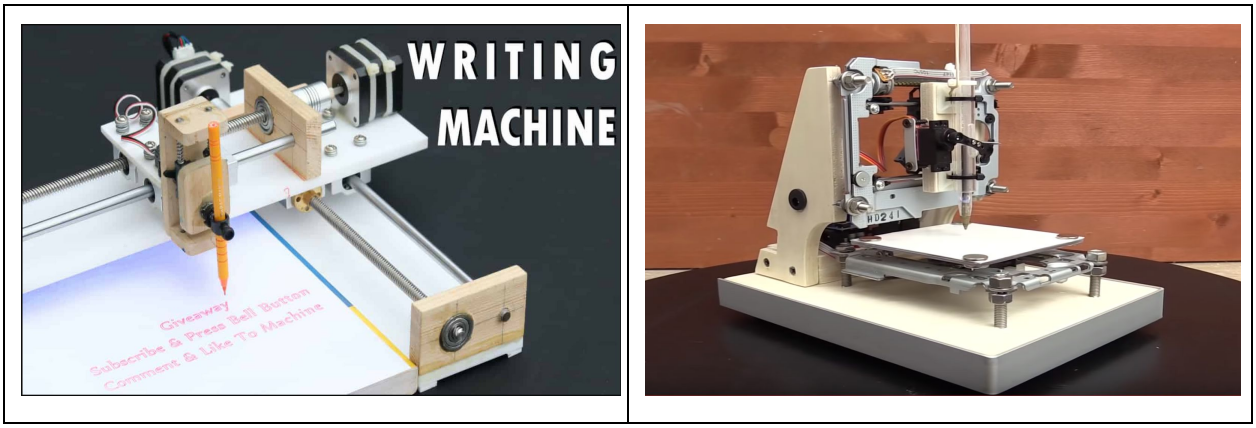

Figure 1. Large scale vs Small scale CNC machines.

To keep costs low, we ordered salvaged parts from old PC’s, specifically the DVD ROM drives, since they came with bi-polar stepper motors along with all the structures needed to turn circular motion into linear motion. On the table below you can find all the materials/devices purchased. Other, materials/tools used on the project came either from us or MakerSpace.

Name	Vendor	Count	Unit Price
DRV8825 Stepper Motor Driver 5pcs	Amazon	1x	$9.99
Micro Stepper Motor with Slider	Amazon	1x	$11.99
HP DVD-ROM Optical Drive	eBay	2x	$9.10
M4 x 50mm Hex bolt 10pcs	eBay	1x	$7.00
Nylon Insert Hex Lock Nut 10pcs	eBay	2x	$3.84
Total Cost			$54.86

Table 1. Table of materials with expenses.

We went through the DRV8825 stepper motor driver's spec sheet and identified the pin schematics, different modes of operation, and the limits of current that can be controlled through a on-chip potentiometer. There were also a lot of online forms and videos that helped with this specific information.

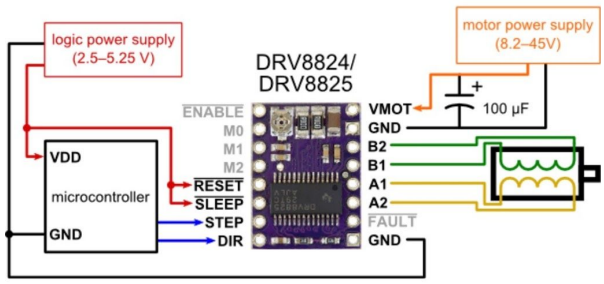

Figure 2. Pin schematic for DRV8825.

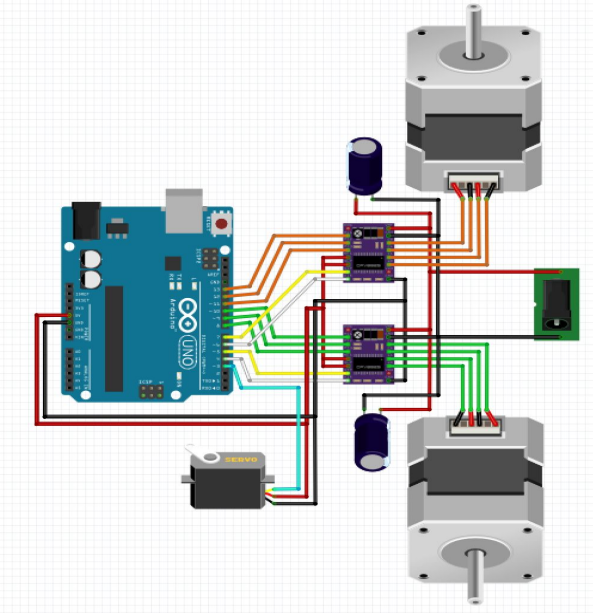

Figure 3. Full project wire diagram.

In order to speed up our building process, we initially started off testing individual components on Arduino boards. Once we collected certain parameters, we moved onto simulating the same output using FRDM boards. Figure 3 shows our fully wired diagram. Once we were happy with our wiring on breadboard, we soughtered our components onto a protoboard. Figure 4 shows our modular protoboard.

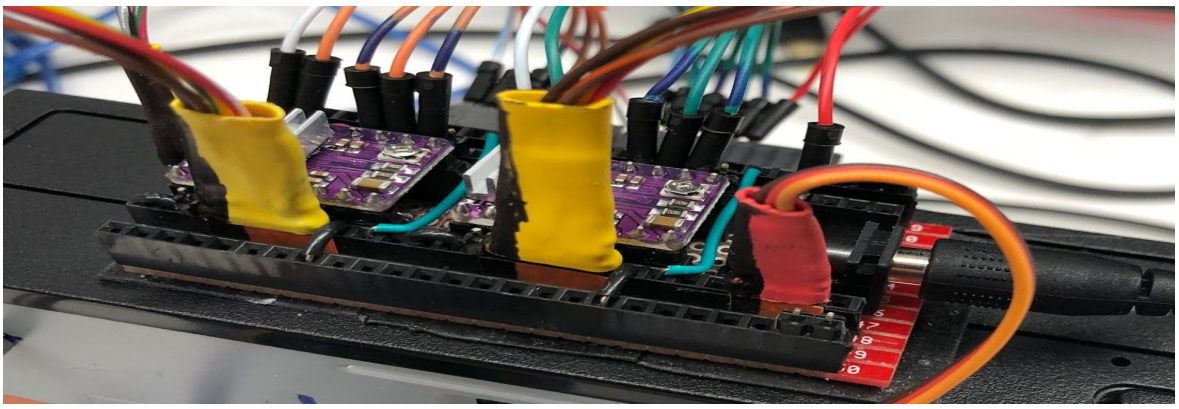

Figure 4. Our controller system on proto-board.

We then used the salvaged parts and some 3D printed parts to create the base structure and the penholder mechanisms. Figure below shows the finished hardware of the project.

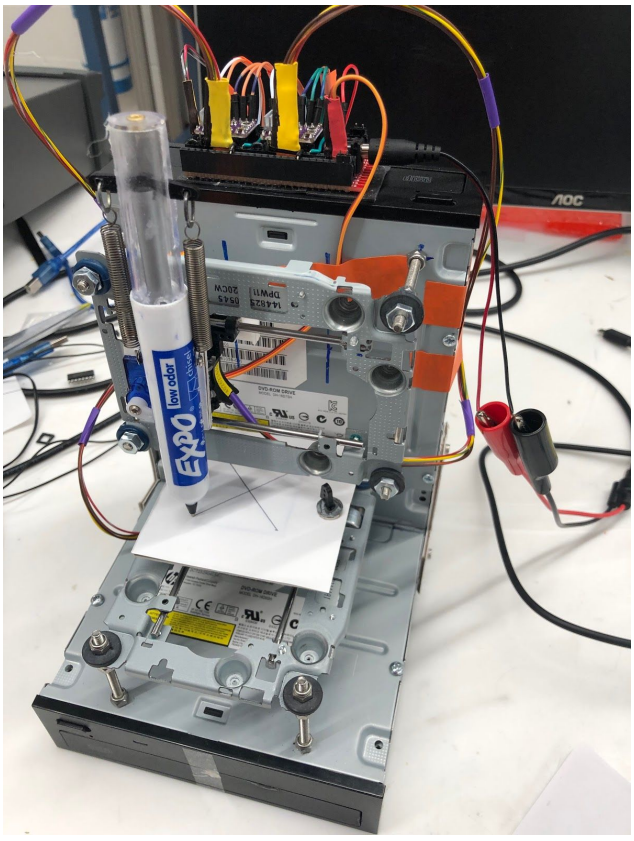

Figure 5. Our CNC machine.

System Diagram

Our coding is done in C++ to leverage the mbed Servo libraries. Because we are in C++, we use a Stepper class to interface with our two stepper motors. The class is as follows:

```
class Stepper{
      public:
            DigitalOut *pinM0;
            DigitalOut *pinM1;
            DigitalOut *pinM2;
            DigitalOut *pinDir;
            DigitalOut *pinStep;
            PinName     M0;
            PinName     M1;
            PinName     M2;
            PinName     Dir;
            PinName     Step;
            float mode;                 // = 1/[1,2,4,8,16,32]
            float minStepDelay;         // = 1ms equivalent
            float minImpulseDelay;      // = mode * stepDelay
            float curr_pos;             // = [0.0 - 250.0]
            int curr_direction;         // = 1 || 0
}
```

We then have a **current_status** structure that tracks our current X and Y positions as well as whether or not the pen is raised. The X and Y positions are pointing to globally declared instances of Stepper's **curr_pos** attribute.

```
struct current_status{
      float *current_x;     // points to x->curr_pos
      float *current_y;     // points to y->curr_pos
      int current_pen;      // 1 || 0
}typedef current_t;
```

We also used publicly available library for controlling the servo, and made an instance of the variable globally available.

```
GLOBALS:
Stepper *y = new Stepper(PTB23,PTA1,PTB9,PTC16,PTC17,0.03125,ONE_MS, 0.0,1);
Stepper *x = new Stepper(PTD3,PTD2,PTD0,PTC12,PTC4,0.03125,ONE_MS, 0.0,1);
current_t current_position = {&(x->curr_pos), &(y->curr_pos), 0};
Servo myservo(PTD1);
```

Software Description

In order to interface with the stepper motors, we used GPIO pins on the microcontroller. Each DRV8825 driver takes 5 digital inputs from the microcontroller: M0-M3, STP, DIR.

- The drivers have three pins used to set the MODE of the driver. 6 GPIO pins are connected to drivers as inputs, two 3-bit encoding to each one of the drivers. We decided to use 1/32 step setting to reduce noise and choppiness on movements. The 1/32-step configuration corresponded to **`0b101`**.
- The motors step when the STP pin of the driver receives a rising edge. When the STP is toggled HIGH then LOW with appropriate delays in-between, the motors will start stepping with a given mode of steps.
- Another GPIO pin is connected to the DIR pin of the motor driver, which dictates the direction of the stepper's rotation. In our case, this meant movement either in left-right(x-axis) or back-forth (y-axis) directions.

We saw the need to declare **`mode`** as an attribute in Stepper class, since **`mode`** was a defining attribute to many others such as timing, length, and position. Timing proved to be extremely important when driving steppers. We needed appropriate wait times for the motor steps to match the frequency of the simulated impulses. So, we defined an attribute in Stepper called **`minStepDelay`** which corresponds to the speed of the motor (time per step). We implemented a function called **`impulse(Stepper stepper)`**, which sets the GPIO pin corresponding to **`stepper->pinStep`** to HIGH then LOW with appropriate delays in-between to simulate a rising edge/square wave.

`Helper Functions:`

To move our servo motor, which raises and lowers the pen, we use the functions **`penUp()`** and **`penDown()`.** These functions use the mbed Servo libraries to interact with our **`Servo`** instance. They update the PWM functionality by writing into the servo instance and update our global **`current_position->pen`** field.

To move our stepper motors, we have an **`impulse()`** function, which takes in a pointer to one of our steppers as an input, and toggles **`pinStep`** twice, giving us a rising edge. Since, we are using microsteps our **`minImpulseDelay`** is the product of **`stepDelay`** and **`mode`**.

The **`initializeStepper()`** function sets the initial values to MODE and DIR pins. This function checks the **`stepper->mode`** field and assigns values to the mode bits accordingly. Also, it initializes **`stepper->pinDir`** to 1 always because that is the positive direction.

Our implementation features three move functions: **`move()`**, **`moveXY()`**, and **`moveTo()`**. The main function is **`moveTo()`** which calls the other move functions. This function takes in the desired X and Y coordinates and calculates the slope of the line from the current position to the goal position. If the slope = INF (no x movement) we call **`move()`** on the y motor. If slope = 0 (no y movement), we call **`move()`** on the x motor. Otherwise, if both motors are moving, we call **`moveXY()`**. The **`move()`** function handles one motor. It takes in a motor, a goal coordinate, and a time to get there. We then set the direction and step the motor, using the time input to determine our delay. The **`moveXY()`** functions moves both motors together. We do this by seeing how many impulses each motor needs to move, then finding the greatest common divisor of these numbers. We then divide the number of impulses by the GCD to get how many each will move before the other does. I.E. if y needs 20 impulses and x needs 10, then y should have 2 impulses for every one of x's. We do this in a while loop that runs until we hit our minimum number of impulses. In that while loop, there are two for loops which will increment each motor by impulses/GCD at a time before letting the other motor move.

Testing

When first testing our GPIO, we used an oscilloscope to check the values of the pins as we toggled them. Once we were confident we could control our GPIO pins, we incorporated the motors. We first took some time to calibrate the motors. Once they were calibrated to our system, we tested them individually, simply moving them back and forth. Now confident that we could step our motors, we then focused on the servo. Because we used the Servo library, moving the servo was easy. Tuning it to consistently raise and lower the pen appropriately was more difficult. We did this through trial and error. Now that we could move our pen and motors, we gave simple input drawings to fully test the system. We tested simple vertical or horizontal lines with only one motor moving at a time. We then tried more complex shapes including diagonal lines with both motors moving together.

Results and Challenges

We succeeded in creating a CNC machine that can draw an image using straight lines. We failed, however, to create a way to input an image to the system. As we proceeded with the project, we realized this would be very difficult to do, and we had already suffered many setbacks, so decided to move forward without this feature. We ran into many hardware issues which were resolved through unit testing and ordering last minute replacement parts. Software problems came about from using both C and C++ code. In the end, we decided to exclusively use C++ in order to keep using the Servo library. Also, because we used mbed, space became an issue. We ended up being able to write all our code without going over on space, but it prevented us from drawing highly complex shapes.

Work Distribution

This project was carried out as a pair. We met often in the Maker Lab in Phillips Hall to construct the system and code. The major components were: the physical system, the GPIO code, and the motor movement code.

Eldor handled the system construction. Jason focused more on the code. Both actively contributed to the coding, particularly the movement code. We each had chances to work on each system and both contributed to the overall design of both hardware and software.

References

The mbed Servo “Hello World” example was used as the basis of our Servo code to control the pen motor (https://os.mbed.com/cookbook/Servo). Other mbed libraries were used by importing mbed into our project.

Inspired by:
https://youtu.be/Q5ma1HDuotk
https://youtu.be/atkGcfnsK3A
https://youtu.be/opZ9RgmOIpc
https://tinyurl.com/y4v99kkc
https://tinyurl.com/y5wrm5nn
https://tinyurl.com/y4k52zl3
https://tinyurl.com/yc4o8kb4