# 3D LED Matrix

## Final Design Document

Stephanie Baldwin

Document Created 1/7/2025

Project Completed 5/19/2025

# Table of Contents

# 1. System Design

This project involved creating a 3D matrix of individually addressable RGB LEDs and its associated controller. The controller was manually programmed with a predefined set of patterns to display on the matrix, and users can change the pattern and brightness modifier for the LED matrix using pushbuttons in the controller circuit. The matrix itself consists of 40 10-long LED strands, giving it an overall size of 5x8x10 pixels. The backend of the matrix assembly houses power distribution lines that connect up to two power supplies to the eight columns of pixels. Five data lines are connected crosswise to the power connections, between rows of pixels and the controller.

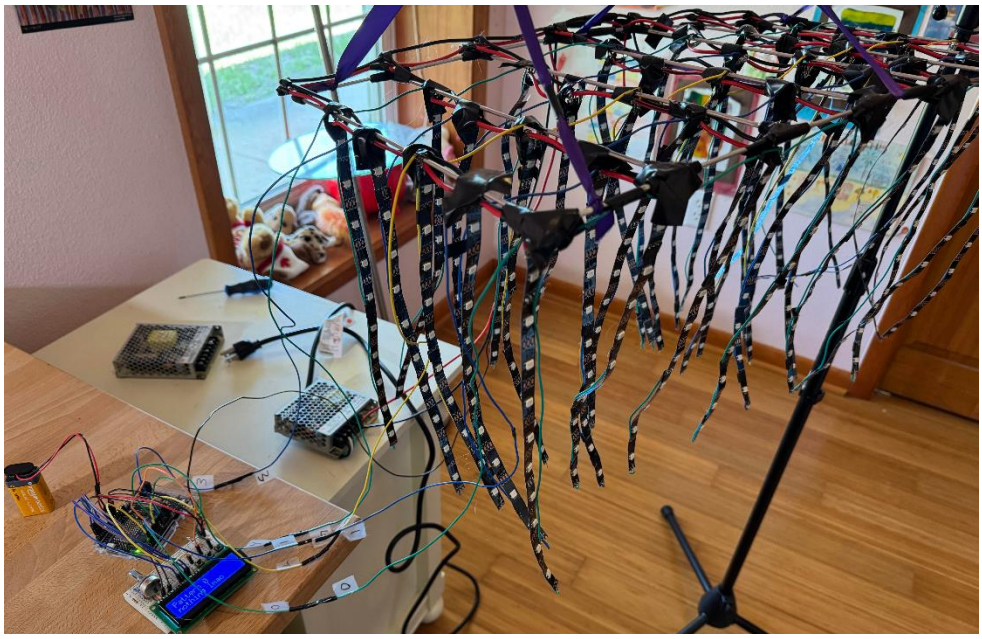

*Figure 1: Full As-Built System*

## 1.1 LED Matrix

The full matrix assembly consists of LED strips, the frame from which the strips are hung, and up to two external power supplies. The frame itself is hung from a suspended rod, and soldered wire connections are used to connect power from the supplies and receive data from the controller.

### 1.1.1 LEDs

The LED matrix consists of 40 LED strips, each of which has 10 individual pixels. The strips are arranged in a rectangular formation that has overall pixel dimensions of 5x8x10. Individual pixels are located using a 3D rectangular coordinate system, defined in Figure 2.

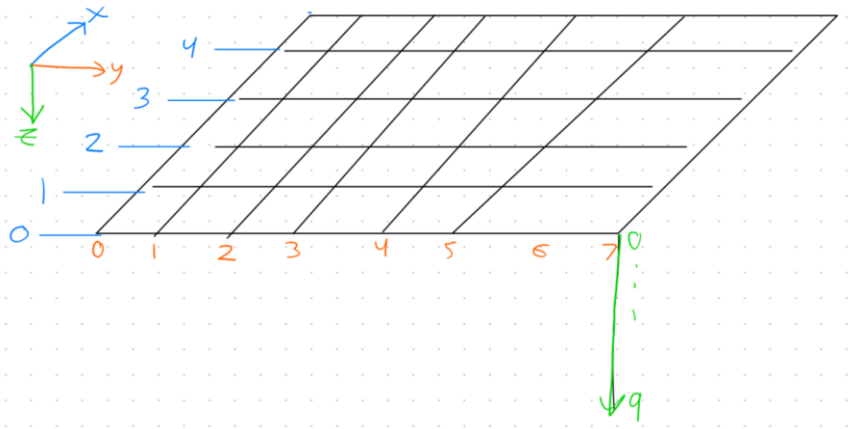


*Figure 2: LED Array Configuration*

Five data lines are wired parallel to the y-axis, and eight lines each for power and ground are wired parallel to the x-axis. The data lines "snake" down the backs of the strips as the stiffness of the wires themselves straightens out the strips (Figure 22), and 470 Ω resistors are placed on the data lines to reduce noise (Figure 3). The power and ground lines branch from a single line connected to the power supply and are connected to the tops of each strip with solder. Additionally, the ground line is connected to the ground of the control circuit so the data signal and the LEDs are using the same reference point. A second power supply can be utilized to allow for brighter settings without using a large power supply that would risk burning the wires (1.1.2.1 Optional Second Supply for High Brightness Settings); in this configuration, the branch between the fourth and fifth columns is broken and each supply individually powers the first or last four columns.

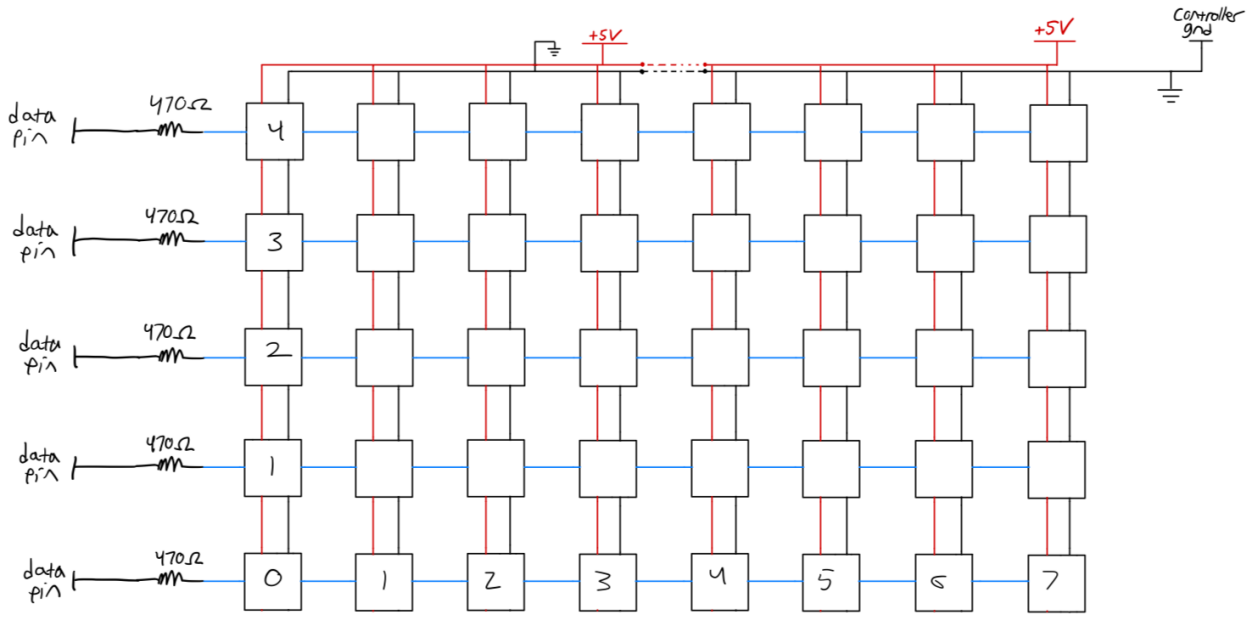


*Figure 3: LED Array Wiring Diagram (Top View)*

#### *1.1.1.1 Chipset Selection*

The LED strips used are WS2812 chips at a density of 30 LEDs/m. This chipset is slower than alternatives such as the APA102, but was selected because of its cost and accessibility. These chips only have three lines: +5V, ground, and data. A relatively low density of 30 LEDs/m was chosen because they make the 10-long strips used in the array 13.1 in long, which makes the array tall enough for pixels to appear as a distinct grid without manually altering the distance between pixels.

### 1.1.2 Power Supplies

Mean Well power supplies are used for this project. This brand was chosen because it provides a range of high-current 5V supplies and has sufficiently detailed documentation. These supplies have line, neutral, and ground terminals to connect to household wall power and +V and -V terminals for +5V and 0V connections to the LEDs.

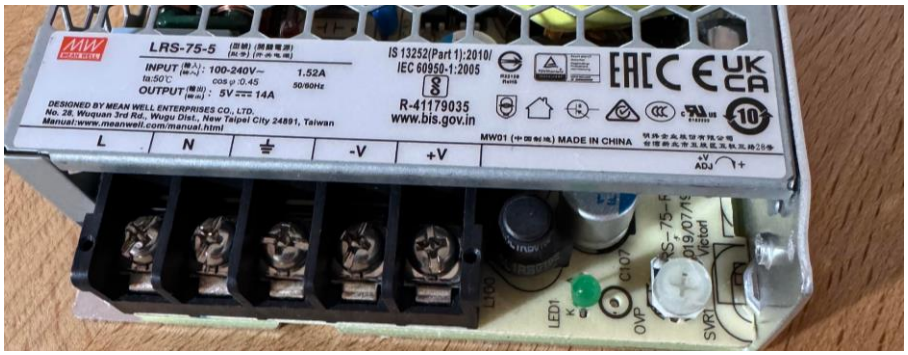


*Figure 4: Power Supply Terminals*

WS2812 strips typically draw 50-60 mA per pixel for white light at full brightness [1]. Power calculations were done with the more conservative 60 mA/pixel, and a 20% power excess was included in the calculations to ensure that voltage variations that may occur when the power supply operates at or near its maximum capacity won't damage the LEDs. It was assumed that the pixels are operating at exactly 5 V.

The array contains 400 pixels in total and thus requires 144 W to safely power (Equation 1).

$$400\ pixels * 60\ \frac{mA}{pixel} * 5\ V * 120\% = 144\ W$$

*Equation 1: LED Power Calculations*

To avoid being blinded, the brightness of the LEDs can be limited to 110 (out of 0-255), or 42% of the total brightness. With this limit, the required power drops to 60.5 W. For this case, a Mean Well LRS-100-5 100W or 75 W supply can safely power the entire structure [2][3].

### *1.1.2.1 Optional Second Supply for High Brightness Settings*

To safely allow the LEDs to operate at full brightness without overloading the wire connections, the power distribution setup allows for two supplies to be connected, with each supply independently powering half of the array (Figure 3). For this configuration, the two supplies used are the LRS-100-5 100W supply from the previous sections and a LRS-75-5 75W power supply. When using one supply, the connections between the halves of the array are held in place with a cylindrical shell made of 3D-printed PLA (Figure 5) and secured with pressure of the wire ends against each other and the shell walls (Figure 6). Extra solder was added to the wire ends to make the connection sufficiently secure when left stationary. This method is also used to connect power and ground lines to the supplies themselves.

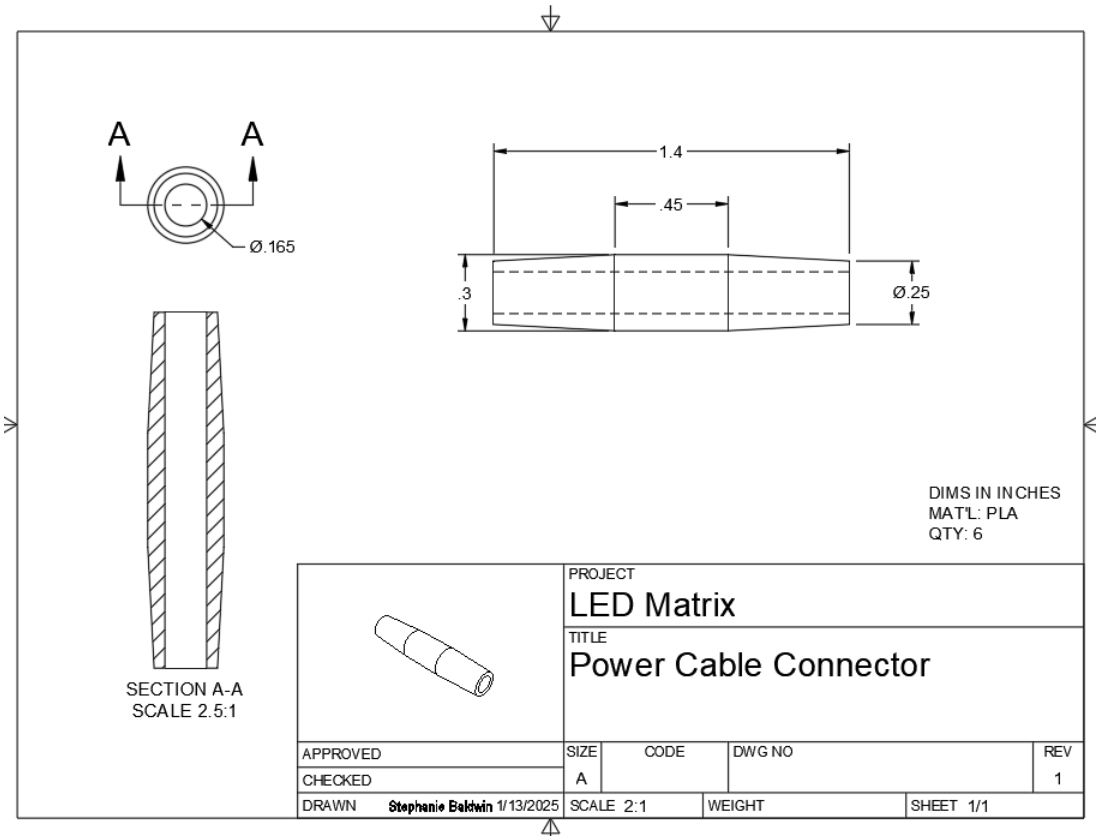


*Figure 5: Power Cable Connector Drawing*

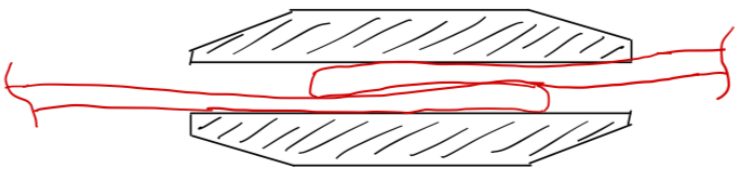

*Figure 6: Temporary Wire Connection Made With PLA Connector Piece (Black) and Wires (Red)*

### 1.1.3 Frame Structure

The frame was the structure from which the LED strips were hung. Power and data wiring interfaced with the LED strips along or across the wires making up the frame, with +5 V and ground travelling along the rods (red and black) and data travelling across the rods (green) (Figure 7). The frame is constructed from 1/8” diameter, 12” long Aluminum rods, and consisted of six total rods forming the outer rectangle (two rods were connected axially to form the longer sides), and six evenly spaced internal rods (Figure 8). This resulted in eight rows upon which the eight LED sets in the y-direction could be hung. Five strips were hung evenly along each of the rods from the outer edge of the frame. The other edge of the frame was left clear for the power wire branching (Figure 3).

*Figure 7: Frame Assembly with Wire Directions Noted*

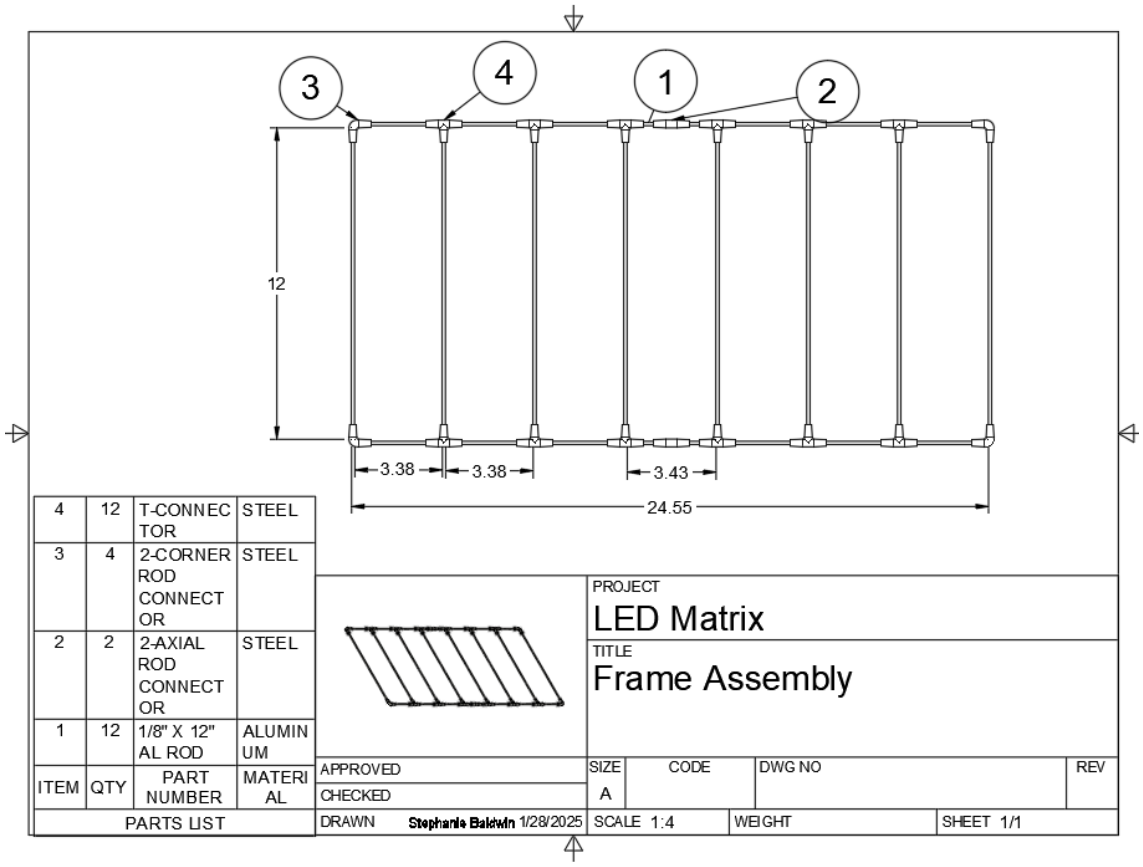


*Figure 8: Frame Assembly Drawing*

### *1.1.3.1 Rod Connectors*

Small cylindrical parts were used to position the rods correctly, and are what the epoxy adheres to in order to hold the frame together. These parts are 3D-printed with PLA with a 50% infill. Three types of connectors were created:

1. a cylindrical part to connect two rods axially at their ends (Figure 9).
2. a corner piece to connect rod ends at a 90˚ angle (Figure 10).
3. a T-shaped piece that one rod passes through and the end of another connects at a 90˚ angle (Figure 11).

Quick-dry hobbyist epoxy was then used to connect the printed pieces to the aluminum rods and assemble the entire frame. These connections were permanent.

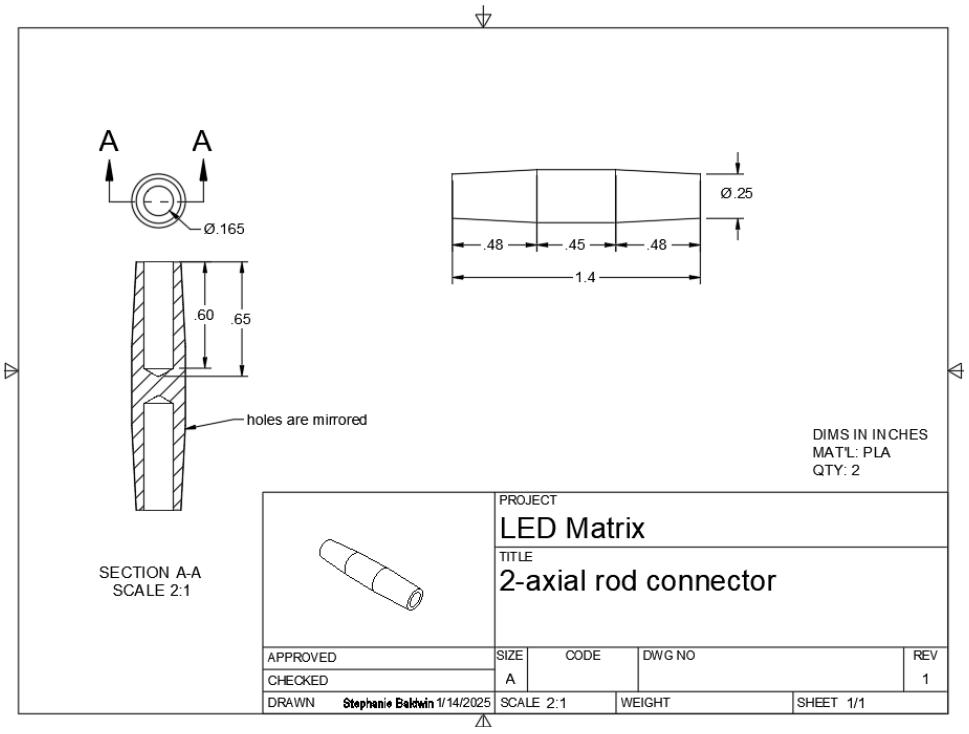


*Figure 9: Axial Rod Connector Drawing*

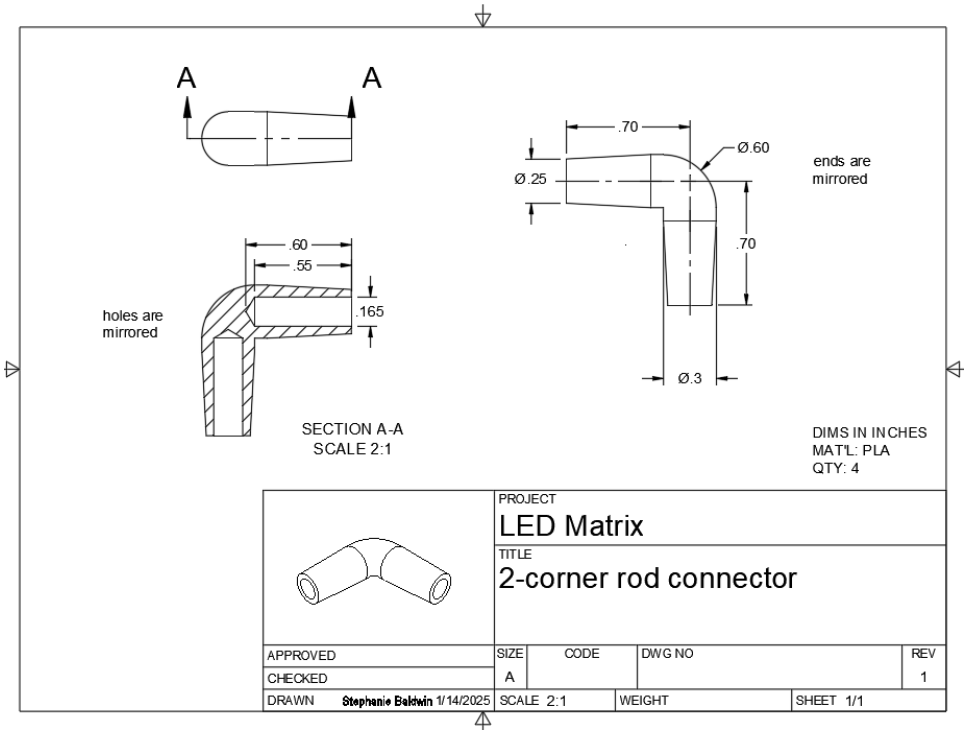


*Figure 10: Corner Rod Connector Drawing*

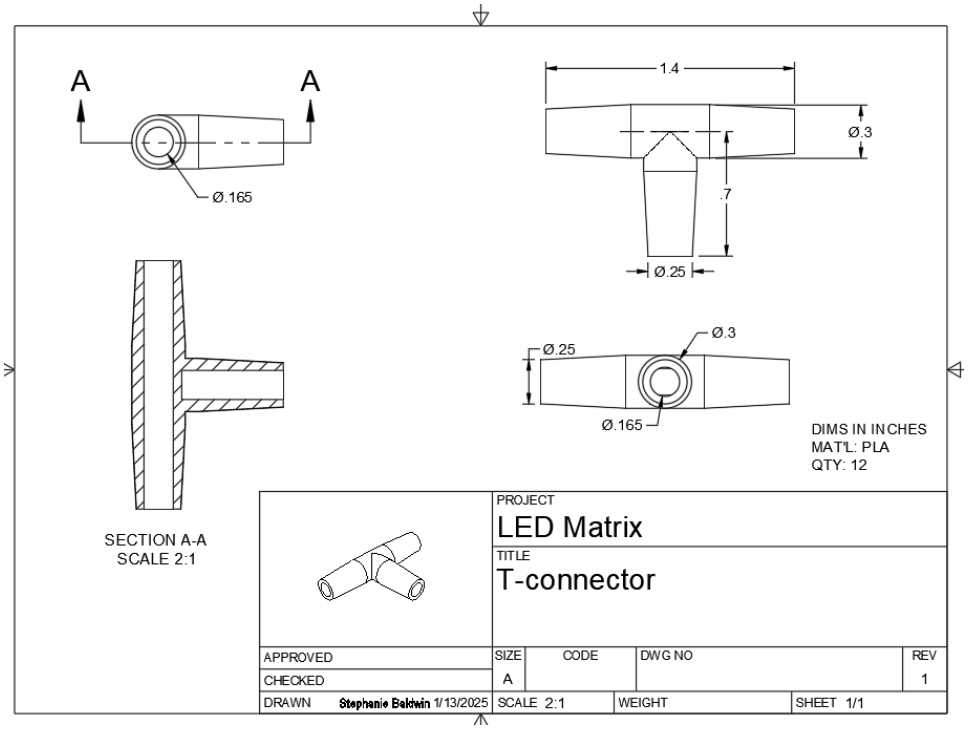


*Figure 11: T-Connector Drawing*

## 1.2 Controller

The controller circuit takes inputs from the user to change the currently displayed pattern or overall LED brightness, computes the necessary pixel settings to produce the appropriate pattern, interfaces with the data inputs of the LEDs to display that pattern on the physical LED array, and displays information about the current pattern number and name to the user via an LCD screen (Figure 12). All computations were done on the Arduino Uno R4 microcontroller. This controller was used because it is compatible with the FastLED library, which handles color gradients and low-level interfaces between the controller and LEDs, has sufficient memory capacity for the project, and was relatively inexpensive [4][5]. Patterns were associated with an integer starting from zero to identify them in the software and to the user.

User inputs were captured using three buttons, where one increments the pattern number, one decrements the pattern number, and one increases the total pixel brightness by a fixed amount. If a button was pressed such that the upper or lower bounds of available pattern numbers was exceeded or the maximum total pixel brightness was exceeded, the resulting value would loop to the opposite value; decreasing the pattern number from zero would change to the largest number and vice versa, and exceeding the total brightness would reset it to zero. The pixel brightness ranges 0-256, and the maximum brightness was set to 110 to avoid exceeding the capabilities of the power supply (one supply was used for testing; the maximum brightness could be higher if multiple supplies were used).

### 1.2.1 Circuit Structure

The physical circuit for the controller consists of three parallel loops connecting the three pushbuttons to the Arduino and to power/ground, and connections between the LCD display and the Arduino, potentiometer (to control the display contrast), and power/ground. Power (+5V) and ground (0V) for this circuit is supplied by the Arduino, and the Arduino itself is powered by the computer used to upload the program to it, or a 9V battery connected to its Vin and GND pins. The ground in this circuit is connected to the ground of the LED circuit to ensure that the Arduino and the LEDs are using the same reference point when sanding and reading the data signals (1.1.1 LEDs). Resistors were placed in the button loops to prevent shorting power to ground, and R1 was placed between VDD and the LED+ pin of the LCD screen to slightly reduce the voltage and limit the current to safe levels for the screen.

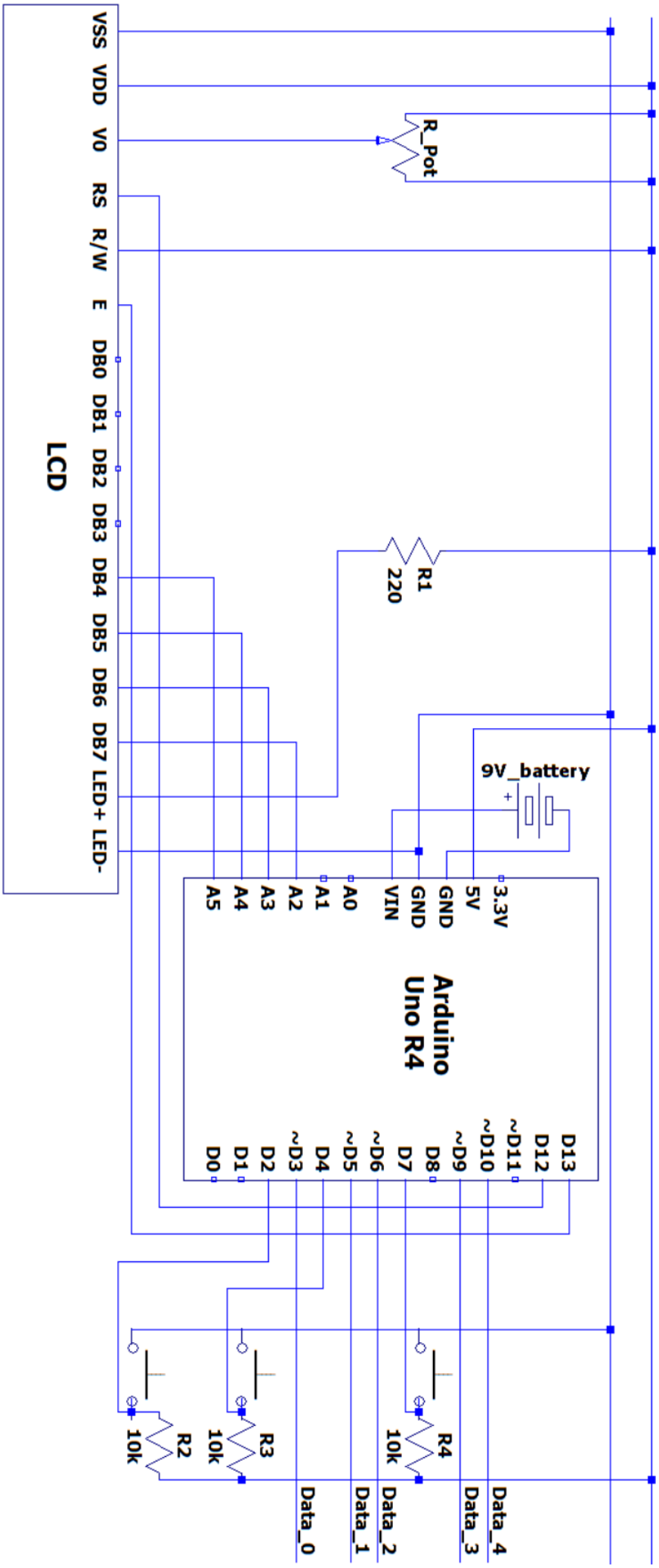


*Figure 12: Control Circuit Schematic*

### 1.2.2 Control Software

When the controller is powered on, it enters a setup stage in which the LCD, buttons, and LEDs are initialized and connected. A 3s delay is added in this stage so user input cannot interfere with component initialization. Once setup is complete, it enters the main control loop, where it will remain until the Arduino is powered off. All updates to the LCD and LEDs are timed such that the fastest something changes is at 60 fps. Once an update time is reached, the software will check the button state and change the current pattern or update the brightness if needed. Then, it will call a function corresponding to the current pattern to get the next frame of animation to display, update the LCD display if it requires scrolling, and show the updated frame of animation on the LED matrix. The scrolling animation was set to occur of 60 update cycles have passed or approximately once per second. The timings aren't exact because the processing time for computing and setting the new pixel colors may exceed the normal period between updates.

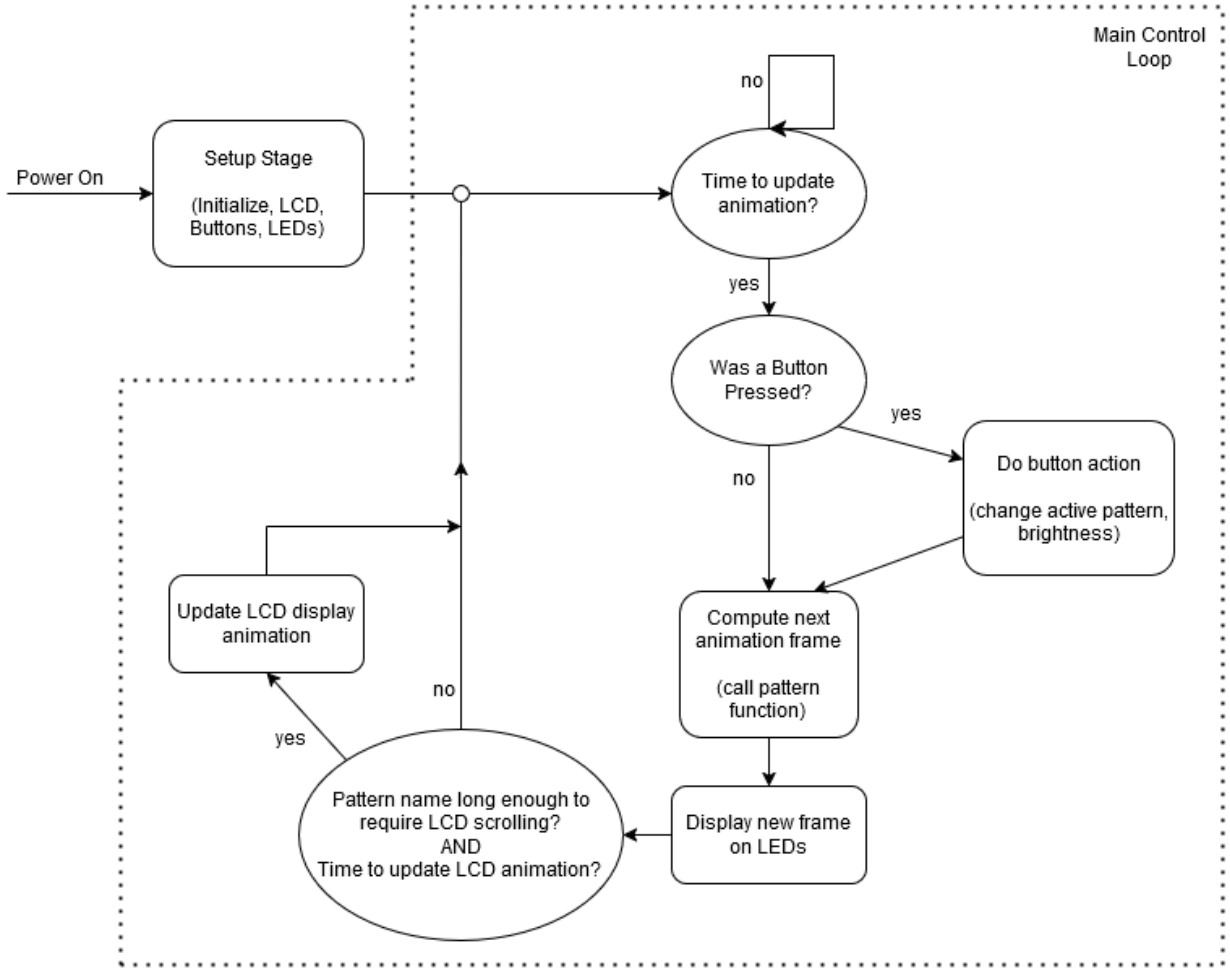


*Figure 13: Control Software Flowchart*

The project made use of external libraries to perform low-level work needed to interface with the various hardware components. FastLED was used to create color gradients and interface with the LEDs, LiquidCrystal was used to set and reset text on the LCD screen, and Bounce2 was used to define debounce periods that no not require delaying the program execution.

### 1.2.3 LED Patterns

Functions defining the behavior of each pattern are called for each LED update cycle. They use local pixel locations and position within the pattern's cycle of animation to compute the next colors for the LED array.

#### *1.2.3.1 Timing Implementation*

Counters were used to keep track of the frame of animation within the pattern. All patterns are displayed in a loop, restarting at the first frame after the last has been displayed. Two integer counter variables were created to fulfill this purpose. The first, named `cyclePosition`, keeps track of the frame of animation within the pattern, with 0 corresponding to the first frame. If a pattern changes at the same rate as the LED updates (~60 fps), `cyclePosition` is the only counter variable used. If a pattern changes less frequently, the second integer variable, named `frameCount`, keeps track of the number of LED update cycles that the current frame has been held for. This counter is incremented during each LED update, and once it reaches the value set for the specific pattern it is reset to zero and `cyclePosition` is incremented.

#### *1.2.3.2 Simple Patterns: Patterns 0-2*

Patterns 0 and 1 have no animation and simple display a uniform color, black for pattern 0 and red for pattern 1. The red CRGB color is scaled by the global brightness. Pattern 2 also displays uniform colors and thus doesn't take pixel coordinates into account. It cycles between red, green, and blue at approximately 2 fps, and selects a color based on its position in three total frames of animation. Colors for the entire array ware set by looping through the list of LEDs and setting its color using `CRGB()`.

#### *1.2.3.3 Patterns 3 and 4: Use of FastLED Palettes*

Patterns 3 and 4 used the Palette feature of the FastLED library to generate rainbow gradients. Pattern 3 displays a uniform color obtained from the built-in rainbow palette `RainbowColors_p`. FastLED palettes have 256 total colors, so the pattern was set to have 256 frames of animation. The `cyclePosition` variable was used to select a CRGB color from the palette such that the LED array color changes smoothly with time.

Pattern 4 uses a similar method, but offsets the local palette index for each column (xz-plane) by 32 to create a wave-like animation. If the offset index exceeds 255, the color

selected from the palette is the offset index subtracted by 256. Pixel colors are set by looping through an individual xz-plane and setting each pixel to the appropriate color.

#### *1.2.3.4 Pattern 5: 3D Wave*

This pattern aimed to create a virtual "wave" that flows diagonally across the array. This wave shape was created using a sinusoidal surface, the equation for which is shown. *A* is the wave amplitude, and was also used to offset the wave from the xy-plane. An amplitude of 2 was selected to make the wave tall enough to be easily visible and ensure that the wave doesn't intersect with the xy-plane. *v* is a parameter used to set the wave density, and is a coefficient of both x and y to make the wave travel diagonally across the xy-plane. A value of 0.6 was selected to make the pattern dynamic enough to be visually interesting, but not so dense that it appears as a solid block. *t* is the time value, and it simply changes the phase of the sinusoid to make the waves appear to move in space. Equation 3 shows how this *t* value is computed. To make the pattern animation consistent across loops, the total number of frames, 710, was selected such that at the end of the loop, $t$ is roughly a multiple of 2π (710/113). The `waveSpeed` parameter controls how quickly the wave appears to move, and was arbitrarily set to 6.

$$z = A\sin(vx + vy - t) + (A + 1)$$

*Equation 2: Sinusoidal Surface for Wave Shape*

$$t = \frac{waveSpeed * cycleposition}{113}, 0 \leq cycleposition \leq 710$$

*Equation 3: Computation for Time Variable in Wave*

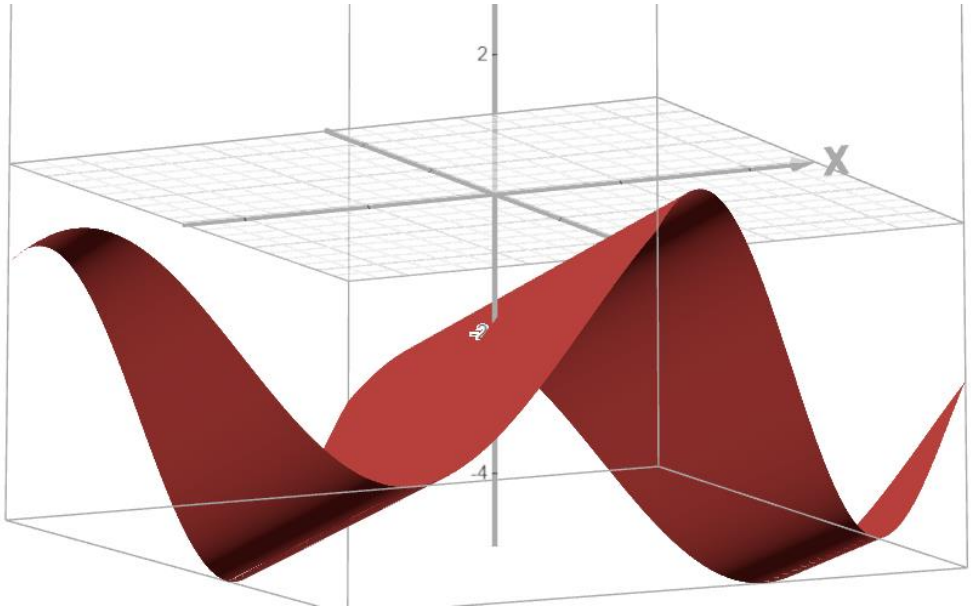


*Figure 14: Visualization of Wave (Made with Desmos)*

A palette that created a gradient from bright blue to black was used to color the wave. Pixels that most closely intersect with the sinusoidal surface are colored bright blue, and pixels beneath it are gradually tapered to black to create an effect of depth. The depth waveDepth, an integer measured in pixels that includes the top surface of the wave, could be changed and was set to 6 so the blue color reached the bottom of the LED array when the wave was at its lowest point. Brightness was also scaled linearly such that pixels further beneath the surface are dimmer.

Pixels were assigned a color sequentially by looping through the entire array. The array was initially set to black, then the loop was created. Equation 2 was evaluated at the current x and y values and the resulting z value was compared to the location of the current pixel. If that difference in z was less than some number `closenessThreshold`, that pixel would be considered to be part of the wave surface. An integer parameter `waveValue` was used to keep track of the vertical distance relative to the bottom of the wave gradient. When a pixel on the surface is found, `waveValue` is set to the largest depth, `waveDepth`. Looping through the array is performed by nesting loops for all possible x, y, and z values, so the next iteration of the loop is of the pixel directly under the pixel currently being worked on. `waveValue` is decremented in each loop until it returns to zero.

The ratio of the current `waveValue` to `waveDepth` is used to select the color and brightness for any pixel that is not black. The color is selected by its palette index, an integer between 0 and 255; the index is rounded down to make the color index an integer (Equation 4). Similarly, the local brightness is scaled using the current ratio and the global brightness set by the user via button press (Equation 5).

$$color = 255 \times \frac{waveValue}{waveDepth}, rounded\ down$$

*Equation 4: Color Selection by Palette Index at Various Wave Depths*

$$brightness = \frac{waveValue}{waveDepth} \times global\ brightness$$

*Equation 5: Brightness Variations with Wave Depth*

# 2. Implementation

## 2.1 Frame Construction

The frame was built following Figure 8. The ends of the aluminum rods were sanded and a mixed two-part epoxy was applied. The appropriate 3D-printed connector was then placed over the rod end and the part was left on a flat surface for at least 30 minutes, until the epoxy

solidified enough for the connector to stay in place. To ensure that all connections could be made, internal connections such as the T-connectors to the interior rods were made before the outer connections.

*Figure 15: As-Built Frame*

The frame was suspended by at least the length of an individual LED strip to accommodate the LED assembly. A microphone stand was repurposed for this, and the frame was attached to it using two sets of strings (Figure 16). This nonpermanent connection method allows for the stand to be reused in other applications.

*Figure 16: Frame Hanging from Stand by Three Strings*

## 2.2 LED Matrix Construction

The LED matrix was created by soldering wires for power, ground, and data between adjacent LED strips. The tops of the strips were secured to the frame using electrical tape; this also

covers the top solder pads of the strip, which have connections to power, ground, and data, reducing the chance of unwanted shorts (Figure 17).

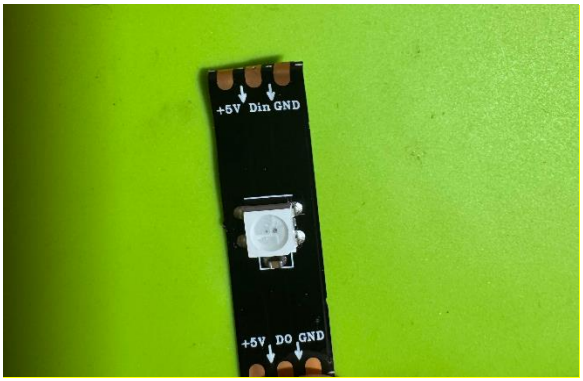


*Figure 17: LED Solder Pads*

Once the LED strips in the same yz-plane were attached, the data lines were attached to the backs of adjacent strips and were soldered to the bottom of the strip to create a continuous line. A 470Ω resistor and male header connections were soldered to the ends of the of the data lines at y=0 so they can connect to the jumper wires leading to the data pins on the Arduino (2.2.2 Interface with Controller). After all of the strips were attached, power and ground were connected across them in the x-direction. This was done by creating three-way solder connections above each strip to connect them to the two adjacent strips in the x-direction.

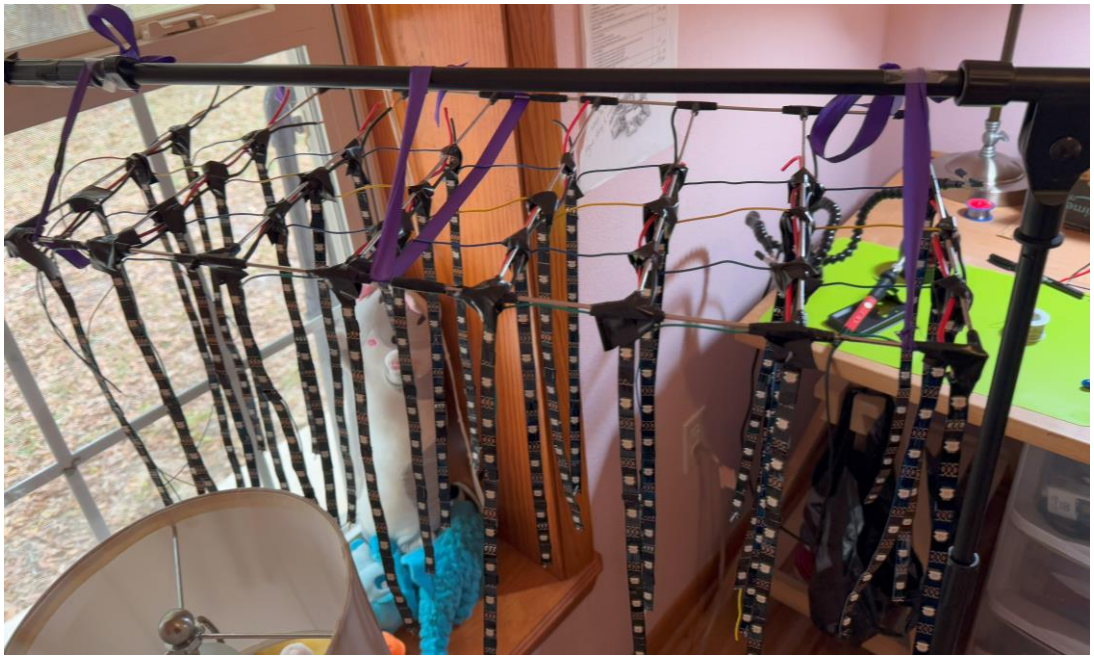

*Figure 18: Completed LED Matrix*

### 2.2.1 Interface with Power Supplies

16 AWG wire was also used to connect the LED array power and ground to the power supply. The array has two sets of wire extensions for power and ground, one for each half of the array. Each extension has a connector on its end to allow for a pressure-based connection to the wires that connect directly to the power supply (Figure 6, Figure 19). The rear edge of the frame is used to hold a network of soldered wires that splits these connections into four or eight lines connecting to the LED strips (Figure 20).

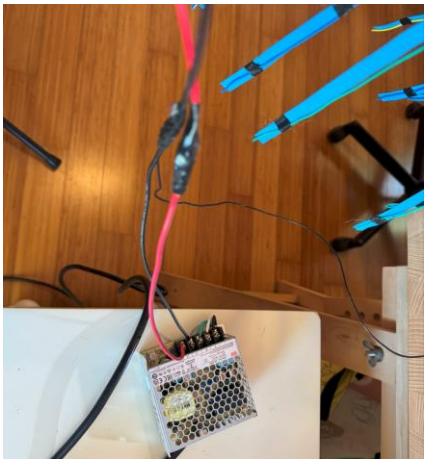

*Figure 19: Interface of LED Matrix and One Power Supply*

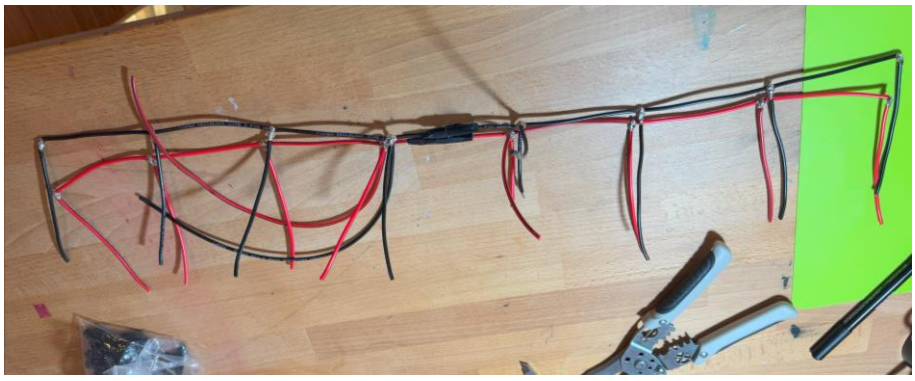

*Figure 20: Power Distribution Wires*

If two power supplies are used, all extensions are connected to the corresponding power and ground terminals of the supplies (Figure 21). If one power supply is used, a similar connector set between the halves is connected together and one set of extensions is used to connect to the supply (Figure 20).

*Figure 21: Interface of LED Matrix and Two Power Supplies*

### 2.2.2 Interface with Controller

Five data lines were used to connect to each yz-plane of LEDs (1.1.1 LEDs). These lines attach to jumper wires connected to the Arduino at the tops of the LED strips at y=0. The line extends to subsequent strips in the y-direction by another wire soldered to the data output pad at the bottom of the strip, which is then attached to the back of the LED strip before travelling out to the top of the next strip (Figure 22). A 470Ω resistor and male header was soldered to each wire leading to the Arduino to connect the LEDs to the controller circuit (Figure 23). The resistor reduces current in the data lines, reducing the chances of excess current damaging any components.

*Figure 22: Data Line Connection Via Strip Backs*

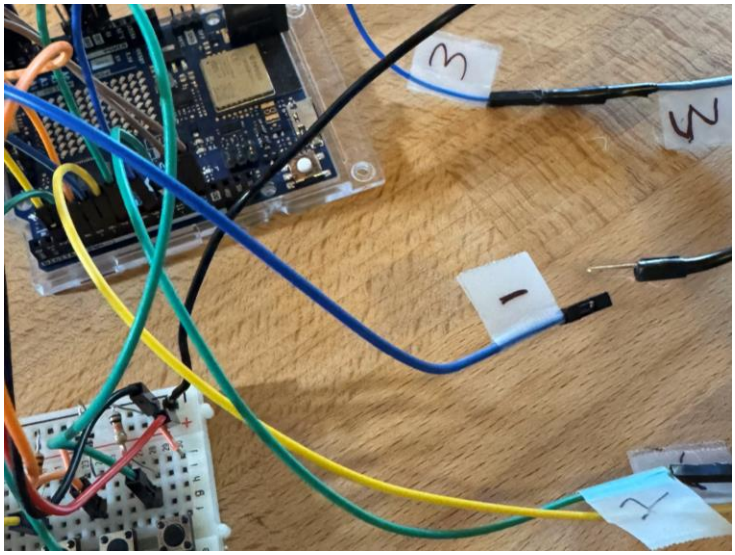

*Figure 23: Data Line Connection to Arduino Jumper Wires*

## 2.3 Controller Construction and Code Implementation

### 2.3.1 Circuit Construction

The controller circuit was constructed on a breadboard (Figure 24). It wasn't moved to a PCB so the parts could be reused in other projects. The LCD screen was connected to VDD and GND, and a 0-10 kΩ potentiometer was connected to VDD, GND, V0 to control the contrast of the display. The 4-bit input mode of the potentiometer was used, so LCD inputs DB3-DB7 were connected to the analog pins A2-A5 on the Arduino. Some 0Ω resistors were used to make wire connections.

Each button was placed between VDD and a 10 kΩ resistor, which was then connected to GND. Jumper wires to digital I/O pins on the Arduino were placed between the button and the resistor. A large resistance of 10 kΩ was selected to prevent a high-current short between VDD and GND. Male-female jumper wires were connected to five of the Arduino's digital PWN pins. The ends not connected to the pins were female in order to connect to the data lines leading to the LED array.

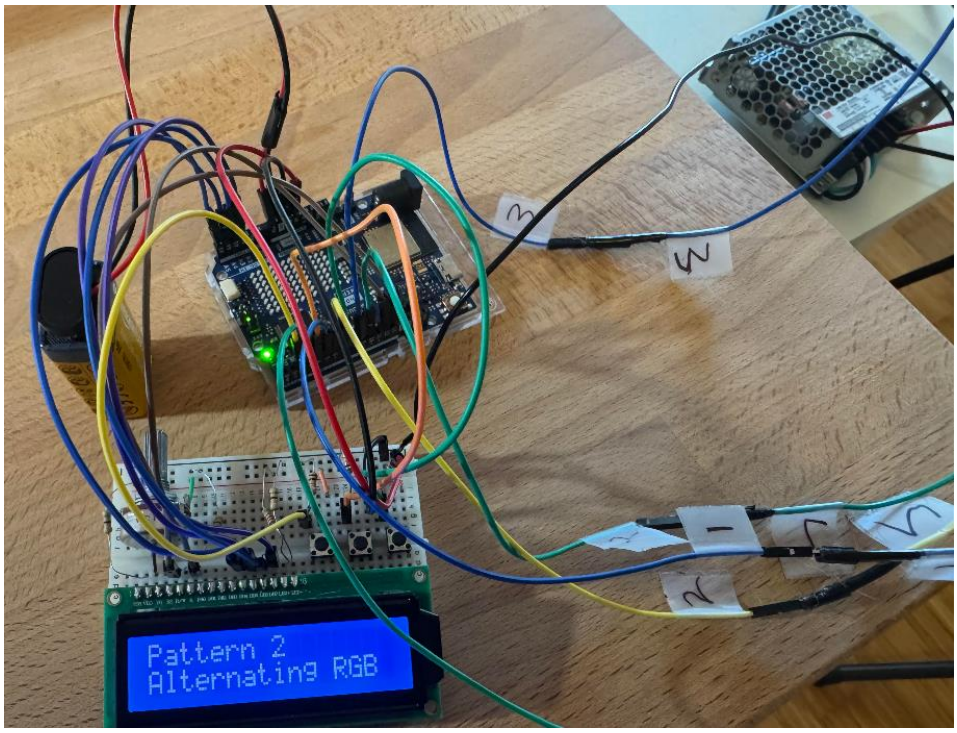


*Figure 24: As-Built Controller Circuit*

### 2.1.2 Code Implementation

The control software was written in C++ with the Arduino IDE, and made use of the built-in `setup()` and `loop()` functions to handle setup tasks and the main control loop respectively. All constants and variables used throughout the program were declared globally, before the `setup()` function and included: array dimensions, pattern names, pin numbers connecting the Arduino to input/output devices, the global LED update period, and maximum allowable brightness. The `setup()` function itself was used to implement the 3s startup delay using `delay()`, declare pin inputs and debounce periods for the buttons, declare the number and type of LEDs for each of the five output lines, and initialize and display a startup message on the LCD screen.

The `loop()` function repeats consecutively, with its speed being limited by the time to execute a single iteration. The `millis()` function was used to track the time passed since the last LED update. During an LED update, the button inputs are checked and their actions are performed if needed, new pixel values are generated based on the current pattern and frame of animation, the LEDs are set to their new colors. If enough LED updates have occurred for roughly 1s to pass and if the current pattern name requires LCD scrolling, the LCD screen will update its position in the scrolling animation on the next LED update.

Computations for generating the new pixel values are contained in separate functions for each pattern, and are based on one or two counters, depending on whether the pattern framerate matches the LED update rate (1.2.3.1 Timing Implementation) and the positional coordinates of each pixel (1.2.3 LED Patterns).

### *2.1.2.1 LCD Scrolling for Long Pattern Names*

The LCD used for this project has a size of 2x16 pixels, so pattern names longer than 16 characters would normally be cut off. A scrolling functionality, where the text shifts periodically so the entire name can be seen over a longer period of time, was added so the entire name can be seen. The text displayed on the LCD at any point in the scrolling cycle is determined by an integer counter `lcdCyclePosition` ranging from zero (the initial displayed text) to the name length. Two spaces were added to the end of the pattern name text to make the end of the name clear to a reader. If wrapping is required, the text pushed to the LCD is determined by concatenating the substring up to the position number and the substring up to the number determined by Equation 6, where *P* is `lcdCyclePosition` and *L* is the length of the pattern name (plus two spaces). This wrapping can be seen visually in Table 1, where the display shows the numbers underlined with blue.

$$substring\ end = P - (L - 16)$$

*Equation 6: End of Second Substring Using for Scrolling Wrapping*

| frame num | sym rep |
|---|---|
| 0 | 01234567 |
| 1 | 01234567 |
| 2 | 01234567 |
| 3 | 01234567 |
| 4 | 01234567 |
| 5 | 01234567 |
| 6 | 01234567 |
| 7 | 01234567 |

*Table 1: Wrapping Example Using an 4-Wide Display*

# 3. Testing and Final Product

## 3.1 Small-Scale Tests

Small-Scale tests were done with just the controller and a few LEDs during manufacturing of the full matrix assembly to ensure that the controller and basic patterns that could be fully observed with a small number of LEDs worked as expected. For these tests, a single strip of three LEDs was connected directly to power and ground the controller circuit.

This setup was used to test the LCD, pushbutton controls, and patterns that only changed all of the LEDs at once (and could be observed with just three pixels) and eventually produced the expected behavior.

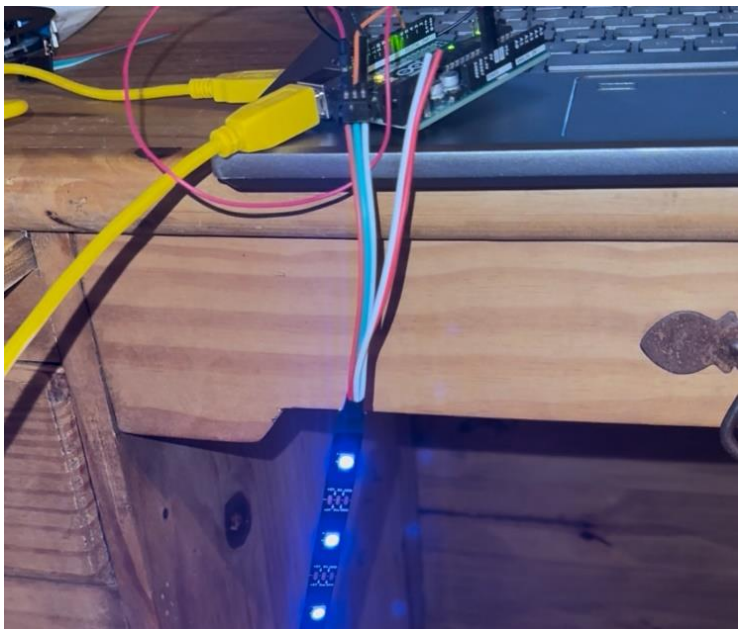

*Figure 25: Small-Scale LED Testing*

## 3.2 Full-Scale Test

Each aspect of the system was tested using the full array. Only the 75W power supply was used for simplicity as one power supply only requires access to one wall outlet, so the maximum brightness was adjusted accordingly. A few solder connections had to be remade after the initial test, where some of the LEDs failed to light; once these connections were fixed, the array functioned as intended.

One of the static patterns was used to verify that all LEDs are functional and test the functionality of the brightness adjustment using the pushbutton:

https://www.youtube.com/shorts/DKKP-Z7yInk

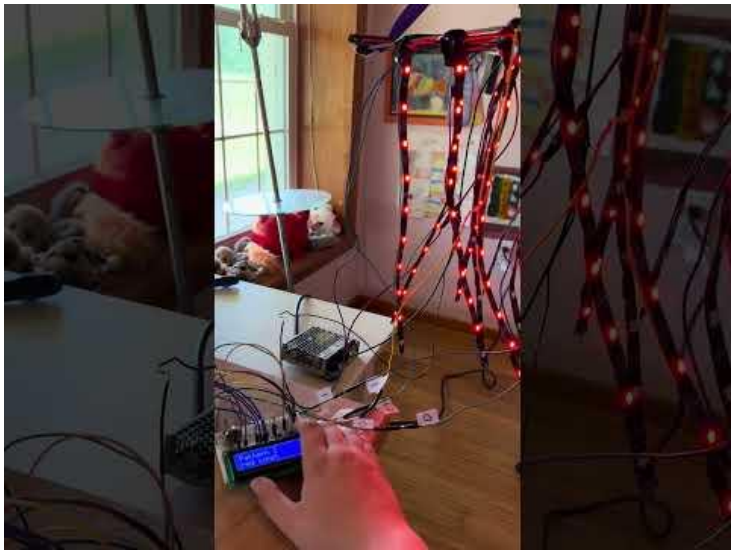

*Figure 26: LED Functionality and Brightness Adjustment Test*

One of the patterns with a name length exceeding the LCD width was selected and a full scrolling cycle was observed to confirm that the LCD scrolling functionality works as intended:

https://www.youtube.com/shorts/fkmqwgtterw

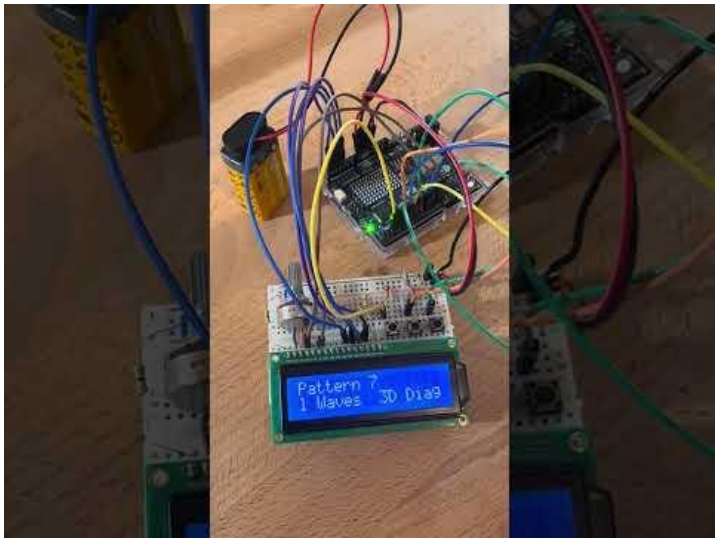


*Figure 27: LCD Scrolling Functionality Test*

All of the system functionalities can be seen in the full integration test, which involved pattern changes, brightness adjustment, and showcase of complex patterns:

https://www.youtube.com/watch?v=wWyCd0KItrE]

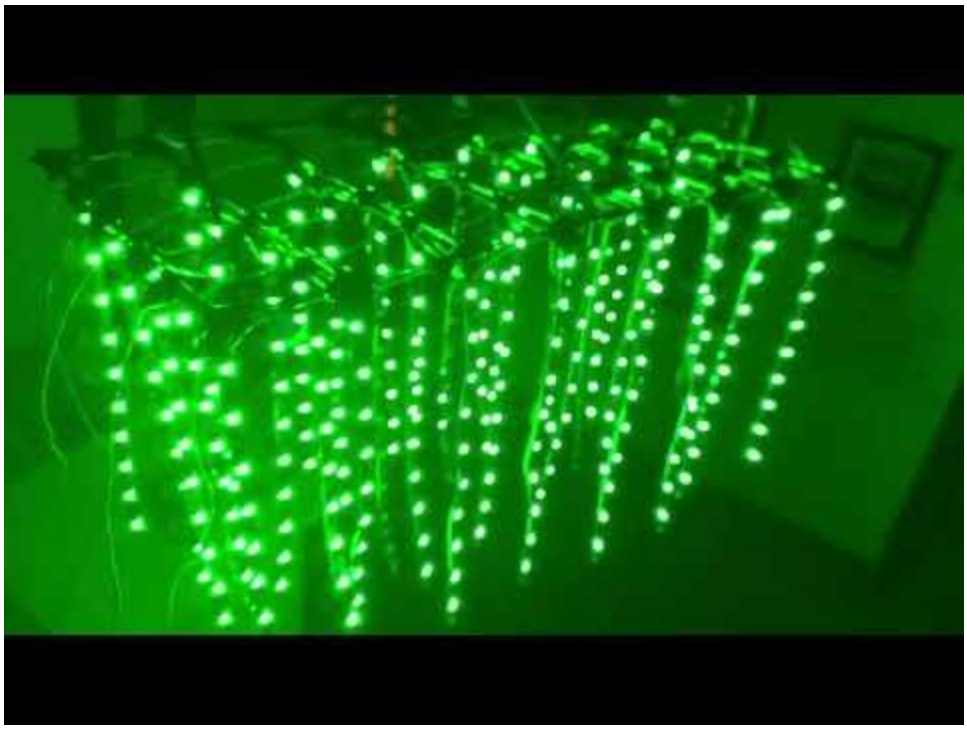

*Figure 28: Full Integration Test*

# 4. Reflection

## 4.1 Things That Could Have Been Done Better

- Didn't consider the manufacturing process enough
    - I kinda made things up as I went along. Also safety lol
- Relied too much on jank (read: electrical tape)
- Didn't do FEA on any of the frame parts. Didn't matter too much for this project but would be important if the frame were expected to handle any substantial load. The epoxy joints and 3D-printed connectors were the weak points and the frame noticeably bent under the weights of the LEDs. Not enough to cause concerns of failure, but enough to be sketchy looking. Getting a weight estimate of the LED assembly would've also been helpful for this analysis.

## 4.2 Improvements for Future Projects

- Always model electronics in CAD
    - At least all the components, wires could be left out if there aren't any concerns of having enough space for them
- Leave zero unknowns about the design before starting manufacturing
    - This includes all interfaces and connections

- Have a rigid project schedule (that has wiggle room for delays and stuff) so the project doesn't stall for months like it did after full-scale tests were complete and all that was left was putting the documentation together.

# Appendix

The code used for this project is contained in a single script and can be accessed publicly on GitHub: https://github.com/stephaniebaldwin/LED-Array.git

# References

[1] https://www.pjrc.com/how-much-current-do-ws2812-neopixel-leds-really-use/
[2] https://www.meanwell.co.uk/assets/pdf/LRS-100-spec.pdf
[3] https://www.meanwell.co.uk/assets/pdf/LRS-75-spec.pdf
[4] https://fastled.io/
[5] https://docs.arduino.cc/hardware/uno-r4-wifi/