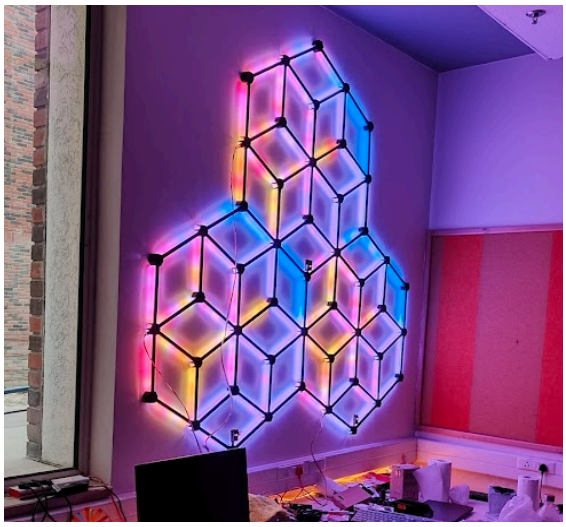
Rapid Prototyping and Experimentation - I
ISM: CS - IS - 3070 - I
Professor Debayan Gupta
With Sai Khurana
Madhav Gupta
15 May, 2024 (Spring '24)

LED Graph Communication Layer(s) - Final Project Report

Github Repository
https://github.com/xSpectre9/Rapid-Protopying-and-Experimation-I
This contains final build code as well as an 'Archive' folder that showcases some code as part of the journey.

Video Demos
(1) Lighting - https://youtu.be/du8vqHZ6CoE
(2) Touch Sensors - https://youtube.com/shorts/lOC0HyyfZBY?feature=share

Background
The Hexagonal LED project has made significant progress but has been on hold due to the lack of a software layer that can send Pranav Iyengar's ISM/Capstone work to the physically built hexagons. While he drew out software simulations of a graph algorithm (Dijkstra's, for instance) and there was a corresponding hardware module built for it as well, the two did not work synchronously.


Three-Hexagon-LED-Array


Touch-Sensor-LED-Hexagon

Aim
(1) To write software that is integrable into Pranav's code that can transfer any software-generated pattern to the physical LED modules. This includes:

(a) The three hexagon build displaying any light pattern generated on the led-graph-mono codebase.
(b) The touch sensor fitted hexagon build displaying any light pattern generated by the led-graph-mono codebase.

(2) To detect a touch input on the physical touch sensor hexagon and send those touch sensor inputs to a python script. This would be used in the two-way communication layer between the hexagon and the led-graph-mono codebase.

Results and Accomplishments

(1) The codebase can break down any LED pattern generated into universes and display them on the corresponding physical hexagon.

Video link 1
Video link 2

(2) A 1:1 mapping of all software LED pixels to hardware pixels has been finished. It has been done for both the 1260 LED three-hexagon module and the 450 LED touch-sensor hexagon module. A 1:1 mapping of the touch sensors has also been done.

(3) A python script can read which of the 19 touch sensors have been pressed on the touch-sensor hexagon, transmitted by the ESP-32 attached. This then gives input to Pranav's led-graph-mono Capstone Project, which in turn dictates the graph algorithm pattern to be displayed on the touch-sensor hexagon.

Video link

Journey Mapping

Here I am going to be describing all the tasks I undertook (highlighted with learning objectives) in order to complete the aim of the project. I will also highlight the major challenges I faced in each of these parts.

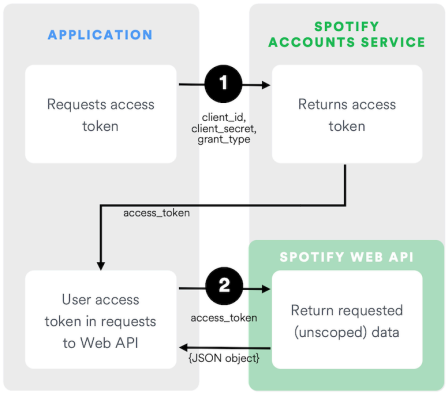
Pre-Work:

I started the ISM by learning the basics of hardware. The components of hardware (and corresponding software) that I learned here were given and moderated by Sai and Deepraj. To expedite the process of starting on and completing the main task of the ISM, I undertook a task-based approach in learning all the basics. For context, I had never worked with programmable hardware, servers, networks or any of the related fields.

Connect to and program a Microcontroller:

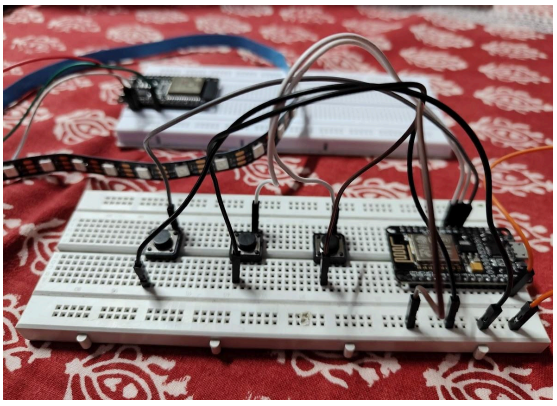
One of the first tasks I undertook was learning the absolute basics – breadboards, microcontrollers, jumper cables. I had to connect a small LED strip to a microcontroller via a breadboard and then light that up. This helped me understand the basics of how an LED strip is represented in software.

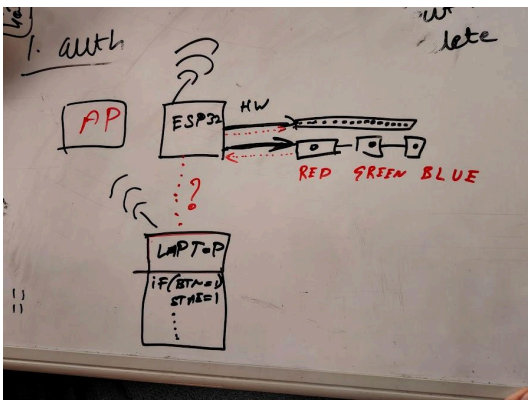
Servers and Client Requests:
While trying to figure out how to communicate with Spotify's API via a Python script, I ran into a few minor issues. As the entire process of how server calls work and workflow was an unfamiliar process to me, it took me a bit longer to fully understand how the code was interacting with the API and what different calls were.



It was easy to get sidetracked while watching tutorials as some of them delved into other concepts (one went into databasing, which was irrelevant for me). Knowing no better, I spent more time trying to learn a concept that had no practical use to me. This, throughout the duration of my ISM remained a struggle point – falling into information traps. As I was starting basically from scratch, almost all the concepts I learned were new to me. For each new thing I learned, I struggled to compartmentalize the useful parts of it (for my purposes) and ended up spending a lot more time on each task and concept.

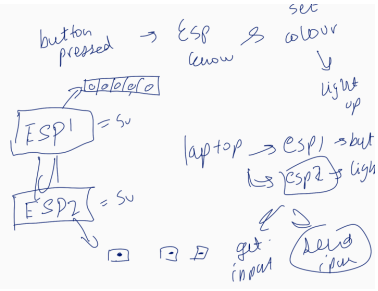
Making Two ESPs Communicate:
This was one of the final pre-learning tasks before I jumped into the final project. It involved reading three push button inputs connected to one ESP, and then having another ESP light up an LED strip based on which button was pressed.

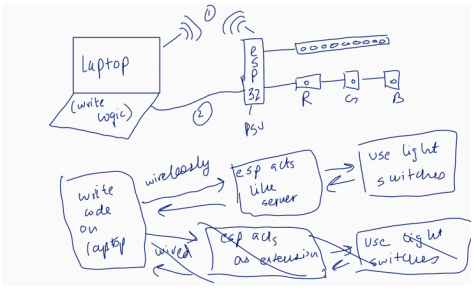


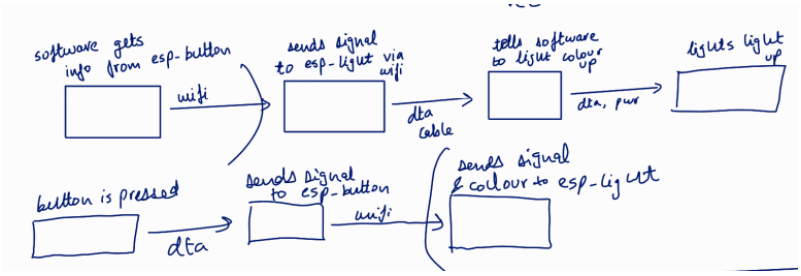


Here, I faced a small issue of not knowing how far to push the button's in. I spent a while trying to troubleshoot why some buttons are getting registered inputs and why some aren't. I went

through a lot of iterations of code to try and fix it, but was stumped. Now that I am here writing this report with a lot more experience, it was less than a trivial problem but at those moments each problem felt a lot deeper than it really was. I didn't realize how far the button pins had to go into the breadboard was the issue, until I took it to Deepraj to ask for guidance.







Above, you can find preliminary control flow diagrams of how I thought the setup would run. Neither of these were accurate.

Talk to a WLED-loaded ESP via a Python Script:
This was the first task that directly worked on the deliverables of the project. I learned directly about DMX channels and universes, how LedFX works (for figuring out sACN functions). Without much structure to the work, I took a lot longer on the task than I should have – similar to how I took longer with API calls. Coding sACN into WLED from a python script is also somewhat niche, so finding beginner friendly tutorials was also a challenge. Here is where I encountered the innermost logic of writing an LED array into DMX universes:

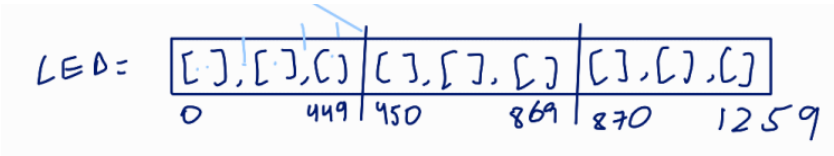
- One universe has 512 DMX channels
- 3 DMX channels represent the (R, G, B) values on one pixel
- 512/3 == 170.67 (not divisible by 3), so how many channels should be sent by each universe? I would be able to reach the answer to this until after implementing multi-threading.

This helps me summarize another challenge I faced – having to skip some logical steps to learn something new and seemingly unrelated, and only then being able to circle back and link the two concepts together. *Exhibit A* for this is in the next subsection.

Implement Multi-Threading:
My next task was to implement multi-threading. While the concept for just the purposes of the project was not very complicated – start 4 simultaneous threads, three for each of the ESPs on

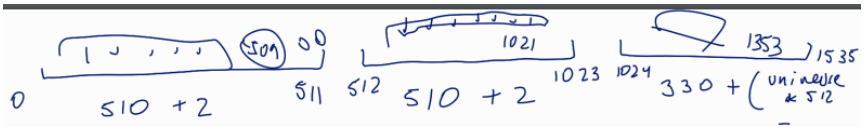
each of the hexagons and one to constantly read the generated text file. While implementing this part of my communication layer, I ran into another information trap. It was difficult to figure out what part of threading suits my the purposes of the ISM, how to implement threads into the program and how the logic of threads translates into the code. My code looked like the file ‘esp_with_python.py’ in the Archives folder of the code repository at this point.

Light up the first hexagon with Python:
This was the major part comprising the communication layer. Going into technical detail, the wiring of the three hexagons has been done to make them one contiguous array of length 1260. Here, one unit of the array contains a tuple, which is the [R, G, B] values that constitute one single pixel on the NeoPixel strip, shown below:

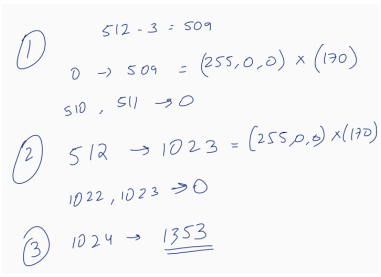


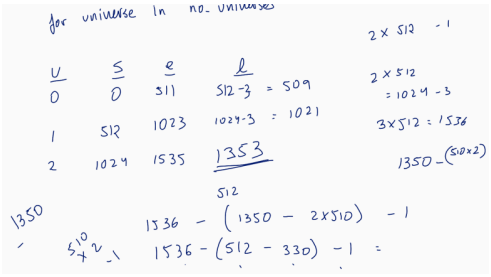
In the figure above, 0 to 449 represents the amount of LEDs being controlled by the first 1 ESP. One of the tuples in this contains the [R, G, B] values for the pixel. There were three such ESPs that controlled the entire three-hexagon module and had to be run at the same time. To make my life easier, I focused on only figuring out how to send the first hexagon input data, as the other two parts of the array were controlled by different ESPs.

WLED, or the software that the ESPs are loaded with that control the light strips, read inputs one DMX universe at a time. One DMX universe is constituted of 512 DMX channels, and three DMX channels represent the [R, G, B] values of one pixel. I was now able to work out the questions I had before implementing multi-threading: for a strip of 450 LEDs needs to be split up into 3 universes, 510 DMX channels (or 170 LEDs) being lit up in the first two universes, and 330 DMX channels (or 110 LEDs) being lit up in the last universe. The non-RGB channels in each universe also had to be padded to zero.

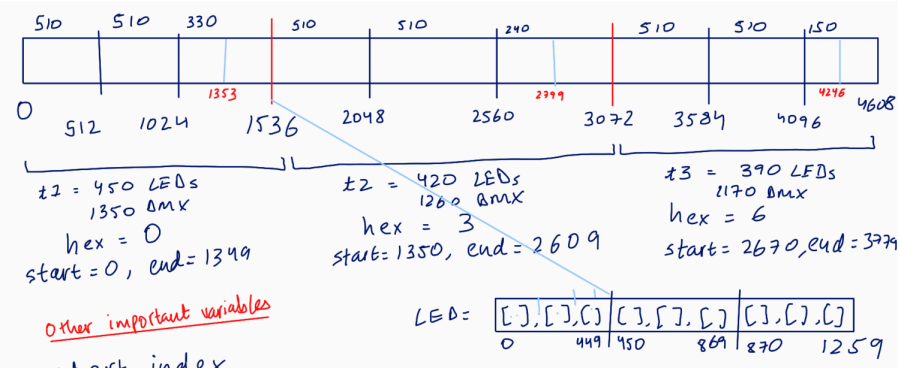


I worked a bunch on writing the algorithm and translate it into code to ensure it ran smoothly. Here’s some of the calculation to make the algorithm generalisable for any number of universes:

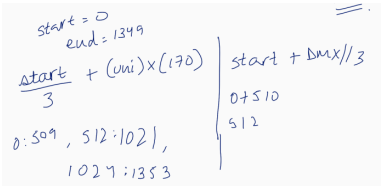


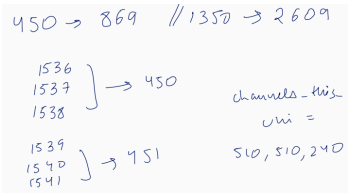


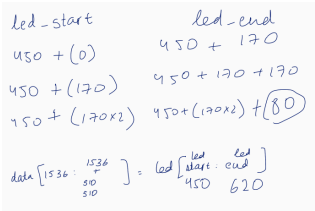
Send universes to the entire three-hexagon module:
I now had to scale up my code to light all three hexagons up at the same time. I quickly realized I made some errors in my logic, and I had to redo some of it to scale my array up to send the contiguously split array into nine universes (three for each hexagon). This can be shown in the figure below:



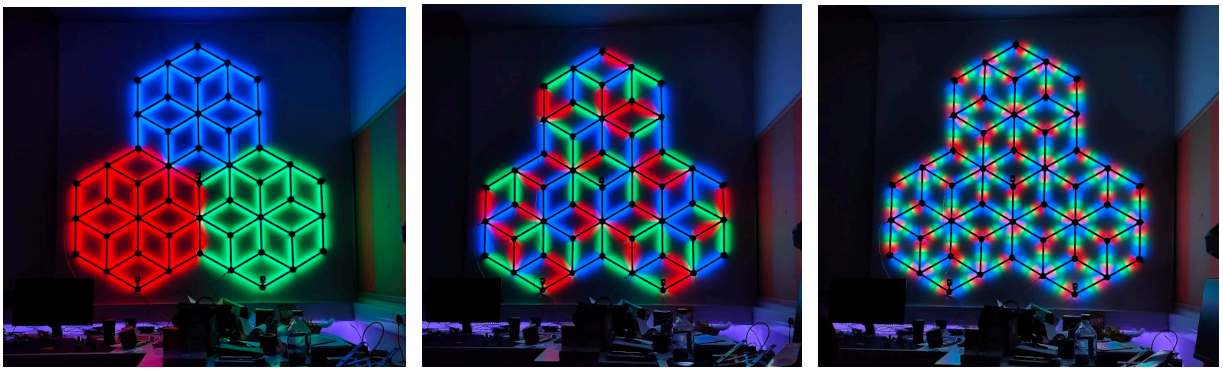
While this didn't take long, I ended up having to redo my slicing logic a few times to make it exactly right.



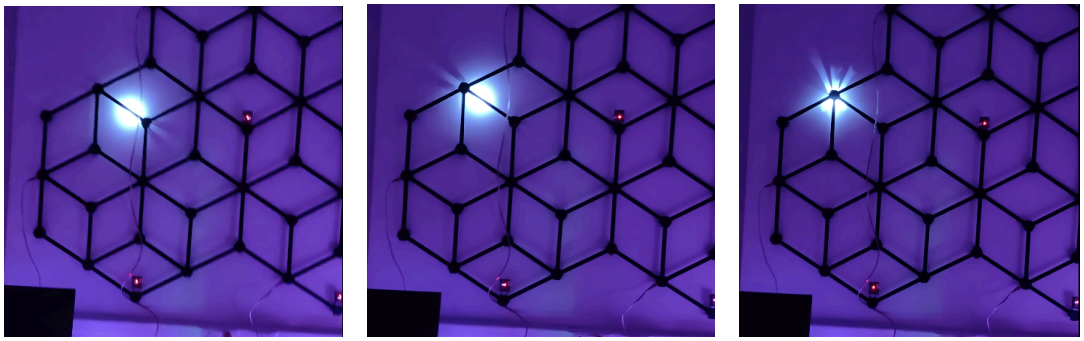




After some tedious troubleshooting, I finally got the three universes running! This marked a major milestone in my project – being able to slice data and send it to the hexagons. This can be found in 'three_hexagons_static.py' in the Archives folder of the repository.



Display a dynamic pattern:
A problem still persisted in how I was generating a pattern for the entire three-hexagon-module which needed to be addressed. It was easy to generate a static pattern and then slice and send it to the 3 ESPs. However, generating and sending a dynamic pattern (like a linear traversal of the hexagons) looked extremely sluggish and needed to be optimized.



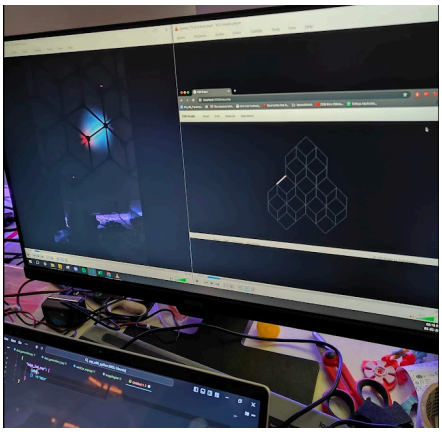
At this time, I was both generating a pattern and simultaneously slicing and sending it to the ESPs. The two performance optimizations that I decided to implement (under Kanishk's inputs) were:

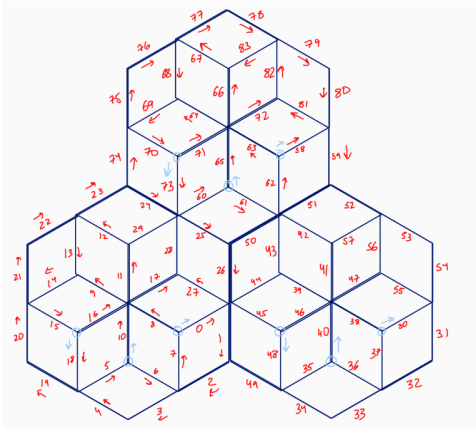
1. Converting all the data structures and transformations to use NumPy arrays (something new) instead of Python lists (something I was quite familiar with).
2. Generating any pattern in another file, storing it as a text file, and then constantly read that text file. Another benefit of this is that led-graph-mono would also be generating the same text file and then feeding it to my codebase to slice and send to the hexagons.

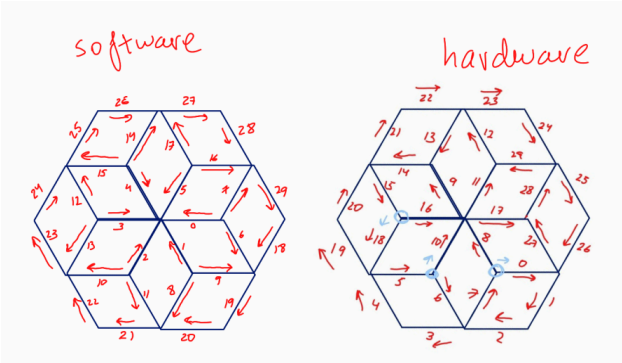
These updates took some time to understand and implement. It was only during this that I began to understand the performance limitations of the software I was writing, and when I also completed the core of the code that would be integrated into led-graph-mono to serve as the communication layer.

Actual Mapping:
The source of these text files is being generated by led-graph-mono, the codebase created by Pranav Iyengar as part of his ISM converted Capstone Project. Now, the first pixel that lit up in

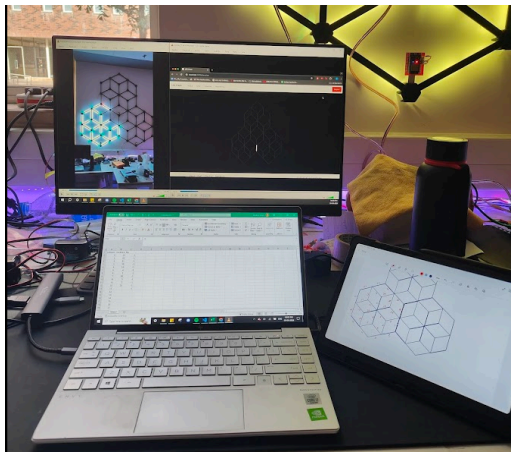
the software simulation didn't equate to the same first pixel lighting up in the actual hardware. This meant that a 1:1 mapping had to be done that equates the first LED in software to the first in hardware. Doing this 1260 times was going to be painstaking, so I devised a method that lights up one edge (15 LEDs) at a time, and notes the direction vector of the edge as well.





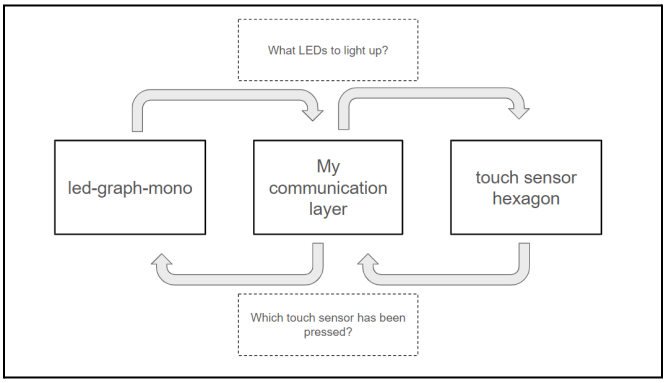


By lighting up one edge at a time, I only had to map 84 edges, which had to be done painstakingly - one edge and direction at a time. I created a JSON file that mapped each hardware edge to match the software ones, for both the three hexagon modules as well as the touch sensor hexagon module.



I had tried using excel at first, but switched to JSON for the completeness of programming. After the mapping was completed, the lighting portion of my project was done. My code could now be integrated into the text files generated by led-graph-mono to display the visual effects. The combination of the two codebases can be found in ‘send_multiple_visualisations.py’ of the repository. A quick reaction video of it finally working on the hardware, a year after the original animation was written in software.

Touch Sensors setup and mapping:
The final part of my ISM is writing a python script that completes the separate two-way channel between the touch sensor hexagon and led-graph-mono.



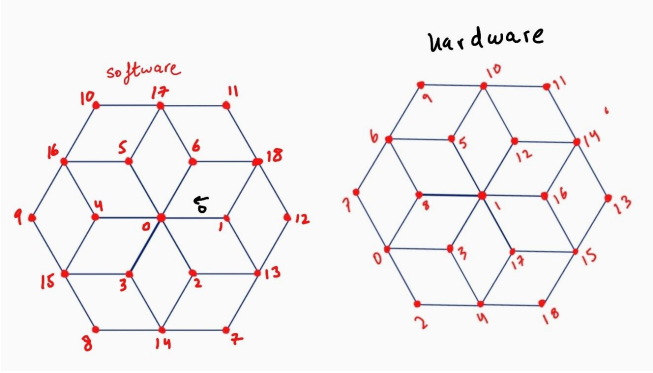
Channel 1 is sending what lights have to be lit up. Channel 2 is detecting which of the 19 touch sensors have been pressed on the graph. The lighting part of this channel was completed. I now had to be able to read which touch sensor is pressed on the hexagon.

I finally got to work on some hardware again for this part. I learned soldering to mount the ESP-32 running ‘TouchHexagonSockets.ino’ as well as crimping to attach it to the touch sensor module. 18 of the touch sensors inputs go into one channel to the ESP via the i2c protocol, and the last one goes directly to a data pin. Here is Sai teaching us how to crimp a wire:

The python script reads which sensor has been pressed and makes a note of it, giving the information to led-graph-mono to then read the data and transmit the corresponding lighting effect. It uses websockets to instantaneously transfer data.

```
gachihyper@LAPTOP-LGBUR42Q:~/Rapid Prototyping and Expermination I$ /bin/python3 "/home/gachihyper/Ra
ver.py"
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: Sensor 17 touched
No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: Sensor 2 touched
No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
Response from ESP32: No sensors touched
```

After confirming that they are working right, I had to map them. The process was similar to the LEDs, except I had to do it 19 times instead of 84.



After the mapping was done, the ISM was completed from my side. All that had to be done was integrating the two codebases.

Major Challenges, Pitfalls and Set-Backs

(1) Technical Proficiency – as an economics major who has only done computer science in the capacity of a minor subject, a lot of concepts in programming and systems designs were unknown to me. This meant it often took me more effort to understand and learn some basics to implement and complete the project. I faced this multiple times, like while learning RestAPI or multi threading, I didn't have a clear view of exactly how the concepts would fit my code.

(2) Time Commitments – as I was working into an already existing project, I had to be cautious of both mine and other stakeholder time commitments. I often needed material and understanding from said stakeholders which resulted in delays. For instance, I could often only test some iterations in my slicing algorithm only the next day as I had to visualize them on the physical array as I had to respect the timings of the lab.

Next Steps

(1) Performance profiling: Any generated effect displayed by my mapping code will not be as smooth as something like what WLED can natively do, or what LedFX can produce.

This requires meticulous performance optimization to ensure that all the components of the core script are running smoothly. Further, a MacBook cannot currently run threads the same way a Windows device running WSL can. My hardware (laptop) too is constrained by its performance – if I am not plugged into a power source, the displayed animations are often laggy. Thus, one must consider how the two software components are being run in the future to ensure smooth animations.

(2) Improvement on functionality: my codebase needed careful intervention to fit into the other codebase. When considering the future scope of the application, which is scaling beyond one or three hexagons and more touch inputs, it needs to be ensured that another machine can plug and play the two codebases and then scale them up for more hexagons.

(3) Scaling up: As the project scales up to more hexagons, research might have to be done into ensuring that the text file being generated doesn't take too long to generate. Another issue that will be encountered is connecting touch sensors to the ESP – I am not sure how many touch sensors one ESP can accommodate and how the i2c protocol would work there. Careful planning would have to be done here.

Conclusion

I hope that this report is helpful for anyone in the future attempting to scale up the existing project. I wanted to supplement the Next Steps section here as well by highlighting the effort and costs it would take for someone to reproduce and scale up the work already done.

(1) Cost of Materials:

Item	**Cost (Estimates)**
450x LEDs - WS2812	Rs. 1,125[1]
1x Wall Laptop Adaptor	Rs. 400
2x ESP-32	Rs. 660
Buck Converter	Rs. 370
19x Touch Sensors	Rs. 665
Other Electrical Components (Deans T, XT60s, PCBs, wiring and cables)	Rs. 700
30x 3D Printed Edges	Rs. 1,800 (estimating 60 rupees an edge)
19x 3D Printed Nodes	Rs. 1,615 (85 per)

[1] Note: As one connects hexagons, the amount of LEDs reduce by 30 for each connected side. So for a three-hexagon setup, hexagon #2 has 420 LEDs and #3 has 390 (and so on)

19x 3D Printed Wall Mounts	Rs. 1,615 (85 per)
Total (Estimated)	**Rs. 8,950**

(2) Time and effort taken to build hardware:

Building the hardware requires a high level of proficiency in skills such as soldering, wiring, CAD/CAM, etc. As I was working with already built modules, I can't comment on how long it would take to build a module, but the work regarding physically building would just be reproducing the already built modules.

(3) Time and effort taken to scale up lighting software:

The piece of software I wrote as part of my ISM can technically read and send any text file to any number of hexagons. The only changes that would have to be made/added are the numbers of hexagons, the number of threads, and the respective global variables. Here, the hexagon global variables would have to be updated/added, as would the initialization of threads *t_left*, *t_right*, etc.

```
# left hexagon = 450 LEDs
left_ip = "192.168.40.5"
left_leds = 450
left_start = 0
left_end = (left_leds * 3) - 1

# right hexagon = 420 LEDs
right_ip = "192.168.40.6"
right_leds = 420
right_start = left_end + 1
right_end = right_start + (right_leds * 3) - 1

# top hexagon = 390 LEDs
top_ip = "192.168.40.7"
top_leds = 390
top_start = right_end + 1
top_end = top_start + (top_leds * 3) - 1
```

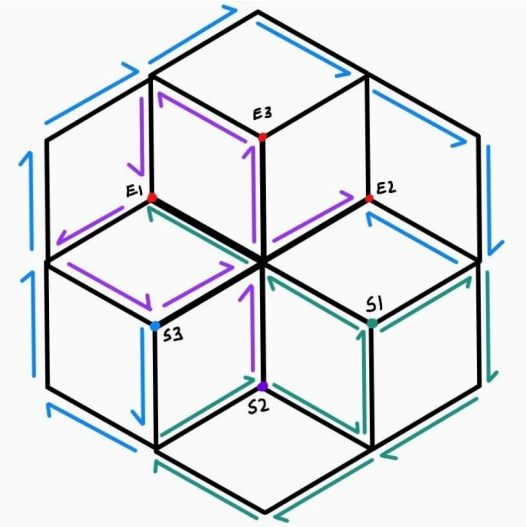
```
t_left = threading.Thread(target=send_to_hexagon, args=(left_ip, left_start, left_end, data))
t_right = threading.Thread(target=send_to_hexagon, args=(right_ip, right_start, right_end, data))
t_top = threading.Thread(target=send_to_hexagon, args=(top_ip, top_start, top_end, data))

t4 = threading.Thread(target=modify_array, args=(no_frames, led_mega))
t4.start()

t_left.start()
t_right.start()
t_top.start()

t4.join()
t_left.join()
t_right.join()
t_top.join()
```

Regarding mapping, careful work would have to be done on the software visualizers end to ensure that it follows how the hardware has been wired (LED #1 on both should light up the same thing)!



(4) Time and effort taken to scale up touch sensor software:

Regarding the touch sensors, research would have to be done into how many inputs from touch sensors can be fed into the same ESP. After that, a similar approach following the i2c protocol can be undertaken to map and light up the correct nodes to display graph algorithms/display artworks.