Music-Visualization and Motion-Controlled LED Cube

By

Hieu Tri Huynh
Islam Kadri
Zihan Yan

Final Report for ECE 445, Senior Design, Spring 2018
TA: Zhen Qin

02 May 2018
Project No. 34

Abstract

The purpose of this project is to design and build a device that allows for music visualization and entertainment through user interaction. This device will collect data through a set of microphones and displays the aesthetic visualizations through real time input onto the LED grid. It also allows the user to play a motion-controlled 3D Snake game that is displayed on the LED grid. The various modules discussed in this report were successfully integrated and allowed us to have a working product.

Contents

1. Introduction

1.1 Purpose

Our project's main inspiration came from an art piece called *Kinetic Rain* in Singapore's Changi Airport. The sculpture cycles over 16 pre-programmed shapes, whereas our device's animations will be based on real time sound's frequency, amplitude and direction of arrival. [1] The construction of the sculpture was made with over 1,200 bronze droplets, steel wires to hold the droplets, and cost several million dollars. [2][3] Our project will be more cost efficient by standardizing the way [2][3] Our project will be more cost efficient by standardizing the way the LED grid is built which will allow for easier fixing and scaling.

Unlike other visualization devices, we use direction of arrival and amplitude, in addition to frequency, to influence animations. Additionally, we'd like the user to be able to interact with the LED grid by making a 3-dimensional Snake game that uses proximity sensors to control the snake's direction. By taking something that is traditionally 2-dimensional and implementing a 3-dimensional version, we can show off the benefits of user interaction as well as increase the entertainment factor of the device.

1.2 High-Level Requirements

1. Create a low budget, high quality, aesthetically pleasing device for under $150.
2. Device can extract audible sound's frequency (50 Hz – 20 kHz), amplitude, and direction of arrival.
3. The LED cube can reflect the sound's properties in an aesthetic and representative way.
4. Device can correctly pick up the user's hand motion and properly control the Snake based on that information when user's hand is within 15 cm of the sensor.

1.3 Subsystem Overview

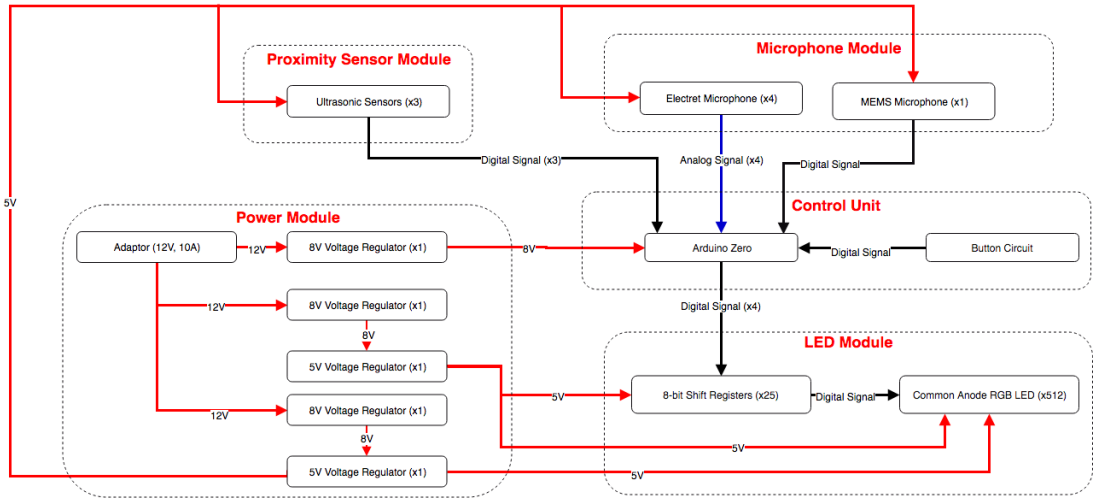

Figure 1. Block Diagram

There are 5 modules in our design. First, the Microphone module will have 1 high quality MEMs digital microphone and 4 electret microphones. These microphones will be able to pick up the sound and send them to the Control Unit, which will extract the information of the signal (frequency spectrum, amplitude, and direction of arrival). With this design, we will satisfy the

second high-level requirement. Next, the Control Unit will output digital signals to drive the 512 LEDs in the LED Module. By doing this, the LED cube can reflect the information of the incoming sound, which satisfies the third high-level requirement. Finally, by having 3 Ultrasonic Sensors in the Proximity Sensor Module arranged in a specific way, the Control Unit will be able to determine the user's hand motion and properly drive the LED module. This will satisfy the last high-level requirement. In conclusion, our design will satisfy all three high-level requirements. The Power Module will supply the power for all components in the system with appropriate voltages.

2. Design

2.1 Physical Design

The LED module consists of 512, 5 mm RGB LEDs, which are arranged into an 8×8×8 grid. We plan to arrange the LEDs such that the horizontal and vertical distances between any 2 LEDs is 1.25 inches. This will be placed on a flat rectangular surface approximately 24×18 inches^2 large, which is also what our sensors will be attached to. The cube will be placed in the upper middle part of the whole design with dimensions of 8.75×8.75×8.75 inches^3. We are going to place the microphones 2 inches away from the three edges of the board. The fourth edge is 2 inches away from the proximity sensor group, which also contains the fourth microphone. Three proximity sensors are arranged in a triangular fashion with 0.75 inches away from each other. This group's dimensions on the board is 2×4 inches^2. The diagrams of the physical design are shown in figure 2 and 3

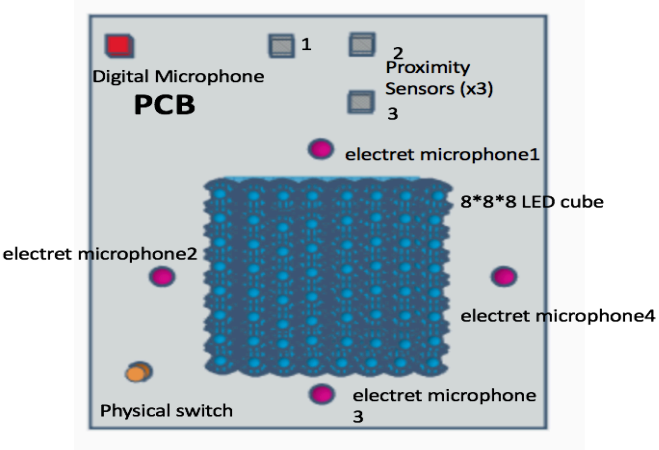

Figure 2. Physical Design Top View

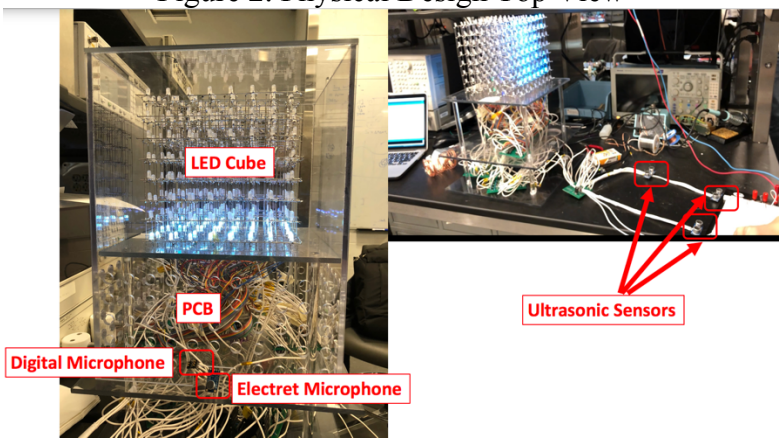

Figure 3. Final Physical Design

2.2 Microphone Module

2.2.1 Design Procedure

The digital MEMS microphone (ICS-43434) was chosen to be the main microphone that collects the sound signal to extract the frequency and amplitude of sound. It can provide a high-quality input signal and has a low probability of error. The 4 electret microphones (MAX9814) were chosen to collect sound to determine the direction of arrival because of their low cost.

2.2.2 Design Detail

The ICS-43434 digital MEMS microphone has a frequency response range from 50 Hz to 20 kHz, which covers most of the frequency range of audible sound [4]. Furthermore, that model has a high Signal-to-Noise Ratio (SNR) of 64 dBA, a high sensitivity of -26 dB with +/-1 dB sensitivity tolerance, and a sample rate range from 23 kHz to 51.6 kHz. Because the highest frequency of audible sound is 20 kHz, we will set the microphone sample rate to be 40 kHz, which satisfies the Nyquist rate [5]. The output of this digital MEMS microphone will be fed to the control unit to extract the amplitude and frequency of sound. The 4 electret microphones are placed at the 4 surrounding sides of the LED cube to collect data for detecting direction of arrival. These electret microphones have a high SNR (61dBA) and sensible sensitivity (-44dB), which is good enough for our purpose [6]. We set the gain of the microphone to be 60 dB, making the output to be amplified enough so that we do not need to have a separate amplifier circuit. The analog output signals of these 4 electret microphones are sent to the control unit, which will process and make decisions about the direction of arrival.

2.3 Proximity Sensor Module

2.3.1 Design Procedure

We choose to use the HC-SR04 ultrasonic sensor over an IR sensor because we want to correctly detect user hand position regardless of the lighting condition. Also, the Ultrasonic sensor can provide high accuracy (about 3 mm) and wide-ranging distance (2 cm – 4 m) [7].

2.3.2 Design Detail

The 3 ultrasonic sensors are placed in an “L” shaped manner as described in the physical design section. By arranging the 3 ultrasonic sensors this way, we can easily extract the user hand’s motion in 6 directions. The HC-SR04 ultrasonic sensor can help us to satisfy the requirement to detect user hand motion when a user’s hand is moving within the 15 cm range. The 3 output signals of these 3 sensors are collected by the control unit and used to determine user hand motion. The circuit for the proximity sensor module is shown in figure4.

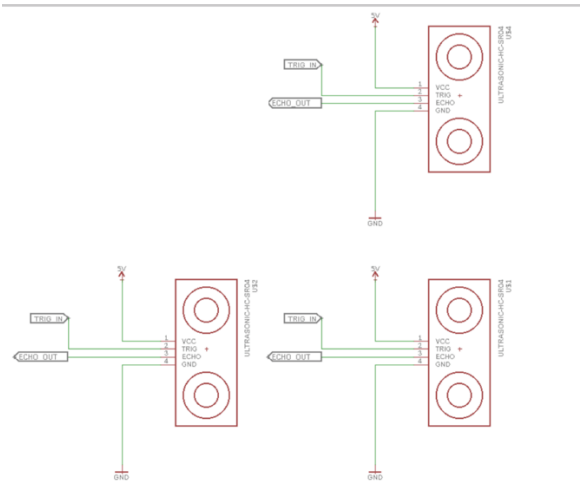

Figure 4. Proximity sensor module circuit

2.4 LED Module

2.4.1 Design Procedure

We chose to use 5 mm common anode RGB LEDs to visualize music input and display the Snake game. The LEDs provide us with a wide range of color and are easy to use. Next, we use 8-bit shift registers (SN74HC595) to control the 512 LEDs. The shift registers reduce the number of the control unit's output signals that are needed to control all 512 LEDs. Furthermore, the SN74HC595 shift register can be controlled with a high clock speed, which can allow us to save microcontroller clock cycles.

2.4.2 Design Detail

Instead of controlling 512 LEDs simultaneously, we only control 64 LEDs on 1 layer at a time. Specifically, we turn on all LEDs that need to be ON on one layer, and then turn them OFF before moving to the next layer. By turning LEDs ON and OFF at rate of 100 Hz, we can prevent them from flickering. To control 64 LEDs on 1 layer, we use 25 8-bit shift registers. These registers are connected in a way such that the data from one register will shift out to the next one, i.e. data from the 1st register will shift out to the 2nd register, data from 2nd register will shift out to the 3rd register, and so on. The circuit of the first 4 shift registers is shown in Fig. 5. The first shift register is used to control the level of LEDs, while the remaining 24 shift registers are used to control the LEDs' color. We will discuss this in detail in the control unit section. Because the output of the shift registers cannot provide enough current for the LEDs, we use the circuit in figure 6, that uses NPN BJT (2N2222) and P-channel MOSFET (FQP27P06), to supply power to the LEDs.

We will connect all 64 cathode pins of the 64 LEDs on the same layer together and use the 1st shift register to control the status of these 8 layers (required for all 512 LEDs). Then for LEDs on different layers but with the same (x, y) coordinates, their red anodes, green anodes, and blue anodes will be connected. The 64 red anodes will be controlled by the next 8 registers. The 64 green anodes will be controlled by the next 8 registers. The 64 blue anodes will be controlled by the last 8 shift registers. The circuit of 8 LEDs with same (x, y) coordinates is shown in Fig. 7. The circuit of 8 LEDS on layer 0 is shown in Fig. 8.

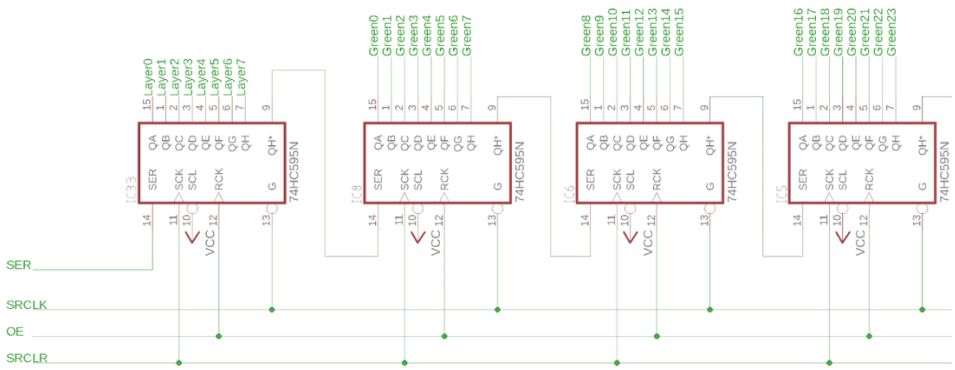

Figure 5. Circuit of The First 4 Shift Registers

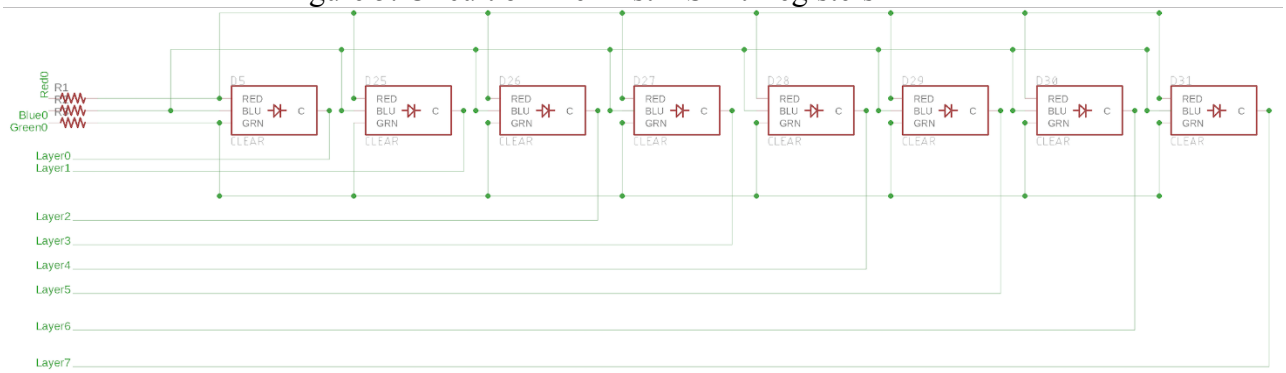

Figure 6. Circuit of 8 LEDs with Same Horizontal Coordinates on 8 Layers

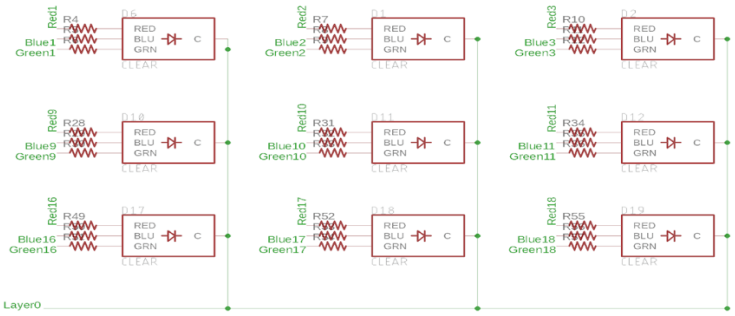

Figure 7. Circuit of 9 LEDs on Layer 0

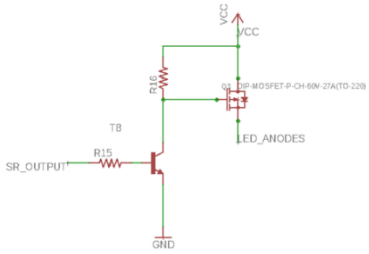

Figure 8. The circuit of 8 LEDS on layer 0

2.5 Power Module

2.5.1 Design Procedure

The 12 V, 10 A adaptor was used to convert from AC to DC and supply power for the whole system. We chose this adaptor because the 512 LEDs consume lots of current and we need to provide a high voltage (8 V) for the Arduino Zero. To step down the voltage from 12 V to 8 V and 5 V, we use linear voltage regulators (7808A and L7805CV). The reason we chose to use these voltage regulators is that they are easy to use and are cheap.

2.5.1 Design Detail

The output of the adaptor will be connected to the input of one of three 8 V voltage regulators to supply the power for the Arduino Zero. The 7808A voltage regulator can supply currents up to

1.5 A, which is sufficient for Arduino Zero. Next, we use the circuit as described in figure9 to step down the voltage from 12 V to 5 V. The reason that we connect 8 V voltage regulator to 5 V voltage regulator is to distribute the heat so that the voltage regulators are not overheated. We have 2 circuits as shown in figure10, which provide up to 3 A of current to supply power for all other components in the system. This design satisfies the requirement because the total current that is consumed by the system is less than 2 A. We will discuss the detail about this in the Test and Verification section.

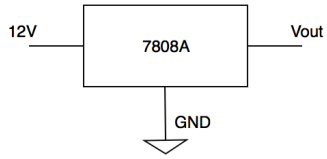

Figure 9. Circuit for voltage regulator from 12V

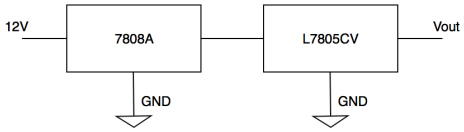

Figure 10. Two voltage regulators in series

2.6 Control Unit

2.6.1 Design Procedure

We chose to use the Arduino Zero to control the whole system because it has I^2S interface, which we need to communicate with the MEMS digital microphone, and is powerful enough for us to run our algorithms.

2.6.2 Design Detail

The Arduino Zero has high clock speed (48 MHz), which is sufficient for us to extract the information from the input sound in real time. To control the LED cube, the Arduino will shift out 200 bits to the 25 shift registers every 10 ms. We use the SPI library to shift out data to shift registers. The first 64 bits are used to control red LEDs, the next 64 bits are used to control green LEDs, the next 64 bits are used to control blue LEDs, and the last 8 bits are used to control the level of LEDs. To control the LED's color, we use the 4-bit Bit Angle Modulation (BAM) technique. Using 4-bit BAM, we can have up to 4096 colors. By using 25 shift registers and 4-bit BAM technique, we can control the color of each individual LED.

To extract the frequency of the input sound, we applied a 64-point FFT on the input that we receive from the MEMS microphone. 64-point FFT gives us a 625 Hz frequency resolution, which satisfies our requirement. After calculating the frequency spectrum of the input sound, we normalize and map the frequency spectrum onto the LED cube in the way shown in figure 11. To extract the amplitude of the input sound, we use the formula as shown in Eq. 1, which can help us to prevent fluctuation caused by noise while being able to update quickly. After calculating the amplitude, we map the amplitude readings to the colors in the following way: [0 – 700000] to GREEN, [700001 – 950000] to BLUE, [950001 – 1200000] to YELLOW, [1200001 – 1450000] to PURPLE, [1450001 – 1700000] to LIGHT BLUE, [>1700001] to WHITE. Finally, to extract

the direction of arrival, we collect the data from 4 electret microphones and use the flow chart in figure 12. For the actual code of these parts, please refer to Appendix B

To detect motion of a user's hand, we use the 4 digital signals received from proximity sensor module. To calculate the distance between the sensor and user's hand, we use the Eq. 2. Then, we apply the algorithm described in figure13 to detect the hand motion. For the actual code, please refer to Appendix B. After detecting the motion of user's hand, the Snake game is controlled based on the flow chart in figure 13.

To switch between 2 modes (gaming and music-visualization modes), we use a de-bounced circuit as shown in figure15.

$$amplitude = (amplitude \times 0.875) + (sample \times 0.125) \quad \text{Eq. 1}$$

$$distance = 330\frac{m}{s} \times t \quad \text{Eq. 2}$$

Figure 11 frequency spectrum onto the LED cube

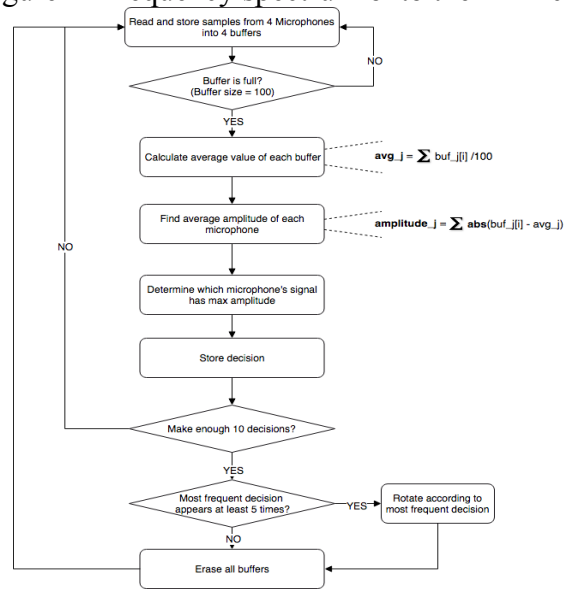

Figure 12. Flow chart to determine arrival of sound

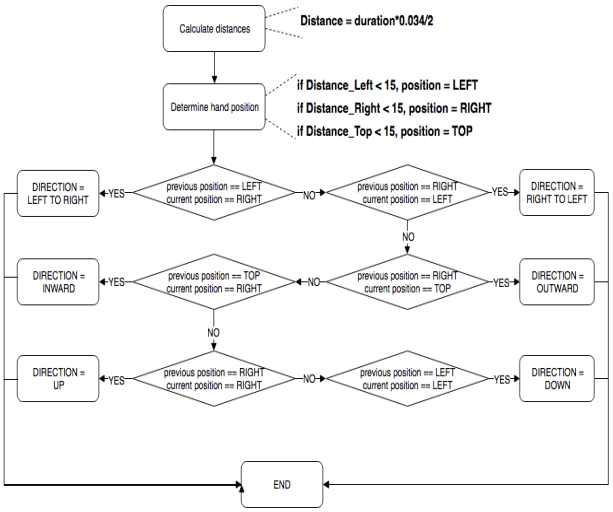

Figure 13. Flow chart to control proximity sensors

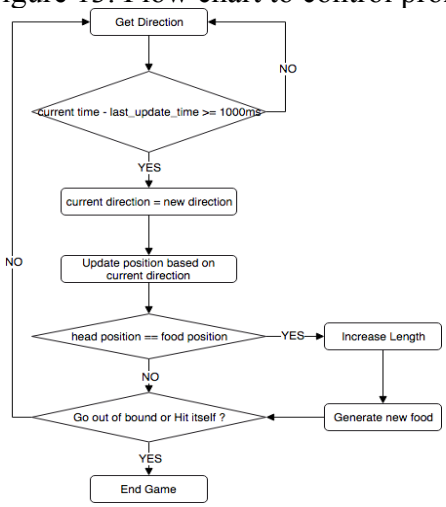

Figure14. Flow chart for snake game

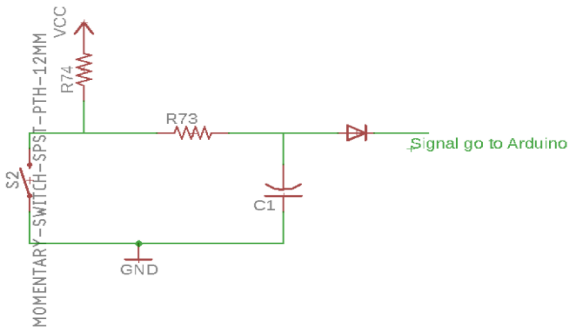

Figure 15. De-Bounced Switch Circuit

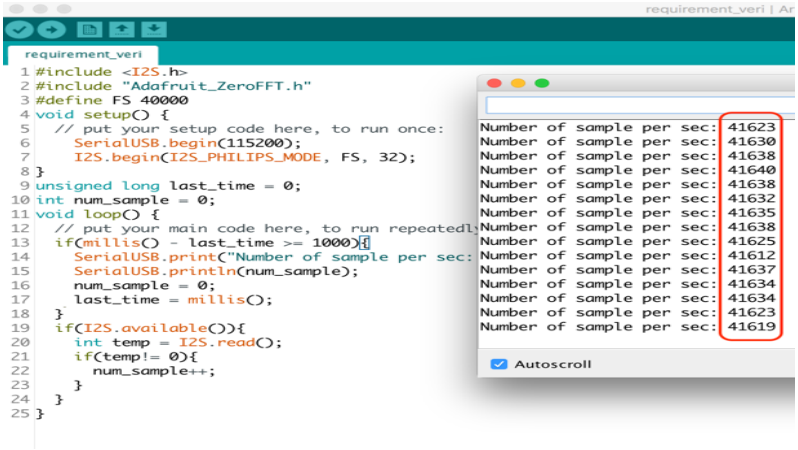

Figure 16. Arduino program to collect and count the number of samples

3. Design Verification

We only discuss the test results in this section. Please refer to Appendix A for the full requirements and verifications table.

3.1 Microphone Module

We wrote an Arduino program to collect and count the number of samples received every second. As shown in figure 16, we received about 41000 samples per second, meaning that the sample rate is higher than 40kHz. Therefore, we satisfied this first requirement.

3.2 Proximity Sensor Module

Table 2 shows the result of the test that we did to verify that our design meets the requirement. As shown in Table 2, the average error percentage of every test case is below 10%. Therefore, we satisfied the requirement for this module.

Table 2

Distance from Sensor (cm)	Average Offset over 10 Samples (cm)	Average Error Percentage (%)
5	0.4	8.00
10	0.6	6.00
15	0.1	0.67
20	0.4	2.00
25	0.4	1.60
30	0.1	0.33
40	0.0	0.00
50	0.4	0.80

3.3 LED Module

For the first requirement, the test result is shown in figure 17. In Figure 17, the rate of 8 clock cycles is 1.5 MHz. Therefore, the rate of 1 clock cycle is 1.5 MHz × 8 = 12 MHz. This shows that we satisfy the first requirement. For the second requirement, figure 18, shows that we can turn on all of 8 LEDs by using 1 shift register controlled by signals from the Arduino Zero. This means that we satisfied the second requirement. For the third requirement, figure 19, shows that we can correctly set the LED's color to be red, orange, green, and blue. For the last requirement, figure21 shows the result when the input voltage is 0V. We can see from the multimeter that the output voltage is 0 V and the LED is off. Figure20 shows the result when the input voltage is 5 V. We can see from the multimeter that the output voltage is 4.98 V and the LED is on. This shows that we satisfy the last requirement.

Figure 17.Test result for first requirement

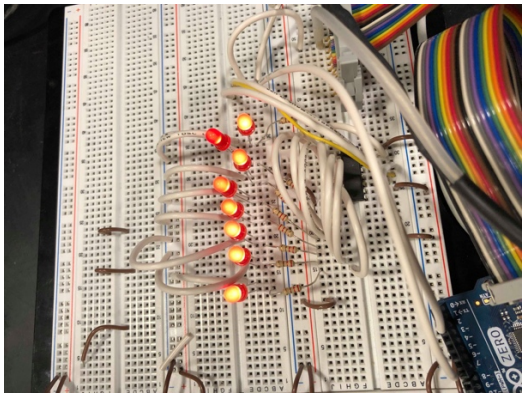

Figure 18.Test result for second requirement

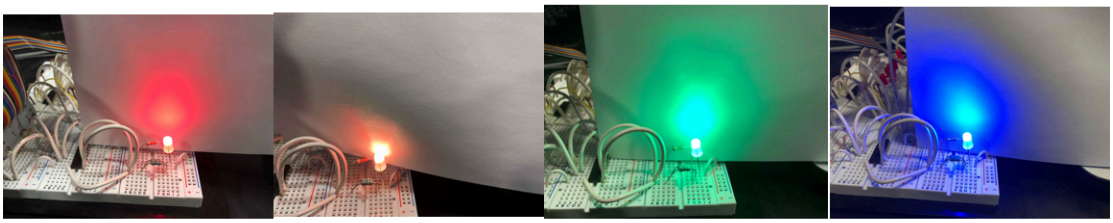

Figure 19. Test result for third requirement

Figure 20. output when the input is 0V

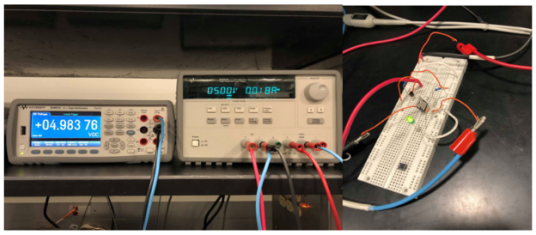

Figure 21. output when the input is 5V

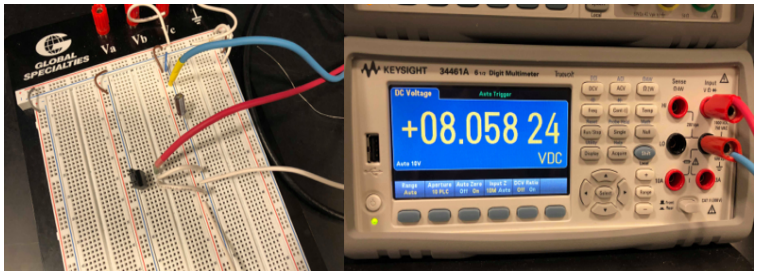

Figure22. 8V voltage regulator test circuit

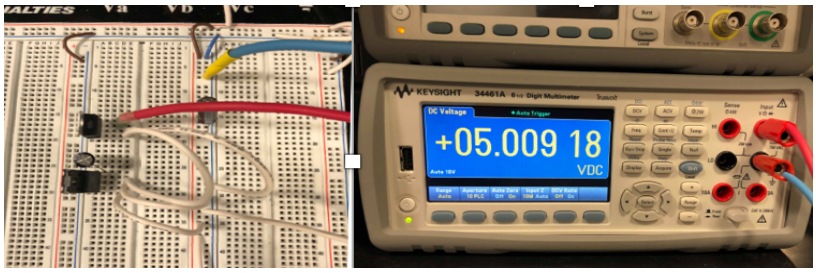

Figure23. 5V voltage regulator test circuit

3.4 Power Module

For the first requirement, figure 22 shows the actual circuit that we used to test the 8 V voltage regulator and the result of the test. We can see that the multimeter shows the output voltage is 8.05824 V, which is within our 8 V +/- 0.5 V. Therefore, we satisfied this requirement. For the second requirement, figure 23 shows the actual circuit that we used to test the 5 V voltage regulator and the result of the test. We can see that the multimeter shows the output voltage is 5.00915 V, which is within our 8 +/- 0.5 V. Therefore, we satisfied this requirement. For the last requirement, figure 24 shows the current consumed by each module. Summing up all the currents consumed by each module, the total current consumed by the whole system is 1.9997 A. Because the adaptor that we use provides a maximum of 10 A, it can sufficiently supply the current for the whole system. Therefore, we satisfied this requirement.

Current consumed by LED Module Current consumed by Sensor Module

Current consumed by Control Unit Current consumed by Microphone Module

Figure24 current consumed by each module

3.5 Control Unit

For the first requirement, Table 2, shows the accuracy rate of detecting the motion of user's hand motion. We can see that over 10 trials, the accuracy rate is higher than 70% for all directions. Therefore, we satisfied the requirement. For the second requirement, figure 25 is the result of the test we performed with a 300 Hz sound generated by the mobile app. As we can see on the right hand side of the figure25, the first frequency bin has the highest amplitude because it has a frequency range from 20 Hz to 625 Hz. Repeating the same test procedure for sound with other frequencies, we also receive results showing that we correctly extract the frequency of the sound. Table 4 shows the average response time and Table3 shows the accuracy rate needed for the third requirement. As shown in table4, the average response time for all 4 directions is less than 10 seconds. Next, table3 shows that the mean time facing correct direction is more than 70% for all direction, meaning that the accuracy is higher than 70%. Therefore, we satisfied the requirement.

Table2 Accuracy rate for proximity sensors

	Left to Right	Right to Left	Up	Down	In	Out
	Correct Output?	Correct Output?	Correct Output?	Correct Output?	Correct Output?	Correct Output?
Trial 1	Yes	No	Yes	Yes	Yes	Yes
Trial 2	Yes	Yes	Yes	Yes	Yes	Yes
Trial 3	Yes	Yes	Yes	Yes	Yes	Yes
Trial 4	Yes	Yes	Yes	Yes	Yes	Yes
Trial 5	Yes	Yes	Yes	Yes	Yes	Yes
Trial 6	No	Yes	Yes	Yes	Yes	Yes
Trial 7	Yes	Yes	Yes	Yes	Yes	Yes
Trial 8	Yes	Yes	Yes	Yes	No	Yes
Trial 9	No	Yes	Yes	Yes	Yes	No
Trial 10	Yes	Yes	Yes	Yes	Yes	Yes
Accuracy Rate (%)	**80%**	**90%**	**100%**	**100%**	**90%**	**90%**

Table3 Mean time facing correct direction and accuracy rate table

	Electret Microphone (Front)	Electret Microphone (Back)	Electret Microphone (Left)	Electret Microphone (Right)
100 Second Trials	Time LEDs Face Front (ms)	Time LEDs Face Back (ms)	Time LEDs Face Left (ms)	Time LEDs Face Right (ms)
Trial 1	87,700	91,600	81,900	85,800
Trial 2	78,100	78,400	79,200	67,100
Trial 3	59,700	82,500	83,200	78,200
Trial 4	74,600	80,500	86,100	84,500
Trial 5	86,300	74,600	62,100	87,800
Mean Time Facing Correct Direction (ms)	77,280	81,520	78,500	80,680
Mean Time Facing Correct Direction (%)	**77.28%**	**81.52%**	**78.50%**	**80.68%**

Table 4 Average Response Time table

	Electret Microphone (Front)	Electret Microphone (Back)	Electret Microphone (Left)	Electret Microphone (Right)
	Time (ms)	Time (ms)	Time (ms)	Time (ms)
Trial 1	9120	7450	9420	6340
Trial 2	8890	8010	8910	7820
Trial 3	9430	6380	9220	5530
Trial 4	9190	9260	8840	7470
Trial 5	8530	6850	9810	8420
Average Response Time (ms)	**9032**	**7590**	**9240**	**7116**

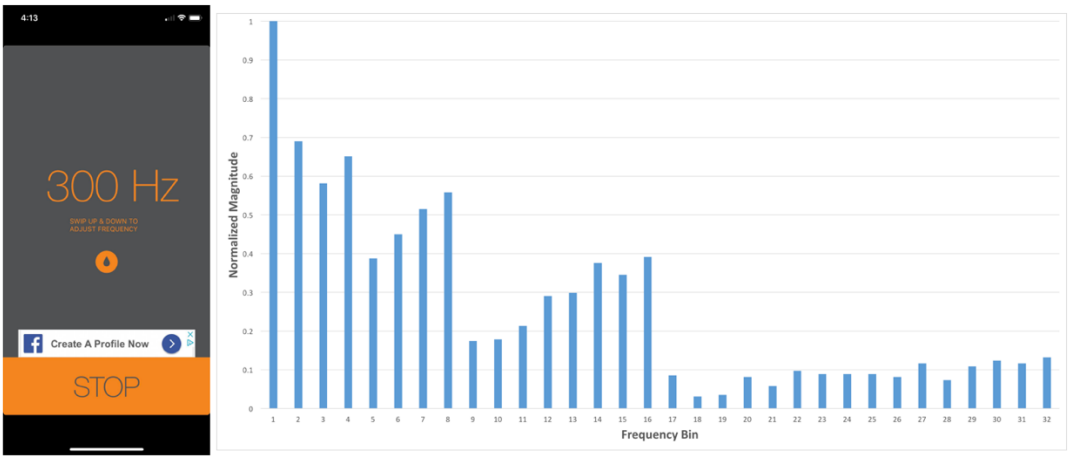

Figure25. Frequency bin responded with 300Hz sound input

4. Cost & Schedule

4.1 Cost Analysis

We assume our hourly salary is $40/hour based on average income for an ECE at Illinois new graduate [8] and work on average 20 hours a week for each member of our team:

$$\text{Total} = 3 \times \frac{\$40}{hr} \times \frac{20\ hr}{week} \times 10 \text{ weeks} \times 2.5 = \$60{,}000$$

Description	Manufacturer	Part Number	Quantity	Cost/Unit	Total Cost
Digital MEMS Microphone	TDK InvenSense	ICS-43434	1	$2.20	$2.20
Electret Condenser Microphone	CUI Inc.	CMA-4544PF-W	4	$0.82	$3.28
8-bit Shift Registers	Fairchild Semiconductor	74HC595	25	$5.49 (10)	$16.47
Low Voltage Audio Power Amplifier	Texas Instruments	LM386-4/NOPB	4	$1.01	$4.04
Linear Voltage Regulator 3.3V	STMicroelectronics	LD1117V33	5	$0.55	$2.75
Linear Voltage Regulator 5V	STMicroelectronics	L7805 TO-220	1	$0.75	$0.75
Arduino Zero Microcontroller	Arduino	N/A	1	$39.00	$39.00
Ultrasonic Range Sensor	ElecFreaks	HC-SR04	3	$3.95	$11.85
Common Cathode RGB 5mm LED	Chanzon	100F5W-YT-RGB-CC	600	$8.96 (100)	$53.76
48W 12V 4A AC Adapter	Kastar	B004FLJ37Q	1	$10.49	$10.49
Total					**$144.59**

Grand total = $60,000 + $144.59 = $60,144.59

4.2 Schedule

Week	Tasks
2/18/2018	• Order parts needed to begin building the project • Finish calculations needed for our device, as well as power requirements • Check in with TA regarding the Arduino Zero usage

	• Finalize schematics of our modules to start work on PCB • Mock Design Review
2/25/2018	Zihan Yan: • Finalize parameters for Microphone Module schematics Islam Kadri: • Begin building LED grid and wiring Hieu Huynh: • Finalize schematics for LED Module Common Tasks: • Prepare for Design Review • Work on soldering assignment • Order parts needed to begin building the project
3/4/2018	Zihan Yan: • Continue building LED grid • Build and test the microphone amplifier circuit Islam Kadri: • Build a simple 2x2x2 LEDs cubes and test the logic of controlling LEDs Hieu Huynh: • Start implementing schematics on PCB Common Tasks: • Talk to Machine Shop regarding a frame for our LED grid, as well as getting some thicker wires • Finish soldering assignment
3/11/2018	Zihan Yan: • Write program to estimate the direction of arrival Islam Kadri: • Write program to calculate Frequency spectrum and amplitude Hieu Huynh: • Write program to drive LEDs cube Common tasks: • Revise PCB and get ready for audit check • Complete LED grid and start setting up sensors/modules • Check in with TA before Spring Break to make sure we are making good progress and to ask about any concerns we may have • Finish building the LEDs cube
3/18/2018	• Spring Break
3/25/2018	Zihan Yan: • Soldering components onto PCB Islam Kadri: • Write software for the Snake Game Hieu Huynh: • Test and debug the Music Mode after putting everything together Common Tasks: • Finish individual progress reports for team members

	• Verify all requirements of all modules • Revise the PCB if needed
4/1/2018	Zihan Yan: • Test the Gaming Mode after putting everything together Islam Kadri: • Start writing Final Report Hieu Huynh: • Continue debugging any problem related to software/hardware Common Tasks: • Make sure project is on schedule and discuss any resolve any enduring issues
4/8/2018	Common Tasks: • Prepare for mock demo to TA • Continue writing Final Report • Test and record a functioning project for official demonstration
4/15/2018	Common Tasks: • Present mock demo • Continue test and record a functioning project for official demonstration • Sign up for demonstration and mock final presentation • Revise final report and practice final presentation and present to class staff
4/22/2018	Common Tasks: • Prepare for final presentation • Perform demo • Finalize final report
4/29/2018	Common Tasks: • Finalize and turn in lab notebooks • Perform lab checkout • Perform final presentation

5. Conclusion

5.1 Accomplishments

In general, we met all the requirements we had at the beginning of the semester. Both features work in the ways we intended and met our expectations. Additionally, our device could display an aesthetically pleasing animation based on real time sound input. We learned a lot about how sensors worked and how to handle different types of data, whether it was analog, digital, or using different communication protocols. We also got over hundreds of hours of experience soldering and debugging connections, which despite the difficulty, proved to be worthwhile due to our success.

5.2 Uncertainties

Although our project met all the requirements, there are a few uncertainties causing by the environmental reasons or the physical property of components. Firstly, each electret microphones have a different sensitivity level when responding to the sound input. In our case, two of the electret microphones were way less sensitive than the other two respective electret microphones: Microphone 1 has the average response time of 9032 ms with the accuracy rate 77.28%, Microphone 2 has the average response time of 7590 ms with the accuracy rate 81.52%, Microphone 3 has the average response time of 9240 ms with the accuracy rate 78.50%, Microphone 4 has the average response time of 7116 ms with the accuracy rate 80.68%. (Table 3) This results in a longer time for directional change for those two mics. Secondly, when the digital microphone takes input sound and maps the frequency pattern onto the LEDs, the background noise also plays a factor and shows up in the frequency. These problems can be fixed by using higher quality microphones. Thirdly, since we soldered and wired the whole grid by hand, there were some connections that were not strong and took several hours to fix.

5.3 Ethical considerations

For this project we abided by IEEE's Code of Ethics [9] and used our moral judgement when needed. We respected all intellectual property laws and were honest with all the estimations and results during the entire process of our project. The public's safety, as well as our own, was a priority and we made sure that we thoroughly tested and mitigated hazards.
We use AC wall outlet to power our device so we need to make sure the voltage regulator is working properly. Our design contains hundreds of LEDs which are wired properly. Also, our device required heavy use with the soldering iron to connect the device's modules together as well as the hundreds of LEDs that we must put together. We followed the lab guidelines. Because the device will be viewed by the public, safety and proper containment of the is always our top priority

5.3 Future work

Although our project met all our requirements, there are still many improvements that could be made. The LED cube building procedure could be improved and standardized so that it would take less time to build the grid and the connections between LEDs and wire connections could be more stable. Increasing the size or the shape of the grid could also be options that could improve the commercial value of our project. Secondly, to provide a more accurate music display, higher quality microphones can be used to capture sound in a more accurate manner. Higher quality ultrasonic sensors or other sensor types such as IR could also be used for better accuracy.

5.4 Ethical considerations

For this project we abided by IEEE's Code of Ethics [9] and used our moral judgement when needed. We respected all intellectual property laws and were honest with all the estimations and results during the entire process of our project. The public's safety, as well as our own, was a priority and we made sure that we thoroughly tested and mitigated hazards.

We used an AC wall outlet to power our device so we need to make sure the voltage regulator is working properly. We need to make sure our design, which contains hundreds of LEDs, are wired properly. Our device required heavy use with the soldering iron to connect the device's modules together as well as the hundreds of LEDs that we must put together. We followed the lab guidelines and made sure to always work in a group of two at the least. Because the device will be viewed by the public, safety and proper containment of the is always our top priority. We had the ECE Machine Shop build a protective case for the wires and the LED grid to ensure top safety for viewers of our project.

References

[1] *Motorola Semiconductor Data Manual,* Motorola Semiconductor Products, Inc., Phoenix, AZ, 2007.

[2] *Double Data Rate (DDR) SDRAM,* datasheet, Micron Technology, Inc., 2000. Available at: http://download.micron.com/pdf/datasheets/dram/ddr/512MBDDRx4x8x16.pdf

[3] Linx Technologies LT Series, web page. Available http://www.linxtechnologies.com/products/rf-modules/lt-series-transceiver-modules/. Accessed January 2012.

[4] W. P. Mondragon, "Principles of coherent light sources: Coherent lasers and pulsed lasers," in *Lasers and Their Applications in Surface Science and Technology,* 2nd ed., J. A. Prufrock, Ed. New York, NY: McGraw-Hill, 2009, pp. 117-132.

[5] G. Liu, "TDM and TWDM de Bruijn nets and shufflenets for optical communications," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 695-701, June 2011.

[6] J. Lewis, "Understanding Microphone Sensitivity," *Understanding Microphone Sensitivity | Analog Devices*. [Online]. Available: http://www.analog.com/en/analog-dialogue/articles/understanding-microphone-sensitivity.html. [Accessed: 03-May-2018].

[7] J. Gamarra-Diezma, A. Miranda-Fuentes, J. Llorens, A. Cuenca, G. Blanco-Roldán, and A. Rodríguez-Lizana, "Testing Accuracy of Long-Range Ultrasonic Sensors for Olive Tree Canopy Measurements," *Sensors*, vol. 15, no. 2, pp. 2902–2919, 2015.

[8] E. I. T. S. Services, "Salary Averages," *Salary Averages :: ECE ILLINOIS*. [Online]. Available: https://ece.illinois.edu/admissions/why-ece/salary-averages.asp. [Accessed: 03-May-2018].

[9] "IEEE Computer Society and ACM Release Approved Software Engineering Code of Ethics," *Entertainment Close-up*, 19-Jan-2016.

Appendix A Requirement and Verification Table

Table 1. System Requirements and Verifications

Requirements	Verification	Verification status (Y or N)
Microphone Module		
1. Digital MEMS microphone must have sampling rate of at least 40 kHz	1. A. Connect pin WS of digital microphone to pin 0 of Arduino Zero B. Connect pin SCK of digital microphone to pin 1 of Arduino Zero C. Connect pin DS of digital microphone to pin 9 of Arduino Zero D. Connect pin VCC of digital microphone to 5V voltage source E. Connect pin GND of digital microphone to Ground F. Load the program shown in figure 16 to Arduino. The program is used to collect data from microphone and count the number of samples every second and display it to the screen E. Verify that the display number is at least 40000 samples per second	**Y**
Proximity Sensor Module		
1. Error from the actual distance versus the distance read from the sensor should be less than 10% (considering only distance under 50 cm)	1. A. Connect pin TRIGGER of ultrasonic sensor to pin 1 of Arduino Zero B. Connect pin ECHO of ultrasonic sensor to pin 1 of Arduino Zero C. Connect pin VCC of ultrasonic sensor to 5 V voltage source D. Connect pin GND of ultrasonic sensor to Ground E. Write program to trigger a PING by setting the trigger pin HIGH for 2 microseconds, then measure the time from sending the ping to the reception of its echo off an object. Calculate distance based on Eq. 1 F. Put a piece of paper at distance 5 cm, 10 cm, 15 cm, 20 cm, 25 cm, 30 cm, 40 cm, 50 cm above the sensor. Run the code described above for each distance	**Y**

	10 times and make sure the display result is at most 10% off.	
LED Module		
1. Shift Register clock is 12 MHz with at most 5% error	1. A. Connect pin VCC of shift register to 5 V voltage source B. Connect pin GND of shift register to GND C. Connect pin CLK of shift register to pin 23 of Arduino Zero D. Connect pin DATA of shift register to pin 24 of Arduino Zero E. Write program using SPI library to shift out data at rate 12MHz F. Use oscilloscope to obtain signal at pin CLK of shift register G. Make sure the clock signal is 12MHz with at most 5% error	**Y**
2. LED can be turned ON/OFF based on Control Unit input signals	2. A. Set up the shift register as described in previous verification procedure B. Connect positive pins of 8 Red LEDs to 8 output pins of the shift register, and negative pins to GND C. Write program to shift out data to turn on all 8 LEDs by shifting out data to the shift register D. Check if all 8 LEDs light up correctly	**Y**
3. LED's color can be controlled based on Control Unit input signals	3. A. Set up the shift register as described in previous verification procedure B. Connect Common Anode pin of RGB LED to 5 V voltage regulator C. Connect the Red, Green, Blue pin of the LED to the output pin 0, 1, and 2 of the shift register D. Write Arduino program to use BAM technique to control the set the color of LED to be Red, Orange, Green and Blue E. Check if the color is correct	**Y**
4. High power circuit driver behaves properly, meaning that when input voltage is 0 V, the output voltage is smaller than 1V; when input voltage is 5 V, the output voltage is 5 +/- 0.5 V	4. A. Set up circuit as shown in figure 20 B. Apply 0 V to the SR_Output, and use multimeter to measure the voltage of LED_Anode. Verify the voltage is less than 1 V and the LED is OFF	**Y**

	C. Apply 5 V to the SR_Output, and use multimeter to measure the voltage of LED_Anode. Verify the voltage is within the range 5+/- 0.5 V and the LED is ON	
Power Module		
1. Output of 8 V voltage regulator is within range 8 +/- 0.5 V when input voltage is 12 V	1. A. Set up circuit as shown in figure 9 B. Use multimeter to measure the output voltage of the voltage regulator C. Verify that the output voltage is within range 8 +/- 0.5 V	**Y**
2. Output of 5 V voltage regulator connected in series with 8 V voltage regulator is within range 5 +/- 0.5 V when input voltage is 12 V	2. A. Set up circuit as shown in figure 10 B. Use multimeter to measure the output voltage of the 5 V voltage regulator C. Verify that the output voltage is within range 5 +/- 0.5 V	**Y**
3. Adaptor can provide enough current for the whole system, meaning that the whole system consumes less than 10 A	3. A. Turn ON all 512 LEDs of the LED module and use the multimeter to measure the current consumed by this module B. Connect VCC pins of 3 Ultrasonic sensors to 5 V voltage source, and GND pins to GND. Use the multimeter to measure the current consumed by this module C. Connect VCC pins of 3 Electret Microphones and MEMS microphone to 5 V voltage source, and GND pins to GND. Use the multimeter to measure the current consumed by this module D. Connect Vin pin of Arduino Zero to 8 V voltage source, and GND pin to GND. Use the multimeter to measure the current consumed by this module E. Sum up all the amount of current consumed by all modules. Verify that the total current is less than 10 A	**Y**
Control Unit		
1. Correctly estimate the movement of user hand in 6 directions (left, right, up,	1. A. Connect the 3 Ultrasonic sensors to the Arduino Zero as shown in figure 4 B. Use the code described in Appendix B to detect motion of the user's hand	**Y**

down, inward, outward) with accuracy at least 70%.	C. Move hand within 15 cm above the sensors in 6 directions, and record the decision made by the program D. Repeat the test 10 times for each direction. Calculate the accuracy rate and make sure that it is at least 70%	
2. Correctly compute the frequency spectrum of the signal in the 20Hz to 20kHz range with frequency resolution of at least 625Hz.	2. A. Connect the MEMS Digital Microphone in the way described in the first verification procedure of Microphone Module B. Use the code described in Appendix B to get the data from the microphone and calculate frequency spectrum C. Normalize and display the magnitude of the frequency bin B. Using an App on the phone to generate sound wave at the following frequencies: 300, 920, 1550, 2175, 2800, 3425, 4050, 4675, 5300, 5925, 6550, 7175, 7800, 8425, 9050, 9675, 10300, 10925, 11550, 12175, 12800, 13425, 14050, 14675, 15300, 15925, 16550, 17175, 17800, 18425, 19050, 19700. Each of the frequency correspond to a frequency bin. B. For each sound wave, verify that the frequency bin with greatest magnitude is the frequency bin corresponding to the frequency of the input sound	**Y**
3. Be able to detect direction and update within 10 seconds with at least 70% accuracy.	3. A. Connect the VCC pins of the 4 Electret Microphones to 5V voltage source B. Connect the GND pins of the 4 Electret Microphones to GND C. Connect the OUT pins of the 4 Electret Microphones to pin A0, A1, A2, A3 of the Arduino Zero C. Put 4 microphones at the positions described in the Physical Design section D. Use the code described in Appendix B to collect the data from the 4 microphones and determine the direction of arrival E. Use an app on the phone to generate	**Y**

	sound wave at frequency 2000 Hz, and put the phone within 5 cm of left microphone for 100 seconds. F. Measure the amount of time that the direction is correct. Verify that the amount of time that the direction is correct is at least 70 seconds, meaning that the accuracy is at least 70% G. Repeat steps E and F for other 3 directions H. Use an app on the phone to generate sound wave at frequency 2000 Hz, and put the phone within 5 cm of left microphone I. Measure how long it takes for the program to detect the sound coming from left direction. Verify that the response time is within 10 seconds J. Repeat steps H and I for the other 3 directions	

APPENDIX B Arduino Code

```
1.  //some code intentionally left out
2.  include < I2S.h > #include "Adafruit_ZeroFFT.h"#
3.  include < SPI.h > #include < Arduino.h > #include "avdweb_SAMDtimer.h" //https://github.com/dhepper/font8x
    8/blob/master/font8x8_basic.h
4.      #
5.  char font8x8_basic[128][8] = {  }; //---------------------------------------------------------------
     //LED DRIVER
6.  byte blue_array[NUM_LED_PER_LEVEL];
7.  byte green_array[NUM_LED_PER_LEVEL];
8.  byte red[NUM_BAM_BIT][NUM_LED_PER_LEVEL];
9.  byte green[NUM_BAM_BIT][NUM_LED_PER_LEVEL];
10. byte blue[NUM_BAM_BIT][NUM_LED_PER_LEVEL];
11. byte anode[DIMENSION];
12. bool is_handle_on_receive;
13. unsigned long start_restart;
14. bool stuck_flag;
15. void led_driver_setup() {
16.     pinMode(LATCH_PIN, OUTPUT);
17.     pinMode(SR_CLOCK_PIN, OUTPUT);
18.     pinMode(DATA_PIN, OUTPUT);
19.     pinMode(BLANK_PIN, OUTPUT);
20.     digitalWrite(BLANK_PIN, HIGH);
21.     for (int i = 0; i < NUM_BAM_BIT; i++) {
22.         for (int j = 0; j < NUM_LED_PER_LEVEL; j++) {
23.             red[i][j] = 0xFF;
24.             green[i][j] = 0xFF;
25.             blue[i][j] = 0xFF;
26.             red_array[j] = 0xFF;
27.             blue_array[j] = 0xFF;
28.             green_array[j] = 0xFF;
29.         }
30.     }
31.     for (int i = 0; i < 8; i++) {
32.         anode[i] = 1 << i;
33.     }
34.     SPISettings mySettting(12000000, MSBFIRST, SPI_MODE0);
35.     SPI.begin();
36. }
37. void interrupt_handler_music(struct tc_module *
38.     const module_inst) {
39.     shift_out_to_reg();
40.     if (I2S.available() && is_handle_on_receive == false) {
41.         int x = I2S.read();
42.         if (x) { //      count_avail ++;
43.             start_restart = millis();
44.         }
45.     } else {
46.         if (millis() - start_restart >= 100 && start_restart != 0) {
47.             stuck_flag = true;
48.         }
49.     }
50. }
51. void interrupt_handler_snake(struct tc_module *
52.     const module_inst) {
53.     shift_out_to_reg();
54. }
55. SAMDtimer my_timer_music = SAMDtimer(5, interrupt_handler_music, 1500, 0);
56. SAMDtimer my_timer_snake = SAMDtimer(4, interrupt_handler_snake, 1500, 0);
57. int BAM_counter;
58. int BAM_bit;
59. int cur_level;
60. int level_idx;
61. void shift_out_to_reg() { //turn off all LEDs
62.     digitalWrite(BLANK_PIN, HIGH);
63.     for (int i = level_idx; i < level_idx + 8; i++) {
```

```
        SPI.transfer(red_array[i]);
    }
    for (int i = level_idx; i < level_idx + 8; i++) {
        SPI.transfer(green_array[i]);
    }
    for (int i = level_idx; i < level_idx + 8; i++) {
        SPI.transfer(blue_array[i]);
    }
    SPI.transfer(anode[cur_level]); //Load data into register
    digitalWrite(LATCH_PIN, HIGH);
    digitalWrite(LATCH_PIN, LOW); //Enable output
    digitalWrite(BLANK_PIN, LOW);
    cur_level = (cur_level + 1) % 8;
    level_idx = (level_idx + 8) % 64;
}
void setLED(int x, int y, int level, bool r, bool g, bool b) {
    if (x < 0 || x >= DIMENSION || y < 0 || y >= DIMENSION || level < 0 || level >= DIMENSION) {
        return;
    }
    int byte_num = level * 8 + y;
    if (r) {
        bitWrite(red_array[byte_num], x, 0);
    } else {
        bitWrite(red_array[byte_num], x, 1);
    }
    if (g) {
        bitWrite(green_array[byte_num], x, 0);
    } else {
        bitWrite(green_array[byte_num], x, 1);
    }
    if (b) {
        bitWrite(blue_array[byte_num], x, 0);
    } else {
        bitWrite(blue_array[byte_num], x, 1);
    }
}
void set_LED(int x, int y, int level, int r, int g, int b) { //Error checking
        if (x < 0 || x >= DIMENSION || y < 0 || y >= DIMENSION || level < 0 || level >= DIMENSION || r < 0
    || r > COLOR_MAX || g < 0 || g > COLOR_MAX || b < 0 || b > COLOR_MAX) {
            return;
        }
        int byte_num = level << 3 + y;
        for (int i = 0; i < NUM_BAM_BIT; i++) {
            bitWrite(red[i][byte_num], x, bitRead(r, i));
            bitWrite(green[i][byte_num], x, bitRead(g, i));
            bitWrite(blue[i][byte_num], x, bitRead(b, i));
        }
    }
int ready_idx;
bool is_doing_fft;
unsigned long spectrum[DATA_SIZE / 2];
unsigned long column[NUM_COLUMN];
int column_height[NUM_COLUMN];
int max_sample_receive = 0;
int num_block = 0;
int currently_writing = 0;
int16_t sample_buf[NUM_BUF][MAX_LENGTH];
int sample_idx = 0;
int prev_column_height[8];#
define AMPLITUDE_PER_LEVEL 1500000 unsigned long avg_amplitude;
void setup_mic() { // start I2S at 8 kHz with 32-bits per sample //Serial.println("Start seting Mic");
    I2S.onReceive(on_receive_handler);
    if (!I2S.begin(I2S_PHILIPS_MODE, FS, 32)) { //    Serial.println("Failed to initialize I2S!");
        while (1); // do nothing
    }
    for (int i = 0; i < DATA_SIZE / 2; i++) {
        spectrum[i] = 0;
    }
    for (int i = 0; i < NUM_COLUMN; i++) {
```

```
        column[i] = 0;
        column_height[i] = 0;
    }
    ready_idx = -1;
    is_doing_fft = false;
    is_handle_on_receive = false;
    start_restart = 0;
    stuck_flag = false;
    for (int i = 0; i < 8; i++) {
        prev_column_height[i] = -1;
    }
    avg_amplitude = 0;
}
void on_receive_handler() {
    is_handle_on_receive = true;
    read_and_store();
    is_handle_on_receive = false;
}
void read_and_store() {
    if (I2S.available()) {
        while (I2S.available()) {
            int32_t sample = I2S.read();
            if (sample != 0) {
                unsigned long temp = 7 * avg_amplitude + abs(sample);
                avg_amplitude = temp / 8;
                sample = sample / 65538; // 65538 = 2147483647/ 32767;
                sample_buf[currently_writing][sample_idx] = (int16_t) sample;
                sample_idx = (sample_idx + 1) % DATA_SIZE;
                if (sample_idx == 0) {
                    if (is_doing_fft == true) { //do nothing
                    } else {
                        ready_idx = currently_writing;
                        currently_writing = (currently_writing + 1) % 2;
                    }
                }
                start_restart = millis();
            }
        }
    } else {
        if (millis() - start_restart >= 100 && start_restart != 0) {
            stuck_flag = true;
        }
    }
}
void calculate_fft(int buf_idx) {
    ZeroFFT(sample_buf[buf_idx], DATA_SIZE); //  set_spectrum(buf_idx);
    set_column_height(buf_idx);
}
int prev_amp_range = -1;
int target_column_height[8];
int temp_column_height[8];
unsigned long column_update_time[8];#
define UP_TIME 20# define DOWN_TIME 50 int need_update_count;
unsigned long last_time_update_freq_to_cube = 0;
void update_freq_to_cube_helper(int amp_range_, int local_cur_dir) {
    for (int i = 0; i < 8; i++) {
        if (target_column_height[i] < temp_column_height[i]) {
            int time_temp = millis();
            if (time_temp - column_update_time[i] >= DOWN_TIME) {
                if (local_cur_dir == DIR_Y_0) {
                    for (int y = 0; y < 4; y++) {
                        setLED(7 - i, y, temp_column_height[i], 0, 0, 0);
                    }
                } else if (local_cur_dir == DIR_Y_7) {
                    for (int y = 4; y < 8; y++) {
                        setLED(i, y, temp_column_height[i], 0, 0, 0);
                    }
                } else if (local_cur_dir == DIR_X_0) {
                    for (int x = 0; x < 4; x++) {
```

```
                        setLED(x, i, temp_column_height[i], 0, 0, 0);
                    }
                } else if (local_cur_dir == DIR_X_7) {
                    for (int x = 4; x < 8; x++) {
                        setLED(x, 7 - i, temp_column_height[i], 0, 0, 0);
                    }
                }
                temp_column_height[i]--;
                column_update_time[i] = time_temp;
            }
        } else if (target_column_height[i] >= temp_column_height[i]) {
            int time_temp = millis();
            if (time_temp - column_update_time[i] >= UP_TIME) {
                if (local_cur_dir == DIR_Y_0) {
                    for (int y = 0; y < 4; y++) {
                        set_led_helper(7 - i, y, temp_column_height[i], amp_range_);
                    }
                } else if (local_cur_dir == DIR_Y_7) {
                    for (int y = 4; y < 8; y++) {
                        set_led_helper(i, y, temp_column_height[i], amp_range_);
                    }
                } else if (local_cur_dir == DIR_X_0) {
                    for (int x = 0; x < 4; x++) {
                        set_led_helper(x, i, temp_column_height[i], amp_range_);
                    }
                } else if (local_cur_dir == DIR_X_7) {
                    for (int x = 4; x < 8; x++) {
                        set_led_helper(x, 7 - i, temp_column_height[i], amp_range_);
                    }
                }
                temp_column_height[i]++;
                column_update_time[i] = time_temp;
            }
        }
    }
}
unsigned long last_time_update_amp_range = 0;
void update_freq_to_cube(int local_cur_dir, bool immediately_update_flag) {
        unsigned long cur_time = millis();
        for (int i = 0; i < 8; i++) {
            target_column_height[i] = column_height[i];
        }
        last_time_update_freq_to_cube = cur_time;
        if (cur_time - last_time_update_amp_range >= 1500 || immediately_update_flag == true) {
            int amp_range = prev_amp_range; //    if(immediately_update_flag == false){
            if (amp_range >= AMPLITUDE_PER_LEVEL * 7) {
                amp_range = 7;
            } else {
                amp_range = avg_amplitude / AMPLITUDE_PER_LEVEL;
                amp_range++;
                amp_range = min(amp_range, 7);
            } //    }
            if (prev_amp_range != amp_range || immediately_update_flag == true) {
                if (local_cur_dir == DIR_X_0) {
                    for (int i = 0; i < 8; i++) {
                        for (int h = 0; h <= temp_column_height[i]; h++) {
                            for (int x = 0; x < 4; x++) {
                                set_led_helper(x, i, h, amp_range);
                            }
                        }
                    }
                } else if (local_cur_dir == DIR_X_7) {
                    for (int i = 0; i < 8; i++) {
                        for (int h = 0; h <= temp_column_height[i]; h++) {
                            for (int x = 4; x < 8; x++) {
                                set_led_helper(x, 7 - i, h, amp_range);
                            }
                        }
                    }
```

```
                } else if (local_cur_dir == DIR_Y_0) {
                    for (int i = 0; i < 8; i++) {
                        for (int h = 0; h <= temp_column_height[i]; h++) {
                            for (int y = 0; y < 4; y++) {
                                set_led_helper(7 - i, y, h, amp_range);
                            }
                        }
                    }
                } else if (local_cur_dir == DIR_Y_7) {
                    for (int i = 0; i < 8; i++) {
                        for (int h = 0; h <= temp_column_height[i]; h++) {
                            for (int y = 4; y < 8; y++) {
                                set_led_helper(i, y, h, amp_range);
                            }
                        }
                    }
                }
                prev_amp_range = amp_range;
            }
            last_time_update_amp_range = cur_time;
        }
        update_freq_to_cube_helper(prev_amp_range, local_cur_dir);
    }
    /* * R * G * B * R + G * R + B * B + G  * R + G + B */
void set_led_helper(int x, int y, int z, int color) {
    if (color == 0) {
        setLED(x, y, z, 1, 0, 0);
    } else if (color == 1) {
        setLED(x, y, z, 0, 1, 0);
    } else if (color == 2) {
        setLED(x, y, z, 0, 0, 1);
    } else if (color == 3) {
        setLED(x, y, z, 1, 1, 0);
    } else if (color == 4) {
        setLED(x, y, z, 1, 0, 1);
    } else if (color == 5) {
        setLED(x, y, z, 0, 1, 1);
    } else if (color == 6) {
        setLED(x, y, z, 1, 1, 1);
    }
}
void set_column_height_helper(int buf_idx, int column_idx, int start_idx, int end_idx) {
    unsigned long sum = 0;
    for (int i = start_idx; i <= end_idx; i++) {
        sum = sum + sample_buf[buf_idx][i];
    } //  sum = sum / (end_idx - start_idx +1);
    column[column_idx] = (column[column_idx] * 7 + sum) / 8;
}
void set_column_height(int buf_idx) {
    set_column_height_helper(buf_idx, 0, 0, 0);
    set_column_height_helper(buf_idx, 1, 1, 1);
    set_column_height_helper(buf_idx, 2, 2, 2);
    set_column_height_helper(buf_idx, 3, 3, 3);
    set_column_height_helper(buf_idx, 4, 4, 4);
    set_column_height_helper(buf_idx, 5, 5, 6);
    set_column_height_helper(buf_idx, 6, 7, 15);
    set_column_height_helper(buf_idx, 7, 16, 31);
    unsigned long _max = 0;
    int max_idx = -1;
    for (int i = 0; i < DATA_SIZE / 2; i++) {
        spectrum[i] = (7 * spectrum[i] + sample_buf[buf_idx][i]) / 9;
        if (_max < spectrum[i] && spectrum[i] > 0) {
            _max = max(_max, spectrum[i]);
            max_idx = i;
        }
    }
    int max_val = 0;
    int sec_max_val = 0;
    bool last_column_flag = false;
```

```
    bool col_6_flag = false;
    bool col_7_flag = false;
    for (int i = 0; i < 8; i++) {
        if (max_val <= column[i]) {
            if (i == 5 && !(max_idx == 5 || max_idx == 6)) {
                column[5] = column[5] / 2;
            }
            if (i == 6 && !(max_idx >= 7 && max_idx <= 15)) { //8, 15
                column[6] = column[6] / 9;
                continue;
            }
            if (i == 7 && !(max_idx >= 16)) {
                column[7] = column[7] / 16;
                col_7_flag = true;
                continue;
            }
            max_val = column[i];
            sec_max_val = max_val;
        }
    }
    if (max_val <= 1) {
        return;
    } else {
        for (int i = 0; i < 8; i++) {
            column_height[i] = max(0, (column[i] * 7) / max_val);
            column_height[i] = min(7, column_height[i]);
        }
    }
    return;
}
int prev_max_div_4 = -1;
void set_spectrum(int buf_idx) {
        int _max = 0;
        int max_freq = -1;
        int max_idx = -1;
        for (int i = 0; i < DATA_SIZE / 2; i++) {
            spectrum[i] = (7 * spectrum[i] + sample_buf[buf_idx][i]) / 8;
            if (_max < spectrum[i] && spectrum[i] > 0) {
                max_freq = FFT_BIN(i, FS, DATA_SIZE);
                _max = max(_max, spectrum[i]);
                max_idx = i;
            }
        }
        int max_div_4 = max_idx / 4;
        if (max_idx >= 0) {
            if (prev_max_div_4 == max_div_4) {
                return;
            }
            prev_max_div_4 = max_div_4;
            for (int x = 0; x < 8; x++) {
                for (int y = 4; y < 8; y++) {
                    for (int h = 0; h < 8; h++) {
                        setLED(x, y, h, 0, 0, 0);
                    }
                }
            }
            for (int y = 4; y < 8; y++) {
                for (int h = 0; h < 8; h++) {
                    setLED(max_div_4, y, h, 0, 0, 1);
                }
            }
        }
    } //------------------------------------------------------------ //SNAKE GAME
byte snake_head_x;
byte snake_head_y;
byte snake_head_z;
byte snake_tail_x;
byte snake_tail_y;
byte snake_tail_z;
```

```
int cur_dir;
int next_dir;
int next_segment_array_idx;
int segment_cur_size;
byte segment_length_array[MAX_SEGMENT];
int segment_dir_array[MAX_SEGMENT];
bool cell_status[8][8][8];
byte fruit_pos_x;
byte fruit_pos_y;
byte fruit_pos_z;
bool is_first_time_update = true; ///
byte snake_pos_array[3][SIZE_SNAKE_ARRAY];
int snake_tail_idx = 0;
int snake_head_idx = 0;
int snake_length;
void init_snake_game() {
    snake_pos_array[0][0] = 3;
    snake_pos_array[1][0] = 3;
    snake_pos_array[2][0] = 3;
    fruit_pos_x = 0;
    fruit_pos_y = 0;
    fruit_pos_z = 0; //Initialize cell_status array
    for (int x = 0; x < 8; x++) {
        for (int y = 0; y < 8; y++) {
            for (int z = 0; z < 8; z++) {
                cell_status[x][y][z] = EMPTY;
            }
        }
    }
    cell_status[snake_pos_array[0][0]][snake_pos_array[1][0]][snake_pos_array[2][0]] = NON_EMPTY;
    setLED(snake_pos_array[0][0], snake_pos_array[1][0], snake_pos_array[2][0], 0, 1, 0);
    snake_length = 1;
    generate_new_fruit();
    init_hand_detection();
    return;
}
void update_func_new() {
    byte next_snake_head_x = snake_pos_array[0][snake_head_idx];
    byte next_snake_head_y = snake_pos_array[1][snake_head_idx];
    byte next_snake_head_z = snake_pos_array[2][snake_head_idx];
    bool is_change_dir = false;
    bool acquire_food = false;
    if (next_dir == -
cur_dir) { //Opposite to the current dir; Not change cur_dir //    Serial.println("Try to move in opposite
 dir");
        next_dir = cur_dir;
    } else if (next_dir != cur_dir) {
        is_change_dir = true;
        cur_dir = next_dir;
    }
    switch (next_dir) {
    }
    setLED(snake_pos_array[0][snake_head_idx], snake_pos_array[1][snake_head_idx], snake_pos_array[2][snak
e_head_idx], 0, 1, 0);
    setLED(next_snake_head_x, next_snake_head_y, next_snake_head_z, 1, 1, 1);
    if (fruit_pos_x == next_snake_head_x && fruit_pos_y == next_snake_head_y && fruit_pos_z == next_snake_
head_z) {
        acquire_food = true;
        snake_length++;
    }
    if (next_snake_head_x >= 8 || next_snake_head_x < 0 || next_snake_head_y >= 8 || next_snake_head_y < 0
 || next_snake_head_z >= 8 || next_snake_head_z < 0) { //Out of bound
        end_game();
        return;
    }
    if (cell_status[next_snake_head_x][next_snake_head_y][next_snake_head_z] == NON_EMPTY && acquire_food
== false) { //hit itself
        int non_empty_count = 0;
        for (int x = 0; x < 8; x++) {
```

```
            for (int y = 0; y < 8; y++) {
                for (int z = 0; z < 8; z++) {
                    if (cell_status[x][y][z] == NON_EMPTY) non_empty_count++;
                }
            }
        }
        end_game();
        return;
    }
    snake_head_idx = (snake_head_idx + 1) % SIZE_SNAKE_ARRAY;
    snake_pos_array[0][snake_head_idx] = next_snake_head_x;
    snake_pos_array[1][snake_head_idx] = next_snake_head_y;
    snake_pos_array[2][snake_head_idx] = next_snake_head_z;
    cell_status[next_snake_head_x][next_snake_head_y][next_snake_head_z] = NON_EMPTY;
    if (acquire_food == false) { //Delete the obsolete cell //Turn off LED of the tail
        cell_status[snake_pos_array[0][snake_tail_idx]][snake_pos_array[1][snake_tail_idx]][snake_pos_arra
    y[2][snake_tail_idx]] = EMPTY;
        setLED(snake_pos_array[0][snake_tail_idx], snake_pos_array[1][snake_tail_idx], snake_pos_array[2][
    snake_tail_idx], 0, 0, 0);
        snake_tail_idx = (snake_tail_idx + 1) % SIZE_SNAKE_ARRAY;
    } else {
        generate_new_fruit();
    }
}
void write_letter(char letter_array[8]) {
    for (int h = 0; h < 8; h++) {
        for (int x = 0; x < 8; x++) {
            int temp = (letter_array[h] >> x) & 0x01;
            setLED(x, 7, 7 - h, 0, temp, temp);
        }
    }
}
void end_game() {
    char E_array[8] = {
        0x7F, 0x46, 0x16, 0x1E, 0x16, 0x46, 0x7F, 0x00
    };
    write_letter(E_array);
    delay(1500);
    char N_array[8] = {
        0x63, 0x67, 0x6F, 0x7B, 0x73, 0x63, 0x63, 0x00
    };
    write_letter(N_array);
    delay(1500);
    char D_array[8] = {
        0x1F, 0x36, 0x66, 0x66, 0x66, 0x36, 0x1F, 0x00
    };
    write_letter(D_array);
    delay(1500); //  Serial.println("END GAME");
    for (int z = 0; z < 8; z++) {
        for (int x = 0; x < 8; x++) {
            for (int y = 0; y < 8; y++) {
                setLED(x, y, z, 1, 0, 0);
            }
            delay(50);
        }
    }
    delay(500);
    init_snake_game();
}
unsigned long last_time_update_snake = 0;
void generate_new_fruit() {
    int temp_x = random(0, 7);
    int temp_y = random(0, 7);
    int temp_z = random(0, 7);
    while (cell_status[temp_x][temp_y][temp_z] == NON_EMPTY) {
        if ((temp_z + 1) % 8 != fruit_pos_z) {
            temp_z = (temp_z + 1) % 8;
        } else if ((temp_x + 1) % 8 != fruit_pos_x) {
            temp_x = (temp_x + 1) % 8;
```

```
        }
        if ((temp_y + 1) % 8 != fruit_pos_y) {
            temp_y = (temp_y + 1) % 8;
        }
    }
    fruit_pos_x = temp_x;
    fruit_pos_y = temp_y;
    fruit_pos_z = temp_z;
    cell_status[fruit_pos_x][fruit_pos_y][fruit_pos_z] = NON_EMPTY;
    setLED(fruit_pos_x, fruit_pos_y, fruit_pos_z, 1, 0, 1);
}
bool can_move(int dir) {
    int x = snake_pos_array[0][snake_head_idx];
    int y = snake_pos_array[1][snake_head_idx];
    int z = snake_pos_array[2][snake_head_idx];
    if (dir == POSITIVE_X) {
        x++;
    } else if (dir == NEGATIVE_X) {
        x--;
    } else if (dir == POSITIVE_Y) {
        y++;
    } else if (dir == NEGATIVE_Y) {
        y--;
    } else if (dir == POSITIVE_Z) {
        z++;
    } else if (dir == NEGATIVE_Z) {
        z--;
    }
    if (x >= 8 || y >= 8 || z >= 8 || x < 0 || y < 0 || z < 0) return false;
    if (cell_status[x][y][z] == EMPTY || (x == fruit_pos_x && y == fruit_pos_y && z == fruit_pos_z)) retur
n true;
    else return false;
}
void auto_snake() {
    if (is_first_time_update == true) {
        last_time_update_snake = millis();
        is_first_time_update = false;
        return;
    }
    if (fruit_pos_x > snake_pos_array[0][snake_head_idx] && cur_dir != -
POSITIVE_X && can_move(POSITIVE_X)) {
        next_dir = POSITIVE_X;
    } else if (fruit_pos_x < snake_pos_array[0][snake_head_idx] && cur_dir != -
NEGATIVE_X && can_move(NEGATIVE_X)) {
        next_dir = NEGATIVE_X;
    } else if (fruit_pos_y > snake_pos_array[1][snake_head_idx] && cur_dir != -
POSITIVE_Y && can_move(POSITIVE_Y)) {
        next_dir = POSITIVE_Y;
    } else if (fruit_pos_y < snake_pos_array[1][snake_head_idx] && cur_dir != -
NEGATIVE_Y && can_move(NEGATIVE_Y)) {
        next_dir = NEGATIVE_Y;
    } else if (fruit_pos_z < snake_pos_array[2][snake_head_idx] && cur_dir != -
NEGATIVE_Z && can_move(NEGATIVE_Z)) {
        next_dir = NEGATIVE_Z;
    } else if (fruit_pos_z > snake_pos_array[2][snake_head_idx] && cur_dir != -
POSITIVE_Z && can_move(POSITIVE_Z)) {
        next_dir = POSITIVE_Z;
    } else {
        if (can_move(POSITIVE_X)) {
            next_dir = POSITIVE_X;
        } else if (can_move(POSITIVE_Y)) {
            next_dir = POSITIVE_Y;
        } else if (can_move(POSITIVE_Z)) {
            next_dir = POSITIVE_Z;
        } else if (can_move(NEGATIVE_X)) {
            next_dir = NEGATIVE_X;
        } else if (can_move(NEGATIVE_Y)) {
            next_dir = NEGATIVE_Y;
        } else {
```

```
            next_dir = NEGATIVE_Z;
        }
    }
    if (millis() - last_time_update_snake >= SNAKE_UPDATE_INTERVAL) {
        update_func_new();
        last_time_update_snake = millis();
    }
}
bool update_speed_flag = true;
void snake_game() {
        detect_function();
        if (is_first_time_update == true) {
            next_dir = POSITIVE_X;
            cur_dir = POSITIVE_X;
            last_time_update_snake = millis();
            is_first_time_update = false;
            return;
        }
        if (millis() - last_time_update_snake >= SNAKE_UPDATE_INTERVAL) {
            update_func_new();
            last_time_update_snake = millis();
            if (snake_length % 5 == 0 && update_speed_flag == true) {
                SNAKE_UPDATE_INTERVAL = SNAKE_UPDATE_INTERVAL - 100;
                update_speed_flag = false;
            }
            if (snake_length % 5 == 1) {
                update_speed_flag = true;
            }
        }
        return;
    } //------------------------------------------------------------
    //HAND_MOTION // defines pins numbers
const int trigPin_L = 4;
const int echoPin_L = 5;
const int trigPin_R = 10;
const int echoPin_R = 11;
const int trigPin_T = 12;
const int echoPin_T = 13; // defines variables
long duration_L;
int distance_L;
long duration_R;
int distance_R;
int max_dist;
int min_dist;
long duration_T;
int distance_T;
bool pass_L;
bool pass_R;
bool pass_T;
bool above_L;
bool above_R;
bool above_T;
bool is_Down;
bool is_Up; //int prev_dist;
unsigned long start_R;
unsigned long start_L;
unsigned long last_time_detect;
void init_hand_detection() {
    pinMode(trigPin_L, OUTPUT); // Sets the trigPin as an Output
    pinMode(echoPin_L, INPUT); // Sets the echoPin as an Input
    pinMode(trigPin_R, OUTPUT); // Sets the trigPin as an Output
    pinMode(echoPin_R, INPUT); // Sets the echoPin as an Input
    pinMode(trigPin_T, OUTPUT); // Sets the trigPin as an Output
    pinMode(echoPin_T, INPUT); // Sets the echoPin as an Input
    pass_L = false;
    pass_R = false;
    pass_T = false;
    above_L = false;
    above_R = false;
```

```
    above_T = false;
    max_dist = 0;
    min_dist = 100;
    is_Down = false;
    is_Up = false;
    last_time_detect = 0;
}
void write_char(char c) {
    char temp_array[8];
    for (int i = 0; i < 8; i++) {
        temp_array[i] = font8x8_basic[c][i];
    }
    write_letter(temp_array);
}#
define THRESHOLD 15 void detect_function() { //  Serial.println("detect_function: Start detec");
        if (millis() - last_time_detect < 300) return; // Clears the trigPin
        digitalWrite(trigPin_L, LOW);
        delayMicroseconds(2); // Sets the trigPin on HIGH state for 10 micro seconds
        digitalWrite(trigPin_L, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin_L, LOW); // Reads the echoPin, returns the sound wave travel time in microsec
    onds
        duration_L = pulseIn(echoPin_L, HIGH); // Calculating the distance
        distance_L = duration_L * 0.034 / 2; ///
        digitalWrite(trigPin_R, LOW);
        delayMicroseconds(2); // Sets the trigPin on HIGH state for 10 micro seconds
        digitalWrite(trigPin_R, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin_R, LOW); // Reads the echoPin, returns the sound wave travel time in microsec
    onds
        duration_R = pulseIn(echoPin_R, HIGH); // Calculating the distance
        distance_R = duration_R * 0.034 / 2; // Clears the trigPin
        digitalWrite(trigPin_T, LOW);
        delayMicroseconds(2); // Sets the trigPin on HIGH state for 10 micro seconds
        digitalWrite(trigPin_T, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin_T, LOW); // Reads the echoPin, returns the sound wave travel time in microsec
    onds
        duration_T = pulseIn(echoPin_T, HIGH); // Calculating the distance
        distance_T = duration_T * 0.034 / 2;
        if (distance_T < THRESHOLD) {
            above_T = true;
        } else {
            above_T = false;
        }
        if (distance_L < THRESHOLD) {
            if (above_L == false) {
                start_L = millis();
                above_L = true;
            } else {
                unsigned long cur_time = millis();
                if (cur_time - start_L > 200) {
                    if (is_Down == false) {
                        next_dir = NEGATIVE_Z;
                    }
                    is_Down = true;
                    above_L = false;
                    while (1) { // Sets the trigPin on HIGH state for 10 micro seconds
                        digitalWrite(trigPin_L, HIGH);
                        delayMicroseconds(10);
                        digitalWrite(trigPin_L, LOW);
                        duration_L = pulseIn(echoPin_L, HIGH); // Calculating the distance
                        distance_L = duration_L * 0.034 / 2;
           if (distance_L > THRESHOLD || (millis() - last_time_update_snake >= SNAKE_UPDATE_INTERVAL)) {
                            pass_L = false;
                            pass_R = false;
                            pass_T = false;
                            last_time_detect = millis();
                            return;
```

```
                        }
                    }
                }
            }
        } else {
            if (is_Down == true) {
                pass_L = false;
                pass_R = false;
                pass_T = false;
            }
            above_L = false;
            is_Down = false;
        }
        if (distance_R < THRESHOLD) {
            if (above_R == false) {
                start_R = millis();
                above_R = true;
            } else {
                unsigned long cur_time = millis();
                if (cur_time - start_R > 200) {
                    if (is_Up == false) {
                        next_dir = POSITIVE_Z;
                    }
                    is_Up = true;
                    above_R = false;
                    while (1) { // Sets the trigPin on HIGH state for 10 micro seconds
                        digitalWrite(trigPin_R, HIGH);
                        delayMicroseconds(10);
                        digitalWrite(trigPin_R, LOW);
                        duration_R = pulseIn(echoPin_R, HIGH); // Calculating the distance
                        distance_R = duration_R * 0.034 / 2;
                        if (distance_R > THRESHOLD || (millis() -
    last_time_update_snake >= SNAKE_UPDATE_INTERVAL)) {
                            pass_L = false;
                            pass_R = false;
                            pass_T = false;
                            last_time_detect = millis(); //            delay(300);
                            return;
                        }
                    }
                }
            }
        } else {
            if (is_Up == true) {
                pass_L = false;
                pass_R = false;
                pass_T = false;
            }
            above_R = false;
            is_Up = false;
        }
        if (above_L == true && above_R == false && pass_R == false) {
            pass_L = true;
        } else if (above_L == false && above_R == true && pass_L == false) {
            pass_R = true;
        }
        if (above_T == true && above_R == false && pass_R == false) {
            pass_T = true;
        }
        bool flag = false;
        if (pass_R == true && above_T == true) {
            next_dir = NEGATIVE_Y;
            flag = true;
            last_time_detect = millis(); //    delay(300);
        } else if (pass_T == true && above_R == true) {
            next_dir = POSITIVE_Y;
            flag = true;
            last_time_detect = millis(); //    delay(300);
        }
```

```
        if (pass_L == true && above_R == true) {
            next_dir = POSITIVE_X;
            flag = true;
            last_time_detect = millis(); //    delay(300);
        } else if (pass_R == true && above_L == true) {
            next_dir = NEGATIVE_X;
            flag = true;
            last_time_detect = millis(); //    delay(300);
        }
        if (flag == true) {
            pass_L = false;
            pass_R = false;
            pass_T = false;
        }
    } //------------------------------------------------------------ //DIRECTION OF ARRIVAL
    #
define BUF_SIZE 50# define dir_arrival_res_buf_count_max 10# define DIR_DETECTION_MIN_VAL 5 int buf1[BUF_SIZE];
int buf2[BUF_SIZE];
int buf3[BUF_SIZE];
int buf4[BUF_SIZE];
int dir_arrival_buf_idx = 0;
int prev_dir = DIR_Y_7;
int dir_arrival_res_buf_idx = 0;
int dir_arrival_res_buf[4];
int dir_arrival_res_buf_count = 0;
void dir_arrival_detect_func() {
        buf1[dir_arrival_buf_idx] = analogRead(A1);
        buf2[dir_arrival_buf_idx] = analogRead(A2);
        buf3[dir_arrival_buf_idx] = analogRead(A3);
        buf4[dir_arrival_buf_idx] = analogRead(A4);
        dir_arrival_buf_idx++;
        if (dir_arrival_buf_idx == BUF_SIZE) {
            int sum1 = 0;
            int sum2 = 0;
            int sum3 = 0;
            int sum4 = 0;
            for (int i = 0; i < BUF_SIZE; i++) {
                sum1 = sum1 + buf1[i];
                sum2 = sum2 + buf2[i];
                sum3 = sum3 + buf3[i];
                sum4 = sum4 + buf4[i];
            }
            int avg1 = sum1 / BUF_SIZE;
            int avg2 = sum2 / BUF_SIZE;
            int avg3 = sum3 / BUF_SIZE;
            int avg4 = sum4 / BUF_SIZE;
            int amp1 = 0;
            int amp2 = 0;
            int amp3 = 0;
            int amp4 = 0;
            for (int i = 0; i < BUF_SIZE; i++) {
                amp1 = amp1 + abs(buf1[i] - avg1);
                amp2 = amp2 + abs(buf2[i] - avg2);
                amp3 = amp3 + abs(buf3[i] - avg3);
                amp4 = amp4 + abs(buf4[i] - avg4);
            }
            int cur_dir = -1;
            amp3 = amp3 - 150;
            if (amp1 > amp2 && amp1 > amp3 && amp1 > amp4) {
                cur_dir = DIR_X_7; //      setLED(7, 1,0,  0,1,0);
            } else if (amp2 > amp3 && amp2 > amp4) {
                cur_dir = DIR_Y_0; //      setLED(6, 0,0,  0,1,0);
            } else if (amp3 > amp4) {
                cur_dir = DIR_X_0; //      setLED(5, 1,0,  0,1,0);
            } else {
                cur_dir = DIR_Y_7; //      setLED(6, 2,0,  0,1,0);
            }
            dir_arrival_res_buf[cur_dir]++;
```

```
            dir_arrival_res_buf_count++;
            if (dir_arrival_res_buf_count == dir_arrival_res_buf_count_max) {
                int max_idx = -1;
                int max_val = -1;
                for (int i = 0; i < 4; i++) {
                    if (dir_arrival_res_buf[i] > max_val) {
                        max_val = dir_arrival_res_buf[i];
                        max_idx = i;
                    }
                }
                if (max_val >= DIR_DETECTION_MIN_VAL && prev_dir != max_idx) {
                    rotate_func(prev_dir, max_idx);
                    prev_dir = max_idx;
                }
                for (int i = 0; i < 4; i++) {
                    dir_arrival_res_buf[i] = 0;
                }
                dir_arrival_res_buf_count = 0;
            }
            dir_arrival_buf_idx = 0;
        }
    } //      #define DIR_X_0 0 //#define DIR_Y_0 1 //#define DIR_X_7 2 //#define DIR_Y_7 3 //------------
    ---------------------------------------------- //MAIN LOOP
bool is_snake_game;
int buttonPin = 8;
int lastButtonState = LOW;
unsigned long lastDebounceTime = 0;
int debounceDelay = 150;
int buttonState;
void setup() {
    pinMode(buttonPin, INPUT);
    is_snake_game = true;
    led_driver_setup();
    my_timer_snake.enableInterrupt(1);
    for (int x = 0; x < 8; x++) {
        for (int y = 0; y < 8; y++) {
            setLED(x, y, 0, 1, 1, 1);
            setLED(x, y, 1, 1, 1, 1);
            setLED(x, y, 2, 1, 1, 1);
            setLED(x, y, 3, 1, 1, 1);
            setLED(x, y, 4, 1, 1, 1);
            setLED(x, y, 5, 1, 1, 1);
            setLED(x, y, 6, 1, 1, 1);
            setLED(x, y, 7, 1, 1, 1);
        }
    }
    while (1);
    init_snake_game();
}
unsigned long last_time_fft = 0;
int max_avg_amplitude = 0;
int temp_max = 0;
int loop_count = 0;
int fft_count = 0;
int last_time_print = 0;
void loop() {
    check_button();
    if (is_snake_game == true) {
        snake_game();
        return;
    } else {
        dir_arrival_detect_func();
        if (stuck_flag == true) {
            for (int x = 0; x < 8; x++) {
                for (int y = 0; y < 8; y++) {
                    for (int h = 0; h < 8; h++) {
                        setLED(x, y, h, 1, 0, 0);
                    }
                }
```

```
            }
            setLED(0, 0, 0, 1, 1, 1);
            while (1);
        }
        update_freq_to_cube(prev_dir, false);
        if (ready_idx == -1) {
            return;
        }
        is_doing_fft = true;
        calculate_fft(ready_idx);
        ready_idx = -1;
        is_doing_fft = false;
        last_time_fft = millis();
        return;
    }
}
void clear_all() {
    for (int x = 0; x < 8; x++) {
        for (int y = 0; y < 8; y++) {
            for (int h = 0; h < 8; h++) {
                setLED(x, y, h, 0, 0, 0);
            }
        }
    }
}
void rotate_func_helper1() {
    for (int x = 0; x < 8; x++) {
        set_LED_column(x, 0, 0);
        delay(20);
    }
}
void rotate_func_helper2() {
    for (int y = 0; y < 8; y++) {
        set_LED_column(7, y, 0);
        delay(20);
    }
}
void rotate_func_helper3() {
    for (int x = 7; x >= 0; x--) {
        set_LED_column(x, 7, 0);
        delay(20);
    }
}
void rotate_func_helper4() {
    for (int y = 7; y >= 0; y--) {
        set_LED_column(0, y, 0);
        delay(20);
    }
}
void rotate_func(int dir_start, int new_dir) {
    int local_amp_range = 0; //RED
    if (dir_start == DIR_Y_0) {
        rotate_func_helper1();
        rotate_func_helper2();
        rotate_func_helper3();
        rotate_func_helper4();
    } else if (dir_start == DIR_Y_7) {
        rotate_func_helper3();
        rotate_func_helper4();
        rotate_func_helper1();
        rotate_func_helper2();
    } else if (dir_start == DIR_X_0) {
                rotate_func_helper4();
                rotate_func_helper1();
                rotate_func_helper2();
                rotate_func_helper3();
            } else { //dir_start == DIR_X_7
                rotate_func_helper2();
                rotate_func_helper3();
```

```
1007.                 rotate_func_helper4();
1008.                 rotate_func_helper1();
1009.             }
1010.             clear_all();
1011.             update_freq_to_cube(new_dir, true);
1012.         }
1013.         void set_LED_column(int x, int y, int amp_range_) {
1014.             for (int h = 0; h <= 8; h++) {
1015.                 setLED(x, y, h, 0, 0, 0);
1016.                 set_led_helper(x, y, h, amp_range_);
1017.             }
1018.         }
1019.         void check_button() {
1020.             int reading = digitalRead(buttonPin);
1021.             if (reading != lastButtonState) { // reset the debouncing timer
1022.                 lastDebounceTime = millis();
1023.             }
1024.             if ((millis() - lastDebounceTime) > debounceDelay) {
1025.                 if (reading != buttonState) {
1026.                     buttonState = reading;
1027.                     if (reading == HIGH) {
1028.                         if (is_snake_game == true) {
1029.                             is_snake_game = false;
1030.                             clear_all();
1031.                             my_timer_snake.enableInterrupt(0);
1032.                             setup_mic();
1033.                             my_timer_music.enableInterrupt(1);
1034.                             clear_all();
1035.                         } else {
1036.                             clear_all();
1037.                             delay(1500);
1038.                             NVIC_SystemReset(); // processor software reset
1039.                         }
1040.                     }
1041.                 }
1042.             } // save the reading. Next time through the loop, it'll be the lastButtonState:
1043.             lastButtonState = reading;
1044.         }
```