LED Cube

By

Michael Lin
Raymond Yeh

Final Report for ECE445, Senior Design, Fall 2013
TA: Joseph Shim

University of Illinois, Urbana-Champaign
07 December 2013
Project No. 4

Abstract

We have designed and built and 4 by 4 by 4 RGB LED cube. The LED cube is capable of displaying 3D interactive and routine animations. Interactive animation uses Bluetooth, IR range finder, and light frequency sensor, which allow users to interact with the displayed lighting pattern; this includes a 3D snake game and hand tracking animation.

Table of Contents

1. Introduction

1.1 Statement of Purpose

LED technology is more advanced and much more efficient than traditional incandescent light bulbs and as such our team decided we wanted to build a device related to LEDs. An LED cube is inherently aesthetically pleasing and ours will be capable of displaying 3D animations and lighting patterns with much increased complexity compared to any 2D display of comparable resolution. Environmental interaction and Android Bluetooth connection will also be able to control the various lighting effects on the cube. Although our plan is for a visually pleasing cube, our implementation can easily be adapted for more practical applications such as displaying 3D models.

1.2 Objective

Goals and Function:

Our project goal is to use 64 RGB LEDs to construct a 4x4x4 cube with a single serial connection. The cube will have three modes of operation-off, programmed animation, and environmental response. There will be several animations that can be cycled through during the operation of the cube. An IR rangefinder and ambient light sensor will be used so the LED cube will react to environmental changes by changing color and brightness of each LED in the cube. The mode of operation of the cube as well as cycling animations will be controlled wirelessly through Bluetooth using an Android application.

Features:

- Display programmed animations capable of displaying any RGB specifiable color
- Bluetooth wireless control
- Environmental control of light patterns

Benefits:

- Aesthetically pleasing
- Easily controllable
- Inexpensive

1.3 Block Diagram

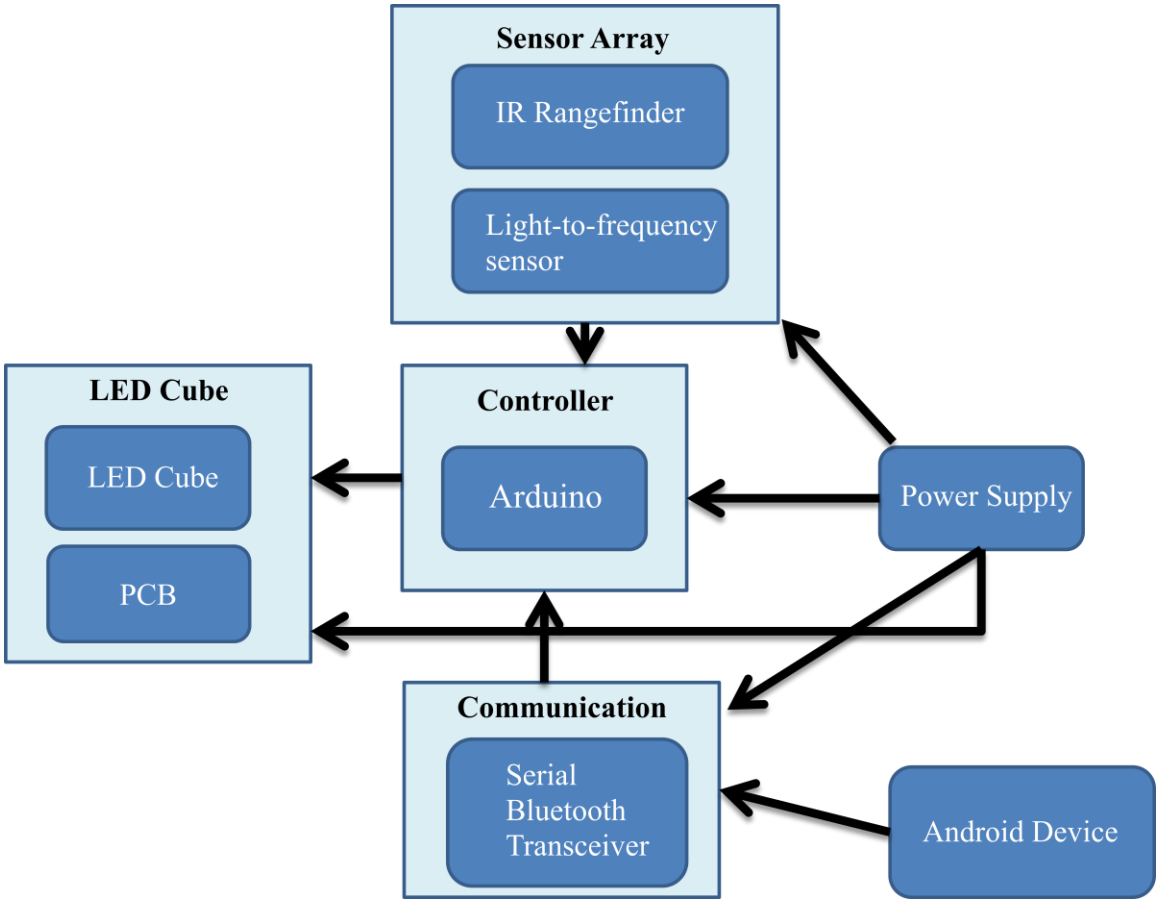


Figure 1: Overall LED Cube Block Diagram

2. Design & Block Descriptions

2.1 Introduction

Our project was designed around making sure all the requirements and verifications that were proposed in the design review, attached in Appendix C, were satisfied. The summaries of each major block are described in the following section.

2.2 Design Details

2.2.1 Power Supply

Inputs	Outputs
• 120 VAC 60Hz from wall outlet	• 5 VDC to modules

The power module supplies power to the Sensor Array, Controller, LED Cube, and Communication module.

2.2.2 LED Cube

Inputs	Outputs
• 4 digital inputs from Controller (1 serial data input, 1 clock input, 1 latch input, 1 blank input)	• LED outputs

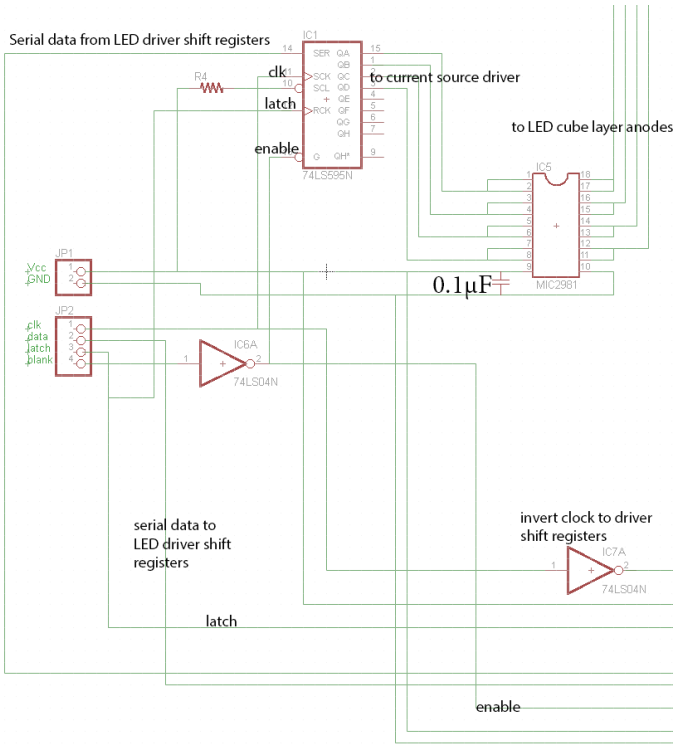


Figure 2: Partial Circuit Schematic for LED Cube

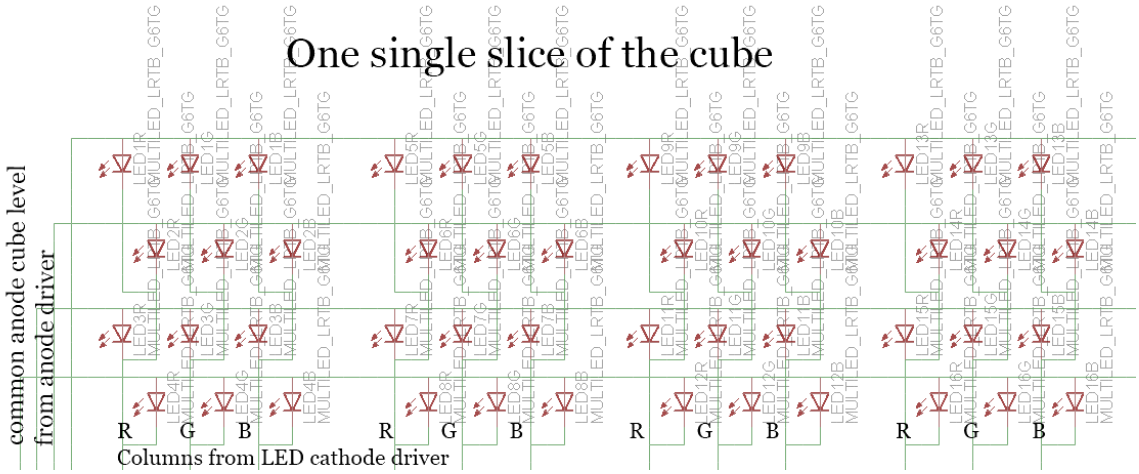


Figure 3: Circuit Schematic for one single slice of LED Cube

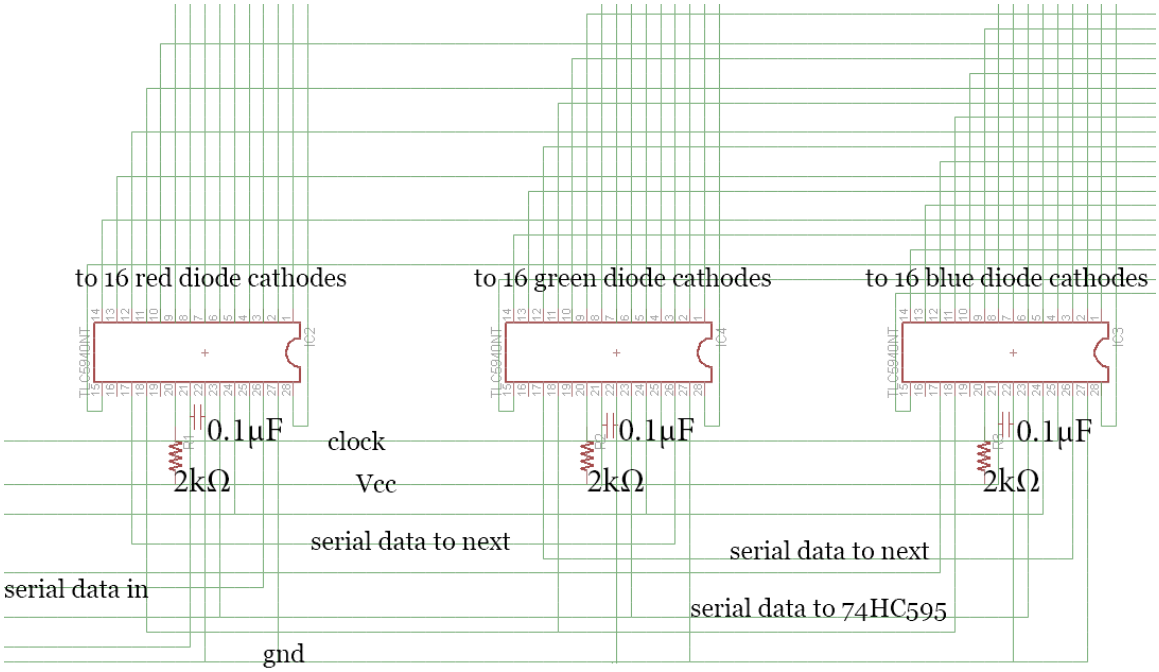


Figure 4: Partial Circuit Schematic for LED Cube

The LED cube takes input from the controller module. It consists of 64 RGB LEDs in a 4x4x4 cube and is controlled by the controller module. This is used to display the light patterns.

2.2.3 Controller

Inputs	Outputs
• 5 VDC from Power Supply • 1 analog input from Sensor Array • 1 digital input from Sensor Array • 1 digital input from Communication Module	• 4 digital outputs to LED cube • 1 digital output to Communication Module

This module will take input from the communication and sensor array module and then output controls to the LED Cube. The controller module will compute and output several different lighting patterns to the LED cube; this includes lighting patterns that depend on sensor module outputs from environmental inputs. The controller will also take input from the communication module to switch between modes of lighting patterns.

We chose to use an Arduino UNO MCU based on its availability, ease of use, and features. The Arduino UNO is easy to source from many online retailers as well as from the ECE Shop. Because it includes an Atmel ATmega16U2 MCU programmed as a USB-to-serial converter, the user can bypass using an external FTDI chip to program the Arduino UNO by directly connecting it to a computer via a USB cable. Features important to choosing the Arduino UNO include the UART TTL serial communication, plethora of GPIOs including both digital and analog inputs, 16 MHz clock speed, and interrupt capabilities. [6] Although not the least expensive MCU on the market, the ease of use made the Arduino UNO a good choice for our project.

LED Indexing

The microcontroller program will output a serial 52 bit control command to the shift registers for the LED cube to display the different lighting patterns. The 52 bits control is indexed as follows with serial communication being MSB first.

$$[0 \ldots .15]\ for\ controlling\ the\ Red\ LEDs$$
$$[16 \ldots .31]\ for\ controlling\ the\ Green\ LEDs$$
$$[32 \ldots .47]\ for\ controlling\ the\ Blue\ LEDs$$
$$[48 \ldots .51]\ for\ controlling\ the\ level\ of\ LEDs$$

The underlying theory to this method of driving the cube is that at each point in time, there is only one "level" of the LEDs that is going to be on; meaning only 16 LEDs on at a time. The cube has been constructed to follow this demultiplexed method of control. Then each layer will have a certain time period to turn on or off. Because the entire cube will refresh with a rate of more than 100Hz, the average person won't see the flickering of the LEDs. With this setup, we will be able to index to all of the LED diodes in the cube individually.

Since the above setup only allows each LED to be turned on and off we use the technique of bit angle modulation in order to display color. The idea is similar to PWM where the duty cycle of a square wave on separate diodes is used to vary the combined color. Bit Angle modulation (BAM) has multiple pulses in the equivalent of a single period of a PWM square wave. [4] For the LED cube, a 4-bit brightness resolution will be used.

Assuming that the refresh rate for the cube is set at 200 Hz (the best refresh rate will be experimentally determined) each cycle takes 0.005 sec.

Bit 3 will then pulse for half of 0.01 sec = 0.0025 sec. Bit 2 will pulse for half of what bit 3 had--0.00125 sec. Continue to divide, then Bit 0 will have the bit taking 0.0003125 second. By using these four bits and adding their pulse times together we achieve an

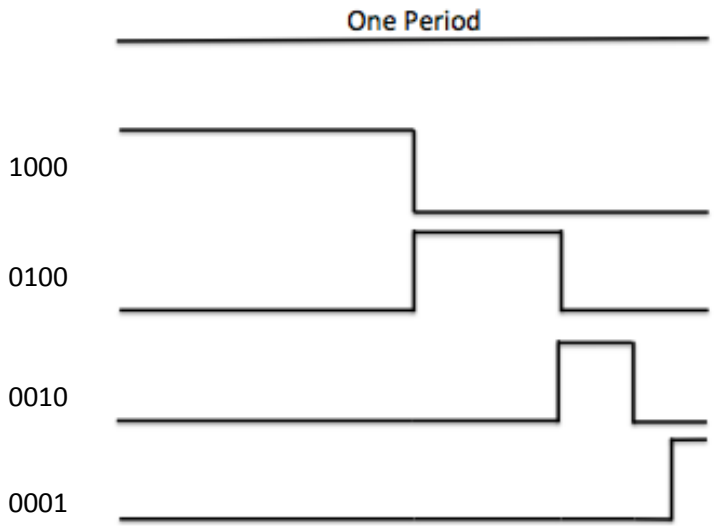


Figure 4-1: Timing Diagram for Bit Angle Modulation

equivalent of PWM. An illustration is shown in figure 4-1.

As can be seen, the on time can be controlled with summation of these 4 cases, and 0000, which is all off. For example, the bits corresponding to a 75% duty cycle will be 1100.

Finally, to ensure that human eyes cannot perceive the flickering of the LEDs in this implementation the refresh rate have to be higher than 60 Hz. According to "Temporal Sensitivity" by Andrew B. Watson, for looking at light people begin to notice interruption when the darkness is about 16 milliseconds or longer [12]; this is equivalent to 62.5Hz. Therefore, if we set the interrupt to be 200Hz, over three times of 62.5Hz, the flickering shouldn't be detectable to human eyes. So by setting the interrupt to be 200Hz, each layer must be updating at 800Hz since there are 4 layers to update inside the 200Hz cube update. The Arduino UNO has a 16 MHz system clock. This means if the interrupt is at every 16 MHz / (800Hz) = 20,000 clock cycles. Then it takes 52 clock cycles at 8 MHz or 104 cycles at 16 MHz to update the shift registers. As can be seen, using a cube refresh rate of 200Hz, we find that the Arduino has 20,000-104 = 19896 clock cycles to complete other tasks.

LED Lighting Patterns

Once each of the LED can be indexed then the lighting pattern will be control by for loops that will iterate through all the LEDs and set the desired color to the LED during the process.

Bluetooth Connection

For the Bluetooth connection, the Serial library for the Arduino to received serial data from the Android device. Then serial library has two key functions, serial.available(), and serial.parseInt(). [23]

serial.available() will return the number of bytes available for reading from the serial port. serial.parseInt() will return the next integer value from the serial port.

Basically, the idea is periodically check the serial.available indicator so that when the number is greater than 0—i.e. there is data to be received—serial data is parsed. The integer value parse will correspond to which mode the LED cube should operate in. If necessary start and end characters will be used to avoid parsing at incorrect location. The necessity for this will have to be experimentally determined.

Controller Design (Software)

The algorithm/idea used to index and set up the code for the microcontroller is described, here I will just explained how the algorithm is implemented.

In order to keep track of all the 64 LEDs and its color, with 4bit resolution BAM, I have used a total of 12 arrays of 64 bits. For each color, 4 arrays of 64 bits are needed because of the 4-bit resolution BAM. Each array is indexed with a linear index $k = x + 4y + 16z$. Thus given the x, y, z position of an LED, then it can map to an index for access the color for that particular LED in the array. So, basically to update the color of the LED cube, at each interrupt, the controller will read the current colors in those 12 arrays, and shift out the corresponding control signals. The following figure shows the overall software procedure, the details of each animation aren't included; please refer to appendix A for the Arduino code written.

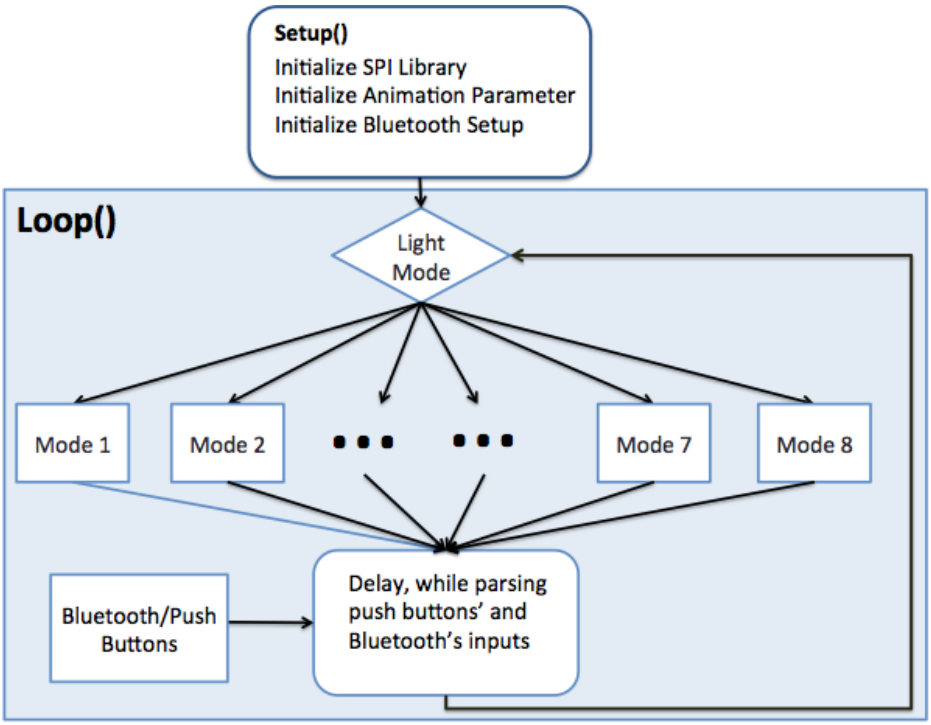


Figure 4-2: Software Control Diagram

2.2.4 Communication

Inputs	Outputs
• 5 VDC from Power Supply • Wireless serial input from Android device • 1 digital input from controller	• 1 digital output to Controller

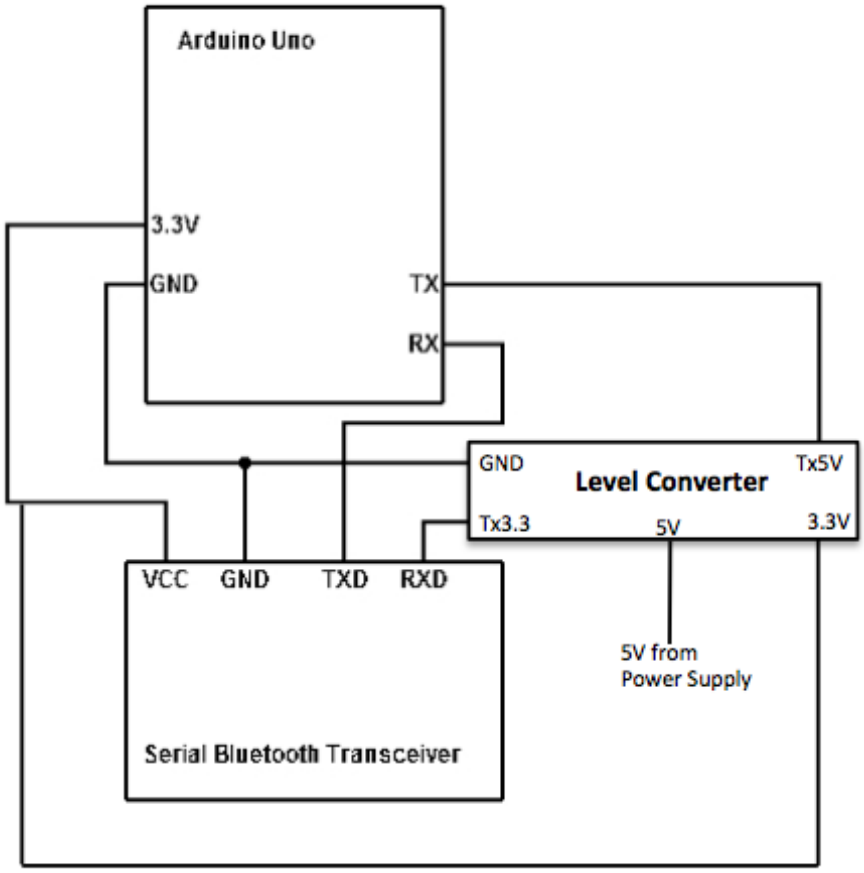


Figure 5: Circuit Schematic for Communication Module

The communication module takes serial data wirelessly from the paired Android device and outputs to the controller; the output to the controller is used to switch modes and lighting patterns.

For Bluetooth, we chose to use the HC-06 Bluetooth module because it satisfies our needs for a Bluetooth transceiver slave in addition to being very inexpensive. Because of our choice to use the HC-06, we had to add a level converter since the Arduino UNO operates at 5V logic levels and the HC-06 operates at 3.3V logic levels. [7] For logic going from the HC-06 to the Arduino UNO, no level converting was necessary since 3.3V is above the 2V voltage threshold for a logical high on the Arduino UNO. To prevent damaging the HC-06 chip, a 5V to 3.3V level converter was implemented on the receiving line of the transceiver by using a voltage divider.

2.2.5 Sensor Array

Inputs	Outputs
• 5 VDC from Power Supply • Environment conditions	• 1 analog output to Controller • 1 digital output to Controller

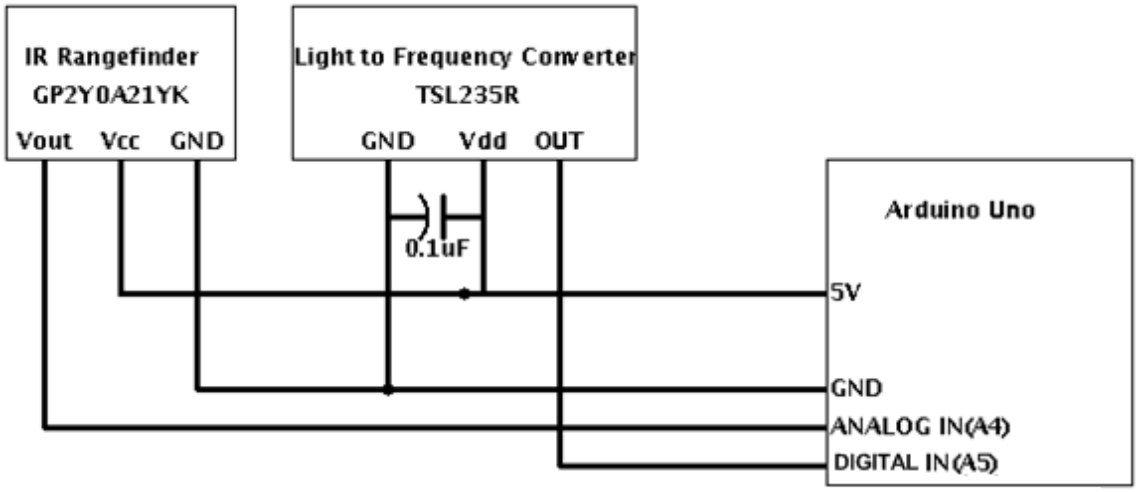


Figure 6: Circuit Schematic for Sensor Array

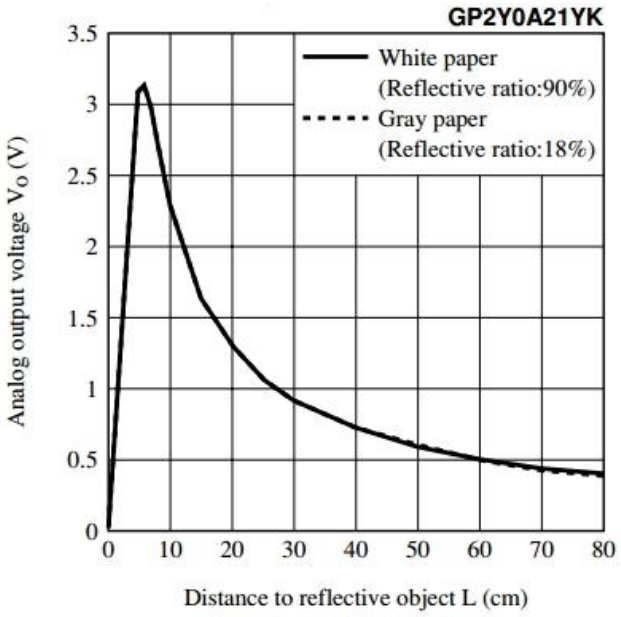


Figure 7: Analog Output Voltage vs. Distance for IR rangefinder

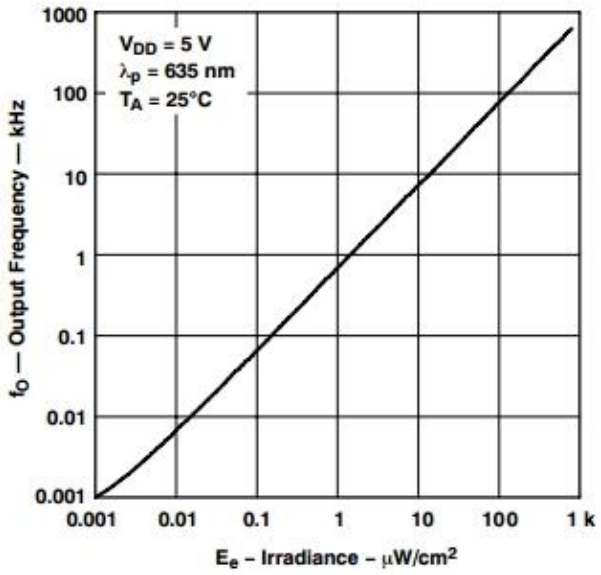


Figure 8: Output Frequency vs. Irradiance for light sensor

The sensor array module outputs to the controller module, this module consists of an IR rangefinder and an ambient light sensor, which both collect data about the environment and will be used to determine lighting patterns. Figure 7 shows the voltage vs. distance relation for the IR rangefinder. [10] Figure 8 shows the frequency vs. Irradiance. [11]; these two shows the expected behavior of the two sensors.

The IR sensor was chosen for the range of distances it could accurately determine since we wanted the range to be between 10cm and 1m. The ambient light sensor was chosen to be a light-to-frequency converter over a photocell for ease of use as well as device to device consistency. Using a photocell would require using a voltage divider type circuit and reading the analog voltage output where using the light-to-frequency converter outputted a square wave with 50% duty cycle with varying frequencies depending on the light incident on the sensor. Replacing a broken light-to-frequency converter is easier than a photocell since the output of a replaced light-to-frequency converter is a better match than the output of a replaced photocell thereby reducing or removing the need to change the code to account for the difference in output.

2.2.6 Android Device

Inputs	Outputs
User input on screen (buttons)	Wireless serial output to Communication module

The Android device module is either an Android tablet or an Android phone. This module will output to the communication module via Bluetooth, in order for the controller to know when switch modes of operations. The Android platform is chosen over iOS because the development process was slightly easier due to our familiarity with Java.

Android Application Design

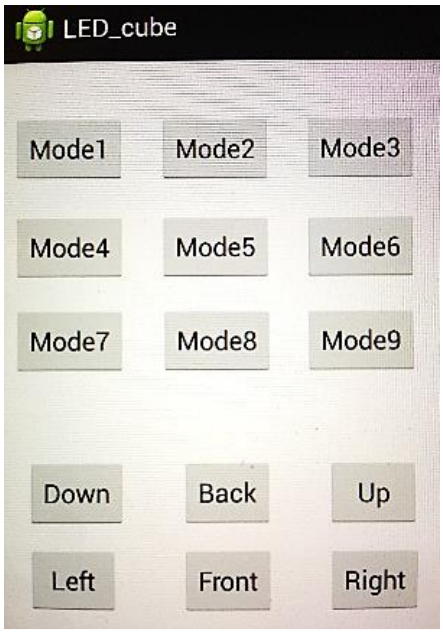


Figure 9: Android Application GUI

On the left, figure 9, is a screenshot of the android application that was developed along with the LED Cube. Each buttons Mode 1 to 9 corresponds to an interactive or routine animation. By clicking the Mode buttons, will switch between the animations. The direction buttons are used for the "3D Snake" game we designed to be played on the LED cube.

The Bluetooth for the Android is coded using Android developer's API. [13, 14]. When each button is pressed, the Bluetooth will send out a single ASCII character through serial Bluetooth to the controller. The controller will react correspondingly.

3. Design Verification

3.1 Testing Procedures & Quantitative Results

Refer to Appendix C for the full requirement and verification table.

3.2.1 Controller Module Testing Procedures & Testing Results

Requirement: a) Compute and shift out data serially to the shift register with a clock speed of at least 8 MHz

Testing Procedure & Results: An oscilloscope was used to measure the shift register clock; the screen capture of the oscilloscope is show in figure 10 below. As can be seen, 8 MHz is achieved.

Figure 10: Oscilloscope screen capture of shift register clock

Requirement: a) Compute and shift out data serially to the shift register.

Testing Procedure & Result: In order to test the Controller and computer and shift out the serial control signals. A prototype of a small cube was built; refer to Appendix B. Using the small prototype, the code of the controller can be verified if it is acting properly. A simple animation of iterating through the rows was done successfully; figure 12 shows a snapshot of the animations.

Figure 11: Snapshot of LED prototype animation

Furthermore, to check the code is actually working properly, the Latch output was measured with an oscilloscope, shown in figure 13. This is behavior of Latch is correct, as one can observe the BAM timing; where the time between each latch roughly doubles.

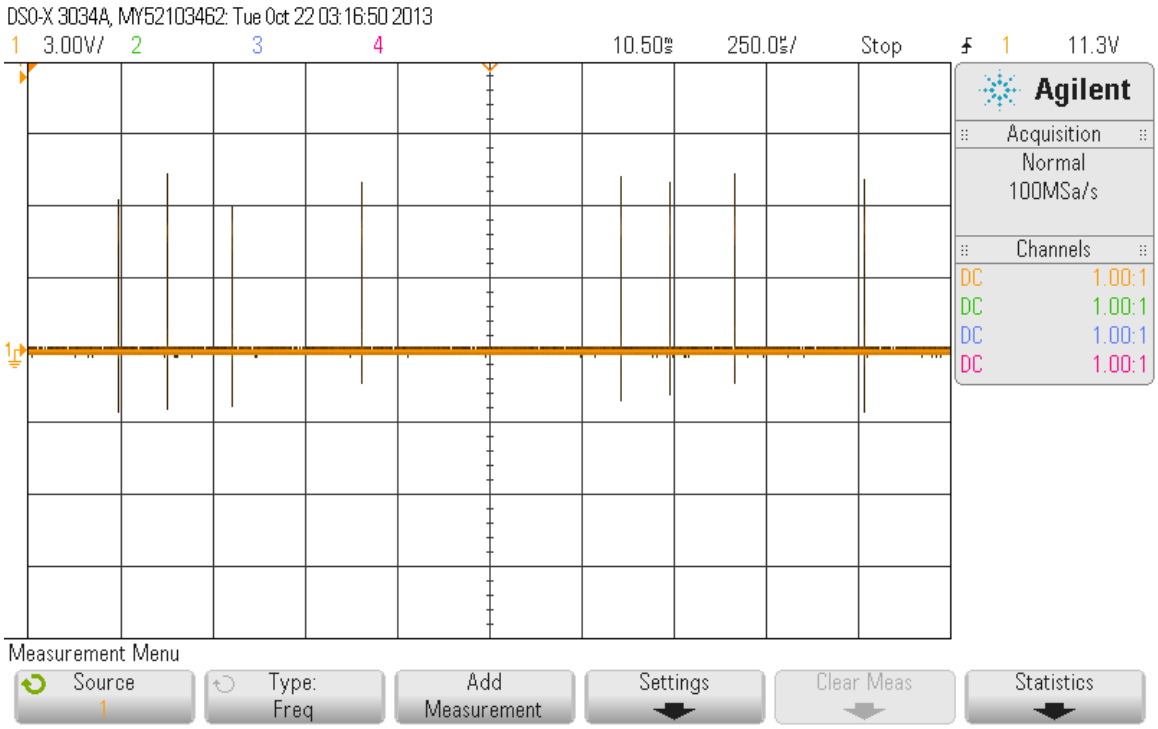


Figure 12: Behavior of the Latch Output from controller to shift

Requirement: b), c) Able to perform environmental sensing animations and receive sensor signals.

Testing Procedure & Results: A test circuit was built with LEDs to display the distance readings from the IR sensor. In figure 11 shows the IR sensor successfully detects the distance and reflects it in the LED outputs. Refer to Appendix B for circuit diagram of the test circuit, and refer to Appendix C for the video link of the result.

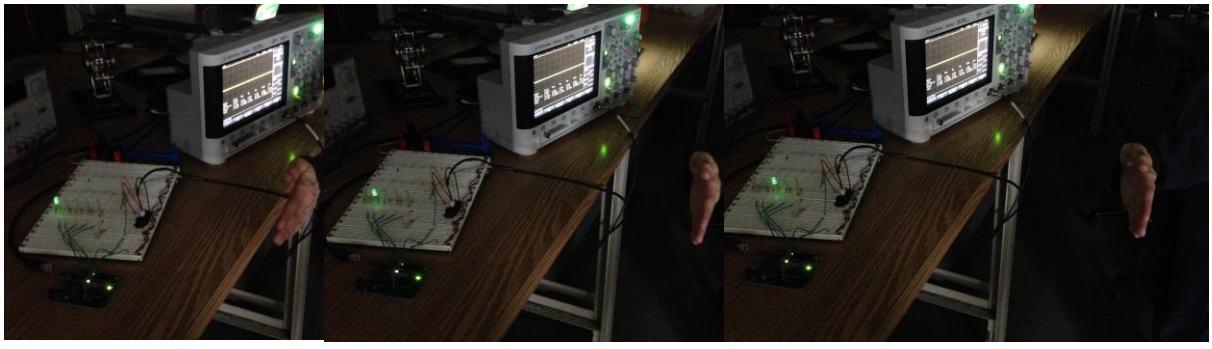


Figure 13: LED placement vs. Distance from IR Sensor

Testing Procedure & Results: Additionally, the light sensor receiving functionality was also tested on the test circuit, the LED will change color, when the light sensor observes a light brighter than a certain threshold. Using iPhone's flashlight to change the brightness, the LED behave as expected, meaning the sensor data can be read properly.

3.2.2 Communication Module Testing Procedures & Testing Results

Requirement: Bluetooth serial communication is received on the Arduino at the baud rate of 9600.

Testing Procedure & Result: Using the app. "Bluetooth SPP" check if the Bluetooth can be detected and connect. The Android was successfully connected with the Bluetooth serial transceiver, shown in figure 14 is a screenshot of the successful connection.

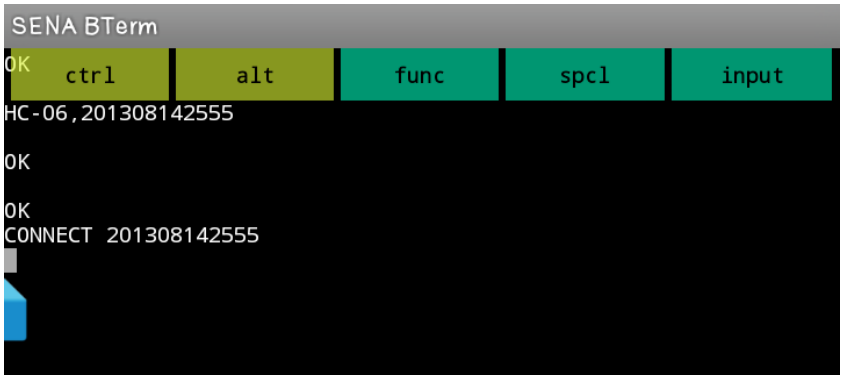


Figure 14: Screenshot of Successful Bluetooth Connection

3.2.3 Sensor Array Module Testing Procedures & Testing Results

Requirement: Light frequency sensor able to output sensor data that reflects the light intensity.

Testing Procedure & Results: Using the flashlight on an iPhone, and room light, two different frequency was observed with and oscilloscope.

Light Condition	Frequency Output
Room Light	19kHz
Room Light + flashlight	432kHz

As can be seen, the frequency output from the sensor reflects relative the light intensity.

Requirement: The IR rangefinder outputs distinct voltages for distances between 10cm and 70cm in increments of 2.5cm.

In order to put the IR range finder's output to use as an actual equation. A notebook was placed at distance 10cm to 70cm with 5 cm increment in front of the sensor, and the corresponding voltage is recorded, shown as follows in Table 1 below

Distance (cm)	Voltage (V)
10 cm	2.08 V
15 cm	1.40 V

20 cm	1.02 V
25 cm	0.90 V
30 cm	0.86 V
35 cm	0.73 V
40 cm	0.69 V
45 cm	0.63 V
50 cm	0.57 V
55 cm	0.53 V
60 cm	0.49 V
65 cm	0.48 V
70 cm	0.43 V

Table 1: Analog Output Voltage vs. Distance for IR rangefinder

Testing Procedure & Results: Using an oscilloscope, measure the relative distance vs. voltage, and Matlab is used to determine the voltage vs. distance equation below

$$Distance = 198e^{-2.7656*voltage} + 9.7252 \qquad (Eq.3.2.3.1)$$

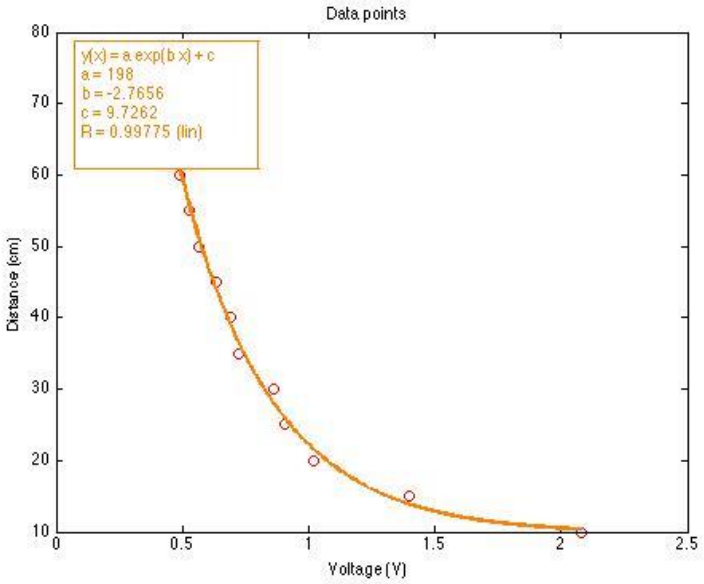


Figure 15: Best-fit line for voltage vs. distance

3.2.3 Power Module Testing Procedures & Testing Results

Requirement: Supply 5±0.5V output voltage and an output voltage ripple not exceeding 0.5V.

Testing Procedure & Results: Using an oscilloscope, measure the output voltage of the power module. As can be seen, a mean voltage of 5.2V with a ripple of at most 0.0886V is achieved by the power supply which is within the requirements.

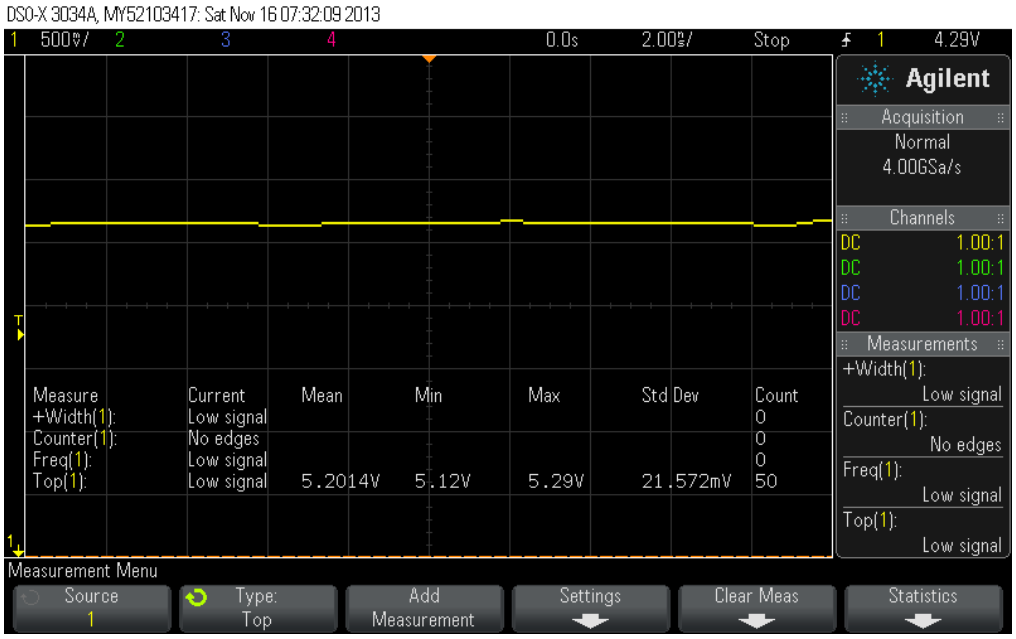


Figure 16: Power Supply Voltage Output

3.2.4 LED Cube Testing Procedures & Testing Results

Requirement: No shorts or non-conducting connections in the cube.

Testing Procedure and Results: Using the continuity check on a multi-meter, check every connection in the cube for continuity. Almost all connections were continuous when checking. There were less than three connections with cold solder joints which did not have good conductivity. These were all fixed and tested again to ensure a good electrical connection.

3.2 Discussion of Results

Our LED cube has passed all the requirement and verification proposed in the designed review during our Final demo. And all the functionality that we designed for was functioning properly and reliably.

Tolerance Analysis

The component that most affects the projects performance is the power component. This is because the power supply has to function properly in order for all the other blocks to work.

For correct operation of the shift registers, the supply voltage cannot go below 3VDC or above 6VDC otherwise the chips will either not output or they will go above their rated heat dissipation limits.

The serial Bluetooth module must receive between 3.6 and 6VDC to ensure operation. If this is not met the Bluetooth module may frequently disconnect or not output serial data. For operation of the project, the power supply has to provide more than 5W. This can be verified by using an ammeter and/or voltmeter on the terminals from the power supply to the LED cube circuit. If the 5W requirement is not fulfilled then the LEDs may not light up or they may be dim and less aesthetically pleasing because of this.
As part of testing for the tolerance analysis, the LED cube was left running for more than 24 hours. After long term use, we checked the chips, traces, and power supply for any excess heat. We found that the LED cube can run reliably for long durations of time without any ill effects or abnormal heating meaning that the power component is stable in supplying 5W at 5V without any problems.

4. Cost

4.1 Cost Analysis

4.1.1 Parts

Item	Part Number	Qty	Unit Cost	Total Cost
Microcontroller board - Arduino UNO	-	1	$27.83	$27.83
16 channel constant current sink SIPO LED driver	MBI5026GN	3	$2.25	$6.75
8 bit SIPO shift register	74HC595	1	$0.63	$0.63
8 channel High-current source driver array	MIC2981	1	$3.03	$3.03
Common Anode 5mm RGB LED	MA475	100	$0.0999	$9.99
Serial Bluetooth Transceiver	RS232	1	$8.86	$8.86
IR Rangefinder	GP2Y0A21YK	1	$13.95	$13.95
JST jumper wire	SEN-08733	1	$1.50	$1.50
Light-to-frequency sensor	TSL235R	1	$2.95	$2.95
0.1 uF capacitor for light sensor	-	1	$0.20	$0.20
Resistor for Red diode current control	-	1	$0.10	$0.10
Resistor for Green diode current control	-	1	$0.10	$0.10
Resistor for Blue diode current control	-	1	$0.10	$0.10
5 VDC 3A USA Switching Power Supply (5.5mm OD, 2.1mm ID)	WSU050-3000	1	$11.34	$11.34
Female Power Barrel Connector	PJ-063AH	1	$1.78	$1.78
SPST Rocker Switch (12 VDC, 20A)	GRB066A802BB1	1	$1.77	$1.77
3.3,5VDC Logic Level Converter	BOB-11978	1	$1.95	$1.95
0.1µF Ceramic Decoupling Capacitors	-	4	$0.20	$0.80
1N5822 Schottky Rectifier	-	2	$0.23	$0.46
			Total:	$94.09

4.1.2 Labor

Name	$/Hour	Hours per week	Number of Weeks	(Total/Person)*2.5
Raymond Yeh	$35.00	15	12	$ 15,750
Michael Lin	$35.00	15	12	$ 15,750
			Total:	$31,500

4.1.2 Grand Total

Total Labor	Total Parts	Grad Total
$31,500	$94.09	$31,594.09

4.2 Equipment Needed

The equipment that we have used throughout the project includes oscilloscope, multi-meter, soldering iron, drill press, sheet metal nibbler, hole punch, Matlab License, and EAGLE License.

5. Conclusions

5.1 Accomplishments

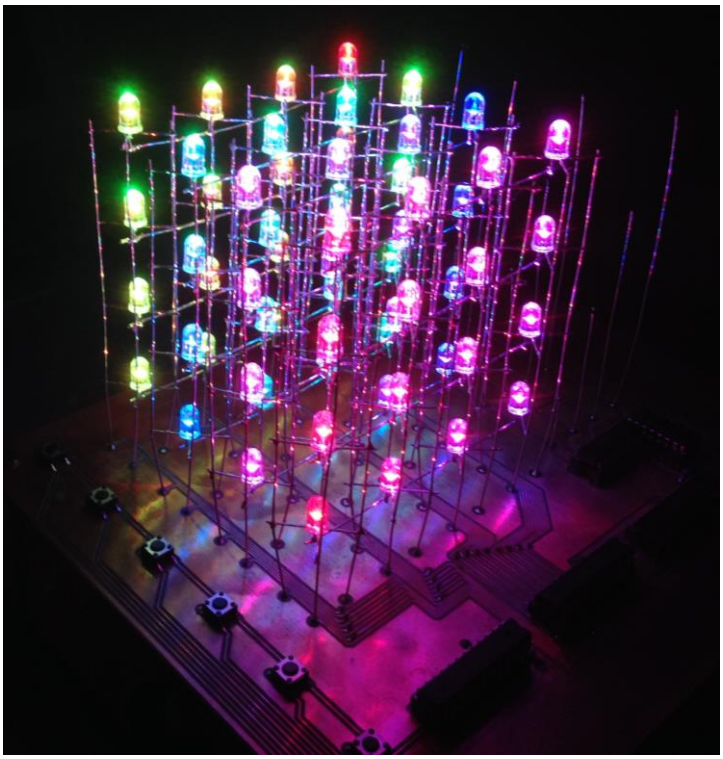


Figure 16: Photo of LED Cube with outine animation

For this project, we have successfully finished and fulfilled all the requirements and verification proposed in the design review. The template the machine shop made for us to use to make the LED columns worked incredibly well. The columns were straight and even with one another. The PCBs made the operation of our project much more reliable and we were able to integrate all the parts of the project together. The additional micro USB jack that was added in addition to the two rectifying 1N5822 diodes allowed us to successfully power the cube with a standard micro USB cell phone type charger as well as the barrel jack power supply. The construction of the cube allows good viewing of all the LEDs in the cube making for great visual presentation of the cube.

On the software side, we have successfully designed three interactive animations, a 3D snake game, hand-tracking animation (IR sensor), and coloring cube animation (Light frequency sensor). Additionally, we have designed a few more routine animations, including ball bounce, and plane sweeps that shows the advantages of a 3D display. Our final product has a reliable Bluetooth connection and overall the LED lighting display is aesthetically pleasing, as can be seen from fig. 16. We are proud of the product that we have made.

5.2 Uncertainties

Overall the product is very reliable and the functionalities are consistent throughout our demo and verification tests.

5.3 Safety & Ethical Considerations

5.3.1 Safety

Building LED Cube

The main safety concern when building the LED cube is soldering the LEDs to the LED cube. This is because soldering is a potentially very hazardous task if not done correctly. The following few safety rules will be follow during soldering.

1. Never touch the tip of the soldering iron.
2. Wear eye protection when soldering
3. Exercise caution when using lead based solder. Avoid ingestion or contact with open wounds as well as contact with children and pregnant women.
4. When using organic acid type flux make sure to clean any flux residue.
5. Always wash hands after soldering.
6. Always solder in a well-ventilated area so toxic fumes are not inhaled.

Operating the LED Cube**:**

When operating the LED cube, be aware that the flashing lights and patterns that are displayed on the LED cube may cause a very small percentage of people to experience a seizure. Please turn off the LED cube immediately if strange or unusual body movement developed.

5.3.2 Ethical

For this project, we will follow the IEEE code of Ethics along with our own moral standards. Our project process and our final product will adhere to the following relevant IEEE Code of Ethics. [1]

a) Our project, the LED cube, is potentially enjoyed by people of all ages and used at many different locations. Therefore, our product must consider the safety, health and welfare of the public in order to protect the users of our product.

1. To accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;

b) The description, data, and design of our project will be honest and realistic. We ensure that all claims about our project are real and truly reflect our project.

3. To be honest and realistic in stating claims or estimates based on available data;

c) **We will accept and seek honest criticism of our technical work in order to improve out project, and giving credit to whoever contributed to our project.**

7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;

d) **We will adhere to the code of ethics; additionally make sure that all group members do follow the code of ethics throughout this project, as well as help each other for professional development.**

10. To assist colleagues and co-workers in their professional development and to support them in following this code of ethics;

5.4 Future Work/Alternatives

Future work we are considering for this project is a larger resolution cube. Currently the dimension for the cube is 4 by 4 by 4. The amount of animation and light patterns are very limited due to the dimension limitation. By making a larger resolution cube, the display will have higher resolution, and thus possible for displaying ASCII characters and more sophisticated animations. We also think that making the project physically smaller is a good next step since currently there is one RGB LED per inch. By using more surface mount parts we can make the project smaller as well as reduce power consumption. Next, we are also considering adding more sensors to the LED cube, to make the animations more interactive. For example, if the LED cube consists of multiple IR range finders, then the animation can respond to the user's motion in more than one direction. Lastly, animation design software should be developed to come with the cube. Then this allows user to design animations without having to write any code.

6. References

[1] Institute of Electrical and Electronics Engineers, Inc. "IEEE Code of Ethics", *ieee.org* [Online] Available: http://www.ieee.org/about/corporate/governance/p7-8.html [Accessed: Sept. 19, 2013]

[2] "Photosensitivity and Seizures", *epilepsyfoundation.org*, 2012[Online]. Available: http://www.epilepsyfoundation.org/aboutepilepsy/seizures/photosensitivity/. [Accessed: Sept.19, 2013]

[3] "Lead Soldering Safety Guidelines", *cmu.edu*, [Online]. Available: http://www.cmu.edu/ehs/chemical/Lead%20Soldering%20Safety%20Guidelines.pdf. [Accessed: Sept 20, 2013]

[4] "Bit Angle Modulation (BAM)", *picbasic.co.uk*, Oct. 28, 2007 [Online]. Available: http://www.picbasic.co.uk/forum/showthread.php?t=7393 [Accessed: Sept 20, 2013].

[5] "How to Cit References: IEEE Documentation Style", *ece.gatech.edu* [Online]. Available:http://www.ece.gatech.edu/academic/courses/ece4007/ECE4007A/deliverables/proposal/2011spring/IEEE_citations/IEEE%20Citation%20Guidelines2.pdf [Accessed: Sept, 28,2013]

[6] "ArduinoBoardUno", *Arduino.cc*, [Online]. Available: http://arduino.cc/en/Main/arduinoBoardUno [Accessed: Sept 20, 2013].

[7] Jaidyn Edwards, "Tutorial #8-HC-05 Bluetooth Module", *duino-robotics.com* [Online]. Available: http://www.duino-robotics.com/arduino-tutorials.html [Accessed: Sept 25, 2013]

[8] Texas Instruments, "16 Channel LED Drive with Dot Correction and Grayscale PWM Control," TLC5940 [Revised Oct. 2007] Available: http://www.ti.com/lit/ds/symlink/tlc5940.pdf [Accessed: Sept 19, 2013]

[9] NXP Semiconductors, "8-bit serial-in, serial or parallel-out shift register with output latches," 74HC595 [Revised Dec, 12 2011] Available: http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf [Accessed: Sept 19, 2013]

[10] Sharp, "General Purpose Type Distance Measuring Sensors," GP2Y0A21YK Available: https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf [Accessed: Sept 23, 2013]

[11] Texas Advanced optoelectronic solution, "Light to frequency converter," TSL235R [Revised Sept. 2007] Available: https://www.sparkfun.com/datasheets/Sensors/Imaging/TSL235R-LF.pdf [Accessed: Sept 23, 2013]

[12] Andrew B.Waston, "Temporal sensitivity", Handbook of Perception and Human Performance [1986] Available:

http://vision.arc.nasa.gov/personnel/pavel/publications/TemporalSensitivity.pdf [Accessed: Oct 5, 2013]

[13] "Bluetooth Device | Android Developer" Dec 2013. [Online] Available: http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html [Accessed: Oct 5, 2013]

[14] "OutputStream | Android Developer" Dec 2013. [Online] Available: http://developer.android.com/reference/java/io/OutputStream.html [Accessed: Oct 5, 2013]

[15] "Creating a simple android app with 2 buttons" June 2011. [Online] Available: http://blog.idleworx.com/2011/06/build-simple-android-app-2-button.html [Accessed: Oct 8, 2013]

[16] "How to make your own Arduino Shield" Sept 2013. [Online] Available: http://www.crispytronics.com/posts/7 [Accessed: Oct 27, 2013]

[17] "Turn your EAGLE schematic into a PCB" [Online] Available: http://www.instructables.com/id/Turn-your-EAGLE-schematic-into-a-PCB/?ALLSTEPS [Accessed: Oct 25, 2013]

[18] "Arduino Shield Scaffold" [Online] Available: http://www.macetech.com/blog/node/69 [Accessed: Oct 12, 2013]

[19] "Arduino Internal Pull-up Resistors for Push Buttons" [Online] Available: http://www.arduino.cc/en/Tutorial/DigitalPins [Accessed: Oct 21, 2013]

[20] "Arduino Debouncing Library" [Online] Available: http://playground.arduino.cc//Code/Bounce [Accessed: Oct 21, 2013]

[21] "Discharge tests of 9 Volt transistors radio style batteries" Oct, 2013. [Online] Available: http://www.powerstream.com/9V-Alkaline-tests.htm [Accessed: Oct 17, 2013]

[22] "How to make a custom library part in Eagle CAD tool" [Online] Available: http://www.instructables.com/id/How-to-make-a-custom-library-part-in-Eagle-CAD-too/ [Accessed: Oct 20, 2013]

[23] "Arduino – Serial", *Arduino.cc,* [Online]. Available: http://arduino.cc/en/reference/serial [Accessed: Oct 5, 2013]

Appendix A: Code for LED Light Control

```
#include <SPI.h> // Use the Serial Peripheral Interface (SPI) library to shift
                 // data into registers for control
#include<Bounce.h> //Debounce for Push buttons
//#include <FreqCounter.h>
//Pin Section
#define blank_pin 4//This is the blank pin
#define enable_pin 2 //This is the latch pin for enable
#define output_pin 11 //The output pin number to connect to shift register
#define output_clk_pin 13 //The clock pin used by shift register
#define test_pin 8//This is the test pin for speed;
//Push buttons
#define push_1 6
#define push_2 7
#define push_3 8
#define push_4 9
#define push_5 10
#define push_6 12

//Variable Section
//Define the following variable to keep track of the LEDs
//Using Bit angle modulation with resolution of 4, thus need 4 array for each color.
//And each array will be 8 byte, because 8*8 = 64 bit total, thus keeps track of each of
the LEDs in the cube.
byte red3[8],red2[8],red1[8],red0[8];
byte green3[8], green2[8], green1[8], green0[8];
byte blue3[8], blue2[8],blue1[8],blue0[8];

int z_index = 1;//This index keeps track of which layer, we are indexing.
int mod_count =0; //This will count through the Bit Angle Mod.
int mod_bit = 0; //This will count which Bit we are modulating
int light_mode = 0;//This keeps track of which light mode, the cube is in.

int IR_Distance = 0;
float IR_Speed = 0;

unsigned long frq = 0;
int test_count = 0;

//Variable Section For Ball Bounce Animation
//Ball Size is 2x2x2
int ball_size = 1;
int x_pos = 0;int y_pos = 0;int z_pos = 0;
int x_velocity = 0; int y_velocity = 0;int z_velocity = 0;
```

```
int ball_count = 0;

//End of Variable Section FOr Ball Bounce

//Variable Section For Snake
int head_x = 1; int head_y = 0;int head_z = 0;
int tail_x = 0;int tail_y = 0;int tail_z = 0;
int food_x = 0;int food_y = 0;int food_z = 0;
int snake_body[64];
int mov_dir = 1; // 1:up   2:down   3:left    4:right    5:front    6:back
int snake_length = 1;
//End of Variable Section For Snake

//Variable Section For Matrix Two
int matrix_body[64];

int count = 0 ; //This is used in the animation
int color_count = 0; //Counts the color for ball bounce

//Variable Section for HSB to RGB
int r_save = 0;
int g_save = 0;
int b_save = 0;

//Variable Section for set cube color
boolean push_array[6];

int cube_x = 0; int cube_y = 0; int cube_z = 0;
int cube_save_r[64]; int cube_save_g[64]; int cube_save_b[64];
boolean flag = 0;

//Push Buttons
Bounce bouncer1 = Bounce(push_1,2);
Bounce bouncer2 = Bounce(push_2,2);
Bounce bouncer3 = Bounce(push_3,2);
Bounce bouncer4 = Bounce(push_4,2);
Bounce bouncer5 = Bounce(push_5,2);
Bounce bouncer6 = Bounce(push_6,2);

//
long max_freq = 0;

//Setup Section
void setup(){
  SPI.setBitOrder(LSBFIRST);//This set the order of bits shifted out of and into the SPI
bus.
```

```
  SPI.setDataMode(SPI_MODE0); //This sets the mode to the Rising edge to shift out
data, and when idle clock is low
  SPI.setClockDivider(SPI_CLOCK_DIV2);//Operating at 8Mhz, half of 16Mhz, of the
board.
  noInterrupts(); //No interrupts until finished setting up.
  //Use Timer 1 for interrupts to referesh the LED cube display
  TCCR1A = 0;
  //TCCR1B = B00000011; //For Timer 1 0x03 clock divides 64.
  TCCR1B = B00001011;
  TIMSK1 = B00000010;

  OCR1A = 25;//This values set to control the refresh rate, for compare match.
  // Refresh rate is 1/(OCR1A*4us) Hz. With OCRA1A = 25, then the multiplex frequency
is 10kHz.
  //TIMSK1 |= (1 << OCIE1A);  // enable timer compare interrupt

  //Setup the Output pins
  pinMode(blank_pin,OUTPUT);
  pinMode(enable_pin,OUTPUT);
  pinMode(output_pin,OUTPUT);
  pinMode(output_clk_pin,OUTPUT);
  pinMode(test_pin, OUTPUT);

  //Setup Push Buttons
  pinMode(push_1,INPUT);
  digitalWrite(push_1,HIGH); //turn on pull upresistors
  pinMode(push_2,INPUT);
  digitalWrite(push_2,HIGH); //turn on pull upresistors
  pinMode(push_3,INPUT);
  digitalWrite(push_3,HIGH); //turn on pull upresistors
  pinMode(push_4,INPUT);
  digitalWrite(push_4,HIGH); //turn on pull upresistors
  pinMode(push_5,INPUT);
  digitalWrite(push_1,HIGH); //turn on pull upresistors
  pinMode(push_6,INPUT);
  digitalWrite(push_6,HIGH); //turn on pull upresistors

  randomSeed(analogRead(2)); //Create Random Seed;
   //Ball Bounce Animation Variables initialization
  x_pos = random(0,2); y_pos = random(0,2); z_pos = random(0,2);
  x_velocity = random(2); y_velocity = random(2); z_velocity = random(2);
  while(x_velocity == 0 && y_velocity == 0 && z_velocity == 0){
    x_velocity = random(0,2); y_velocity = random(0,2); z_velocity = random(0,2);}

  //Setup Snake Animation Variables;
  snake_body[head_x+4*head_y+16*head_z] = mov_dir;
```

```
  snake_body[tail_x+4*tail_y+16*tail_z] = 4;
  new_food_pos();
  while(snake_body[food_x+4*food_y+16*food_z]!= 0){new_food_pos();}

  //Setup Bluetooth Connection Section
  Serial.begin(9600);
  Serial.flush();

  //Begin Multiplexing and interrupt
  SPI.begin(); // Begin the SPI library
  interrupts();// Begin the interupt.

  //Get max frequency for light frequency calibration
  //max_freq = getFrequency(5);
}

//Loop Section
void loop(){
  switch(light_mode){
    case 0:
      intro_animation();
      //color_wheel();
      BT_parse();
      delay(50);
      //delay(100);
      break;
    case 1:
      //box_display();
      snake();//animation2();//Animation 2
      //delay(3000);
      BT_parse();
      break;
    case 2:
      set_color_cube();
      BT_parse();
      break;
    case 3:
      ball_bounce();
      BT_parse();
      delay(300); //Velocity Control
      break;
    case 4:
      //set_color_IR();
      //animation2();
      animation1(); //IR Display
      BT_parse();
```

```
      delay(20);
      break;
    case 5:
      clear_Cube();
      matrix_code();
      delay(50);
      BT_parse();
      break;
    case 6:
      color_wheel();
      delay(77);
      BT_parse();
      break;
    case 7:
      box_display();
      BT_parse();
      delay(60);
      //delay(250);
      break;
    case 99:
      snake_dead();
      delay(30);
      BT_parse();
      break;
    default:
      delay(100);//if doesn't match this means
  }
}//End of Loop Section

//Intro Animation
void intro_animation(){
  for (int i = 0; i<4; i++){
   for (int j = 0; j <4; j++){
     for (int k =0 ;k<4;k++){
       LED_setup(i,j,k,0,0,0);
       if((i == 0) && (j==0)){LED_setup(i,j,k,0,15,0);}
       if((i == 0) && (j==3)){LED_setup(i,j,k,0,15,0);}
       if((i == 3) && (j==0)){LED_setup(i,j,k,0,15,0);}
       if((i == 3) && (j==3)){LED_setup(i,j,k,0,15,0);}

       if((i == 0) && (k==0)){LED_setup(i,j,k,0,15,0);}
       if((i == 0) && (k==3)){LED_setup(i,j,k,0,15,0);}
       if((i == 3) && (k==0)){LED_setup(i,j,k,0,15,0);}
       if((i == 3) && (k==3)){LED_setup(i,j,k,0,15,0);}

       if((j == 0) && (k==0)){LED_setup(i,j,k,0,15,0);}
```

```
      if((j == 0) && (k==3)){LED_setup(i,j,k,0,15,0);}
      if((j == 3) && (k==0)){LED_setup(i,j,k,0,15,0);}
      if((j == 3) && (k==3)){LED_setup(i,j,k,0,15,0);}
    }
   }
  }
}

//Bluetooth Parse Function
boolean BT_parse(){
  //Serial.flush();
  // Wait for incoming data
  int ard_command = 0;
  if (Serial.available() > 0)
  {
     ard_command = Serial.read(); // read the command
     //Serial.flush();
     if(ard_command == '0') {//Initial Mode
       light_mode = 0;
       clear_Cube();
     } //Change Light Mode

     //----------- 1 ------------// Snake
     if(ard_command == '1') {
       randomSeed(analogRead(2)); //Create Random Seed;
       light_mode = 1;
       clear_Cube();
       //Snake Variable Initialization
       head_x = 1; head_y = 0; head_z = 0;
       tail_x = 0; tail_y = 0; tail_z = 0;
       food_x = 0; food_y = 0; food_z = 0;
       for(int kk = 0; kk<64; kk++){snake_body[kk]=0;}//Clear Out Snake Body}
       mov_dir = 4; // 1:up   2:down   3:left    4:right    5:front    6:back
       snake_length = 1;
       snake_body[head_x+4*head_y+16*head_z] = mov_dir;
       snake_body[tail_x+4*tail_y+16*tail_z] = 4;
       new_food_pos();
       while(snake_body[food_x+4*food_y+16*food_z]!= 0){new_food_pos();}
}//Change Light Mode
     //----------- 2 ------------// Set Color Cube
     if(ard_command == '2') {
       light_mode = 2; clear_Cube();
       for(int kk = 0; kk<64;kk++){
         cube_save_r[kk] = 0;
         cube_save_g[kk] = 0;
         cube_save_b[kk] = 0;
```

```
      }
      cube_x = 0;
      cube_y = 0;
      cube_z = 0;
} //Change Light Mode

    //---------- 3 ----------// Ball Bounce
    if(ard_command == '3') {
      light_mode = 3;
        randomSeed(analogRead(2)); //Create Random Seed;
        //Ball Bounce Animation Variables initialization
        x_pos = random(0,2); y_pos = random(0,2); z_pos = random(0,2);
        x_velocity = random(2); y_velocity = random(2); z_velocity = random(2);
        while(x_velocity == 0 && y_velocity == 0 && z_velocity == 0){
           x_velocity = random(0,2); y_velocity = random(0,2); z_velocity = random(0,2);}
        ball_count = 0;
} //Change Light Mode

    //---------- 4 ----------// IR Distance Sensing Moving
    if(ard_command == '4') {
      light_mode = 4;
      clear_Cube();
    } //Change Light Mode
    if(ard_command == '5') {
      light_mode = 5;
      clear_Cube();
    } //Change Light Mode
    if(ard_command == '6') {
      light_mode = 6;
      clear_Cube();
    } //Change Light Mode

    if(ard_command == '7') {
      light_mode = 7;
      clear_Cube();
    }
    return 1;
  } // if serial empty then continue to multiplexing
  return 0;
}//End of BT_Parse


//The interrupt routine
ISR(TIMER1_COMPA_vect){//This is the timer compare interrupt

if (test_count == 0)
```

```
{
  test_count =1;
  digitalWrite(test_pin, HIGH);
}
else
{
  test_count =0;
  digitalWrite(test_pin, LOW);
}

//****Multiplexing Section****//
//PORTD |= 1<<blank_pin;
if(mod_count == 0){
  bit_shift(mod_bit);
  //level_shift(z_index);
  level_shift();
  PORTD |= 1<<enable_pin;//High
  PORTD &= 0<<enable_pin;//Low
  //digitalWrite(enable_pin,HIGH);
  //digitalWrite(enable_pin,LOW);
  mod_bit++;
}
else if(mod_count == 1){
  bit_shift(mod_bit);
  //level_shift(z_index);
  level_shift();
  PORTD |= 1<<enable_pin;//High
  PORTD &= 0<<enable_pin;//Low
  //digitalWrite(enable_pin,HIGH);
  //digitalWrite(enable_pin,LOW);
  mod_bit++;
}
else if (mod_count ==3){
  bit_shift(mod_bit);
  //level_shift(z_index);
  level_shift();
  PORTD |= 1<<enable_pin;//High
  PORTD &= 0<<enable_pin;//Low
  //digitalWrite(enable_pin,HIGH);
  //digitalWrite(enable_pin,LOW);
  mod_bit++;
}
else if(mod_count == 7){
  bit_shift(mod_bit);
  //level_shift(z_index);
  level_shift();
```

```
  PORTD |= 1<<enable_pin;//High
  PORTD &= 0<<enable_pin;//Low
  //digitalWrite(enable_pin,HIGH);
  //digitalWrite(enable_pin,LOW);
  mod_bit++;
}

mod_count++;

//According to which mod_bit it is currently at. Then shift out the corresponding control
signals.

if( mod_count == 15){
  mod_count =0;
  mod_bit = 0;
  z_index = z_index+1;//Go to the nex level
}

if(z_index == 4){
  z_index = 0; //If reached the top, then reset to level 0
}
}//End of ISR

void LED_setup(int x, int y, int z, byte red, byte green, byte blue){
//This function will take the index of the LED, in the 3 dimensional cube, (x,y,z),
//and the color to set the LED to in RGB, on the scale of 0 to 15.

  //Overflow prevention and sanity check to avoid undesired behavior in both position and
color
  if(x<0){x = 0;}
  if(y<0){y = 0;}
  if(z<0){z = 0;}
  if(x>3){x = 3;}
  if(y>3){y = 3;}
  if(z>3){z = 3;}
  if(red<0){red = 0;}
  if(green<0){green =0;}
  if(blue<0){blue = 0;}
  if(red>15){red = 15;}
  if(green>15){green = 15;}
  if(blue>15){blue = 15;}

//Now using the x,y,z a linear index will be created using the following formula
16*z+4*y+x
//First, now determine which Byte to write to, as the color array are in byte.
```

```
//So have 8 byte each correspond to [0...7][8...15]...[56...63]. So, each layer users two byte.

int byte_num = int((z*16+4*y+x)/8); // Thi byte_num correspond to write byte to write to. typecast to int so round down.
int bit_num = int(z*16+4*y+x)-8*byte_num;//This correspondes to the bit number to be written
//Then following the Bit Angle Mod, write the values into the array
//Write Red
bitWrite(red3[byte_num], bit_num, bitRead(red,3));
bitWrite(red2[byte_num], bit_num, bitRead(red,2));
bitWrite(red1[byte_num], bit_num, bitRead(red,1));
bitWrite(red0[byte_num], bit_num, bitRead(red,0));

//Write Green
bitWrite(green3[byte_num], bit_num, bitRead(green,3));
bitWrite(green2[byte_num], bit_num, bitRead(green,2));
bitWrite(green1[byte_num], bit_num, bitRead(green,1));
bitWrite(green0[byte_num], bit_num, bitRead(green,0));

//Write Blue
bitWrite(blue3[byte_num], bit_num, bitRead(blue,3));
bitWrite(blue2[byte_num], bit_num, bitRead(blue,2));
bitWrite(blue1[byte_num], bit_num, bitRead(blue,1));
bitWrite(blue0[byte_num], bit_num, bitRead(blue,0));

}//End of LED Index

void bit_shift(int mod_bit_in){
   switch(mod_bit_in){
   case 3:///2*zindex because
    for(int i=2*z_index; i<2*z_index+2; i++)
      SPI.transfer(red3[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
      SPI.transfer(green3[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
      SPI.transfer(blue3[i]);
     break;
   case 2:
    for(int i=2*z_index; i<2*z_index+2; i++)
      SPI.transfer(red2[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
      SPI.transfer(green2[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
      SPI.transfer(blue2[i]);
     break;
```

```
   case 1:
     for(int i=2*z_index; i<2*z_index+2; i++)
     SPI.transfer(red1[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
     SPI.transfer(green1[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
     SPI.transfer(blue1[i]);
     break;
   case 0:
     for(int i=2*z_index; i<2*z_index+2; i++)
     SPI.transfer(red0[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
     SPI.transfer(green0[i]);
    for(int i=2*z_index; i<2*z_index+2; i++)
     SPI.transfer(blue0[i]);
     break;
  }
}//End of Bit_shift

void level_shift(){
 switch(z_index){
 case 0:
   SPI.transfer(B10000000);
   break;
 case 1:
   SPI.transfer(B01000000);
   break;

 case 2:
   SPI.transfer(B00100000);
   break;

 case 3:
   SPI.transfer(B00010000);
   break;
 }

}//End of level_shift

//********Sensor Processing Section******//
void IR_Distance_Get(){
 int sensorValue = analogRead(A0);
 float voltage = sensorValue * (5.0 / 1023.0);
 // print out the value you read:
 int distance = int((198*pow(2.71828,voltage*(-2.7656))+0.7262)*2)+5;//type cast to
int
```

```
  //int distance =int((198*pow(2.71828,voltage*(-2.7656))));//type cast to int
  //Then quantize into 5 cases.
  if(distance > 10  && distance < 40){
    IR_Distance = 3;
  }
  else if(distance > 40 && distance <60){
    IR_Distance = 2;
  }
  else if(distance > 60 && distance <80){
    IR_Distance = 1;
  }
  else{
    IR_Distance = 0;
  }
  delay(10);
}

int IR_Distance_Get2(){
  int sensorValue = analogRead(A0);
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  int distance = int((198*pow(2.71828,voltage*(-2.7656))+0.7262)*2)+5;//type cast to
int
  //Then quantize into 5 cases.
}

void IR_Speed_Get(){
  int sensorValue1 = analogRead(A0);
  delay(5);
  int sensorValue2 = analogRead(A0);
  float voltage1 = sensorValue1 * (5.0 / 1023.0);
  float voltage2 = sensorValue2 * (5.0 / 1023.0);
  float distance1 = float((198*pow(2.71828,voltage1*(-2.7656))+0.7262)*2);
  float distance2 = float((198*pow(2.71828,voltage2*(-2.7656))+0.7262)*2);
  IR_Speed = (distance2 - distance1)/5; //IR Speed is in cm/ms = 10m/s
}

long getFrequency(int pin) {
  #define SAMPLES 600
  int ret = 0;
  long freq = 0;
  for(unsigned int j=0; j<SAMPLES; j++) freq+= 1000000/pulseIn(pin, HIGH, 250000);

  if(freq/SAMPLES<30000){
    return 30000;
  }
```

```
  if(freq/SAMPLES>400000){
    return 400000;
  }
  return freq / SAMPLES;
}

long get_color(){
  long freq_to_c = getFrequency(5);
  return frequency_to_color(freq_to_c,150000);
}

long frequency_to_color(long freq_to_c, long max_c){
  long k = (freq_to_c/max_c)*4096;
  return k;
}

//****Animation Section****//

//Animation 1 Code
void animation1(){
  //This animation uses IR Sensor.
  IR_Distance_Get();
  for (int i = 0; i<4; i++){
    for (int j = 0; j <4; j++){
      for (int k =0 ;k<4;k++){
        switch(IR_Distance){
        case 0:
          if(j == 0){
            LED_setup(i,j,k,10,0,8);
          }
          else{
            LED_setup(i,j,k,0,0,0);
          }
          break;
        case 1:
          if(j == 1){
            LED_setup(i,j,k,10,0,5);
          }
          else{
            LED_setup(i,j,k,0,0,0);
          }
          break;
        case 2:
          if(j == 2){
            LED_setup(i,j,k,10,0,4);
          }
```

```
        else{
          LED_setup(i,j,k,0,0,0);
        }
        break;
      case 3:
        if(j == 3){
          LED_setup(i,j,k,10,0,2);
        }
        else{
          LED_setup(i,j,k,0,0,0);
        }
        break;
      }}}}
  delay(77);
}//End of animation1

void animation2(){
  //This animation uses light Frequency Sensor.
  frq = getFrequency(5);
  for (int i = 0; i<4; i++){
    for (int j = 0; j <4; j++){
      for (int k =0 ;k<4;k++)
      {
        if(frq<50000){
          LED_setup(i,j,k,0,0,8);
        }
        else{
          LED_setup(i,j,k,0,8,8);
        }
      }
    }
  }

}

//Animation 3 Code
void animation3(){
   for(int kk = 0; kk<3;kk++){
     for (int b = 0; b<15;b++){
       animation3_2(b,kk,count);//Animation 1
     }
     for (int b = 15; b>0;b--){
       animation3_2(b,kk,count);//Animation 1
     }
   }
   count++;
```

```
   if (count ==4){
     count = 0;
   }
}
void animation3_2(int b,int kk, int count){
  //This animation will flash green level by level
   for (int i = 0; i<4; i++){
     for (int j = 0; j <4; j++){
       for (int k =0 ;k<4;k++){
         if(i==0 && j ==0){
           if (count == k){
             if(kk==0){
               LED_setup(i,j,k,b,0,0);
             }
             if(kk==1){
               LED_setup(i,j,k,0,0,b);
             }
             if(kk==2){
               LED_setup(i,j,k,b,0,b);
             }
           }
           else{
             LED_setup(i,j,k,0,0,0);
           }
         }

        if(i==1 && j ==0){
           if(count ==k){
             if(kk==0){
               LED_setup(i,j,k,0,0,b);
             }
             if(kk==1){
               LED_setup(i,j,k,b,0,b);
             }
             if(kk==2){
               LED_setup(i,j,k,b,0,0);
             }
           }
           else{
             LED_setup(i,j,k,0,0,0);
           }
          }
       }
     }
   }
  delay(50);
```

```
}//End of animation3_2

//Variable Section For Ball Bounce Animation
//Ball Size is 2x2x2
//int ball_size = 1;
//int x_velocity, y_velocity, z_velocity, x_pos, y_pos,z_pos;

void ball_bounce(){
  if(ball_count == 15)
  {
    x_pos = random(0,2); y_pos = random(0,2); z_pos = random(0,2);
    x_velocity = random(2); y_velocity = random(2); z_velocity = random(2);
    while(x_velocity == 0 && y_velocity == 0 && z_velocity == 0){x_velocity =
random(0,2); y_velocity = random(0,2); z_velocity = random(0,2);}
    ball_count = 0;
  }
  ball_count++;
  x_pos = x_pos+x_velocity;
  y_pos = y_pos+y_velocity;
  z_pos = z_pos+z_velocity;

  //At the left edge && //At the right edge
  if(x_pos == 0 || x_pos == 2) //Limit to 2 so the ball wouldn't go out of range.
  {
    x_velocity = -1*x_velocity;
    if(x_velocity != 0){
      color_count++;
    }
  }
  //At the front edge && //At the back edge
  if(y_pos == 0 || y_pos == 2) //Limit to 2 so the ball wouldn't go out of range.
  {
    y_velocity = -1*y_velocity;
    if(y_velocity != 0){
      color_count++;
    }
  }

  //At the top edge && //At the bottom edge
  if(z_pos == 0 || z_pos == 2) //Limit to 2 so the ball wouldn't go out of range.
  {
    z_velocity = -1*z_velocity;
    if(z_velocity != 0){
      color_count++;
    }
  }
```

```
 if(color_count > 2)
 {
  color_count = color_count % 3;
 }
 // color_count = 0;
 //Index the LED Base on the ball position
 clear_Cube();
 int r,g,b;
 //Base on the case, it will display the correct col
 switch(color_count){
  case 0:
    r = 8; g = 0; b = 0;
    LED_setup(x_pos,y_pos,z_pos,r,g,b);
    LED_setup(x_pos,y_pos,z_pos+1,r,g,b);
    LED_setup(x_pos,y_pos+1,z_pos,r,g,b);
    LED_setup(x_pos,y_pos+1,z_pos+1,r,g,b);
    LED_setup(x_pos+1,y_pos,z_pos,r,g,b);
    LED_setup(x_pos+1,y_pos,z_pos+1,r,g,b);
    LED_setup(x_pos+1,y_pos+1,z_pos,r,g,b);
    LED_setup(x_pos+1,y_pos+1,z_pos+1,r,g,b);
    break;
  case 1:
    r = 0; g = 8; b = 0;
    LED_setup(x_pos,y_pos,z_pos,r,g,b);
    LED_setup(x_pos,y_pos,z_pos+1,r,g,b);
    LED_setup(x_pos,y_pos+1,z_pos,r,g,b);
    LED_setup(x_pos,y_pos+1,z_pos+1,r,g,b);
    LED_setup(x_pos+1,y_pos,z_pos,r,g,b);
    LED_setup(x_pos+1,y_pos,z_pos+1,r,g,b);
    LED_setup(x_pos+1,y_pos+1,z_pos,r,g,b);
    LED_setup(x_pos+1,y_pos+1,z_pos+1,r,g,b);
    break;
  case 2:
    r = 0; g = 0; b = 8;
    LED_setup(x_pos,y_pos,z_pos,r,g,b);
    LED_setup(x_pos,y_pos,z_pos+1,r,g,b);
    LED_setup(x_pos,y_pos+1,z_pos,r,g,b);
    LED_setup(x_pos,y_pos+1,z_pos+1,r,g,b);
    LED_setup(x_pos+1,y_pos,z_pos,r,g,b);
    LED_setup(x_pos+1,y_pos,z_pos+1,r,g,b);
    LED_setup(x_pos+1,y_pos+1,z_pos,r,g,b);
    LED_setup(x_pos+1,y_pos+1,z_pos+1,r,g,b);
    break;
 }
}
```

```
//This Function Clears the LED Cube. Erase old values.
void clear_Cube(){
  noInterrupts();
  for (int i = 0; i<4; i++){
   for (int j = 0; j <4; j++){
     for (int k =0 ;k<4;k++)
     {
        LED_setup(i,j,k,0,0,0);
}}}
  interrupts();
}//End of clear_Cube

void snake(){
  //Display
 for (int i = 0; i<4; i++){
   for (int j = 0; j <4; j++){
     for (int k =0 ;k<4;k++)
     {
       if(snake_body[i+4*j+16*k] != 0){
         LED_setup(i,j,k,0,5,0); // color the body
       }
       else{
         LED_setup(i,j,k,0,0,0); //else blank
       }
     }
   }
 }
 LED_setup(food_x,food_y,food_z,8,0,0); //Display the food
 LED_setup(head_x,head_y,head_z,0,0,8);//draw the head
 delay(1000);
  //Check Bluetooth at the very end for the next move.
  if (Serial.available() > 0)
  {
     int mov_dir_c = Serial.read(); // read the command
     //Update the moving direction, ignore if the reverse is pressed
     if(mov_dir_c == 'u' && mov_dir != 2) {mov_dir = 1;}
     if(mov_dir_c == 'd' && mov_dir != 1) {mov_dir = 2;}
     if(mov_dir_c == 'l' && mov_dir != 4) {mov_dir = 3;}
     if(mov_dir_c == 'r' && mov_dir != 3) {mov_dir = 4;}
     if(mov_dir_c == 'f' && mov_dir != 6) {mov_dir = 5;}
     if(mov_dir_c == 'b' && mov_dir != 5) {mov_dir = 6;}
     if(mov_dir_c == '0') {light_mode = 0;}
  }

 //Move the Head
 int head_idx = head_x + 4*head_y + 16*head_z; //index for the current head position
```

```
switch(mov_dir){
  case 1:
    head_z = head_z + 1;
     snake_body[head_idx] = 1;
    break;
  case 2:
    head_z = head_z - 1;
    snake_body[head_idx] = 2;
    break;
  case 3:
    head_x = head_x - 1;
    snake_body[head_idx] = 3;
    break;
  case 4:
    head_x = head_x + 1;
    snake_body[head_idx] = 4;
    break;
  case 5:
    head_y = head_y + 1;
    snake_body[head_idx] = 5;
    break;
  case 6:
    head_y = head_y - 1;
    snake_body[head_idx] = 6;
    break;
}
head_idx = head_x + 4*head_y + 16*head_z; //index for the updated head position
int food_idx = food_x + 4*food_y + 16*food_z; //index for the current tail position
int got_food = 0; //flag for food

//Check if got food
if(head_idx==food_idx)
{
  got_food = 1;
}

int tail_idx = tail_x + 4*tail_y + 16*tail_z;
if(!got_food)
{ //Update the tail location, if didn't get food.
  switch(snake_body[tail_idx]){ //&& snake)length <10;
  case 1:
    tail_z = tail_z + 1;
    break;
  case 2:
    tail_z = tail_z - 1;
    break;
```

```
  case 3:
    tail_x = tail_x - 1;
    break;
  case 4:
    tail_x = tail_x + 1;
    break;
  case 5:
    tail_y = tail_y + 1;
    break;
  case 6:
    tail_y = tail_y - 1;
    break;
  }
  snake_body[tail_idx] = 0; //Tail removed
}
else
{
  snake_length = snake_length +1;
  //Generate new food location;
  //new_food_pos();
  snake_body[food_idx] = -1;
  while(snake_body[food_idx] != 0){
    new_food_pos();
    food_idx = food_x + 4*food_y + 16*food_z;
  }
  //snake_body[food_x+4*food_y+16*food_z] = 7; //Update the food location on body
}
//Check if there is Collision
if(!got_food && (snake_body[head_idx]!=0))
{
  light_mode = 99;
  clear_Cube();
  return;
  //there is a collision with body
}
if(head_x >3 || head_x < 0){
  light_mode = 99;
  clear_Cube();
  return;
}
if(head_y >3 || head_y < 0){
  light_mode = 99;
  clear_Cube();
  return;
}
if(head_z >3 || head_z < 0){
```

```
   light_mode = 99;
   clear_Cube();
   return;
  }
   //there is a collision with wall;

}//End of Snake Function

void new_food_pos(){
   //New Food Position from 0 to 3.
   food_x = int(random(0,4)); food_y = int(random(0,4)); food_z = int(random(0,4));
}

void snake_dead(){ //Snake Dead display all red
 for (int i = 0; i<4; i++){
  for (int j = 0; j <4; j++){
    for (int k =0 ;k<4;k++){
      if(i+4*j+16*k < snake_length){
         LED_setup(i,j,k,10,0,0);//Display a unique color to indicate in change animation
mode
      }}}}
  delay(30);
}

int kk = 0;
void matrix_code(){
   int x_p[16], y_p[16];
   int drop_c = random(1,16);
   //int rand_c[16];
   //int rand_b[16];
   //int rand_r[16];
   int rand_c[drop_c];
   int rand_b[drop_c];
   int rand_r[drop_c];
   int delay_time = random(200,250);
   for(int i =0 ; i<drop_c; i++){
     x_p[i] = random(0,4);
     y_p[i] = random(0,4);
     rand_c[i] = random(3,5);
     rand_b[i] = random(3,10);
     rand_r[i] = random(3,16);
   }

   for (int kk = 0; kk<8; kk++){
     switch(kk){
       case 0:
```

```
  for(int v = 0; v<drop_c; v++){
    LED_setup(x_p[v],y_p[v],3,rand_r[v],rand_c[v],rand_b[v]);
  }
 break;
 case 1:
  for(int v = 0; v<drop_c; v++){

LED_setup(x_p[v],y_p[v],3,int(0.9*rand_r[v]),int(0.9*rand_c[v]),int(0.9*rand_b[v]));
    LED_setup(x_p[v],y_p[v],2,rand_r[v],rand_c[v],rand_b[v]);
  }
 break;
 case 2:
  for(int v = 0; v<drop_c; v++){

LED_setup(x_p[v],y_p[v],3,int(0.8*rand_r[v]),int(0.8*rand_c[v]),int(0.8*rand_b[v]));

LED_setup(x_p[v],y_p[v],2,int(0.9*rand_r[v]),int(0.9*rand_c[v]),int(0.9*rand_b[v]));
     LED_setup(x_p[v],y_p[v],1,rand_r[v],rand_c[v],rand_b[v]);
  }
 break;
 case 3:
  for(int v = 0; v<drop_c ; v++){

LED_setup(x_p[v],y_p[v],3,int(0.7*rand_r[v]),int(0.7*rand_c[v]),int(0.7*rand_b[v]));

LED_setup(x_p[v],y_p[v],2,int(0.8*rand_r[v]),int(0.8*rand_c[v]),int(0.8*rand_b[v]));

LED_setup(x_p[v],y_p[v],1,int(0.9*rand_r[v]),int(0.9*rand_c[v]),int(0.9*rand_b[v]));
    LED_setup(x_p[v],y_p[v],0,rand_r[v],rand_c[v],rand_b[v]);
  }
 break;
 case 4:
  for(int v = 0; v<drop_c; v++){
    LED_setup(x_p[v],y_p[v],3,0,0,0);

LED_setup(x_p[v],y_p[v],2,int(0.5*rand_r[v]),int(0.5*rand_c[v]),int(0.5*rand_b[v]));

LED_setup(x_p[v],y_p[v],1,int(0.7*rand_r[v]),int(0.7*rand_c[v]),int(0.7*rand_b[v]));

LED_setup(x_p[v],y_p[v],0,int(0.8*rand_r[v]),int(0.8*rand_c[v]),int(0.8*rand_b[v]));

//LED_setup(x_p[v],y_p[v],0,int(0.9*rand_r[v]),int(0.9*rand_c[v]),int(0.9*rand_b[v]));
  }
 break;
 case 5:
  for(int v = 0; v<drop_c;v++){
```

```
          LED_setup(x_p[v],y_p[v],3,0,0,0);
          LED_setup(x_p[v],y_p[v],2,0,0,0);

LED_setup(x_p[v],y_p[v],1,int(0.4*rand_r[v]),int(0.4*rand_c[v]),int(0.4*rand_b[v]));

LED_setup(x_p[v],y_p[v],0,int(0.5*rand_r[v]),int(0.5*rand_c[v]),int(0.5*rand_b[v]));

//LED_setup(x_p[v],y_p[v],1,int(0.6*rand_r[v]),int(0.6*rand_c[v]),int(0.6*rand_b[v]));

//LED_setup(x_p[v],y_p[v],0,int(0.7*rand_r[v]),int(0.7*rand_c[v]),int(0.7*rand_b[v]));
        }
      break;
      case 6:
        for(int v = 0; v<drop_c;v++){
          LED_setup(x_p[v],y_p[v],3,0,0,0);
          LED_setup(x_p[v],y_p[v],2,0,0,0);
          LED_setup(x_p[v],y_p[v],1,0,0,0);

LED_setup(x_p[v],y_p[v],0,int(0.2*rand_r[v]),int(0.2*rand_c[v]),int(0.2*rand_b[v]));

//LED_setup(x_p[v],y_p[v],2,int(0.4*rand_r[v]),int(0.4*rand_c[v]),int(0.4*rand_b[v]));

//LED_setup(x_p[v],y_p[v],1,int(0.5*rand_r[v]),int(0.5*rand_c[v]),int(0.5*rand_b[v]));

//LED_setup(x_p[v],y_p[v],0,int(0.6*rand_r[v]),int(0.6*rand_c[v]),int(0.6*rand_b[v]));
        }
      break;
      case 9:
      for(int v = 0; v<drop_c;v++){

LED_setup(x_p[v],y_p[v],3,int(0.1*rand_r[v]),int(0.1*rand_c[v]),int(0.1*rand_b[v]));

LED_setup(x_p[v],y_p[v],2,int(0.2*rand_r[v]),int(0.2*rand_c[v]),int(0.2*rand_b[v]));

LED_setup(x_p[v],y_p[v],1,int(0.3*rand_r[v]),int(0.3*rand_c[v]),int(0.3*rand_b[v]));

LED_setup(x_p[v],y_p[v],0,int(0.4*rand_r[v]),int(0.4*rand_c[v]),int(0.4*rand_b[v]));
        }

      break;
      case 8:
        for(int v = 0; v<drop_c;v++){

LED_setup(x_p[v],y_p[v],3,int(0.0*rand_r[v]),int(0.0*rand_c[v]),int(0.0*rand_b[v]));

LED_setup(x_p[v],y_p[v],2,int(0.0*rand_r[v]),int(0.0*rand_c[v]),int(0.0*rand_b[v]));
```

```
LED_setup(x_p[v],y_p[v],1,int(0.1*rand_r[v]),int(0.1*rand_c[v]),int(0.1*rand_b[v]));

LED_setup(x_p[v],y_p[v],0,int(0.2*rand_r[v]),int(0.2*rand_c[v]),int(0.2*rand_b[v]));
        }
       break;
       case 7:
        for(int v = 0; v<drop_c; v++){
          LED_setup(x_p[v],y_p[v],0,0,0,0);
          LED_setup(x_p[v],y_p[v],1,0,0,0);
          LED_setup(x_p[v],y_p[v],2,0,0,0);
          LED_setup(x_p[v],y_p[v],3,0,0,0);
        }
       break;
     }
   delay(delay_time);
  }
}



void HSVtoRGB(float h, float s, float v){
  //rnage S,V is 0 to 100. H is 0 to 360;
  int i = 0;
  float f,p,q,t;
  float r_temp, g_temp, b_temp;
  if(s ==0){
    r_temp = v;
    g_temp = v;
    b_temp = v;
    return;
  }

  h = h/60; // divide into 0 to 5
  i = int(h);
  f = h-i;
  p = v*(1-s);
  q = v*(1-s*f);
  t = v*(1-s*(1-f));

  switch(i){
    case 0:
      r_temp = v;
      g_temp = t;
      b_temp = p;
      break;
```

```
    case 1:
      r_temp = q;
      g_temp = v;
      b_temp = p;
      break;
    case 2:
      r_temp = p;
      g_temp = v;
      b_temp = t;
      break;
    case 3:
      r_temp = p;
      g_temp = q;
      b_temp = v;
      break;
    case 4:
      r_temp = t;
      g_temp = p;
      b_temp = v;
      break;
    default:
      r_temp = v;
      g_temp = p;
      b_temp = q;
      break;
  }

  r_save = int (r_temp/15); //type cast into range of 0,15;
  g_save = int (g_temp/15);
  b_save = int (b_temp/15);
}//End of HSV to RGB

void set_color_cube(){
  //display cube
  int idx = 0;
  for(int i =0 ; i<4;i++){
    for(int j =0; j<4;j++){
      for(int k =0; k<4;k++){
        idx = i+4*j+16*k;
        if(idx!=cube_x+4*cube_y+16*cube_z){
          LED_setup(i,j,k,cube_save_r[idx],cube_save_g[idx],cube_save_b[idx]);
        }
        else{
           LED_setup(i,j,k,0,0,0);}}}}
  int c_r = 0;
  int c_g = 0;
```

```
 int c_b = 0;
 long color_freq = getFrequency(5);

if(color_freq > 350000){ //color 1
   c_r = 10;
   c_g = 0;
   c_b = 0;
 }
 else if(color_freq > 320000){ //color 1
   c_r = 10;
   c_g = 0;
   c_b = 5;
 }
 else if(color_freq > 300000){
   c_r = 10;
   c_g = 0;
   c_b = 10;
 }
 else if(color_freq > 220000){
   c_r = 5;
   c_g = 0;
   c_b = 10;
 }
 else if(color_freq > 200000){
   c_r = 0;
   c_g = 0;
   c_b = 10;
 }
 else if(color_freq > 150000){
   c_r = 0;
   c_g = 5;
   c_b = 10;
 }
 else if(color_freq > 120000){
   c_r = 0;
   c_g = 10;
   c_b = 10;
 }
 else if(color_freq > 100000){
   c_r = 0;
   c_g = 10;
   c_b = 5;
 }
 else{
   c_r = 0;
   c_g = 10;
```

```
  c_b = 0;
 }
 //Blink
 if(flag){
  LED_setup(cube_x,cube_y,cube_z,0,0,0);
 }
 else{
  LED_setup(cube_x,cube_y,cube_z,c_r,c_g,c_b);
 }
 button_delay(500);
 if(push_array[5]==0){cube_x = cube_x +1;}
 if(push_array[4]==0){cube_y = cube_y +1;}
 if(push_array[3]==0){cube_z = cube_z +1;}
 if(push_array[2]==0){
  cube_save_r[cube_x+4*cube_y+16*cube_z] = c_r;
  cube_save_g[cube_x+4*cube_y+16*cube_z] = c_g;
  cube_save_b[cube_x+4*cube_y+16*cube_z] = c_b;}

 if(cube_x>3){cube_x = 0;}
 if(cube_y>3){cube_y = 0;}
 if(cube_z>3){cube_z = 0;}
 //Toogle the flag
 if(flag)
  flag = 0;
 else
  flag = 1;

}//End of set_color_cube

void button_delay(int time){
 for(int i =0 ; i< 6;i++){
  push_array[i] = 1; //Initialize to 0
 }
 for(int i = 0 ; i<time/25; i++){
  bouncer1.update(); bouncer2.update(); bouncer3.update();
  bouncer4.update(); bouncer5.update(); bouncer6.update();
  delay(25);
  if(push_array[0] == 1) {push_array[0] = bouncer1.read();}
  if(push_array[1] == 1) {push_array[1] = bouncer2.read();}
  if(push_array[2] == 1) {push_array[2] = bouncer3.read();}
  if(push_array[3] == 1) {push_array[3] = bouncer4.read();}
  if(push_array[4] == 1) {push_array[4] = bouncer5.read();}
  if(push_array[5] == 1) {push_array[5] = bouncer6.read();}
 }

}//End of button_delay
```

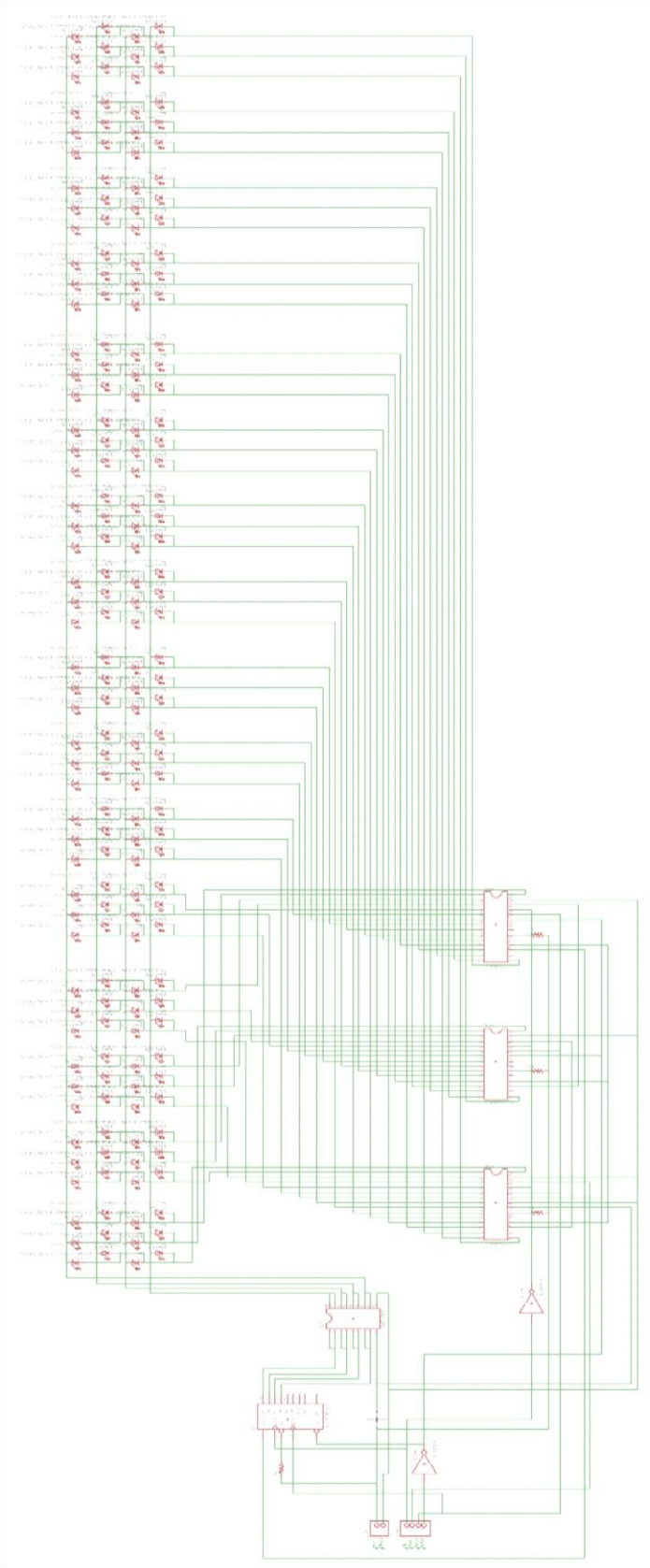
```
void box_display_help(int box_size, int s_case, int r_c, int g_c, int b_c)
{
  for(int i = 0; i <4;i++){
    for(int j =0; j<4;j++){
      for(int k = 0;k<4;k++){
        switch(s_case){
        case 0: //Case (0,0,0)
          if(i<box_size && j<box_size && k<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 1: //Case (3,0,0)
          if((3-i)<box_size && j<box_size && k<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 3: //Case(0,3,0)
          if(i<box_size && (3-j)<box_size && k<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 2: //Case(3,3,0)
          if((3-i)<box_size && (3-j)<box_size && k<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 4: //Case (0,0,3)
          if(i<box_size && j<box_size && (3-k)<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 5: //Case (3,0,3)
          if((3-i)<box_size && j<box_size && (3-k)<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 7: //Case(0,3,3)
          if(i<box_size && (3-j)<box_size && (3-k)<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
        case 6: //Case(3,3,3)
          if((3-i)<box_size && (3-j)<box_size && (3-k)<box_size){
            LED_setup(i,j,k,r_c,g_c,b_c);}
          break;
}}}}}//End of box_display_help

int rc = 15; int gc = 0; int bc = 0;
int routine = 0;
void box_display(){
  for(int v = 0; v<3;v++){
    for(int pp = 0; pp<4;pp++){
```

```
    plane_sweep(v,pp);
    delay(500);
    clear_Cube();
    color_wheel_helper();}

  if(BT_parse()){goto change_case;}
  for(int pp = 3;pp>-1;pp--){
    plane_sweep(v,pp);
    delay(100);
    clear_Cube();
    color_wheel_helper();
  }
  if(BT_parse()){goto change_case;}
  for(int pp = 0; pp<4;pp++){
    plane_sweep(v,pp);
    delay(100);
    clear_Cube();
    color_wheel_helper();
  }

  if(BT_parse()){goto change_case;}

  for(int pp = 3;pp>-1;pp--){
    plane_sweep(v,pp);
    delay(100);
    clear_Cube();
    color_wheel_helper();
    if(BT_parse()){
    goto change_case;}
  }
 }

for(int v = 0; v<8;v++){
  for(int i = 0 ; i<4;i++){
    box_display_help(i,v,rc,gc,bc);
    delay(100);
    clear_Cube();
    color_wheel_helper();
  }
  if(BT_parse()){goto change_case;}
  for(int i = 4; i>0 ;i--){
    box_display_help(i,v,rc,gc,bc);
    delay(100);
    clear_Cube();
    color_wheel_helper();
  }
```

```
   if(BT_parse()){goto change_case;}
  }
  change_case:;
}//End of box_display;

void color_wheel(){
  for(int i =0 ; i <4; i++){
    for(int j = 0; j<4;j++){
      for(int k = 0; k<4;k++){
        LED_setup(i,j,k,rc,gc,bc);
        color_wheel_helper();
        delay(30);
      }}}
} //End of color_wheel

void plane_sweep(int dir_save, int plane){
  for(int i = 0 ; i<4;i++){
    for(int j = 0 ; j<4;j++){
      for(int k = 0 ; k<4;k++){
        switch(dir_save){
          case 0:
            if(i == plane){LED_setup(i,j,k,rc,gc,bc);}
          break;

          case 1:
            if(j == plane){LED_setup(i,j,k,rc,gc,bc);}
          break;

          case 2:
            if(k == plane){LED_setup(i,j,k,rc,gc,bc);}
          break; }
      }}}
}

void color_wheel_helper(){
   if(routine == 0){
     bc = bc +1;
     if(bc == 16){ routine = routine +1;}
   }
   else if(routine == 1){
     rc = rc -1;
     if(rc == 0){routine = routine+1;}
   }
   else if(routine == 2){
     gc = gc+1;
     if(gc == 16){routine = routine+1;}
```

```
  }
  else if(routine == 3){
    bc = bc-1;
    if(bc == 0){routine = routine+1;}
  }
  else if(routine == 4){
    rc = rc + 1;
    if(rc == 16){routine = routine+1;}
  }
  else if(routine == 5){
    gc = gc-1;
    if(gc == 0){routine =0;}
  }
} //End of color_wheel_helper
```

Appendix B: Overall LED Cube Schematic



Appendix C: Requirements and Verifications

Requirements	Verification
1. Power a) Supply at least 5W. b) Supply 5±1V output voltage with an output voltage ripple not exceeding 0.5V.	a) Test each module to make sure that the maximum power consumed by each part does not exceed: Microcontroller-0.5W, TLC5940-0.13W, LED cube-2.88W, 74HC595-0.35W, MIC2981-0.03W, IR rangefinder-0.2W, light sensor-0.02W, Serial Bluetooth-0.2W. b) Using an oscilloscope to confirm the output voltage of the power supply stays within the 5±1V and ripple of 0.5V bounds.
2. Controller a) Compute and shift out data serially to the shift registers with a clock speed of at least 8MHz. b) Compute and output control signal to for displaying environmental sensing lighting patterns. c) Receive input from the sensor array, both ambient light and range sensor. d) Switches between lighting animations and modes when signal received from the communication block. e) The minimum base clock of the microcontroller must be greater than 8MHz. f) Support at least 8 GPIO including at least one hardware serial receive port and at least one analog input.	a) A test circuit with the shift-registers will be built and using LED or multi-meter to confirm that the shift registers output the data inputted by the user. b) A test circuit with the IR range sensor and light frequency sensor will be built. And observe if the control signal changes according to our specifications. The ambient light and range sensors will be tested separately to make sure that both can be used to control the brightness of a single LED. c) Connect the range sensor and light frequency sensor to the Arduino and use this sensor data to drive LEDs on the Arduino to confirm that the Arduino is correctly receiving sensor data and outputting the control signals to drive the cube in the expected manner (color and motion). d) Confirm Bluetooth connection by LED and check that serial data is received on the Arduino by running simple code to change LED status. Observe that cube changes lighting animations and modes when the change animation signal is received. e) Use an oscilloscope to test if the frequency of the clock on the

	microcontroller exceeds 8MHz f) Check that microcontroller supports 8 GPIO.
3. LED Cube a) No shorts or non-conducting connections in the cube. b) The diodes in the RGB LEDs used should emit at least 1000 mcd of brightness each at 3VDC 20mA. c) When control signal is sent from the controller module the LEDs light up with the correct color and brightness. d) Shift registers and LED drivers shift in data at a clock speed of at least 8MHz. e) Shift registers and LED drivers should have an output delay of less than 100µs f) Decoupling capacitors placed at the power input to DIP chips must keep a voltage of 5±1 across their terminals while the cube is displaying an animation or image.	a) Test each LED individually before soldering into a cube, test columns before put into cube, test planes before put into cube, then use a test input of the controller and observe if the entire cube lights up. Testing is done by providing 3VDC 20mA to each diode individually and confirming that the tested diode turns on and other diodes do not. b) Use light meter to measure brightness of diodes being driven with 3VDC 20mA. c) Use test inputs from the controller to observe if the LED cube demonstrates the correct color and brightness output according to the user input. d) Test different clock speeds for the serial data line to shift into the cube to ensure that it continues to correctly shift in data with at least an 8MHz clock. e) Use an oscilloscope on inputs and outputs to the shift registers and LED drivers to make sure that the propagation delay on the output of less than 100µs f) Probe the voltage across the decoupling capacitors with an oscilloscope to ensure their voltages are within the requirement.
4. Communication a) Bluetooth serial communication is received on the Arduino at the baud rate of 9600. b) No errors occur in the received data when the Bluetooth module loses connection or power. c) Level shifter outputs at least 2V for a high voltage and less than 0.8 for a low voltage. d) Level shifter has a propagation delay of less than 100ns	a) Check if the input is received by sending out test inputs from the Android Device, by using the app. “Bluetooth SPP”. b) Confirm the operation of the microcontroller continues without errors—i.e. unexpected microcontroller behavior—when the Bluetooth module loses connection. c) Input 5VDC and 0VDC and check that the output of the level shifter using voltmeter. d) Use oscilloscope to ensure that input to

	output propagation delay is less than 100ns
5. Sensor Array a) The IR rangefinder outputs distinct voltages for distances between 10cm and 50cm in increments of 2.5cm. b) Light frequency senor able to output sensor data that reflects the light intensity.	a) With an input voltage of 5±1V to the sensor, move a white box or sheet of paper along the distance measurement axis and at points 2.5cm apart from between 10cm and 50cm check that the voltages are distinct by using a multi-meter on the output. Also measure the output voltage with the multi-meter and record these values along with their corresponding distance to determine an equation that models the distance-output voltage relationship of the sensor. This equation can later be used by the controller to determine distances of objects. b) Use a ceiling light or lamp above the light sensor and measure the output signal frequency on an oscilloscope to confirm that the frequency of the output signal is linearly proportional to the amount of light between 0.001 and 1000 μW/cm^2 with at least 85% confidence interval.
6. Android Device a) Able to connect and transmit Bluetooth signals to the Bluetooth transceiver in the communication block.	a) Check if the communication block receives the signal by using an LED on the Arduino. If the signal is received the LED goes on.

Appendix D: Android Application Main Activity code

```
package com.example.led_cube;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

import android.os.Bundle;
import android.os.Handler;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.util.Log;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity {

        private BluetoothAdapter btAdapter = null;
        private BluetoothSocket btSocket = null;
        private OutputStream mmOutStream = null;
        private static final String TAG = "MainActivity";

        //UUID
        private static final UUID SPP_UUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");

        //Defined the MAC ADDRESS for our device
        private static String mac_address = "20:13:09:30:09:71";

        //OnClickListener for the two buttons
   public void onMyButtonClick1(View view){
        Toast.makeText(this, "Button 1 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("0");
   }//End of onMyButtonClick1
   public void onMyButtonClick2(View view){
        Toast.makeText(this, "Button 2 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("1");
   }//End of onMyButtonClick2

   public void onMyButtonClick3(View view){
        //Toast.makeText(this, "Button 3 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("f");
   }//End of onMyButtonClick3

   public void onMyButtonClick4(View view){
        //Toast.makeText(this, "Button 4 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("b");
   }//End of onMyButtonClick4
```

```
public void onMyButtonClick5(View view){
        //Toast.makeText(this, "Button 5 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("r");
}//End of onMyButtonClick5

public void onMyButtonClick6(View view){
        //Toast.makeText(this, "Button 6 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("l");
}//End of onMyButtonClick6

public void onMyButtonClick7(View view){
        //Toast.makeText(this, "Button 7 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("d");
}//End of onMyButtonClick7

public void onMyButtonClick8(View view){
        //Toast.makeText(this, "Button 8 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("u");
}//End of onMyButtonClick8

public void onMyButtonClick9(View view){
        Toast.makeText(this, "Button 9 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("2");
}//ENd of onMyBUttonClick9

public void onMyButtonClick10(View view){
        Toast.makeText(this, "Button 10 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("3");
}//ENd of onMyBUttonClick10

public void onMyButtonClick11(View view){
        Toast.makeText(this, "Button 11 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("4");
}//ENd of onMyBUttonClick10

public void onMyButtonClick12(View view){
        Toast.makeText(this, "Button 12 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("5");
}//ENd of onMyBUttonClick12

public void onMyButtonClick13(View view){
        Toast.makeText(this, "Button 12 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("6");
}//ENd of onMyBUttonClick12

public void onMyButtonClick14(View view){
        Toast.makeText(this, "Button 12 Clicked", Toast.LENGTH_SHORT).show();
        SPP_Write("7");
}//ENd of onMyBUttonClick12
```

```
   public void onMyButtonClick15(View view){
         Toast.makeText(this, "Button 12 Clicked", Toast.LENGTH_SHORT).show();
         SPP_Write("8");
   }//ENd of onMyBUttonClick12



   @Override
   //On Create Method
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      btAdapter = BluetoothAdapter.getDefaultAdapter(); //Get a Bluetooth Adapter to start the
BT process
      //Check the Bluetooth connection
      if(btAdapter.isEnabled()){
         Log.d(TAG,"Bluetooth is Enabled");
      }
      else{
         //Else Ask User to Turn on Bluetooth
         Intent enableBtIntent  = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
         startActivityForResult(enableBtIntent , 1);
      }
      Log.d(TAG,"onCreate Ended");
   }//End of onCreate
   @Override
   public void onResume() {
    super.onResume();
    //Use the btAdapter to get a device, using the mac address
    BluetoothDevice device = btAdapter.getRemoteDevice(mac_address); //Obtained a BT device
    Log.d(TAG,"BluetoothDevice Got");
    //Next Use this BT Device to get a BT Socket
    try {
          btSocket = device.createRfcommSocketToServiceRecord(SPP_UUID);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                Log.e(TAG,"Socket Failed, Exit",e);
                finish();
        }

        btAdapter.cancelDiscovery(); //Must Cancel Discovery before connecting, according to
android website.
        //Next Connect to the btsocket
        Log.d(TAG,"Before Connect to Socket");
        try {
                        btSocket.connect();
                        Log.d(TAG,"Before Connect to Socket");
                } catch (IOException e) {
                    try {
                       btSocket.close();//Close btSocket if not connected
                     } catch (IOException e2) {
```

```
                        Log.d(TAG,"Cannot Connect");
                        finish();//Failed Connection close the program
                      }
                }
    //Once the btSocket is connected, now create an output stream for communication
        try {
                        mmOutStream = btSocket.getOutputStream();
                } catch (IOException e) {
                        finish();//Unable to get outstream APP closes
                }
   }//End of onResume
   @Override
   public void onPause() {
    super.onPause();

    Log.d(TAG, "In onPause()");

    if (mmOutStream != null) {
     try {
      mmOutStream.flush();
     } catch (IOException e) {
      finish();
     }
    }
    try     {
     btSocket.close();
    } catch (IOException e2) {
     finish();
    }
   }//End of onPause
   private void SPP_Write(String message){
        byte [] byte_msg = message.getBytes(); //the write for outsteam takes btye.
        Log.d(TAG,"SPP_Write");
        try {
                        mmOutStream.write(byte_msg);
                } catch (IOException e) {
                        Log.d(TAG,"Fail to write SPP");
                        finish();
                }
   }

}//End of Activity
```

Appendix E: Project Pictures

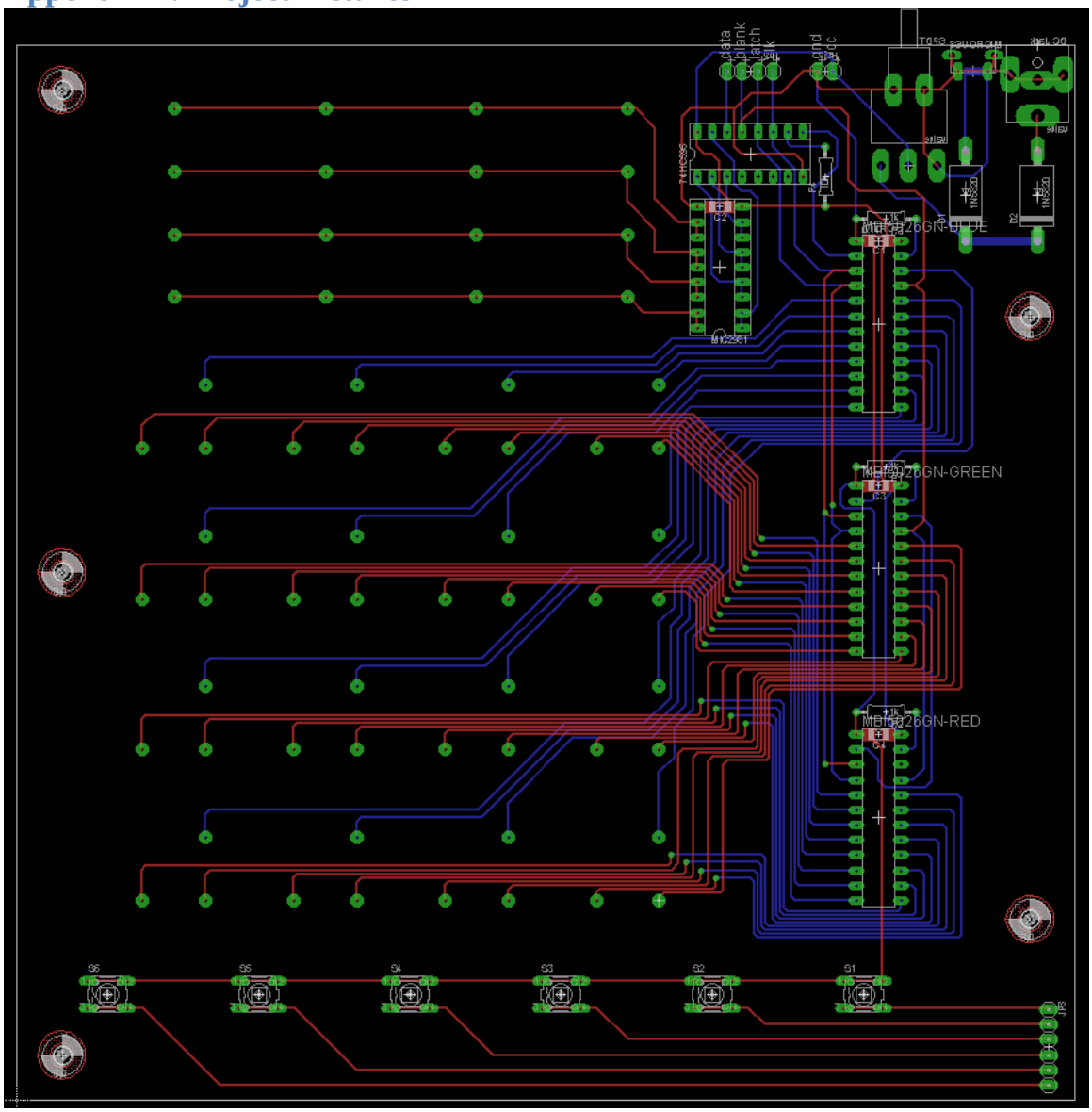


Figure E.1: Top Printed Circuit Board with Components

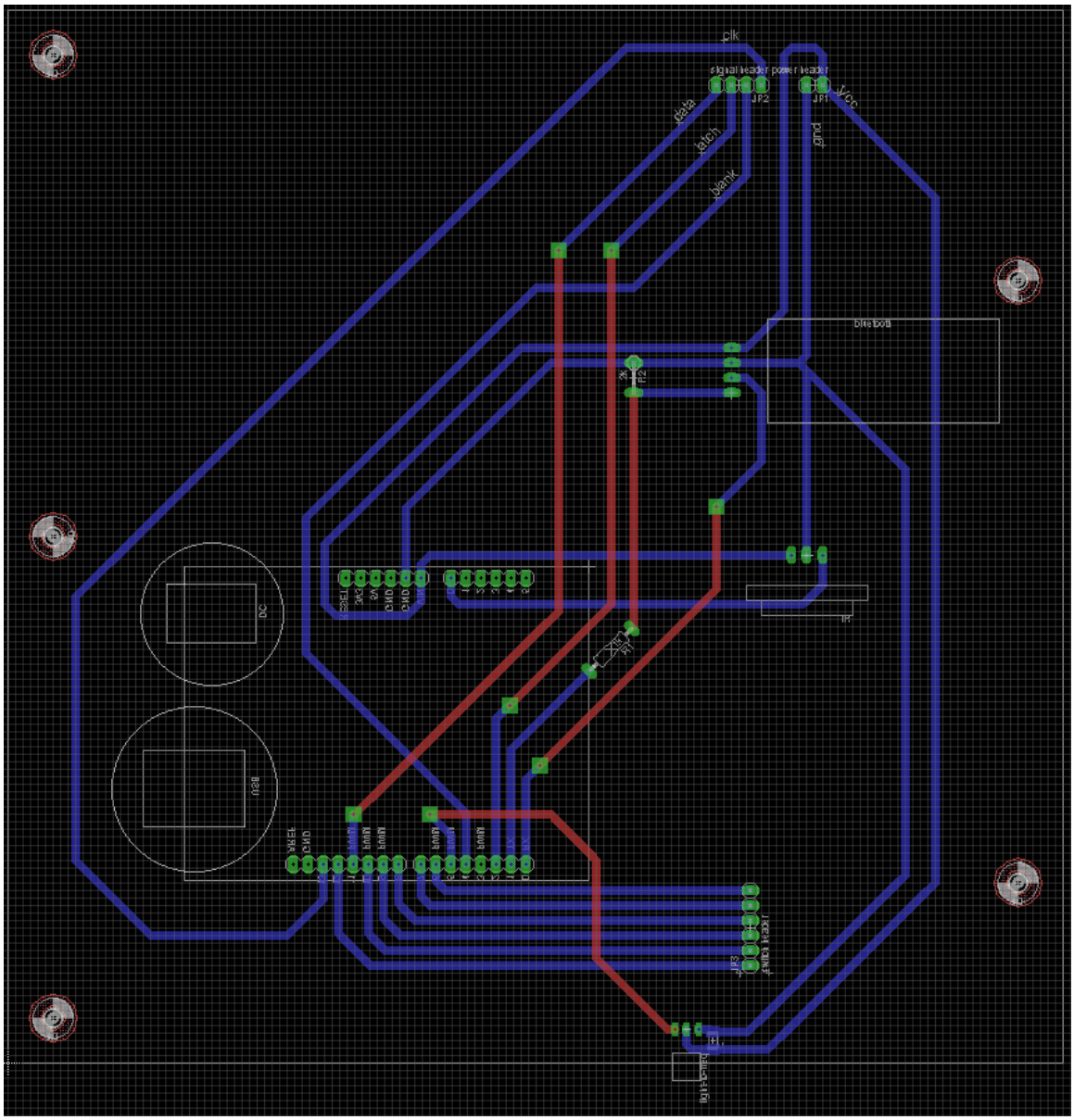


Figure E.2: Bottom Printed Circuit Board with Components

Figure E.3: Soldering LED Layer

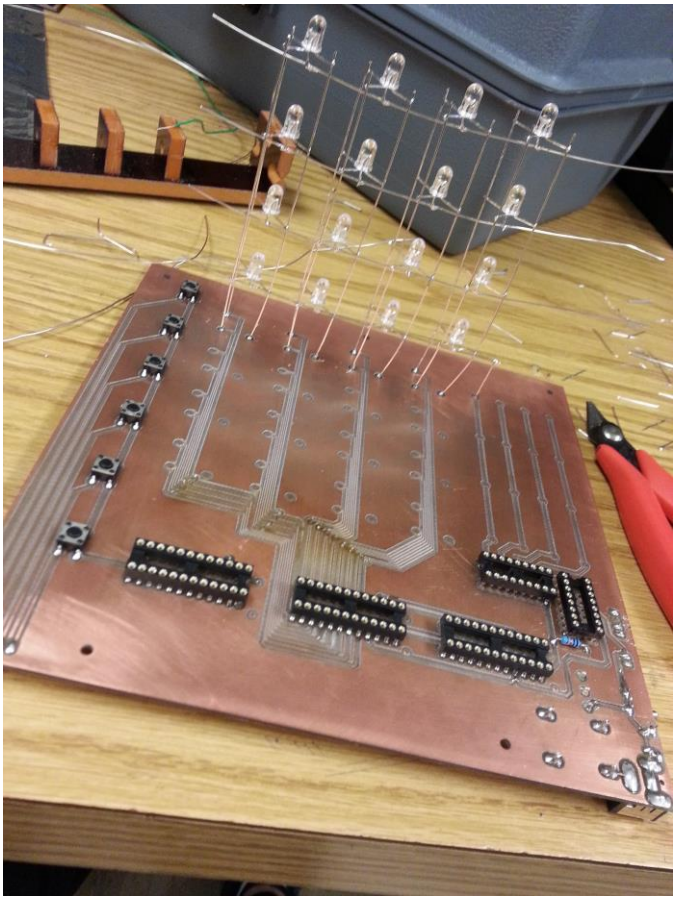


Figure E.4: Soldering LED cube onto top PCB

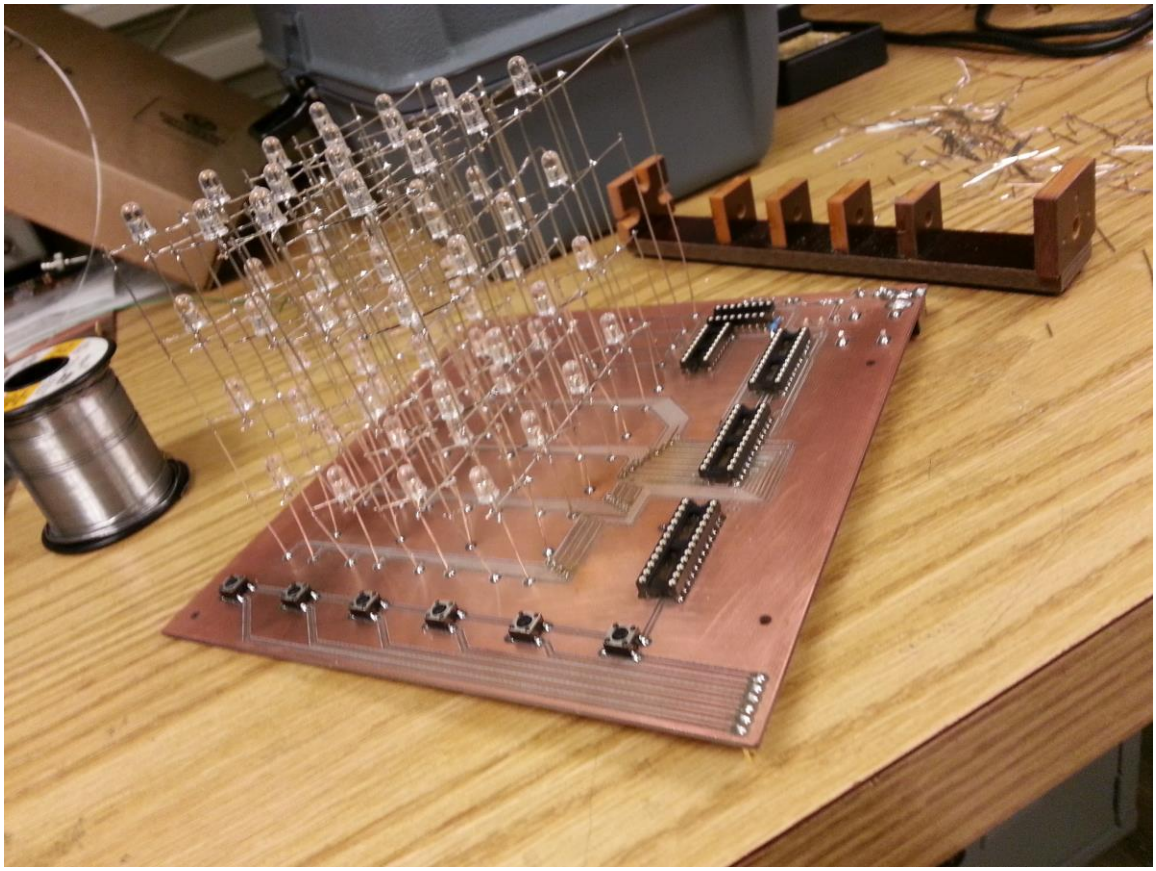


Figure E.5: Complete Soldering LED Cube

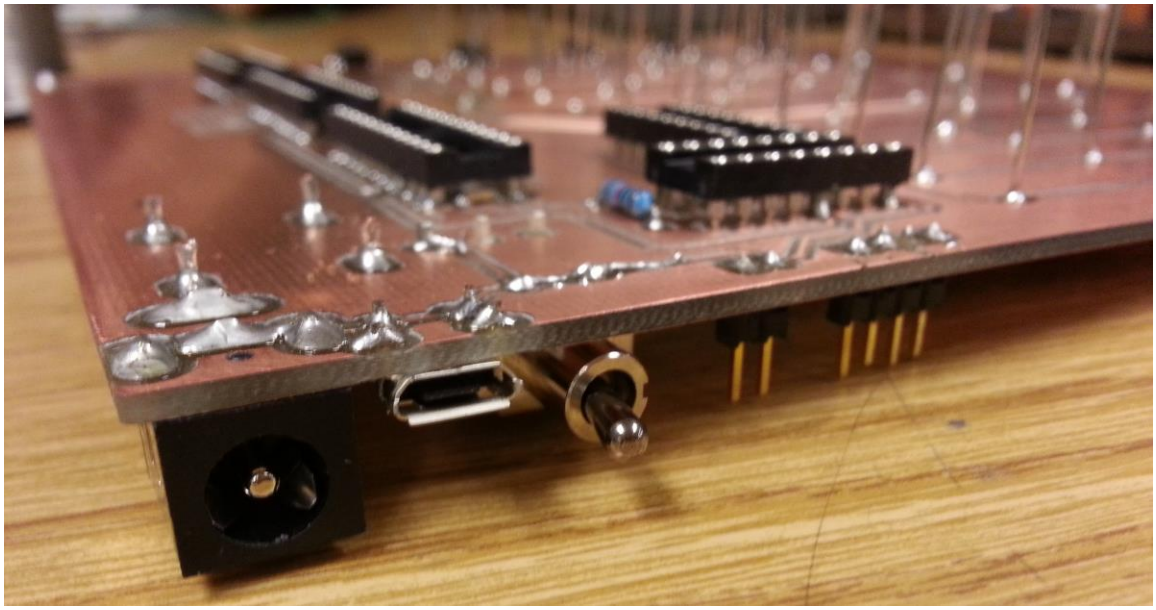


Figure E.6: Top PCB close-up – power input.

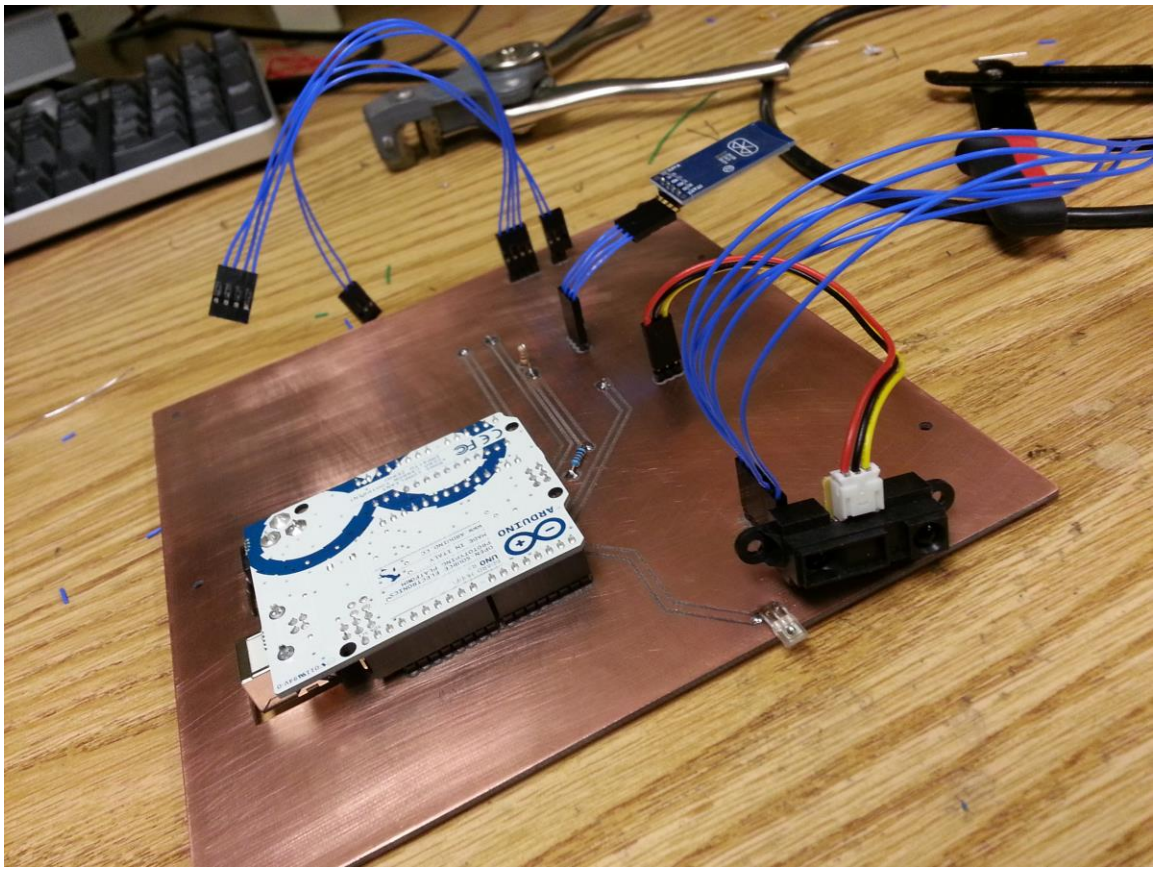


Figure E.7: Bottom PCB - Jumpers, Bluetooth, Sensors, Arduino

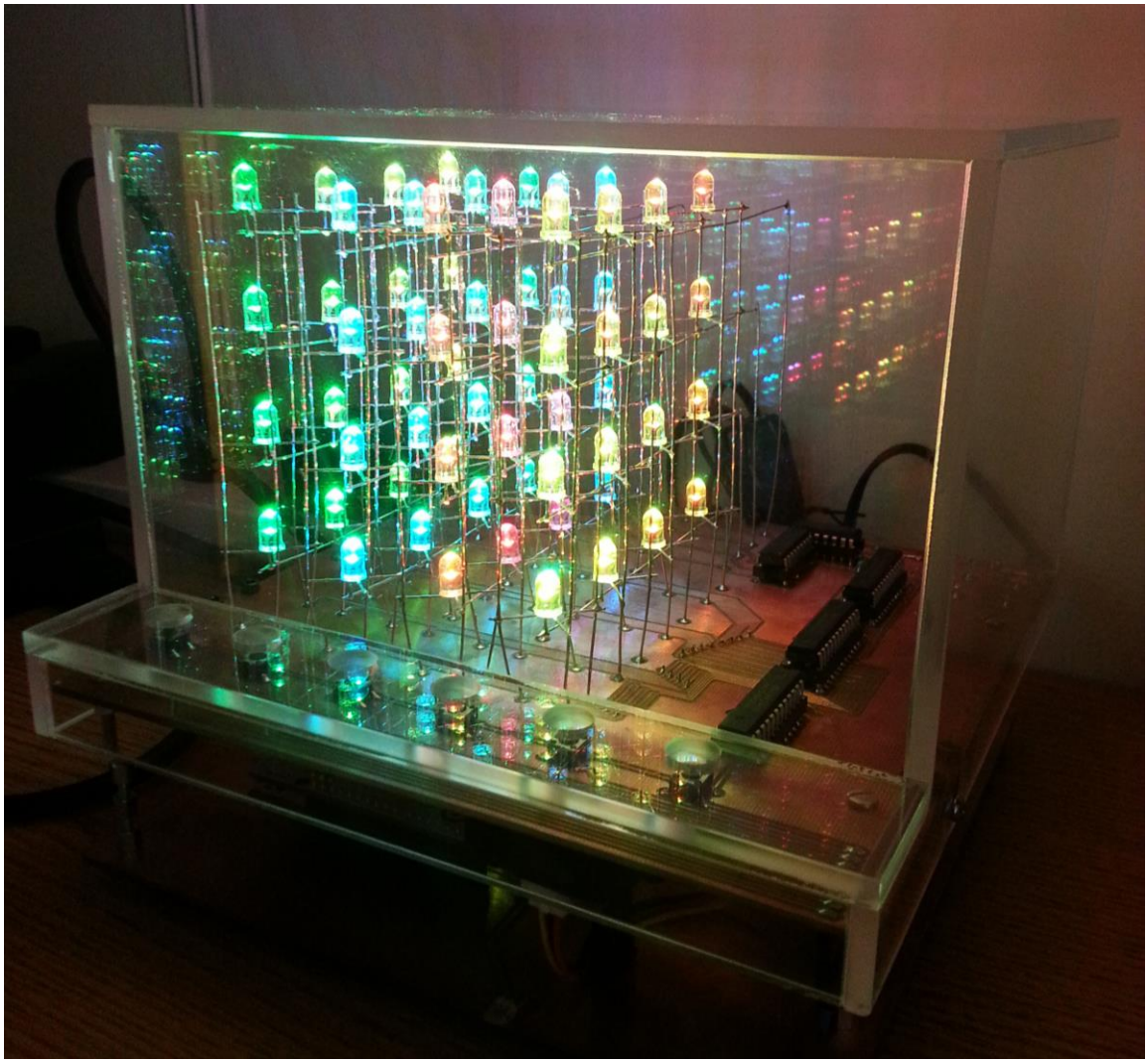


Figure E.8: Final Product

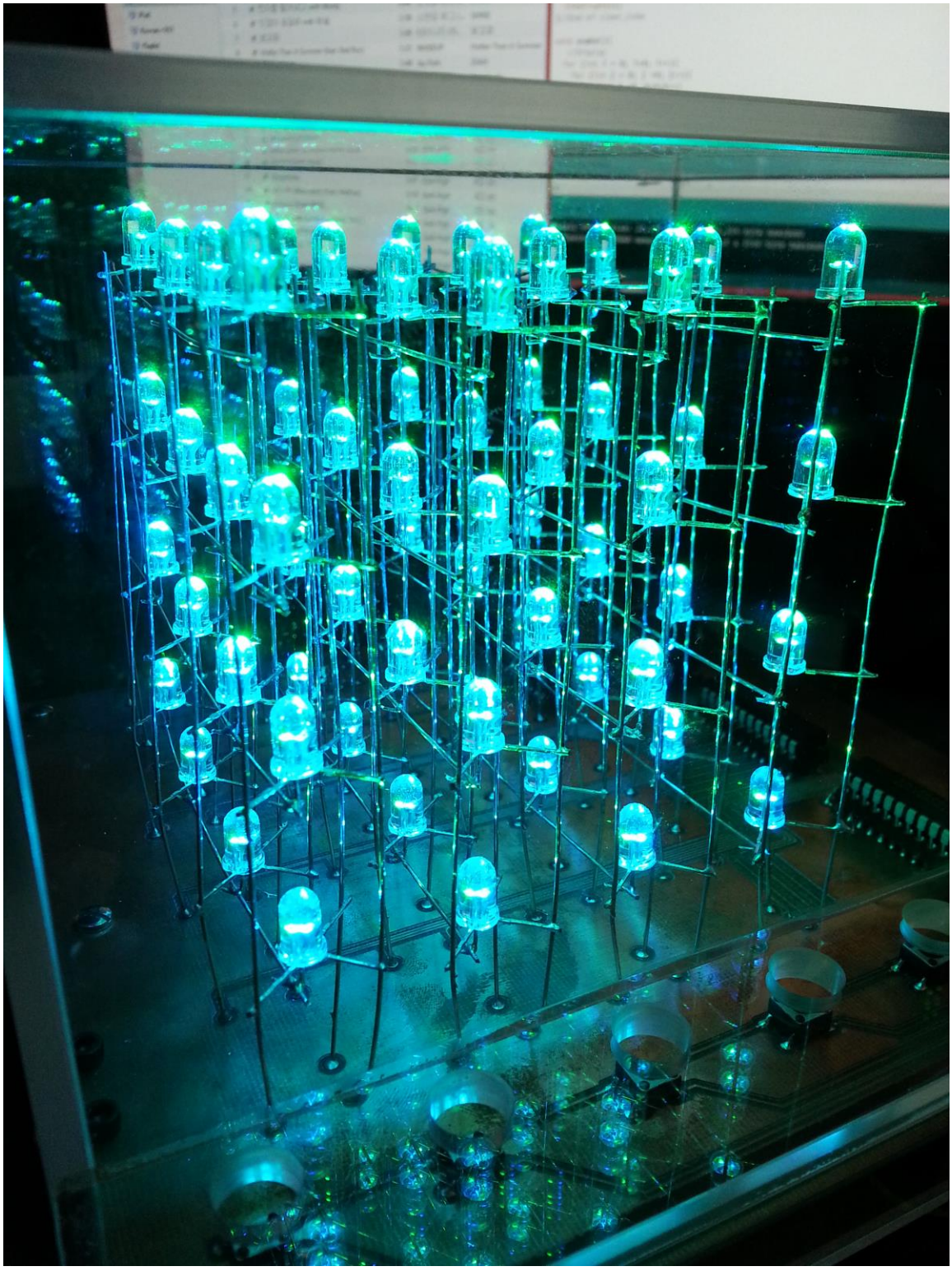


Figure E.9: Final Product-Close-up