Design Document

Tweedle-Dee and Tweedle-Dum

Team B08: Sean Nima, James Joh, Mac Murray, Michael Parrish

Date: 09/25/2022

Note: Content in this document is subject to change. Changes will be updated in the Revision Record on Page 3.

Table of Contents

Revision Record

Date	Author	Comments
Sept 9th, 2022	Team	Document Created (System Design)
Sept 24th, 2022	Team	Added Hierarchal Design
Oct 15th, 2022	Team	Added Software Simulation
Oct 15th, 2022	Team	Added Mechanical Design
November 10th, 2022	Team	Finalized Design Document

Project Description

We intend to design a character Tweedle-Dee and Tweedle-Dum character box from Alice in Wonderland. The box will be designed to fit the theme of the characters dancing. They will be on a “stage:” with lights shining on them as they rotate around. The characters are intended to move as sound is playing and then return to their original positions when they finish their “performance”.

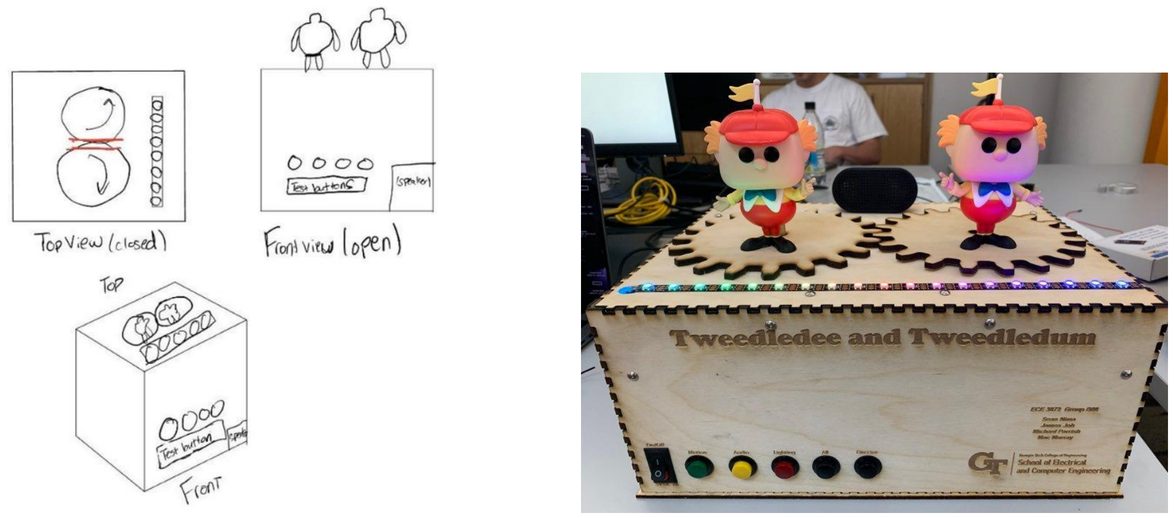


Figure 1: Initial and Final Design of the Tweedle-Dee & Tweedle-Dum Box

System Design

The inputs of the system currently include 120V AC input power, a PWM (pulse-width modulator) audio source, and user input. The outputs of this system include a speaker and the movement produced by a figurine rotating through a system of gears. The system of gears will rotate through the PWM control of a motor. The system consists of five sub-systems including the control, power, motion, lights, and audio. We are searching to include a 120V AC to 5V DC adapter and a switch for our power system. For our control, we are looking to use a Raspberry Pi Zero 2 W. The audio will consist of an EMB-3008A speaker. The lights will be using a WS2812B LED strip. The motion control will be done using a DG01D motor.

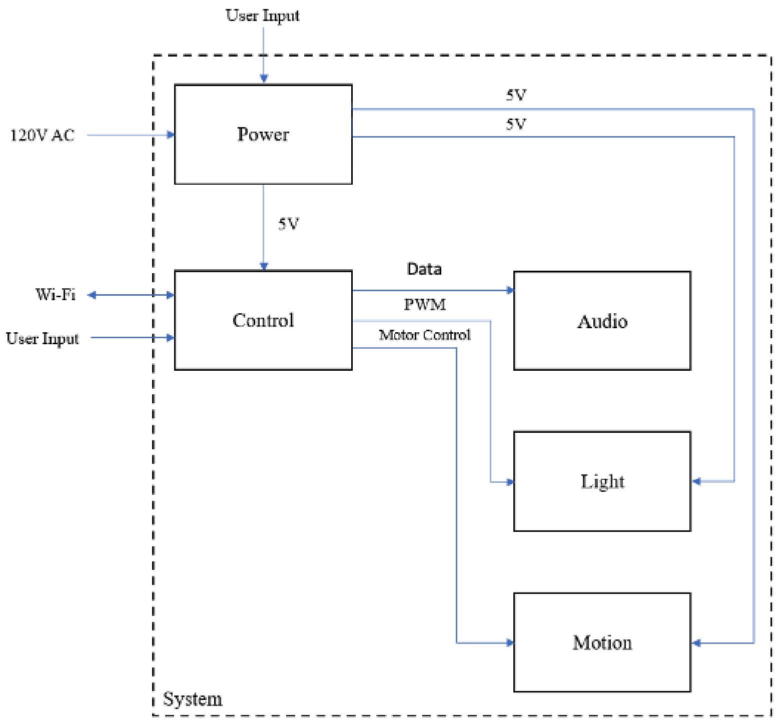
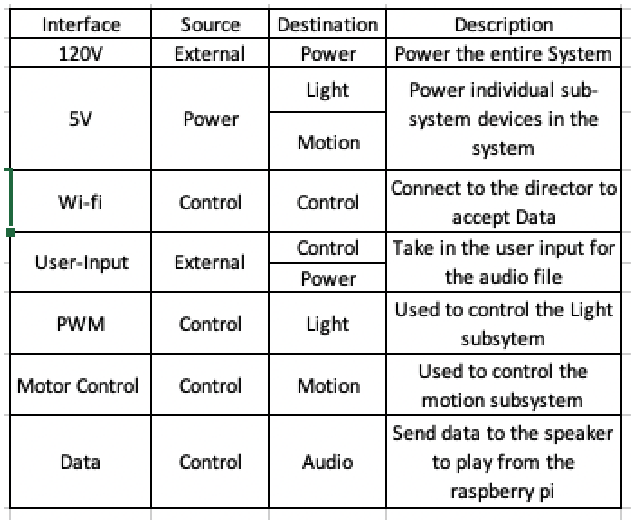


Figure 2: System Diagram



Interface	Source	Destination	Description
120V	External	Power	Power the entire System
5V	Power	Light	Power individual sub-system devices in the system
		Motion	
Wi-fi	Control	Control	Connect to the director to accept Data
User-Input	External	Control	Take in the user input for the audio file
		Power	
PWM	Control	Light	Used to control the Light subsytem
Motor Control	Control	Motion	Used to control the motion subsystem
Data	Control	Audio	Send data to the speaker to play from the raspberry pi

Table 1: Sub-system inputs and outputs

Sub-System Designs

Our design incorporates five sub-systems: power, control, motion, lights, and audio. The controller will convert the text provided by the user into a mp4 file in which the audio system will output. The motor and lights will be controlled via the controller through signals.

Control Sub-System

The control system will be powered through 5V power passed from the power source. It will be responsible for communicating control systems via output pins to the motion, lights, and audio sub-systems. A Raspberry Pi Zero 2W will be the controller in use. The Raspberry Pi will communicate with buttons and user input.

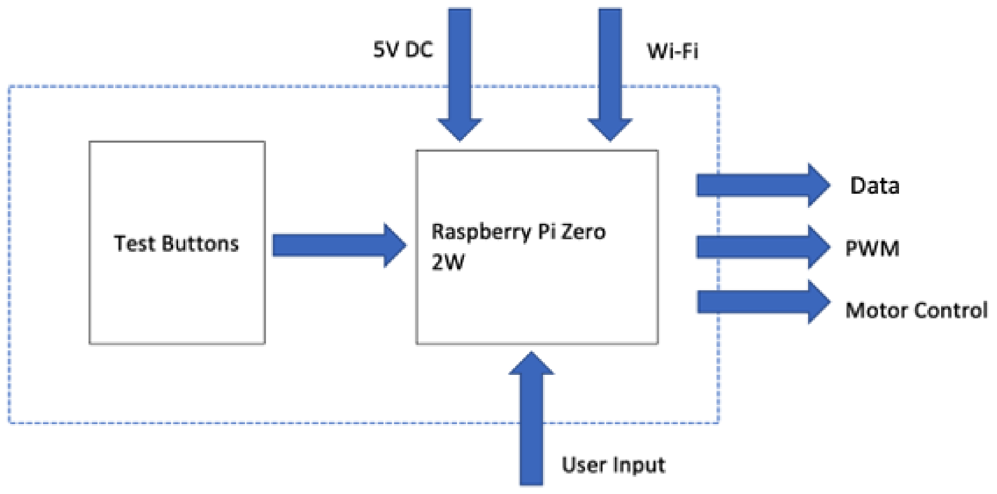


Figure 3: User Interface Sub-system

Audio Sub-System

The audio system will take data in from the Raspberry Pi to output. This data will be determined by the user.

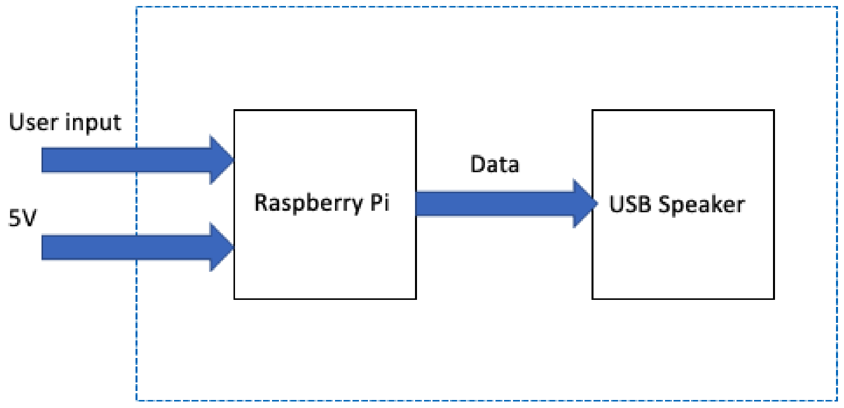


Figure 5: Audio Sub-System

Motion Sub-System

The motion subsystem will take in power and a signal from the control system into the motor driver. The motor driver will then communicate with the motor in order to make the gears spin.

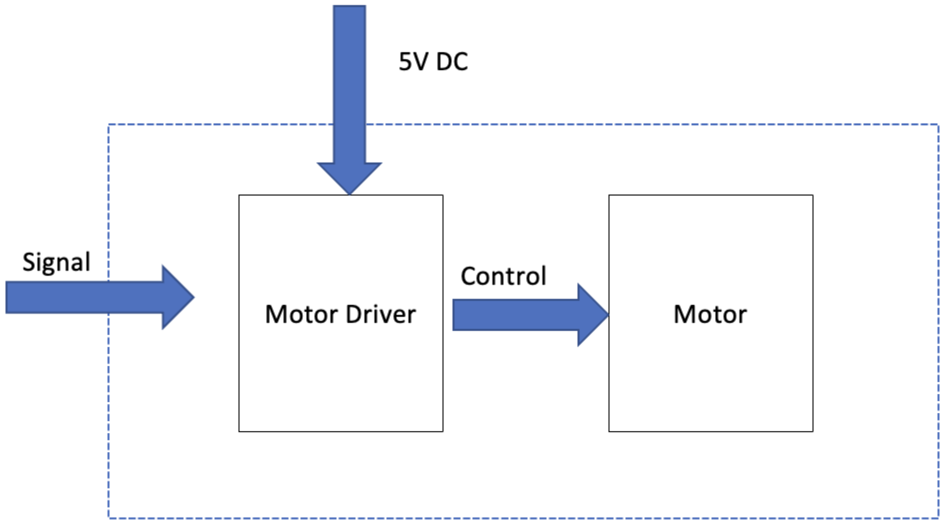


Figure 6: Motion Sub-System

Power Sub-System

The power subsystem will take 120 Volts AC and distribute 5V DC to all other sub-systems that are in this design. The power will go through a switch to determine whether the system should be on or off and a fuse for safety reasons.

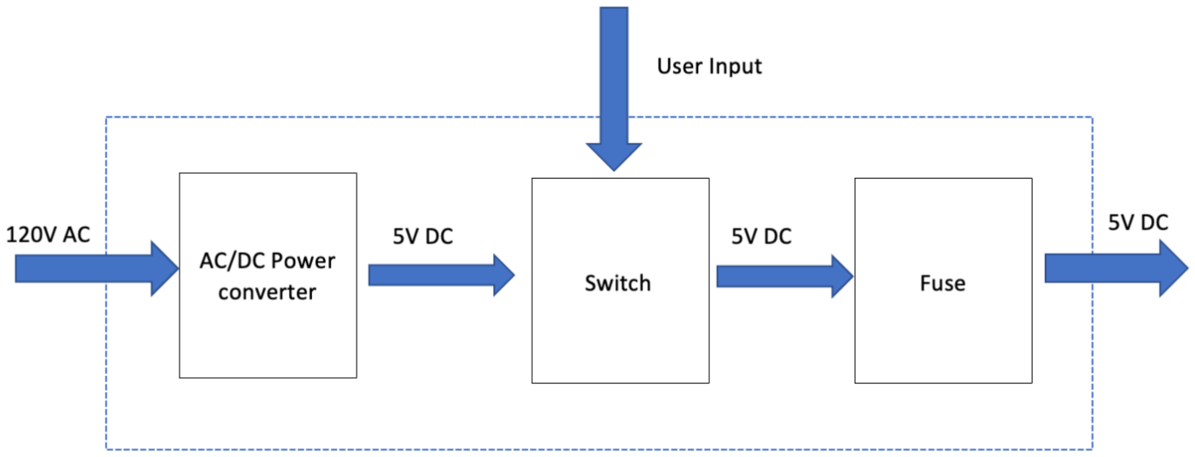


Figure 7: Power Sub-System

Lights Sub-System

The lights subsystem will consist of a WS2812B Led strip only. The LED strip will communicate directly from the controller to perform functions and be powered by the power sub-system.

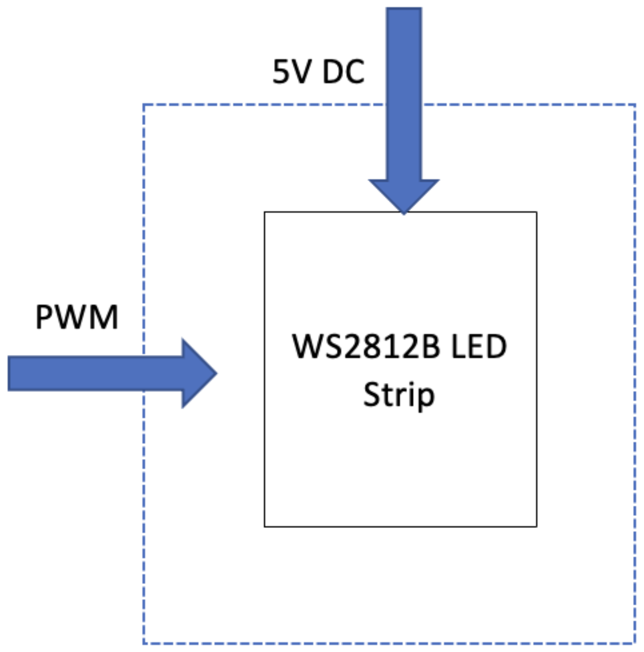
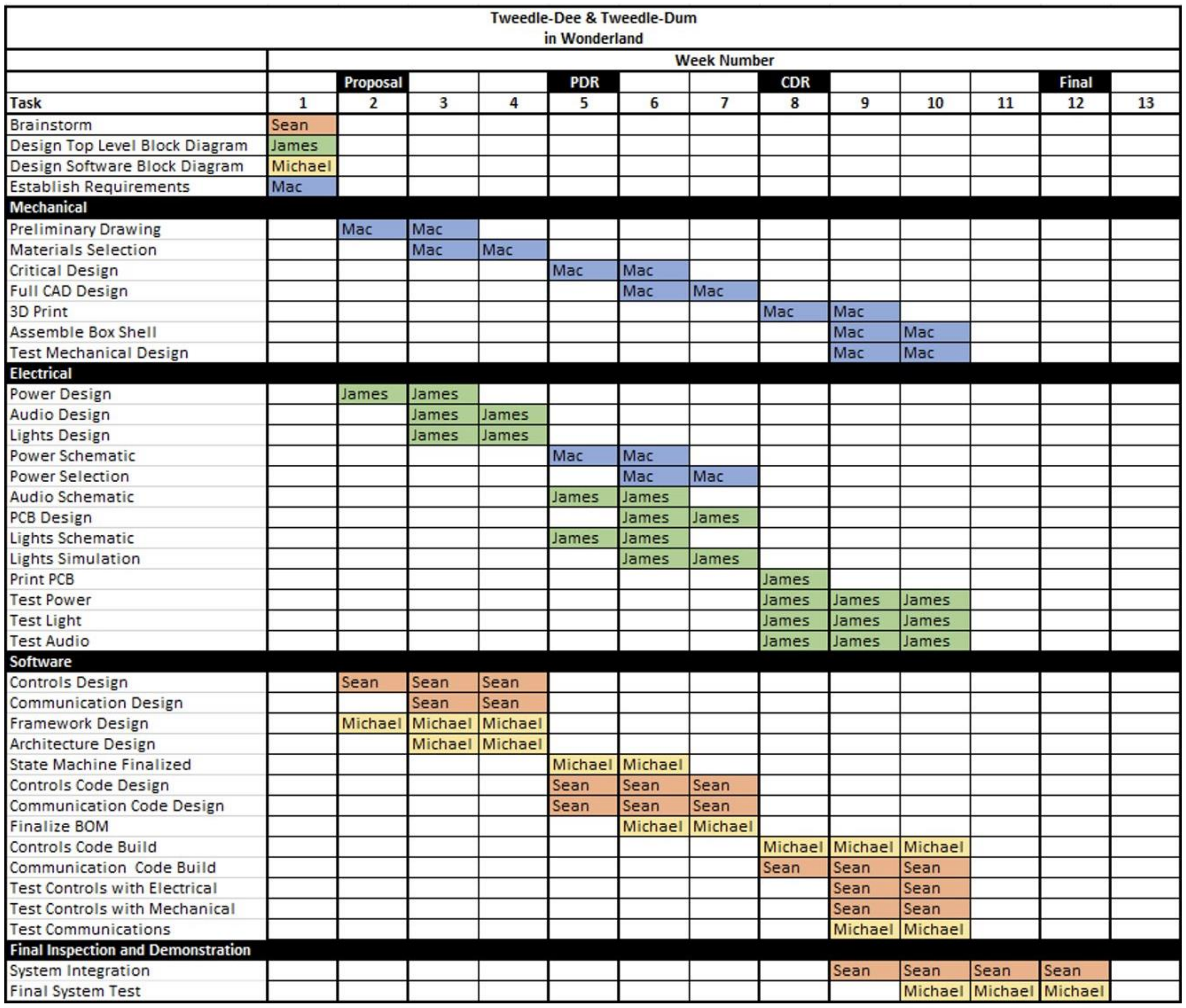


Figure 8: Lights Sub-System

Schedule

The schedule for completion of the project is shown in Table 2 where the task lead is indicated on each task.



Tweedle-Dee & Tweedle-Dum in Wonderland													
	Week Number												
		Proposal			PDR			CDR				Final	
Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Brainstorm	Sean												
Design Top Level Block Diagram	James												
Design Software Block Diagram	Michael												
Establish Requirements	Mac												
Mechanical													
Preliminary Drawing		Mac	Mac										
Materials Selection			Mac	Mac									
Critical Design					Mac	Mac							
Full CAD Design						Mac	Mac						
3D Print								Mac	Mac				
Assemble Box Shell									Mac	Mac			
Test Mechanical Design									Mac	Mac			
Electrical													
Power Design		James	James										
Audio Design			James	James									
Lights Design			James	James									
Power Schematic					Mac	Mac							
Power Selection						Mac	Mac						
Audio Schematic					James	James							
PCB Design						James	James						
Lights Schematic					James	James							
Lights Simulation						James	James						
Print PCB								James					
Test Power								James	James	James			
Test Light								James	James	James			
Test Audio								James	James	James			
Software													
Controls Design		Sean	Sean	Sean									
Communication Design			Sean	Sean									
Framework Design		Michael	Michael	Michael									
Architecture Design			Michael	Michael									
State Machine Finalized					Michael	Michael							
Controls Code Design					Sean	Sean	Sean						
Communication Code Design					Sean	Sean	Sean						
Finalize BOM						Michael	Michael						
Controls Code Build								Michael	Michael	Michael			
Communication Code Build								Sean	Sean	Sean			
Test Controls with Electrical									Sean	Sean			
Test Controls with Mechanical									Sean	Sean			
Test Communications									Michael	Michael			
Final Inspection and Demonstration													
System Integration									Sean	Sean	Sean	Sean	
Final System Test										Michael	Michael	Michael	

Table 2: Schedule

Software Design

The software design will be created in the form of a state machine. There are multiple states as shown below in Figure 9, but the main two states are Setup and IDLE. Setup is responsible for initializing the design and ensure all communication between devices on board are working properly. The IDLE state is responsible for waiting for a function to be called, whether it's a test or a director call, then performing the action that is requested. More on the framework is shown in Figure 10.

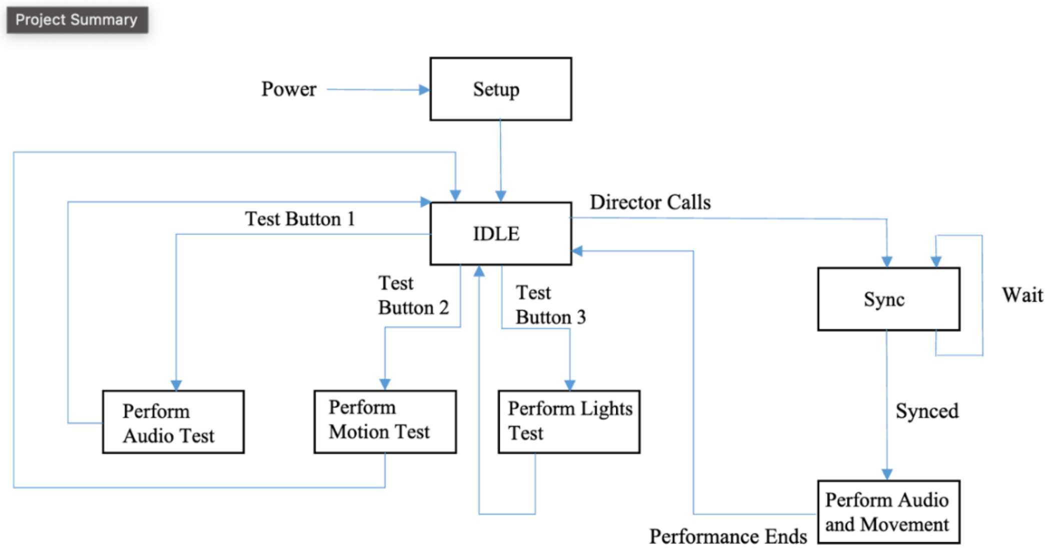


Figure 9: Software Design

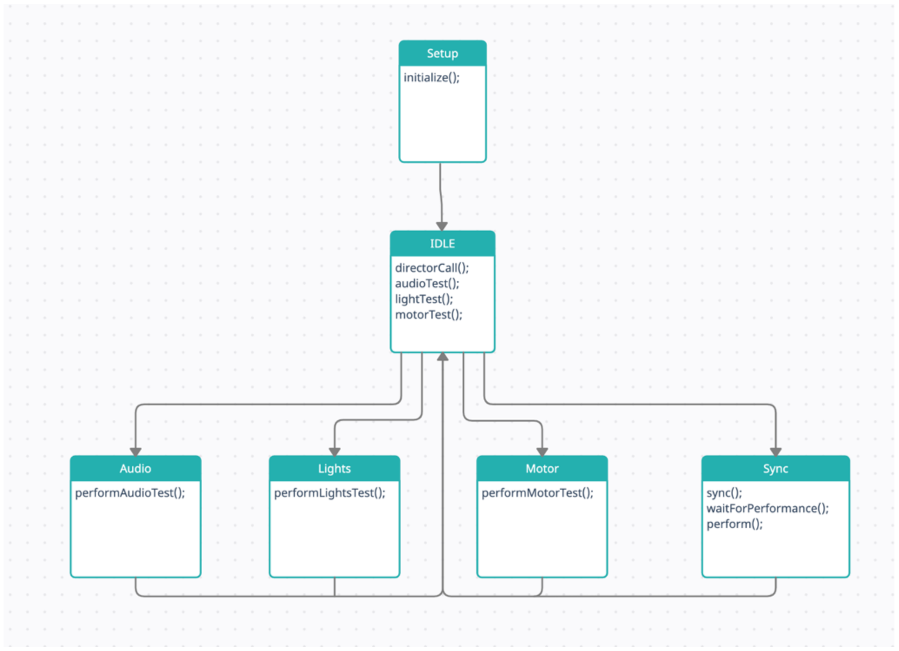
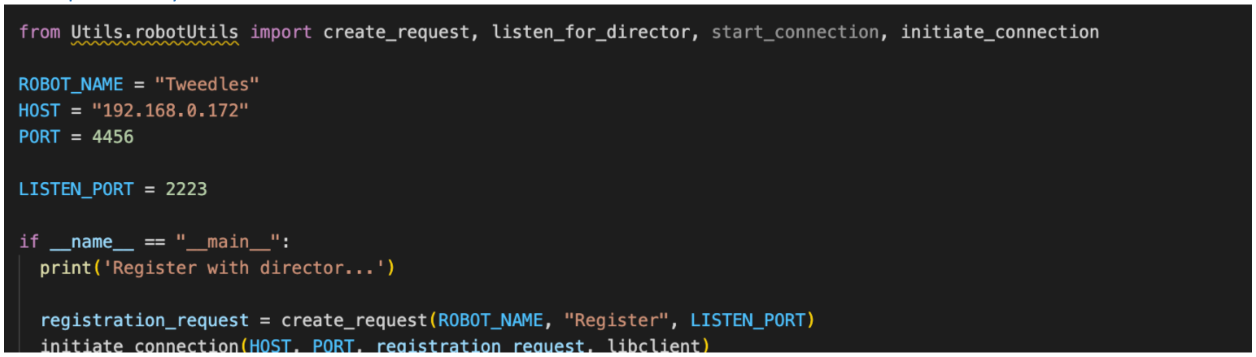


Figure 10: Software Framework

Software Simulation

In the software simulation, we tested all the coding functionalities to run error free on the raspberry pi. We were looking to run the code error free and see if the code operates properly.

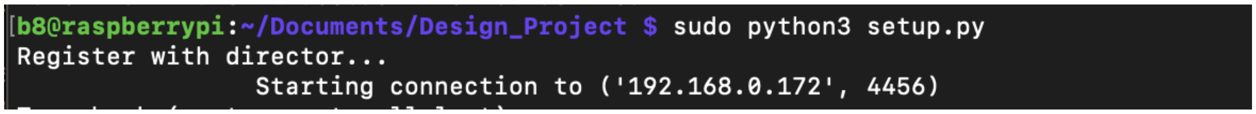
Setup Sub-system



```python
from Utils.robotUtils import create_request, listen_for_director, start_connection, initiate_connection

ROBOT_NAME = "Tweedles"
HOST = "192.168.0.172"
PORT = 4456

LISTEN_PORT = 2223

if __name__ == "__main__":
    print('Register with director...')

    registration_request = create_request(ROBOT_NAME, "Register", LISTEN_PORT)
    initiate_connection(HOST, PORT, registration_request, libclient)
```

Figure 11: Setup Test Code

Figure 11 shows code that is supposed to connect the sub-system to the director online. We expected the code to look for the director on a given IP address. We wanted to make sure this code operated properly and can at least search for the director right now.

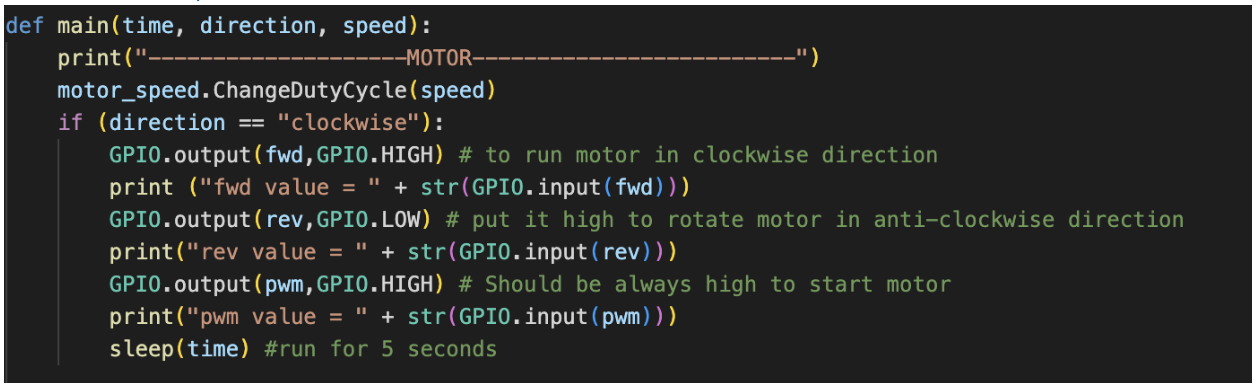


```
[b8@raspberrypi:~/Documents/Design_Project $ sudo python3 setup.py
Register with director...
                    Starting connection to ('192.168.0.172', 4456)
```

Figure 12: Setup Test

The result operated as expected as shown in figure 12. It started to search for the director on the expected IP address that the director would have been located on during this testing.

Motion Sub-System

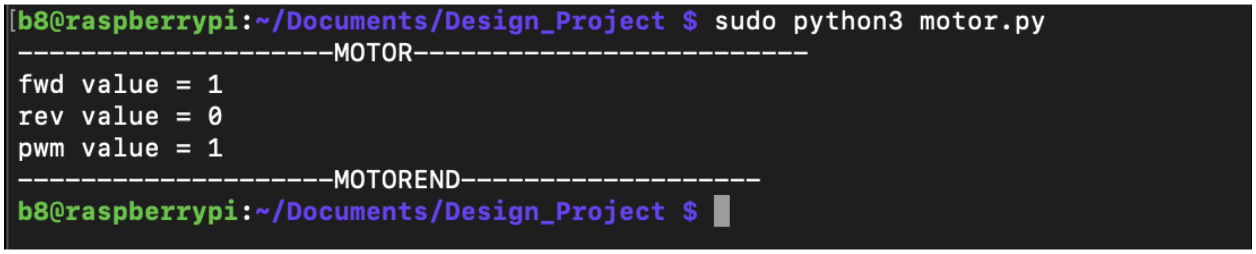


```python
def main(time, direction, speed):
    print("--------------------MOTOR-------------------------")
    motor_speed.ChangeDutyCycle(speed)
    if (direction == "clockwise"):
        GPIO.output(fwd,GPIO.HIGH) # to run motor in clockwise direction
        print ("fwd value = " + str(GPIO.input(fwd)))
        GPIO.output(rev,GPIO.LOW) # put it high to rotate motor in anti-clockwise direction
        print("rev value = " + str(GPIO.input(rev)))
        GPIO.output(pwm,GPIO.HIGH) # Should be always high to start motor
        print("pwm value = " + str(GPIO.input(pwm)))
        sleep(time) #run for 5 seconds
```

Figure 13: Motor Test Code

Figure 13 shows the main function which takes in the time that the motor is supposed to run, the direction the motor should spin in, and at what speed the motor should spin for. Since we are running the motor using a PWM control loop, we have 3 pins: a fwd, rev, and pwm pin. To rotate the motor clockwise, the fwd bit should be high and the rev bit should be low and vice versa to rotate the motor counterclockwise. At any point, the motor should only run if the PWM bit is on.

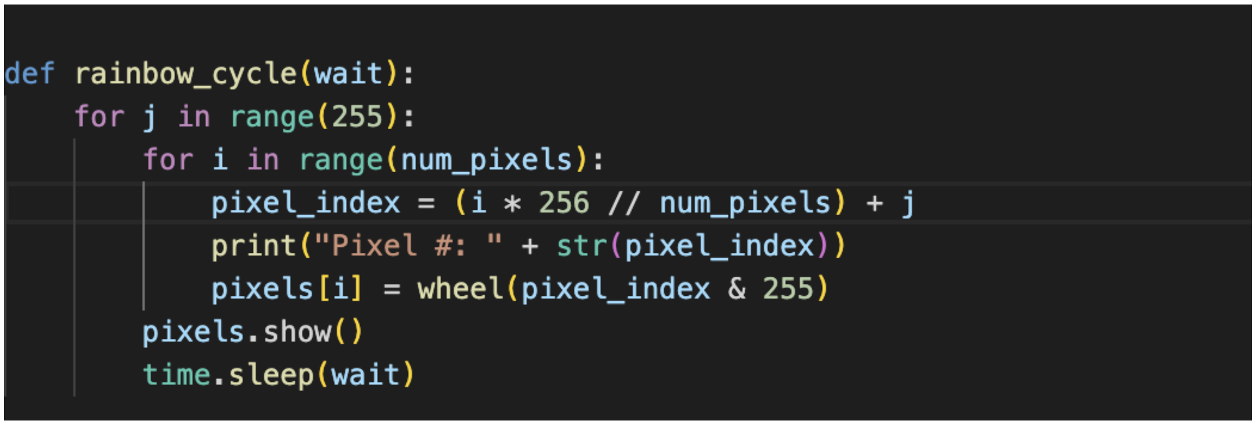
The sub-system was simulated using the function call main(5,”clockwise”, 50). We expected the results to print fwd:1, rev:0, pwm:1 and wait a few seconds before printing the end line. As shown in figure 14, the motor function behaved as expected.



```
b8@raspberrypi:~/Documents/Design_Project $ sudo python3 motor.py
--------------------MOTOR-------------------------
fwd value = 1
rev value = 0
pwm value = 1
--------------------MOTOREND-------------------
b8@raspberrypi:~/Documents/Design_Project $ 
```

Figure 14: Motor Test

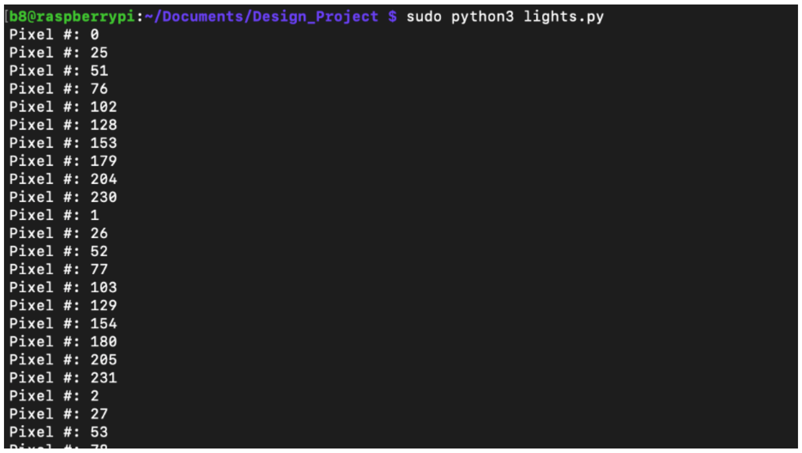
Lights Sub-System



```
def rainbow_cycle(wait):
    for j in range(255):
        for i in range(num_pixels):
            pixel_index = (i * 256 // num_pixels) + j
            print("Pixel #: " + str(pixel_index))
            pixels[i] = wheel(pixel_index & 255)
        pixels.show()
        time.sleep(wait)
```

Figure 15: Lights Test Code

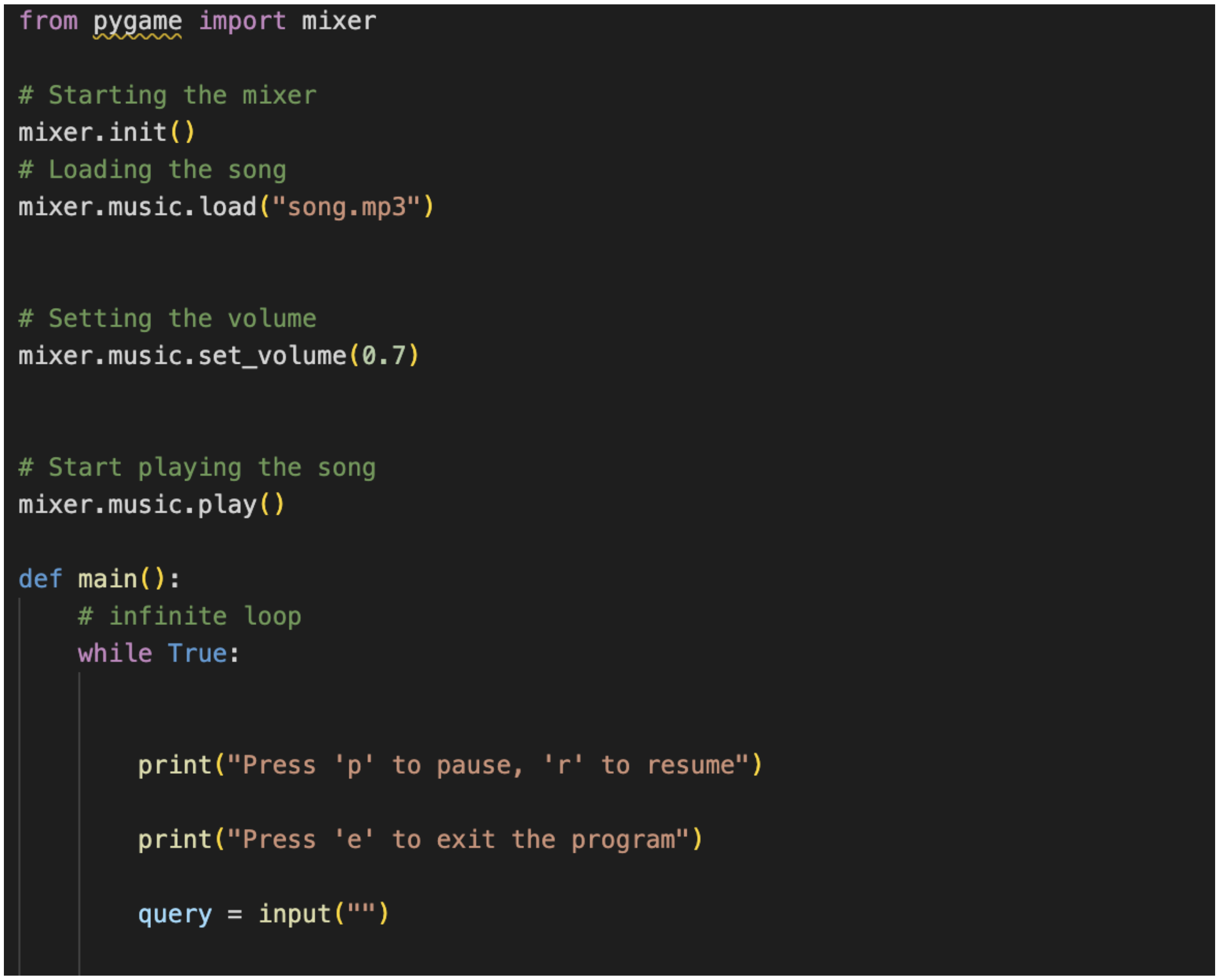
Figure 15 outlines the main testing point of the lights sub-system. It is responsible for finding a pixel_index and producing an RGB value for each pixel. The code is expected to print out the Pixel #: (some random number) in the code to show that the lights hypothetically would be lit up with some value. Figure 16 shows that the result occurred as expected for the lights sub-system.



```
b8@raspberrypi:~/Documents/Design_Project $ sudo python3 lights.py
Pixel #: 0
Pixel #: 25
Pixel #: 51
Pixel #: 76
Pixel #: 102
Pixel #: 128
Pixel #: 153
Pixel #: 179
Pixel #: 204
Pixel #: 230
Pixel #: 1
Pixel #: 26
Pixel #: 52
Pixel #: 77
Pixel #: 103
Pixel #: 129
Pixel #: 154
Pixel #: 180
Pixel #: 205
Pixel #: 231
Pixel #: 2
Pixel #: 27
Pixel #: 53
```

Figure 16: Lights Test

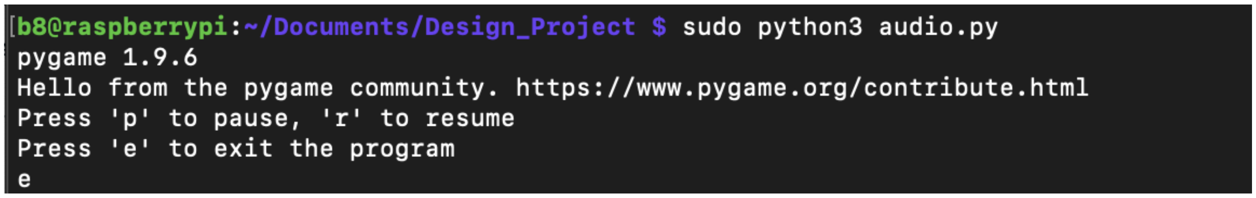
Audio Sub-System



```
from pygame import mixer

# Starting the mixer
mixer.init()
# Loading the song
mixer.music.load("song.mp3")


# Setting the volume
mixer.music.set_volume(0.7)


# Start playing the song
mixer.music.play()

def main():
    # infinite loop
    while True:


        print("Press 'p' to pause, 'r' to resume")

        print("Press 'e' to exit the program")

        query = input("")
```

Figure 17: Audio Test Code

Figure 17 shows the code that was used to test the audio. The code imports the pygame library and is used to play a .mp3 file. For this code, we want it to just register to play an audio clip. The code will have operated properly if we see the text inside the print statements.



```
[b8@raspberrypi:~/Documents/Design_Project $ sudo python3 audio.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Press 'p' to pause, 'r' to resume
Press 'e' to exit the program
e
```

Figure 18: Audio Test

Figure 18 shows the audio test acted as expected. We see the text as expected in the print statements. The pygame and “Hello from the pygame…” appeared as part of the mixer.init() function for library credit.

Contingency Plan

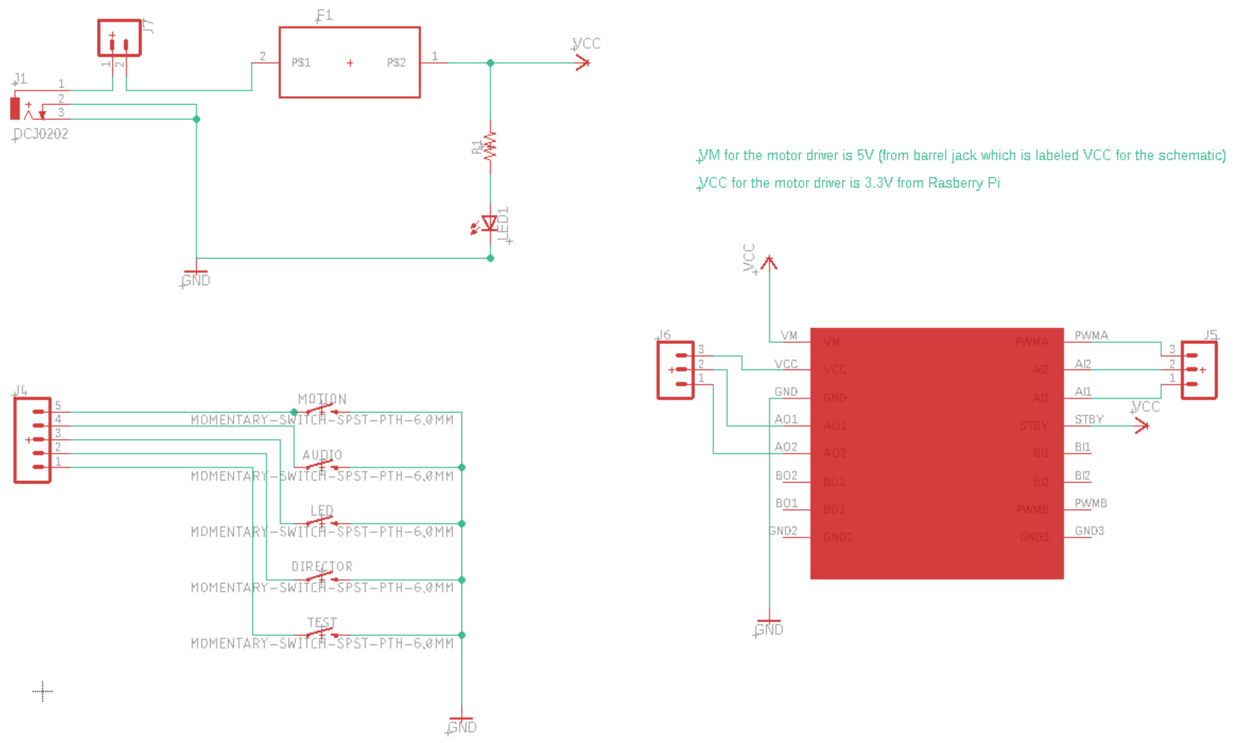
Although all the software behaved as expected, the software has not been integrated with the hardware. We have already prepared a contingency plan for the software in this case. We will end up using an MBED controller to control the audio, motor, and lights sub-sections of the code and a raspberry pi to communicate strictly with the director and MBED. The resulting system hierarchy will be documented and updated only if this contingency occurs.

Note: For our final project, we did not use the contingency plan.

Electrical Design

The electrical design of the robot is centered around a Raspberry Pi Zero W and a custom-built printed circuit board. The printed circuit board hosts the main power source for the system and distributes the 5V to the various subsystems. The 5V is supplied from a 120V AC to 5V DC and 2 Amps wall adapter. The PCB also connects the motor driver to conveniently operate our motor without having to use an unnecessary number of wires. Additionally, it acts as a control board as all 5 test pushbuttons are connected by the PCB. An analog circuit of a simple resistor and LED is included to indicate when the power switch has been turned on or off.

Figure 19 shows the schematic of the PCB designed on EAGLE. A custom part was needed to be created for motor driver and fuse. A Dual TB6612FNG Motor Driver was not able to be found as an already existing EAGLE part so measurements were made to specifically design this part to import as a new library into EAGLE. A custom fuse was designed as well to fit our Littel fuse holder to be soldered onto the PCB.



J7
F1
PS1
PS2
VCC
J1
DCJ0202
R1
LED1
GND
VM for the motor driver is 5V (from barrel jack which is labeled VCC for the schematic)
VCC for the motor driver is 3.3V from Rasberry Pi
J4
MOTION
AUDIO
LED
DIRECTOR
TEST
MOMENTARY-SWITCH-SPST-PTH-6.0MM
J6
J5
VM
VCC
GND
AO1
AO2
BO2
BO1
GND2
PWMA
AI2
AI1
STBY
BI1
BI2
PWMB
GND3

Figure 19: PBC Schematic

Figure 20 is a snapshot of the actual board design our PCB. All the traces were made on the bottom layer of the board so that the connections will be more secure when soldering the pieces. The components will be placed on the front, and the actual connections will be soldered on the back side. A rats-nest feature was used for all the GND connections to be intertwined onto the copper plane to minimize the number of traces. A silkscreen of our team's name and number was included to easily identify our PCB.

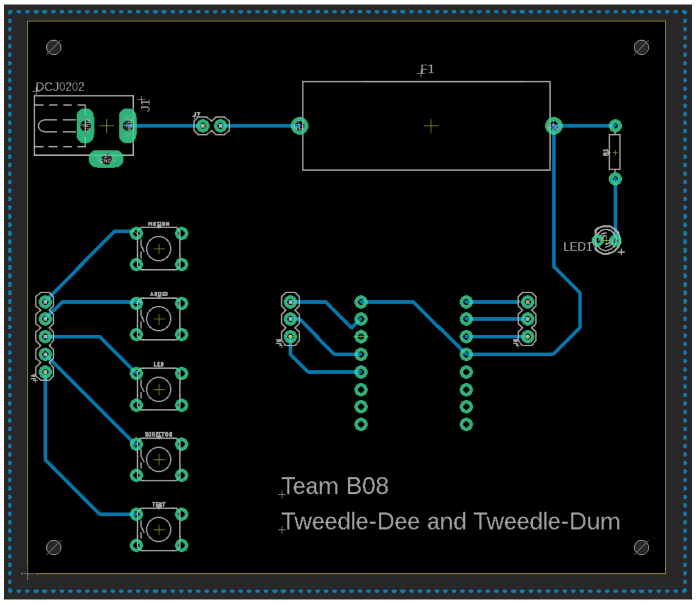


Figure 19: Board Design

Figure 21 is a picture of how the PCB was physically wired to the Raspberry Pi Zero 2 W and other subsystem components such as the wire, LED strip, and the speaker. Mostly female-male wires were used for the connections, and zip ties were used for wire management.

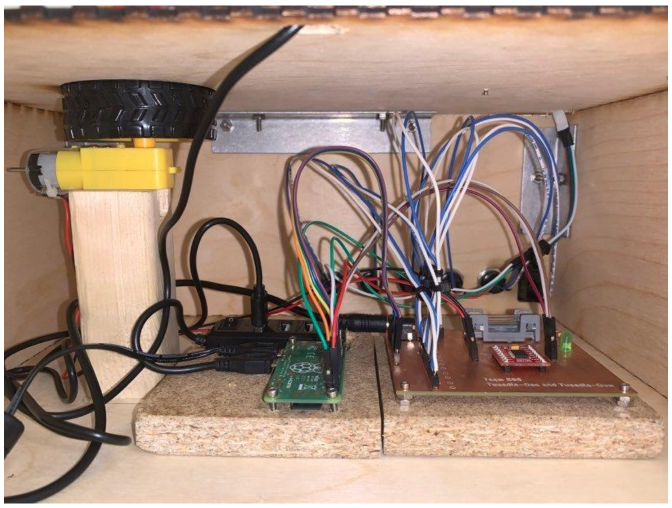


Figure 21: Physical Electrical Components

Mechanical Design

The mechanical design features Tweedledee and Tweedledum rotating upon two platforms to create a shrinking and growing effect. The platforms will rest upon a wooden box that contains all the electrical components such as a PCB, a Raspberry Pi, and a DC motor which will be attached to one of the platforms to control the motion and speed of the characters. The box will have an on/off switch, 5 test pushbuttons, an LED strip, and a speaker.

Before the project proposal, our team brainstormed ideas for the mechanical design and created the sketch in Figure 22.

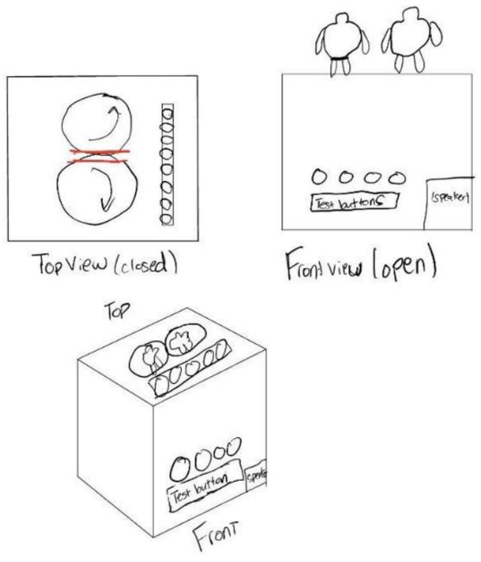


Figure 22: Initial Mechanical Design

To create the rotating platforms, the group decided that gears would be the best method to achieve this movement. A 3D model gear was created in SolidWorks with a diameter of about

12cm, and a prototype was laser cut with scrap wood. Figure 23 shows the CAD design and the prototype gears.

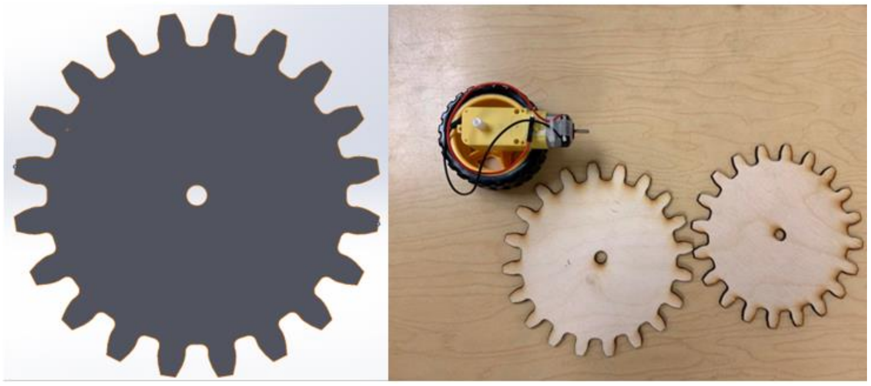


Figure 23: Initial Mechanical Design

After seeing the gear prototypes and simulating the movement of Tweedledee and Tweedledum, we concluded that the gears should be bigger. We increased the gear size by 5cm to 17 cm each. With this measurement in mind, design measurements were added to the mechanical design as seen in Figure 24.

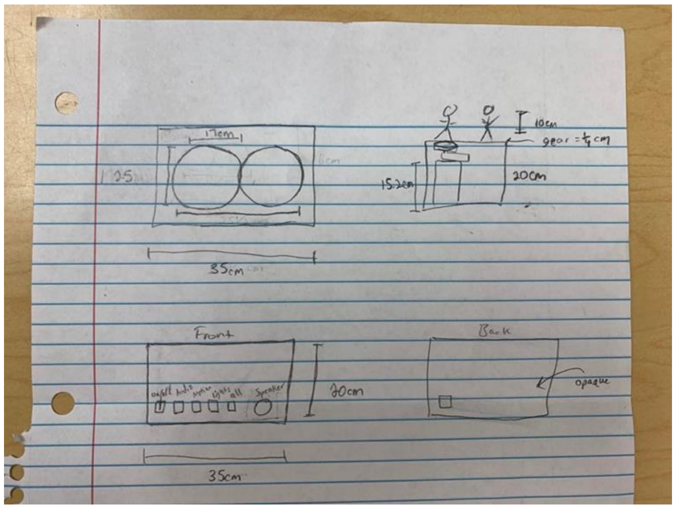


Figure 24: Mechanical Design with Measurements

A prototype of the front side of the box was made with cardboard to make sure the dimensions of our design were sufficient. Overall, the design of the box was bigger than we wanted, so we decreased the height to 15cm and the length to 34cm and created the prototype in Figure 25.

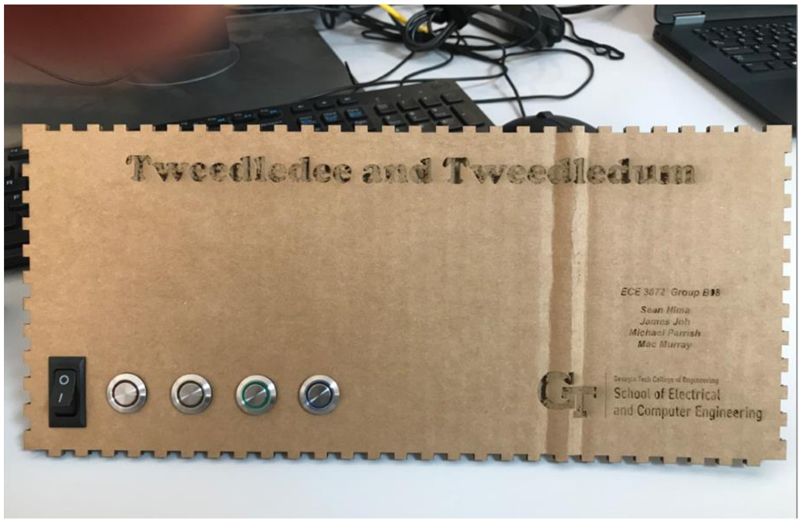


Figure 25: Frontside of Box Prototype

We decided to use all elements above in the mechanical design for the box and below is the final product.

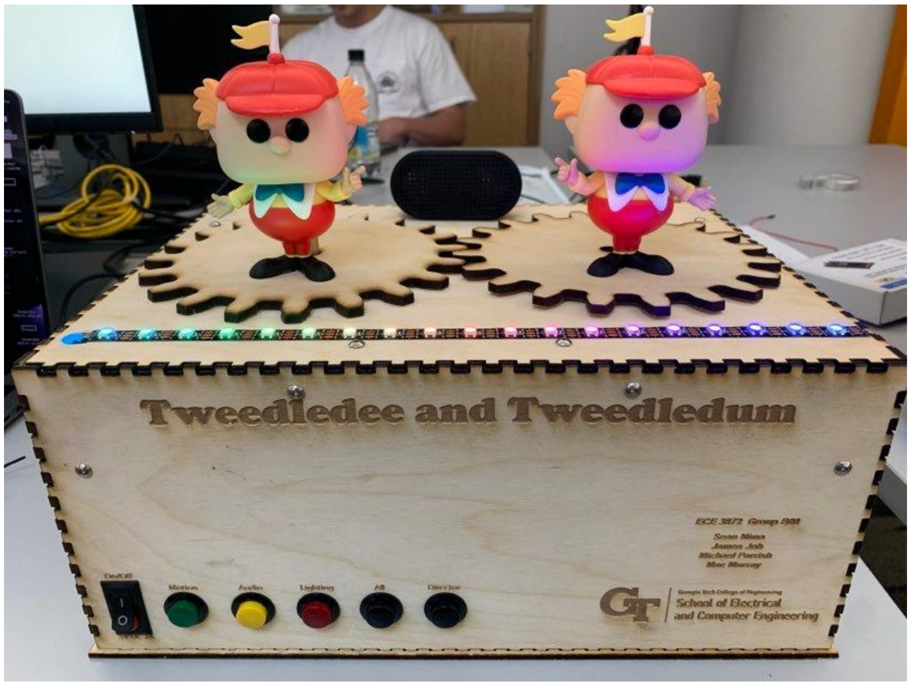


Figure 26: Frontside of Box Final

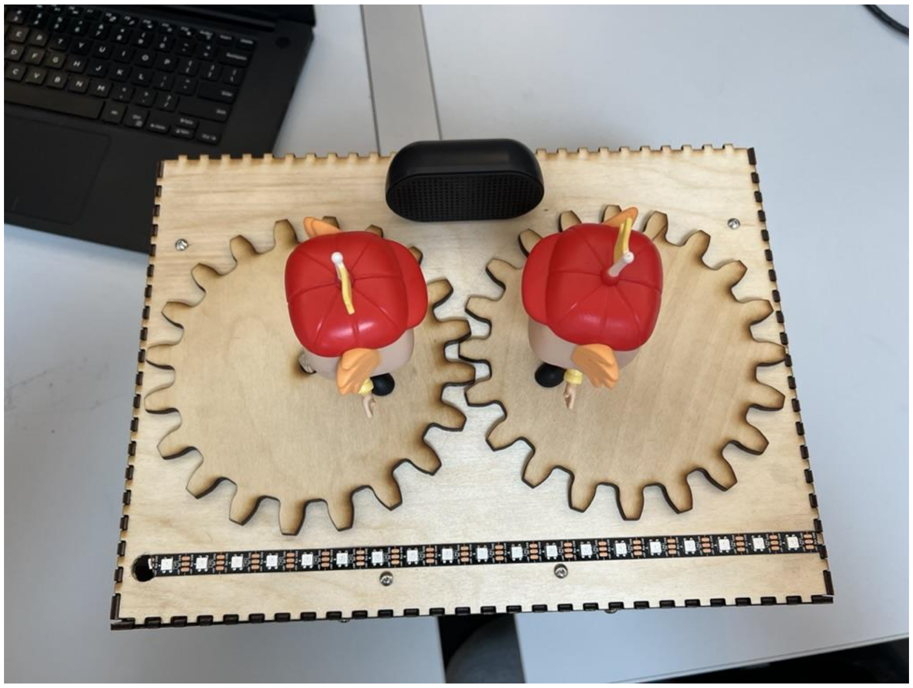


Figure 27: Top View of Box Final

Integration

All the sub-systems must be able to communicate and operate in sync with the other systems. To ensure they work properly, we will include pushbutton test for individual sub-systems along with a failsafe push-button for the sub-systems to operate together on. The software had already been simulated and shown to operate properly, so our main goal is to have the hardware aspects of the system behave properly.

Overall, integration went smoothly. We ran into an issue with the audio sub-system and ended up having to change it completely to be able to run our system completely in parallel. Our team decided to move from an audio amplifier system to a USB-speaker since we learned after research and testing that the amplifier was not very compatible with the micro-controller we had inside of our model. We also ran into issues with PCB plates burning out due to incorrect routings and failure of systems to operate due to poor soldering.

To keep track of all our software files, we used a GitHub repository located at https://github.gatech.edu/snima3/ece3872design.

Conclusion

Over the course of 12 weeks, our team was able to successfully design and build what we call the Tweedle-Box. During this time, we broke our project up into smaller tasks and assigned leads that had existing background knowledge in that area. Despite this, our team ended up facing many challenges throughout the way. In software, we had known that learning python would have to be required and we struggled as a group coding the robot at first but after enough research, we ended up getting proper code working. For our PCB design, we ran into cases where our PCB circuit had fried out and we needed to create a new one as soon as possible. Our mechanical design went through multiple different iterations as we were building it to create housing for the ever-changing parts that were implemented into the system.

If given the project again, our team would likely approach things differently. We would spend more time exploring the Junior Design Museum to learn what sub-system methods worked for the people in the past to have working parts off the bat. Although most of our systems worked, we spent a lot of time on the audio sub-system to get it working with an audio driver but ended up using a one-hour solution with a USB-speaker. We also would do a better job of being more thorough inside of our design notebook at the beginning of the project phase. At the beginning, we exclaimed a lot of different ideas with no follow-up and ended up burning most of the ideas that took us multiple days to come up with due to lack of information.